Fall 2020

EE 210
Mini-Project #: 01

Discrete Convolution & Discrete-time Fourier Transform

Last Name: **Aldacher**
First Name: **Muhammad**
ID: **011510317**

Date: **10/24/2020**

TABLE OF CONTENTS

SECTION 1: ABSTRACT **5**

SECTION 2: INTRODUCTION **5**

SECTION 3: DESIGN **5**

3.1 Discrete Convolution 5
3.2 Discrete-Time Fourier Transform 5

SECTION 4: RESULTS **6**

4.1 Discrete Convolution 6
4.2 Discrete-Time Fourier Transform 9

SECTION 5: CONCLUSION **15**

APPENDIX A: MATLAB CODES **16**

A.1: Discrete Convolution – "my_conv" Function: 16
A.2: Discrete Convolution – Main Code: 16
A.3: DTFT – "my_dtft" Function: 17
A.4: DTFT – Main Code: 17

REFERENCES: **20**

Section 1: Abstract

This project is about designing generalized MATLAB codes that perform discrete convolution and discrete-time Fourier transform (DTFT) to audio and voice signals. Signal-processing MATLAB functions like "conv", "filter", and "fir1" are used to manipulate the input voice signal with different filters and study the output spectrum. This project can be used as the basis for future signal-processing and filter development applications.

Index Terms— Discrete-time, Fourier-Transform, Convolution, Signal-Processing, Matlab, Filters.

Section 2: Introduction

Discrete-time Fourier Transform (DTFT) is a type of Fourier transform that deals with discrete data in the time domain and is used to find its spectrum in frequency domain. Unlike continuous signals, discrete-time signals consist of samples of data evenly spaced in time domain by a sampling period T_S. The discrete signal is a result of a multiplication of the continuous signal with a train of delta functions evenly spaced by T_S. Using DTFT, the discrete signal is represented by a repetition of the spectrum of the continuous signal that repeats every f_S, which is the sampling frequency. In the case of DTFT, the spectrum in the frequency domain is continuous. The concept of DTFT is a crucial component in the signal-processing toolbox which is vital for every application in today's technology.

Section 3: Design

This section will explain the idea behind designing the MATLAB codes for both parts of the project. The MATLAB codes used are provided in Appendix A.

3.1 Discrete Convolution

The discrete convolution deals with 2 discrete-time signals in the manner shown in equation 1. Convolutions are basically multiply-and-accumulate (MAC) operations, where one of the 2 signals is flipped around the y-axis, then is moved towards the other signal in steps. At each step, corresponding samples are multiplied and the output sample is produced by summing those products. The length of the resulted output will be the sum of the 2 signals' lengths minus one as shown in equation 2.

$$y[n] = x[n] * h[n] = \sum_{\tau=-\infty}^{\infty} x[\tau].h[n-\tau] \quad (1)$$

$$\text{length(y)} = \text{length(x)} + \text{length(h)} - 1 \quad (2)$$

3.2 Discrete-Time Fourier Transform

The discrete-time Fourier transform follows equation 3. The code is implemented by using 2 for loops, the 1st loop is to go over each n and do the multiplication of the input x with the complex component, and the 2nd loop is to go over each Ω and accumulate the products. Ω represents the digital filter defined in equation 4.

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n] \cdot e^{-j\Omega n} \quad (3)$$

$$\Omega = \frac{2\pi f}{fs} \quad (4)$$

Section 4: Results

4.1 Discrete Convolution

In part (a) and (b) of the project, it is required to plot the input x[n], the filter's impulse response h[n], and the output y[n]. Using the stem function, x[n] and h[n], defined in equations 5 and 6, are plotted as in Fig. 1. The output y[n] is generated using the "my_conv" function as the convolution of x[n] and h[n]. The output y[n] is shown in Fig. 2.

$$x = ones(1,20) \tag{5}$$

$$h = \frac{1}{4}.ones(1,4) \tag{6}$$

$$y = x * h \tag{7}$$

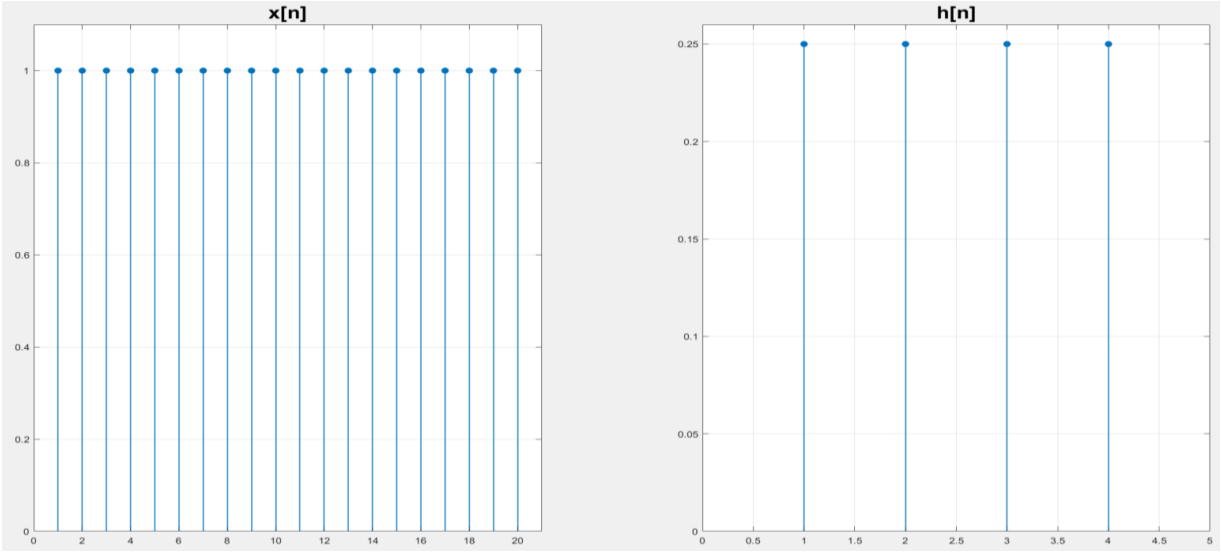


Fig.1. Stem of x[n] and h[n]

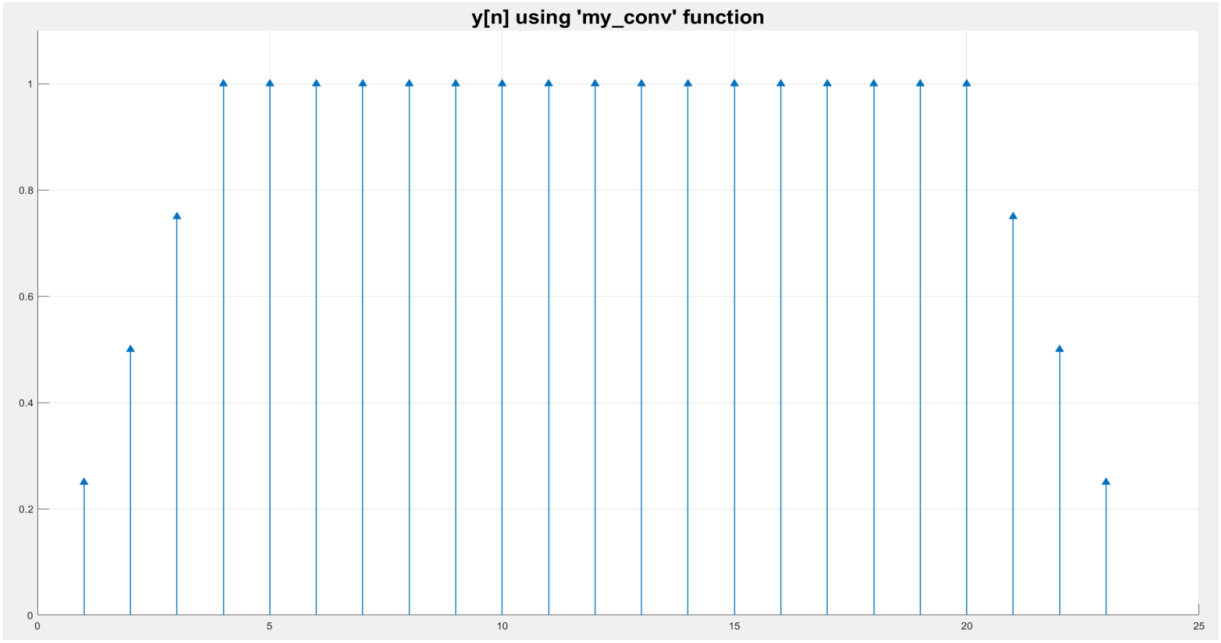


Fig.2. Output y[n] using "my_conv" on x[n] and h[n]

In part (c), a comparison is made between the output y[n] generated by the user-defined "my_conv" function and the MATLAB built-in function called "conv". From Fig. 3, the output is the same in both cases, proving that the user-defined function "my_conv" is working correctly.

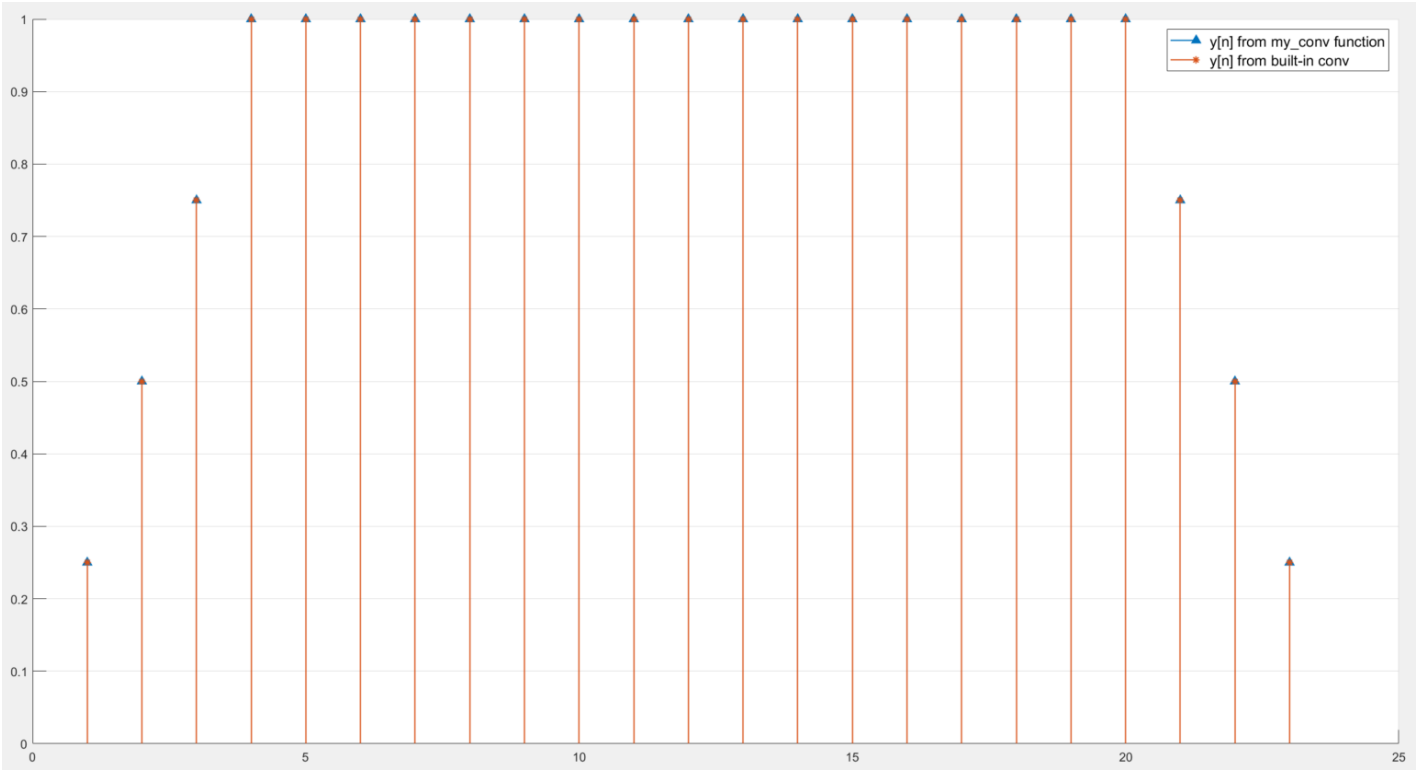


Fig.3. Comparison between y[n] generated by "my_conv" function and y[n] generated by the built-in function "conv"

For part (d), the "help filter" command in Matlab is used to understand how the "filter" function is used. Basically, the "filter" function produces the output y[n] by passing the input x[n] through a digital filter representation from the coefficients vector of the input x and the coefficients vector of the output y in the difference equation shown in equation 8. The format for the command is **y = filter(B,A,x)**, where B and A are the input and the output coefficients vectors respectively.

$$a(1).y(n) = b(1).x(n) + b(2).x(n-1) + \ldots + b(n_b+1).x(n-n_b) - a(2).y(n-1) - \ldots - a(n_a+1).y(n-n_a) \quad (8)$$

Since $H(Z) = \frac{Y(Z)}{X(Z)}$, then B represents the coefficients vector of the numerator and A represents the coefficients vector of the denominator in the filter's transfer function. So, writing **y = filter(h,1,x)** actually does a convolution between x and h, then produces y.

In part (e), a comparison is made between the output y[n] generated by the user-defined "my_conv" function and the MATLAB built-in function called "filter". Fig. 4 shows the difference between y[n] generated by the 2 methods.

Fig.4. Comparison between y[n] generated by "my_conv" function and y[n] generated by the built-in function "filter"

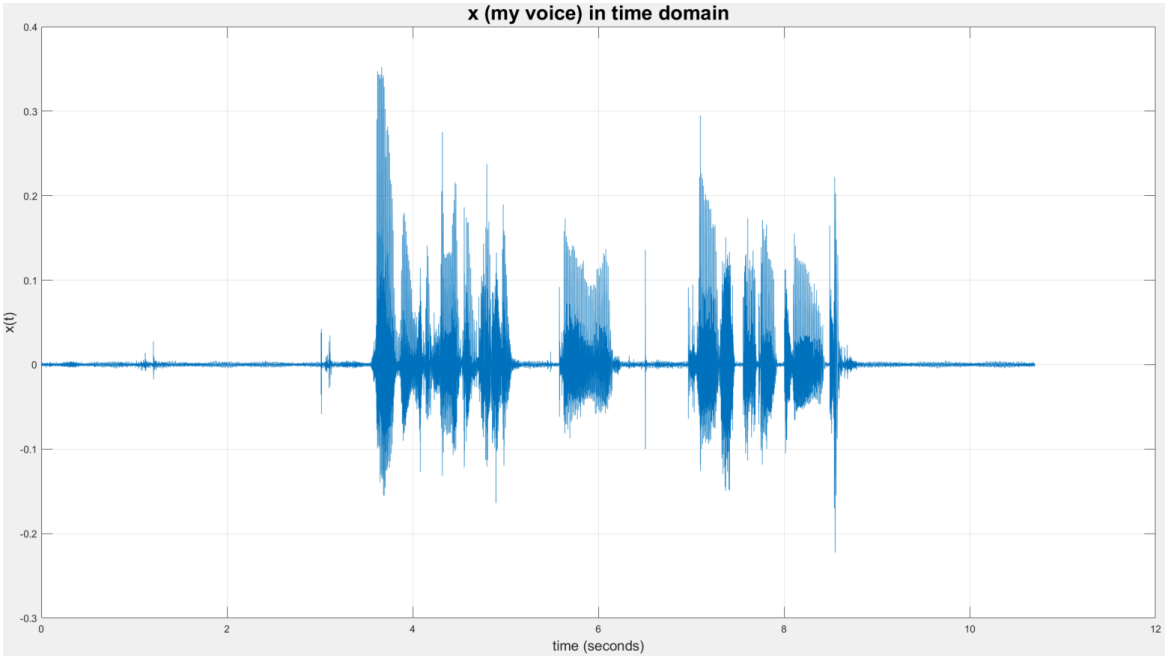
"my_conv" (or "conv") function performs the convolution y[n] = h[n]*x[n] using the command **y=conv(h,x)** and produces the complete output y[n] with a number of samples equal to the sum of sample numbers of x[n] & y[n] minus 1, as shown in equation 9. **"filter"** function performs the convolution y[n] = h[n]*x[n] using the command **y=filter(h,1,x)** and produces an output y[n] with a number of samples equal to the number of samples of the input x[n], as shown in equation (10).

Using "my_conv"/"conv": **length(y) = length(x) + length(h) -1** (9)

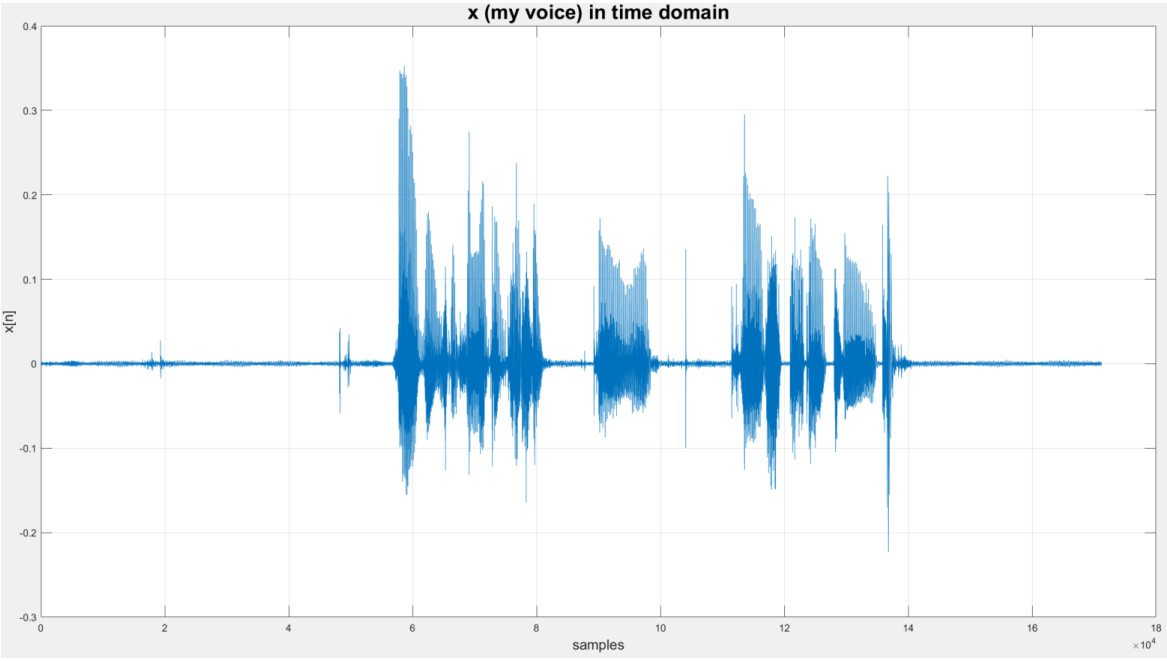
Using "filter": **length(y) = length(x)** (10)

4.2 Discrete-Time Fourier Transform

In part (f) of the project, it is required to plot a recorded voice signal x[n] in time domain, then perform the discrete-time Fourier transform on this signal using the user-defined function that contains 2 for-loops and plot the magnitude and the phase of its spectrum for part (g) of the project. Fig. 5 shows the voice signal plotted in time domain, while Fig. 6 shows the magnitude and the phase of the signal's frequency response.

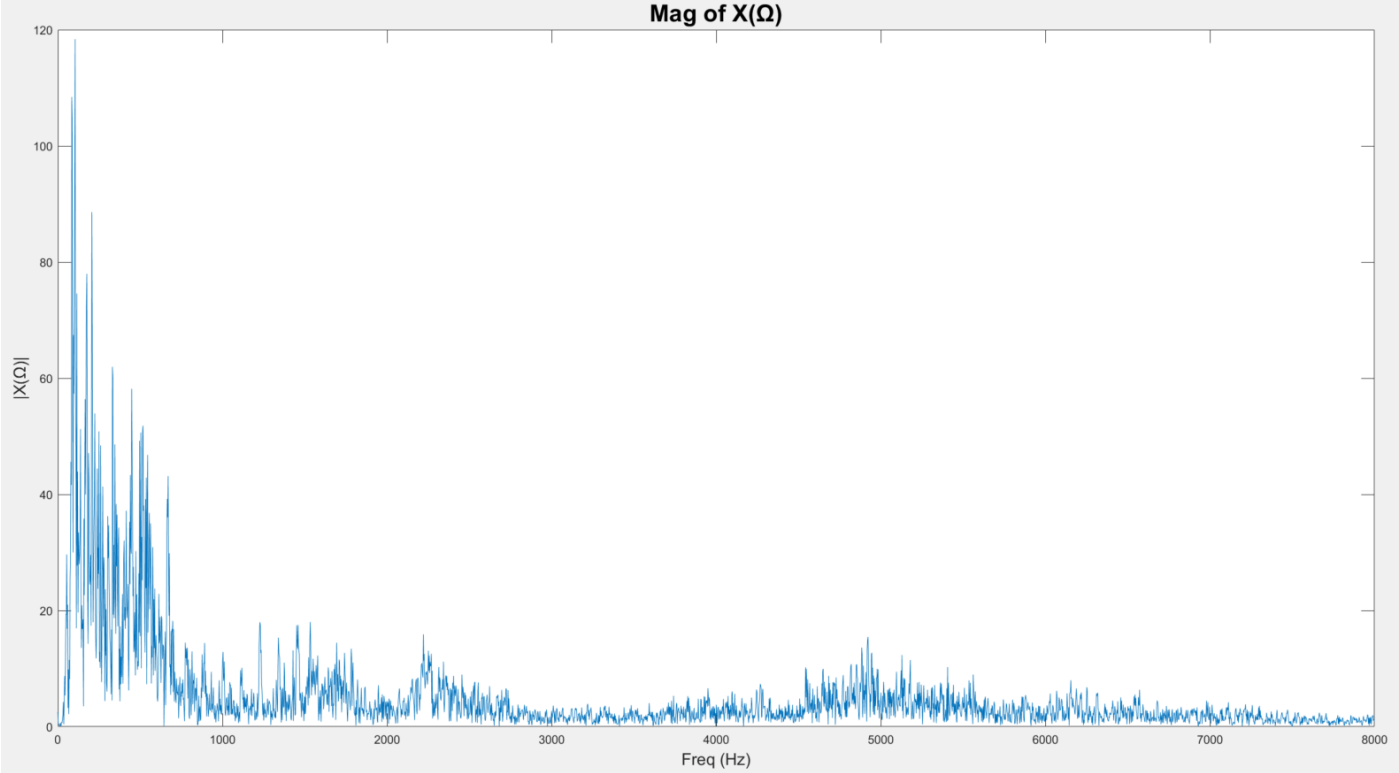


(a)

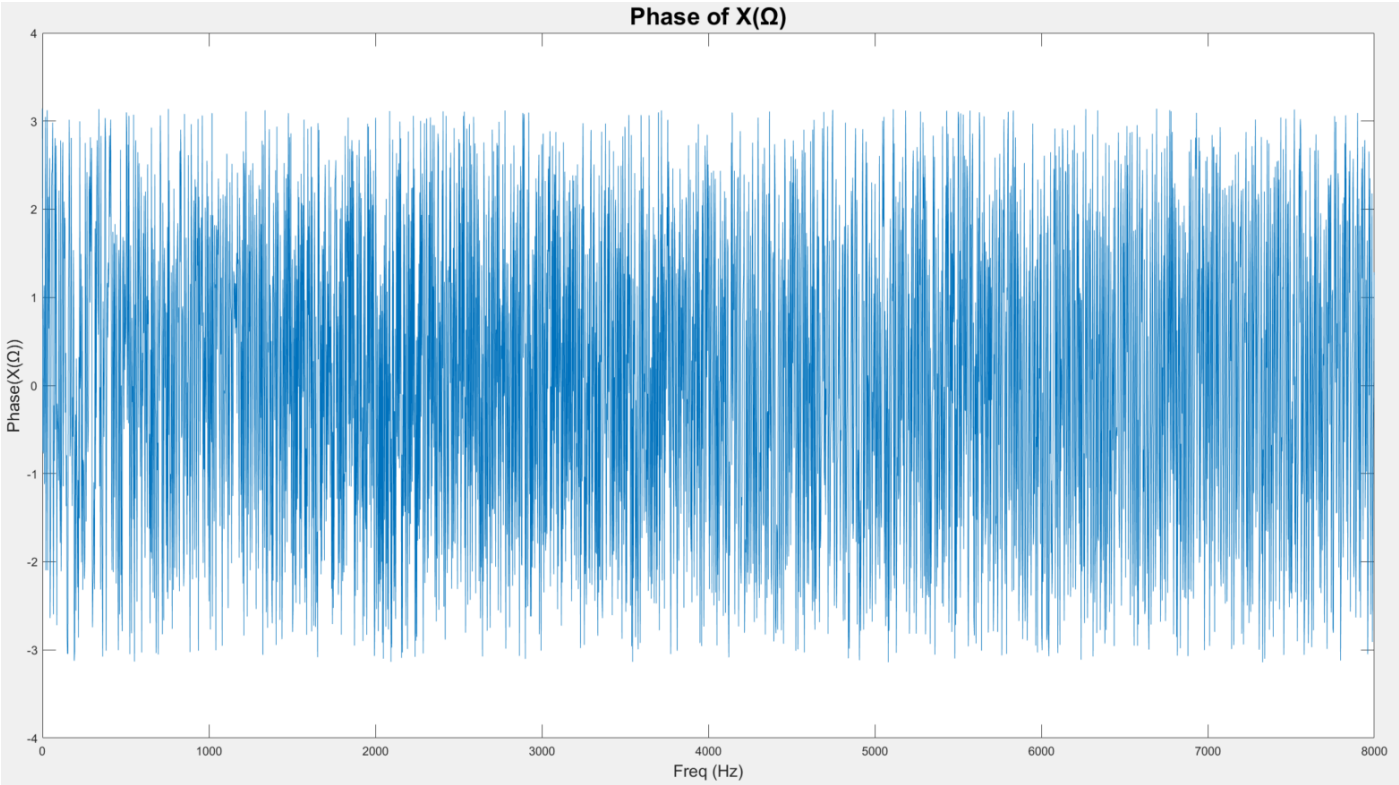


(b)

Fig.5. Voice plotted in time domain (a) vs. time in seconds (b) vs. samples



(a)



(b)

Fig.6. Frequency response of the voice signal, X(Ω), ($0 \leq f \leq fs/2$)
(a) Magnitude (b) Phase

Using the "fir1" function in the following format: **h = fir1(N,Wn,'low')**, a digital filter representation is created. The filter produced is of an N^{th} order filter and of length N+1, with a normalized cut-off frequency of Wn. The 3rd term 'low' determines that the filter is a lowpass filter. For the example mentioned in part (i), in the command **h1 = fir1(100,500/8000,'low')**, 100 represents the order of the filter & determines the length of h1 to be 101. 500/8000 represents the cutoff frequency of the filter normalized to fs/2. Since our chosen sampling frequency is 16000, then fs/2 = 8000, then 500 is the actual cutoff frequency of the lowpass filter.

For parts (j) and (k) of the project, the filters defined in equations 11, 12, 13, and 14 are plotted in time domain using the "stem" function in Fig. 7. Their corresponding frequency responses are obtained by performing DTFT and are plotted in Fig.8.

$$h_1 = \text{fir1}(100, 500/8000, \text{'low'}) \quad (11)$$
$$h_2 = \text{fir1}(100, 1000/8000, \text{'low'}) \quad (12)$$
$$h_3 = \text{fir1}(100, 2000/8000, \text{'low'}) \quad (13)$$
$$h_4 = \text{fir1}(100, 3000/8000, \text{'low'}) \quad (14)$$

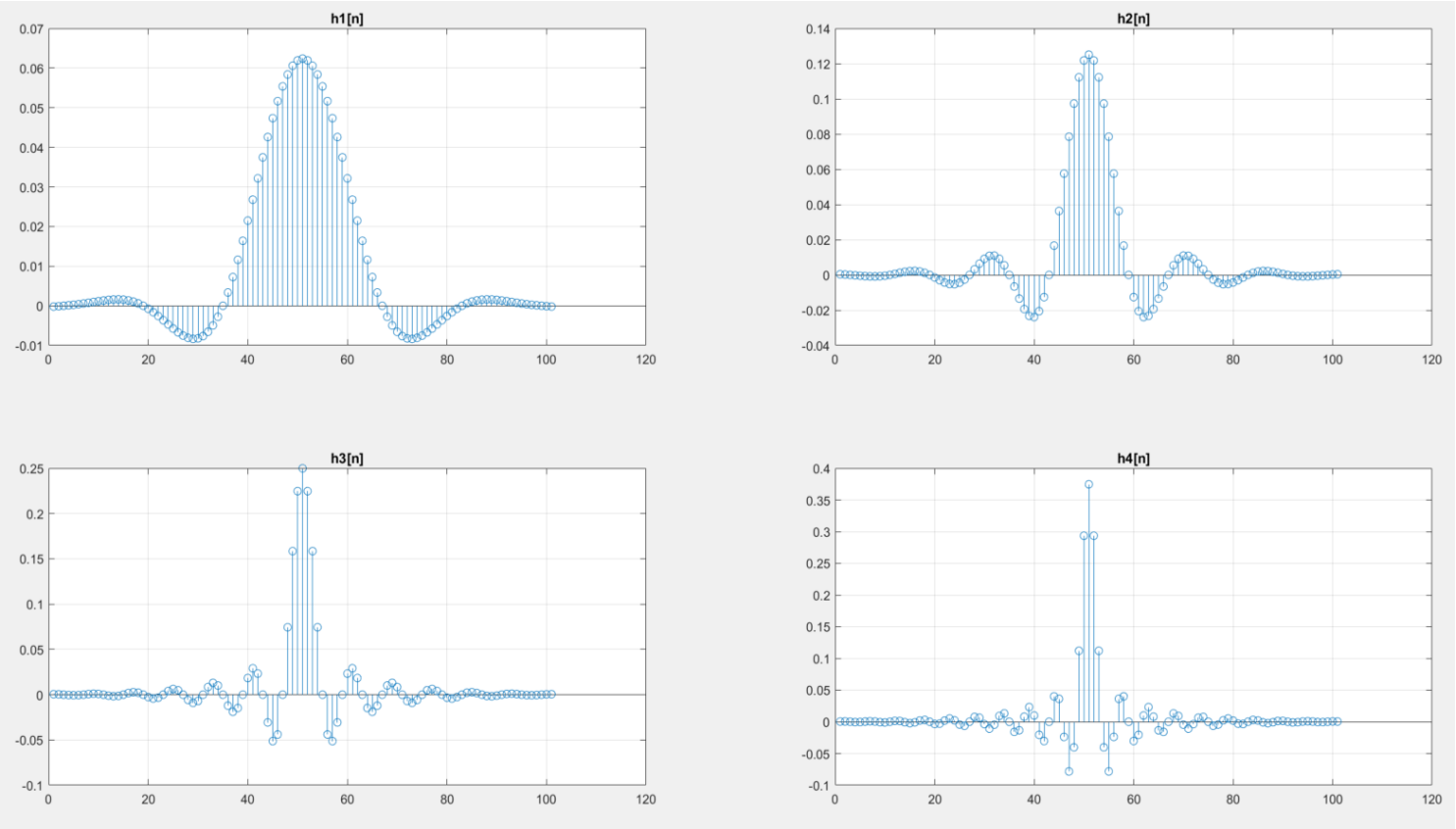
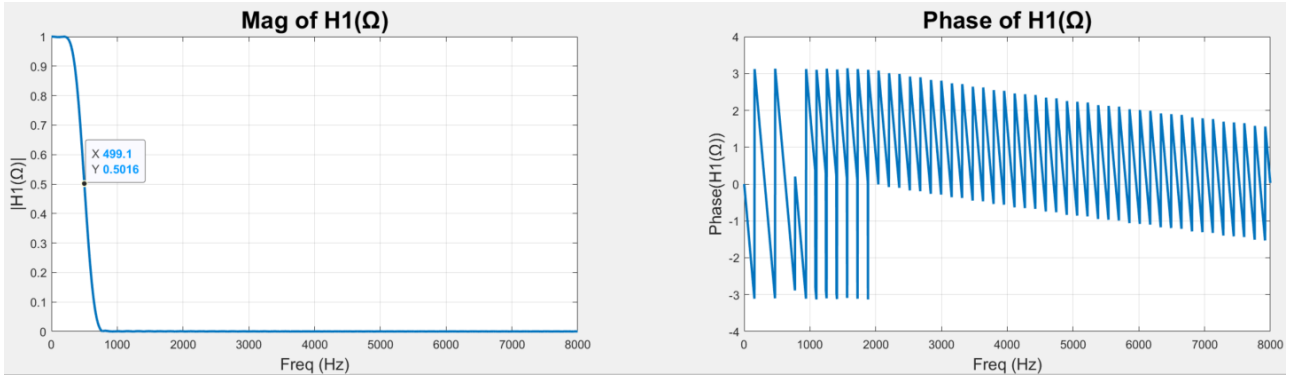
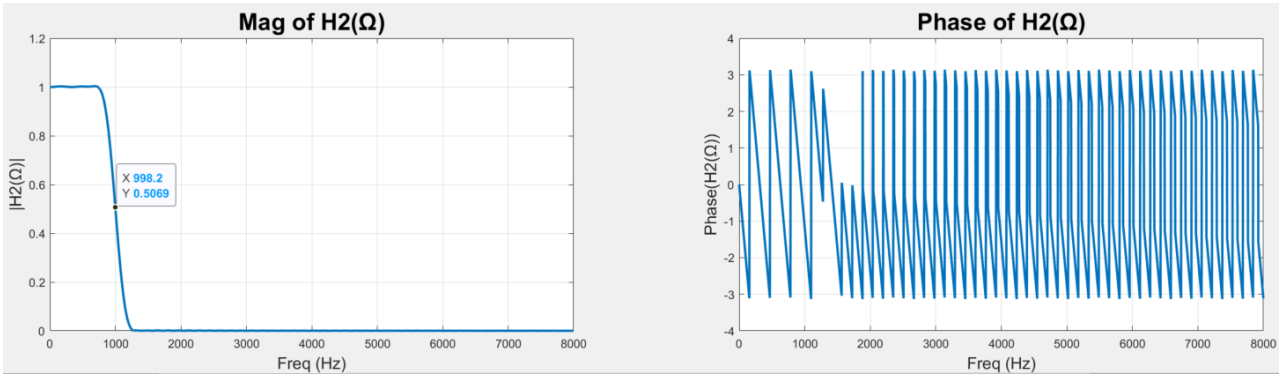


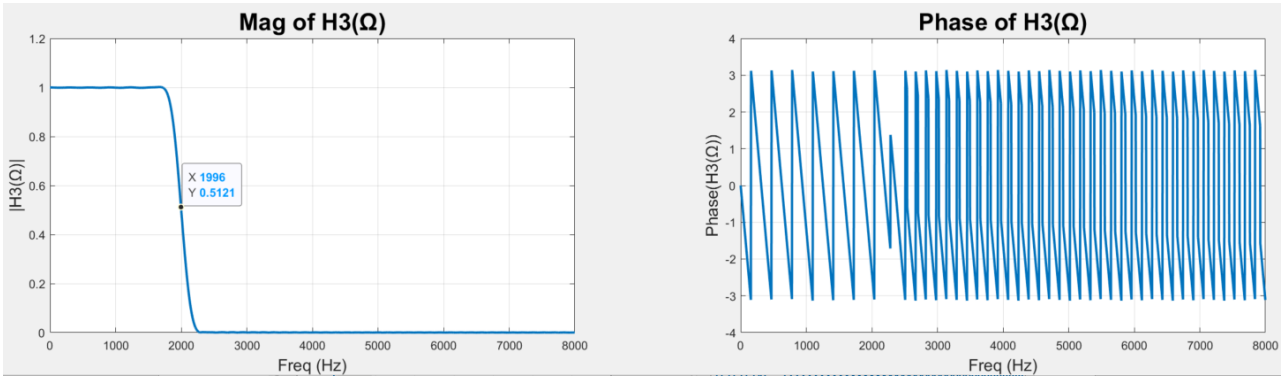
Fig.7. Stem of h1[n], h2[n], h3[n], and h4[n] (in time domain)



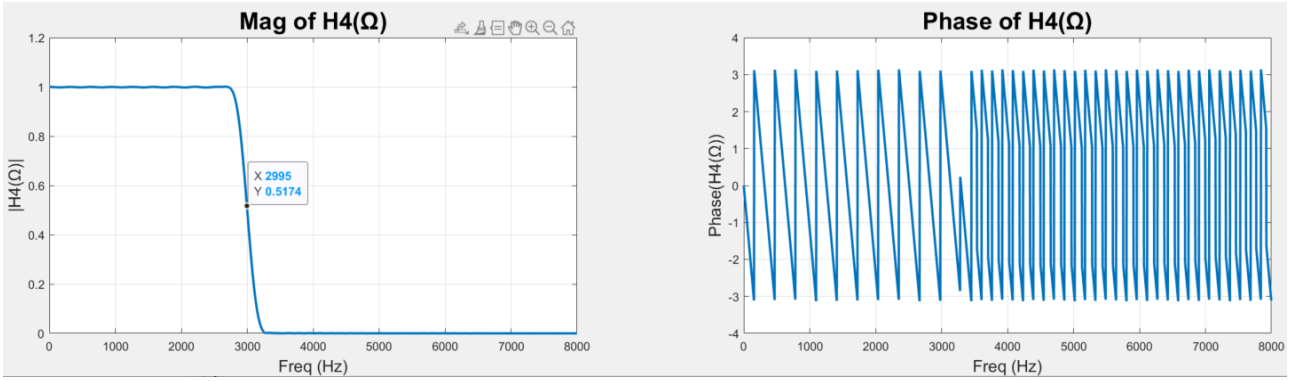
(a)



(b)



(c)



(d)

Fig.8. Frequency response of each filter H(Ω) in frequency domain (Magnitude & Phase)

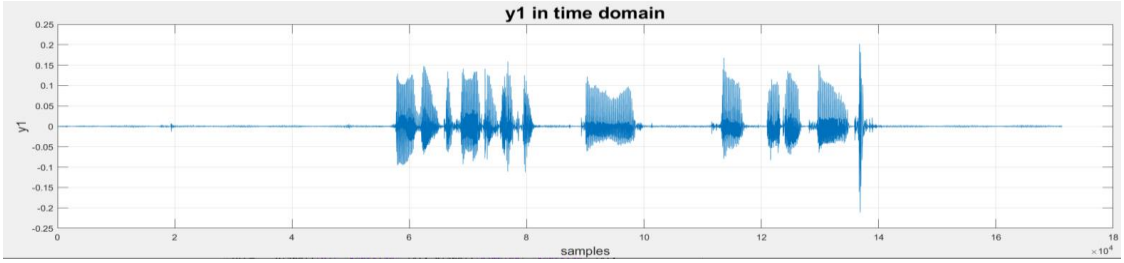
In part (l), the voice signal x[n] is convoluted with each filter h[n] using the previously created function “my_conv” to produce 4 outputs y1,y2,y3, and y4, as described in equations 15, 16, 17, and 18, respectively. The outputs are plotted in time domain in Fig. 9. After performing DTFT on the outputs, the frequency responses of the outputs are plotted in Fig. 10. Although the time-domain plots show very slight differences, the magnitudes in the frequency response clearly show the effect of the different filters on the spectrum.

$$y1[n] = x[n] * h1[n] \quad (15)$$

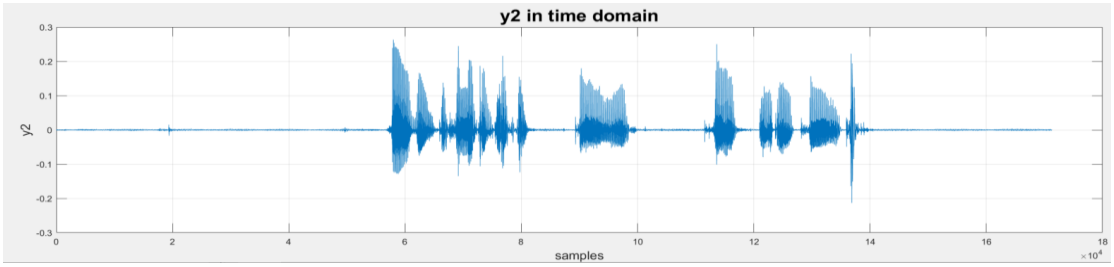
$$y2[n] = x[n] * h2[n] \quad (16)$$

$$y3[n] = x[n] * h3[n] \quad (17)$$

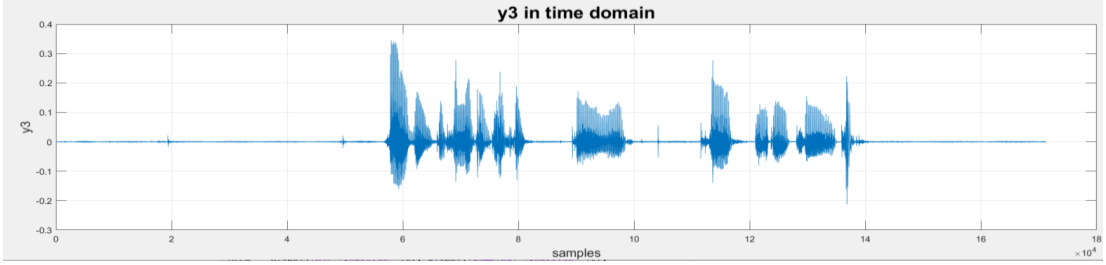
$$y4[n] = x[n] * h4[n] \quad (18)$$



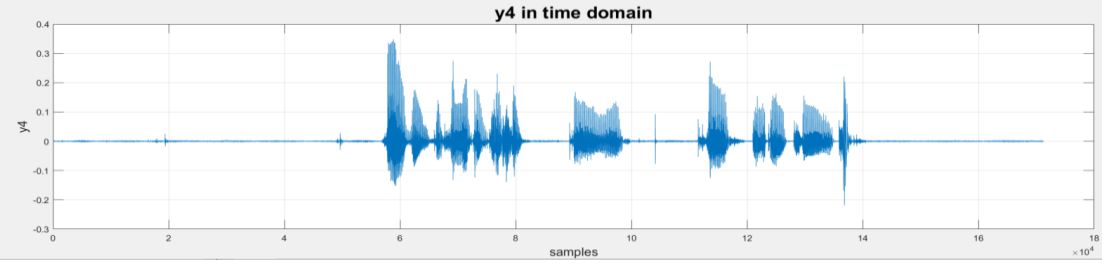
(a)



(b)

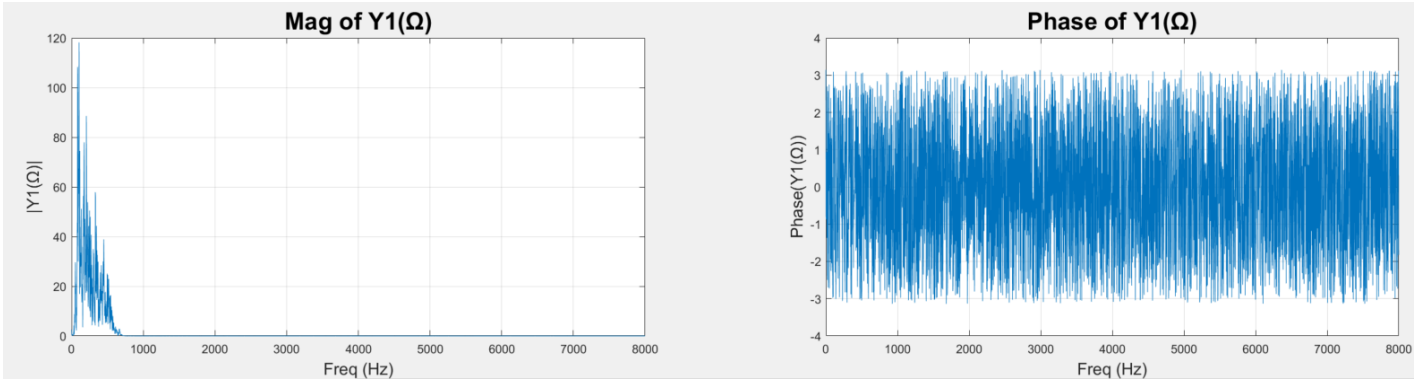


(c)

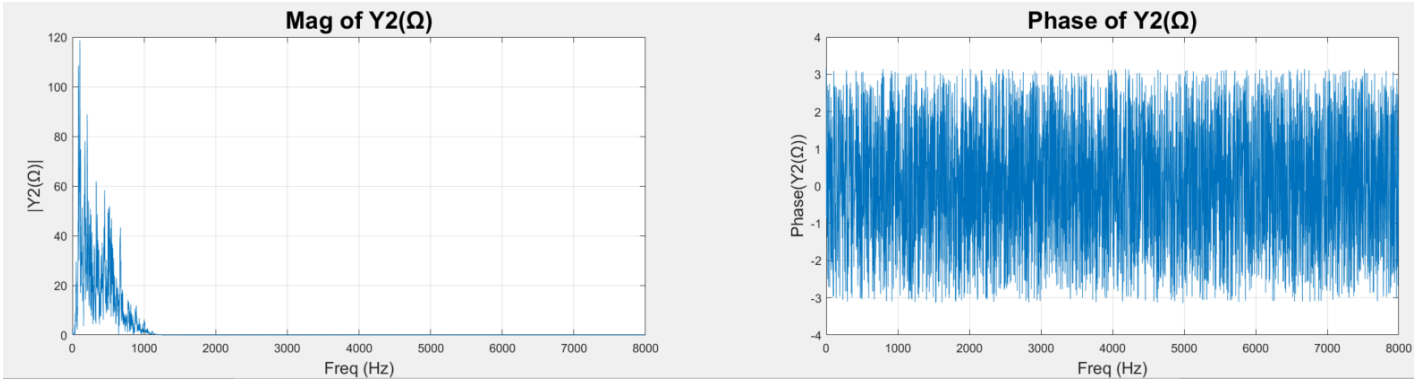


(d)

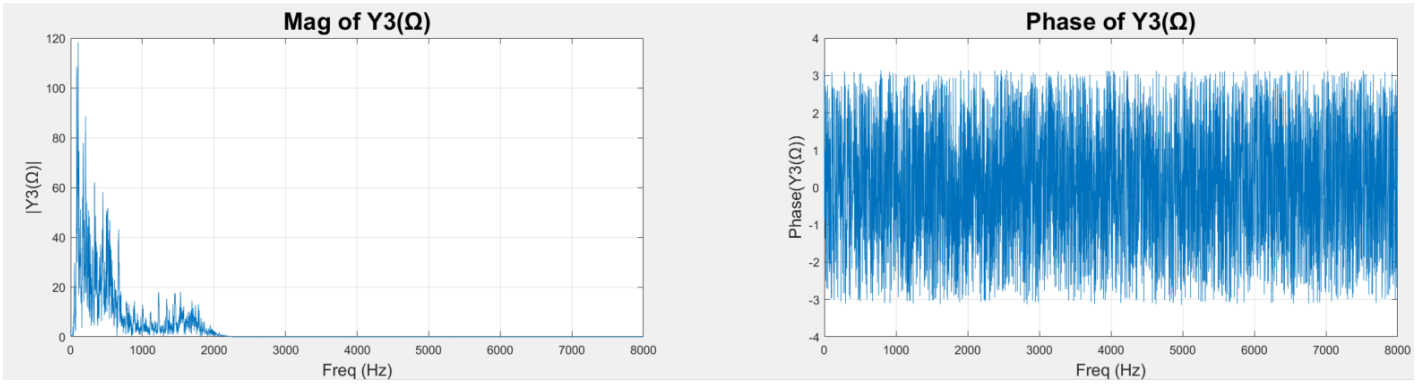
Fig.9. Plots of the convolution of the voice signal (x[n]) with the 4 different filters in time domain



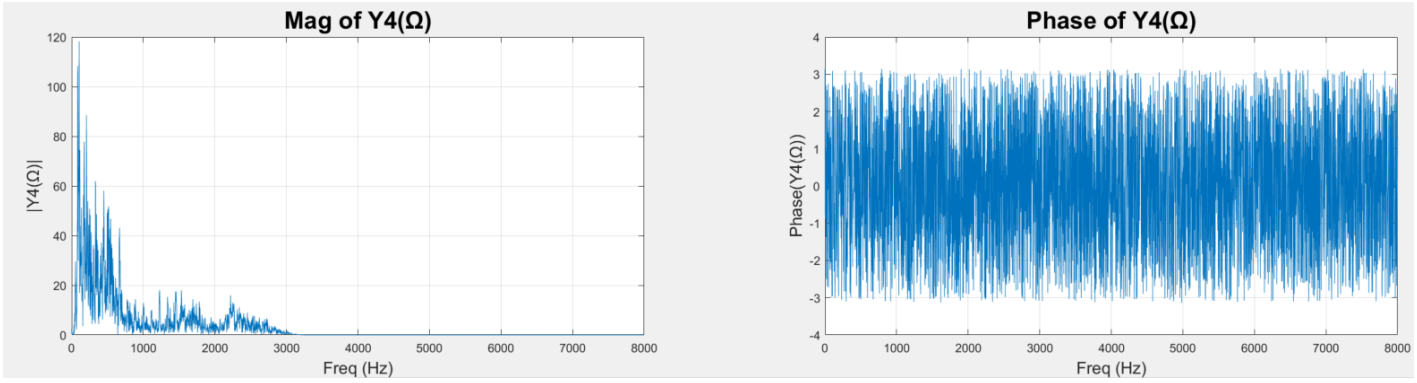
(a)



(b)



(c)



(d)

Fig.10. Plots of the frequency response of the output signals Y(Ω)

Listening to the outputs of the convolution y1, y2, y3, & y4, it is can be noticed how the filters with different cutoff frequencies were able to affect the voice signal. As the cutoff frequency of the filter decreases, more of the frequency components of the original voice signal is filtered out & the output voice signal becomes deeper & less clear. So, y4 is slightly less clear than the original signal x, but the voice is still very recognizable because most of the energy is concentrated in the frequency range below 3000 Hz. The cutoff frequency of h4 is 3000 Hz. But, going to y1, the output voice signal becomes less & less recognizable because more of the energy is being filtered out. Since most of the energy is concentrated in the low frequencies & the cutoff frequency of h1 is 500 Hz, we can still understand the voice message but with worse quality. (m)

Section 5: Conclusion

This project is about performing convolutions on an audio signal with different filters and using discrete-time Fourier transform to study the output signal's spectrum in frequency domain. The learning outcomes of this project include understanding and implementing the convolution process, taking the Fourier transform of a signal, and creating digital filters. During the coding process, the use of MATLAB functions like "conv", "filter", and "fir1" is exercised which helped develop a deeper understanding on how these functions in the signal-processing toolbox can be used. The work and codes in this project have been generalized so they could be used in future related projects. (n)

Appendix A: MATLAB codes

A.1: Discrete Convolution – "my_conv" Function:

```
function [y] = my_conv(x,h)
Nx = length(x);
Nh = length(h);

x = [x zeros(1,Nh-1)];
h = [h zeros(1,Nx-1)];
count = 1;

for i = 1: Nx+Nh-1;
    y(i)=0;
    for j = 1:i;
        y(i)=y(i)+(x(j)*h(i-j+1));
    end
end
```

A.2: Discrete Convolution – Main Code:

```
clc; clear all; close all;

x = ones(1,20);
h = ones(1,4)/4;

subplot(1,2,1); stem(x,'filled','Linewidth',1);
title("x[n]","FontSize",20); axis([0 21 0 1.1]); grid on
subplot(1,2,2); stem(h,'filled','Linewidth',1);
title("h[n]","FontSize",20); axis([0 5 0 0.26]); grid on

y1 = my_conv(x,h);
y2 = conv(x,h);

figure; hold on; grid on;
stem(y1,'^','filled','Linewidth',1);
stem(y2,'*','filled','Linewidth',1);
legend('y[n] from my\_conv function','y[n] from built-in conv')

y3 = filter(h,1,x);
figure; hold on; grid on;
stem(y1,'^','filled','Linewidth',1);
stem(y3,'*','filled','Linewidth',1);
legend('y[n] using my\_conv function','y[n] using filter function')
```

A.3: DTFT – "my_dtft" Function:

```
function [X, OM] = my_dtft(x)

cnt=1;
for my_OM = 0:0.001:pi;
    my_x =0;

    for n = 1:length(x);
       my_x = my_x + x(n)*exp(-j*my_OM*n);
    end

   X(cnt) = my_x;
    OM(cnt) = my_OM;
   cnt = cnt+1;
end
end
```

A.4: DTFT – Main Code:

```
clc;clear all; close all;
[z,fs] = audioread('MiniProject01.wav');
% sound(z,fs)

%----------------------------------------
% Resampling to fs = 16KHz
x = resample(z,16000,fs);
fs = 16000;
audiowrite('newaudio.wav',x,fs)
[x,fs] = audioread('newaudio.wav');
t=0:(1/fs):(length(x)/fs); t=t(1:end-1);
% sound(x,fs)
figure; plot(t,x);title('x (my voice) in time domain',"FontSize",20); grid on;
ylabel('x(t)',"FontSize",14); xlabel('time (seconds)',"FontSize",14);

figure; plot(x);title('x (my voice) in time domain',"FontSize",20); grid on;
ylabel('x[n]',"FontSize",14); xlabel('samples',"FontSize",14);

%----------------------------------------
% Performing DTFT
[X, OM] = my_dtft(x);

X_mag = abs(X);
X_ph = angle(X);
freq = OM*fs/(2*pi);

figure; plot(freq,X_mag);
title('Mag of X('+string(char(937))+')',"FontSize",20);
ylabel('|X('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
figure; plot(freq,X_ph);
title('Phase of X('+string(char(937))+')',"FontSize",20);
ylabel('Phase(X('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);

%----------------------------------------
% the FIR filters
h1 = fir1(100,500/8000,'low');
h2 = fir1(100,1000/8000,'low');
h3 = fir1(100,2000/8000,'low');
h4 = fir1(100,3000/8000,'low');

figure;
subplot(2,2,1);stem(h1); title('h1');
subplot(2,2,2);stem(h2); title('h2');
```

```
subplot(2,2,3);stem(h3); title('h3');
subplot(2,2,4);stem(h4); title('h4');

[H1, OM] = my_dtft(h1);
[H2, OM] = my_dtft(h2);
[H3, OM] = my_dtft(h3);
[H4, OM] = my_dtft(h4);

H1_mag = abs(H1); H1_ph = angle(H1);
H2_mag = abs(H2); H2_ph = angle(H2);
H3_mag = abs(H3); H3_ph = angle(H3);
H4_mag = abs(H4); H4_ph = angle(H4);
freq = OM*fs/(2*pi);

figure;
subplot(1,2,1); plot(freq,H1_mag,'Linewidth',2);
title('Mag of H1('+string(char(937))+')',"FontSize",20); grid on;
ylabel('|H1('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
subplot(1,2,2); plot(freq,H1_ph,'Linewidth',2);
title('Phase of H1('+string(char(937))+')',"FontSize",20); grid on;
ylabel('Phase(H1('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);

figure;
subplot(1,2,1); plot(freq,H2_mag,'Linewidth',2);
title('Mag of H2('+string(char(937))+')',"FontSize",20); grid on;
ylabel('|H2('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
subplot(1,2,2); plot(freq,H2_ph,'Linewidth',2);
title('Phase of H2('+string(char(937))+')',"FontSize",20); grid on;
ylabel('Phase(H2('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);

figure; grid on;
subplot(1,2,1); plot(freq,H3_mag,'Linewidth',2);
title('Mag of H3('+string(char(937))+')',"FontSize",20); grid on;
ylabel('|H3('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
subplot(1,2,2); plot(freq,H3_ph,'Linewidth',2);
title('Phase of H3('+string(char(937))+')',"FontSize",20); grid on;
ylabel('Phase(H3('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);

figure; grid on;
subplot(1,2,1); plot(freq,H4_mag,'Linewidth',2);
title('Mag of H4('+string(char(937))+')',"FontSize",20); grid on;
ylabel('|H4('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
subplot(1,2,2); plot(freq,H4_ph,'Linewidth',2);
title('Phase of H4('+string(char(937))+')',"FontSize",20); grid on;
ylabel('Phase(H4('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);

%----------------------------------------
% Convolution

y1 = my_conv(x',h1); sound(y1,fs);
y2 = my_conv(x',h2); sound(y2,fs);
y3 = my_conv(x',h3); sound(y3,fs);
y4 = my_conv(x',h4); sound(y4,fs);

figure; plot(y1);title('y1 in time domain',"FontSize",20); grid on;
ylabel('y1',"FontSize",14); xlabel('samples',"FontSize",14);
figure; plot(y2);title('y2 in time domain',"FontSize",20); grid on;
ylabel('y2',"FontSize",14); xlabel('samples',"FontSize",14);
```

```
figure; plot(y3);title('y3 in time domain',"FontSize",20); grid on;
ylabel('y3',"FontSize",14); xlabel('samples',"FontSize",14);
figure; plot(y4);title('y4 in time domain',"FontSize",20); grid on;
ylabel('y4',"FontSize",14); xlabel('samples',"FontSize",14);

[Y1, OM] = my_dtft(y1);
[Y2, OM] = my_dtft(y2);
[Y3, OM] = my_dtft(y3);
[Y4, OM] = my_dtft(y4);

Y1_mag = abs(Y1); Y1_ph = angle(Y1);
Y2_mag = abs(Y2); Y2_ph = angle(Y2);
Y3_mag = abs(Y3); Y3_ph = angle(Y3);
Y4_mag = abs(Y4); Y4_ph = angle(Y4);
freq = OM*fs/(2*pi);

figure;
subplot(1,2,1); plot(freq,Y1_mag);
title('Mag of Y1('+string(char(937))+')',"FontSize",20); grid on;
ylabel('|Y1('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
subplot(1,2,2); plot(freq,Y1_ph);
title('Phase of Y1('+string(char(937))+')',"FontSize",20); grid on;
ylabel('Phase(Y1('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);

figure;
subplot(1,2,1); plot(freq,Y2_mag);
title('Mag of Y2('+string(char(937))+')',"FontSize",20); grid on;
ylabel('|Y2('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
subplot(1,2,2); plot(freq,Y2_ph);
title('Phase of Y2('+string(char(937))+')',"FontSize",20); grid on;
ylabel('Phase(Y2('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);

figure; grid on;
subplot(1,2,1); plot(freq,Y3_mag);
title('Mag of Y3('+string(char(937))+')',"FontSize",20); grid on;
ylabel('|Y3('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
subplot(1,2,2); plot(freq,Y3_ph);
title('Phase of Y3('+string(char(937))+')',"FontSize",20); grid on;
ylabel('Phase(Y3('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);

figure; grid on;
subplot(1,2,1); plot(freq,Y4_mag);
title('Mag of Y4('+string(char(937))+')',"FontSize",20); grid on;
ylabel('|Y4('+string(char(937))+')|',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
subplot(1,2,2); plot(freq,Y4_ph);
title('Phase of Y4('+string(char(937))+')',"FontSize",20); grid on;
ylabel('Phase(Y4('+string(char(937))+'))',"FontSize",14);
xlabel('Freq (Hz)',"FontSize",14);
```

References:

[1] A. Oppenheim and R. Schafer, "Discrete-Time Signal Processing", 3rd Ed., Pearson, 2010.

[2] *MATLAB Signal Processing Toolbox* [Online], Available: https://www.mathworks.com/help/signal/index.html, Accessed: October 22, 2020.