Asam Mahmood

Abstract

This project's aim is to filter the nose from the signals.The is mainly used for filtering operations and its functionality can easily be enhanced as well. In this report a design of filter for the noise removal is illustrated with MATLAB coding. This project provides knowledge about the applications of Filters in audio processing and gives some details from past studies. This will indicate that filter responses can merge together and can form large models or designs. The major objective of the project is to analyze and perform the audio processing.

INTRODUCTION:

This paper portrays the acknowledgment of audio processing with the design of filters. Our framework was executed on the MATLAB. The basic aim was to execute and analyze the audio processing.

Aim of Design

This Design aims to get familiar with the audio processing and design the filter for signal processing along with simulation results using the MATLAB. The motive of this design is to build audio processing and test the functionality of the project using the plots and simulation results.

Objectives of the Design

- To understand the basics audio signal processing and its operation.
- To implement and analyze the filter code on the MATLAB.
- To generate the test simulations.

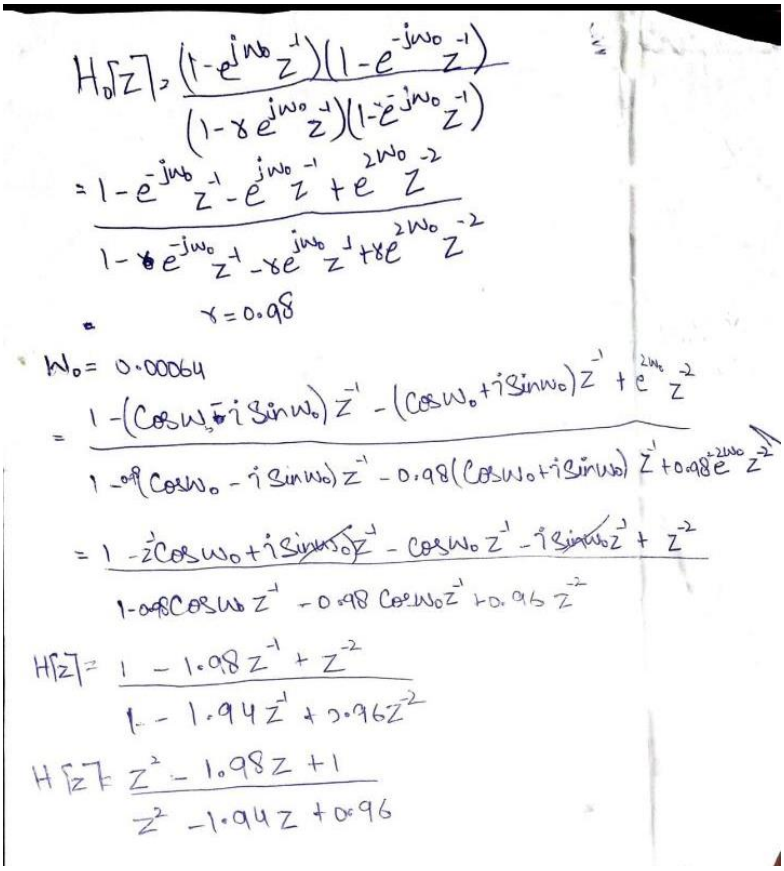
Design Creation Procedure in MATLAB

For creating new projects:

- Go to file
- Select a new project
- File name any other than reserved ones.
- Specify the location and hit next.
- Go to a new project
- Select a new source

Q1. Find ω_0. Explain how you found it.

ANS: At first I thought the value of W0 is 0.314 and I find this value by comparing the value of the equation with 2nd order general equation.Then I went through equation and implementation I found it to be .0006



Q1. (ii) Choose r near but less than 1.r = 0.98 may be a good choice. Experiment with it.

Ans: I did choose the value of r=0.98 as per given instructions.

1-A Read the file TONE.wav into a vector called **speech**. Note its sample rate **sf**. Plot **speech** versus **time** and play it using *soundsc*. Note the length of **speech** as **lengthspeech**. Compute the **nfft**-point DFT of **speech** where **nfft** is 4 times the smallest power of 2 larger than **lengthspeech**. Save it in **dftspeech**.Plot the magnitude **dftspeech** versus normalized frequency ω between 0 and 2π, versus normalized frequency ω between $-\pi$ and π and the actual frequency $F = \frac{\Omega}{}$ between +/- half the sample rate. Comment about the file content based on listening to the file and observing the plots. Note the largest spectral magnitude and its actual and normalized frequency.That is the major interfering tone

```
%clear all
close all
clc
[speech,sf]=audioread('TONE.wav');%reading the audio signal
sound(speech,sf); %Listen to audio file
speechlength = length(speech);% total length of speech
x = speechlength/sf; % x is the sampling period
time = linspace(0, x, speechlength);%finding the length of time
figure(1)
subplot(2,2,1)
plot(time,speech), title('TONE,wave in frequency domain'),ylabel('Amplitude'),
xlabel('Time (secs)');
dft_speech = fft(speech);
nfft = (0:length(speech)-1)*100/length(speech);
subplot(2,2,2)
abs_value=fftshift((abs(dft_speech(:,1))));
plot(nfft,abs_value)
 title('TONE,wave in time domain'),ylabel('Amplitude'), xlabel('dfft speech in
freq (HZ)');
```

1-A Create a sinusoid of the same frequency and listen to it. Compare the tones.

```
 t=0:0.0005:1;
 x=sin(2*pi*100*t);
 sound(x,sf)
 subplot(2,2,3)
 plot(x)
 title('sine wave'),  ylabel('Amp'), xlabel('Time (secs)');
dft_x = fft(x);
subplot(2,2,4)
abs_valu=fftshift((abs(dft_x)));
plot(abs_valu)
 title('sine wave in freq domain'),ylabel('Amplitude'), xlabel('dfft sine in
freq (HZ)');
```

1-A Follow the instructions below to design a filter to remove the interfering tone. The filter has the transfer function

$$(1 - e^{j\omega 0} z^{-1})(1 - e^{-j\omega 0} z^{-1})\, H0(z) = (1 - re^{j\omega 0} z^{-1})(1 - re^{-j\omega 0} z^{-1}).$$

```
 %filter design and simulations
 w0=0.314;
 r=0.98;
 b2=0.96;
 b1=-1.95;
 b0=1;
 a0=1;
 a1=1.99;
 a2=1;
 a=[b2 -b1 b0];
 b=[a2 -a1 a0];
 figure(2)
```

```
 [zero pole gain] = tf2zp(a,b)
 zplane(zero,pole)
 grid on
%apply butter worth filter
[b, a] = butter(6,500/(Fs/2));
freqz(b,a)
% use filter command
```

v. Use MATLAB function *filter* to filter **speech**. Record the output in **clean1**

```
clean1 = filter(b, a, speech);
% Play the sound.
player = audioplayer(clean1, Fs);
```

1-v-a. Plot clean1 versus time

```
info = audioinfo('TONE.wav')
[speech, sf]=audioread('TONE.wav');;
dt = 1/sf;  % Sampling time [s]
t = 0:seconds(1/sf):seconds(info.Duration);
t = t(1:end-1);
plot(t,clean1 );          % Filtered output
title('Filtered Signal ');
xlabel('Time (s)');
ylabel ('Amplitude');
```

1-v-b. Listen to **clean1**.

```
play(player);
```

1-v-c.Compute the **nfft**-point DFT of **clean1** and save it in **dft-clean1**. Plot its magnitude.

```
%compute nff-dfft
y = fft(clean1);
dft_clean1 = abs(y);
%figure(6)
subplot(2,2,1)
plot(dft_clean1)
```

1-B Repeat the filtering as above using $H_1(z)$, $H_2(z)$, $H_3(z)$ to notch the next 3 interfering tones at ω_1, ω_2, ω_3 in cascade. Use time and frequency domain plots and listening tests as necessary and write the results.

To find ω_1, ω_2, ω_3 you may use ”the human in the loop” system, i.e. find the frequencies manually. Alternatively, you may write code to find a local peak. You may contact me for discussion on how.

```
 clean2=filter(a,b,x);
%  figure(4)
 subplot(2,2,2)
```

```
 plot(clean2,'g')
% plr1 = audioplayer(clean2, Fs);
% play(plr1);


 clean3=filter(a,b,speech);
%  figure(5)
 subplot(2,2,3)
 plot(clean3,'yellow')
%  plr = audioplayer(clean3, Fs);
%  play(plr);


%plot H1(z) ,H2(z) and H3(z)
%for H1(z)
 w1=0.12;
 r1=0.98;
 b5=0.96;
 b4=-1.95;
 b3=1;
 a3=1;
 a4=1.99;
 a5=1;
 a=[b2 -b1 b0];
 b=[a2 -a1 a0];
 [zero pole gain] = tf2zp(a,b)
 zplane(zero,pole)
 grid on
%apply butter worth filter
[b1, a1] = butter(6,500/(Fs/2));
freqz(b1,a1)
% use filter command
clean1 = filter(b, a, speech);
% Play the sound.
player = audioplayer(clean1, Fs);
play(player);

%for H2(z)
 w2=0.12;
 r2=0.98;
 b8=0.96;
 b7=-1.95;
 b6=1;
 a8=1;
 a7=1.99;
 a6=1;
 a=[b2 -b1 b0];
 b=[a2 -a1 a0];
 figure(5)
 [zero pole gain] = tf2zp(a,b)
 zplane(zero,pole)
 grid on
%apply butter worth filter
[b, a] = butter(6,500/(Fs/2));
freqz(b,a)
% use filter command
```

```
clean2 = filter(b, a, speech);
% Play the sound.
player = audioplayer(clean1, Fs);
play(player);

%for H3(z)
 w3=0.12;
 r3=0.98;
 b11=0.96;
 b10=-1.95;
 b9=1;
 a11=1;
 a10=1.99;
 a9=1;
 a=[b2 -b1 b0];
 b=[a2 -a1 a0];
 figure(5)
 [zero pole gain] = tf2zp(a,b)
 zplane(zero,pole)
 grid on
%apply butter worth filter
[b1, a1] = butter(6,500/(Fs/2));
freqz(b1,a1)
% use filter command
clean3 = filter(b, a, speech);
% Play the sound.
player = audioplayer(clean3, Fs);
play(player);
%represent into time domain and frequency domain
%y = sin(2*pi*player);
figure(6)
subplot(3,3,1)
plot(clean3)
subplot(3,3,2)
stem(clean3)
```

1-C Obtain the overall filter $H(z)$ in terms of $H_i(z)$, i = 0, 1, 2, 3 and plot its poles and zeros on the z-plane. Plot also its spectral magnitude superposed over the original speech spectrum to show how the filter's zeros and the interfering tones align

```
% (c) obtain overall filter
sys1=clean1+clean2+clean3;
sound(sys1)
%%
```

Question 2

(a) Analyze the segments first.

i.Compute the **fr nfft**-point DFT of each frame by using fft on the matrix **speech frames**. Choose the DFT length using the same criterion you used to choose **nfft** adjusted to the

frame length.

ii.Make a movie out of plots each frame versus time and frequency.
iii.Generate a spectrogram of the frames using *imagesc*.

Comment on your observations

```
%portion 2
clc
[speech,sf]=audioread('TONE.wav');%reading the audio signal
%[ht]=Frames(speech, 200, 0.8,fs); % 2 a i

[xm]=getframe(speech);
nof=size(xm,2);
nfft_fr=2^18;
fxm=fft(xm,nfft_fr);  % 2 a ii

afxm=abs(fxm(1:nfft_fr/2));
t_fr=[0:nfft_fr/2-1]/nfft_fr*sf;
figure
imagesc(t_fr,speech,20*log10(afxm)), axis xy,colormap(jet),colorbar % 2 a iii
figure
for k=1 :nof
    plot(speech,afxm(:,k)) % 2 a iv
    pause
end
```

2-b

Assume that the signal in TONE.wav is available to you in frames in pseudo real-time. Remove the dominant interfering tone one frame at a time. After you filter each frame, collate the output to generate an alternative ”clean” signal that you can listen to. I will discuss this in class.

```
clean = filter(b, a, speech);
% Play the sound.
player = audioplayer(clean3, Fs);
play(player);
```

COMMENTS ON THE OBSERVATIONS

It was observed during the experiment while I was doing on the frames i-e (by filtering images frame by frame) I noticed that the resolution of an image get changed in terms of voice after each successive experiment I noticed that the voice was bit clearer than the previous one.

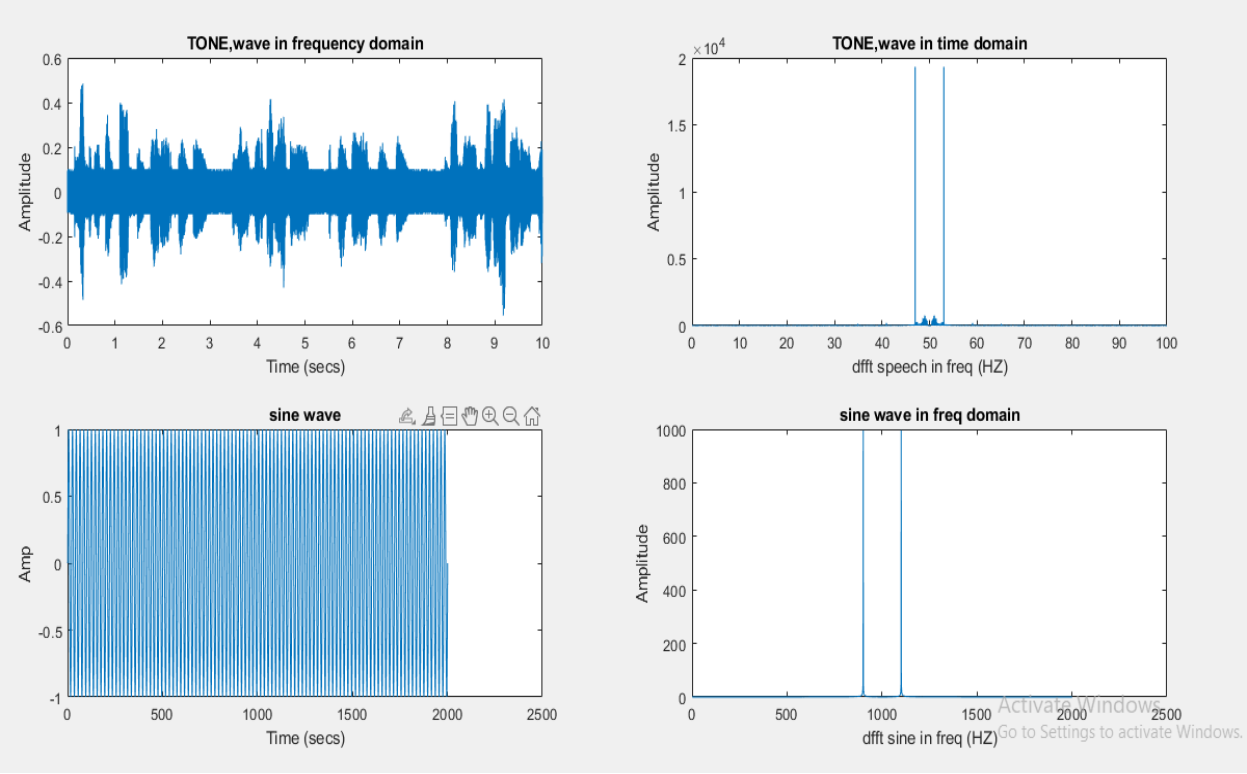


FIGURE1

1.The figure1 has 4 subplots like plot of noise, plot of noise in frequency, the plot of sine wave and the plot of sine wave in frequency domain. The figure2 gives the details of poles and zeros and their locations in the complex plane. The figure3 gives the details of (b) and (c) parts by showing the combined plot filter responses.I also added a script below to print out each graph independently for each section for question 1-A & 1-C.

Q1(iii) Note the poles and zeros of $H(z)$ and plot them on the complex z-plane using MATLAB.

Ans: here are the poles and zeros of the function H(z) bellow.

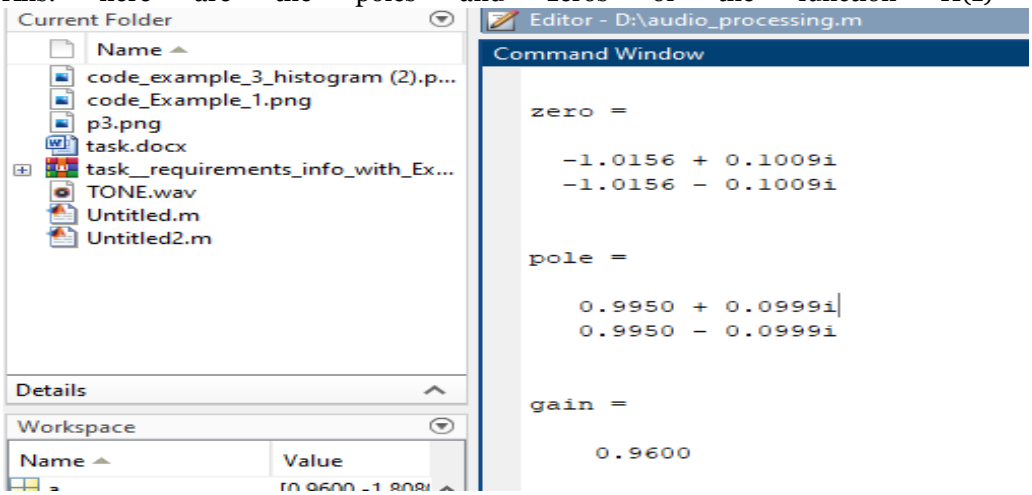
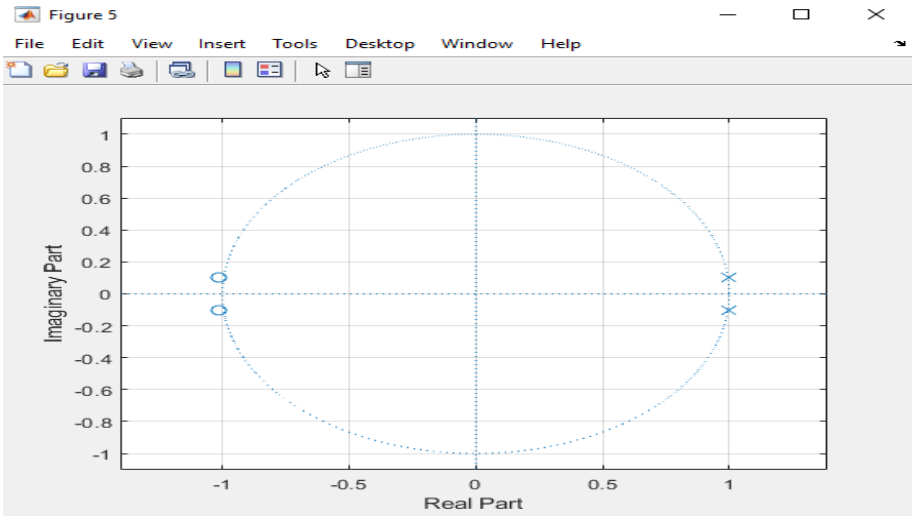


Figure2



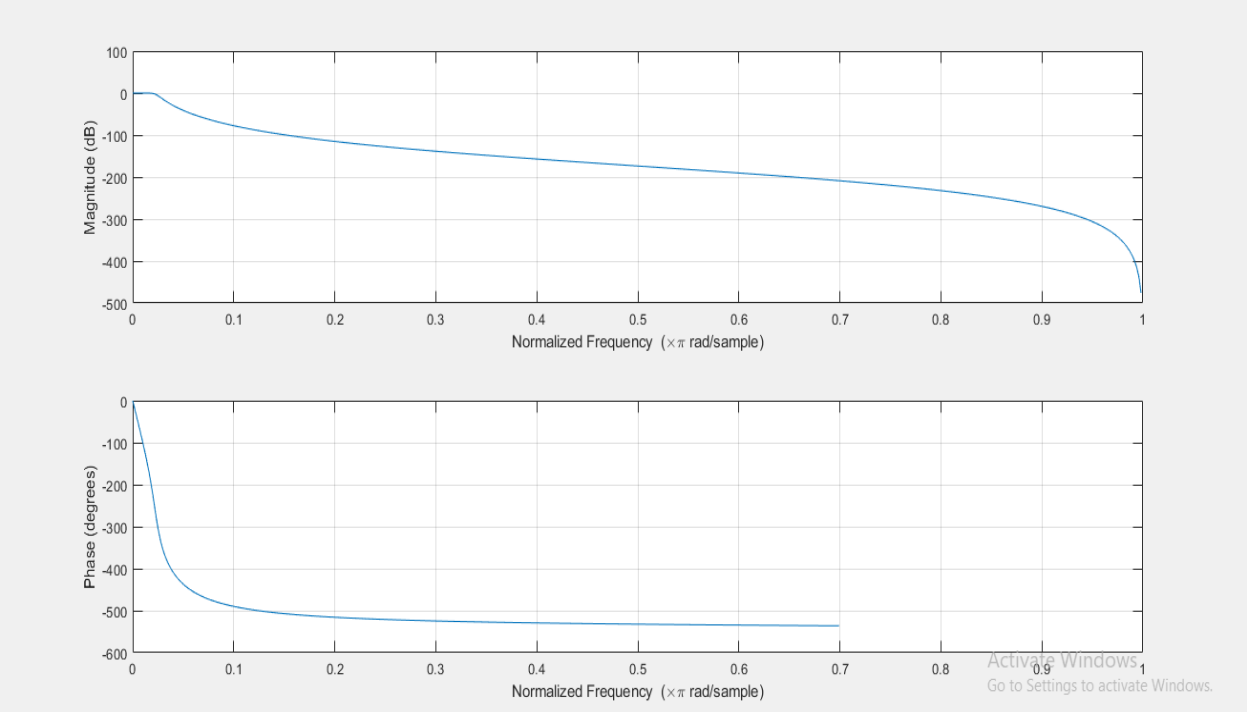


Figure3

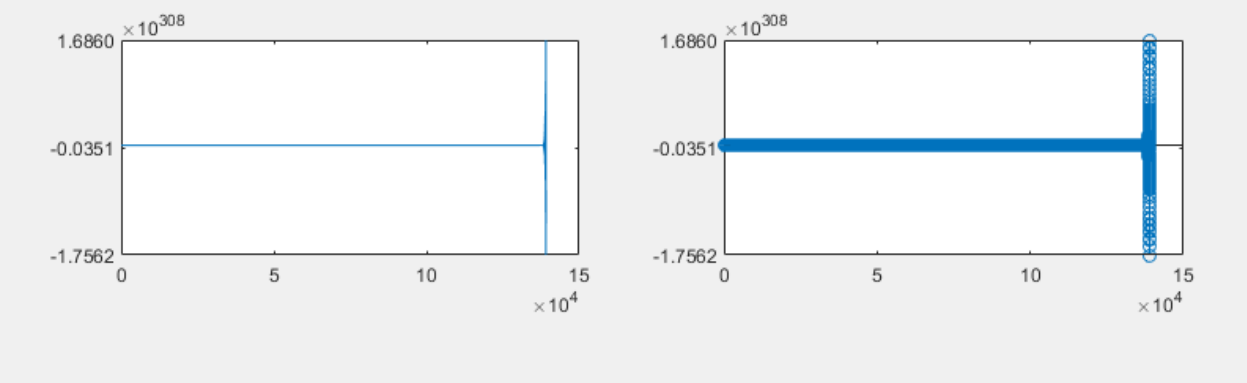
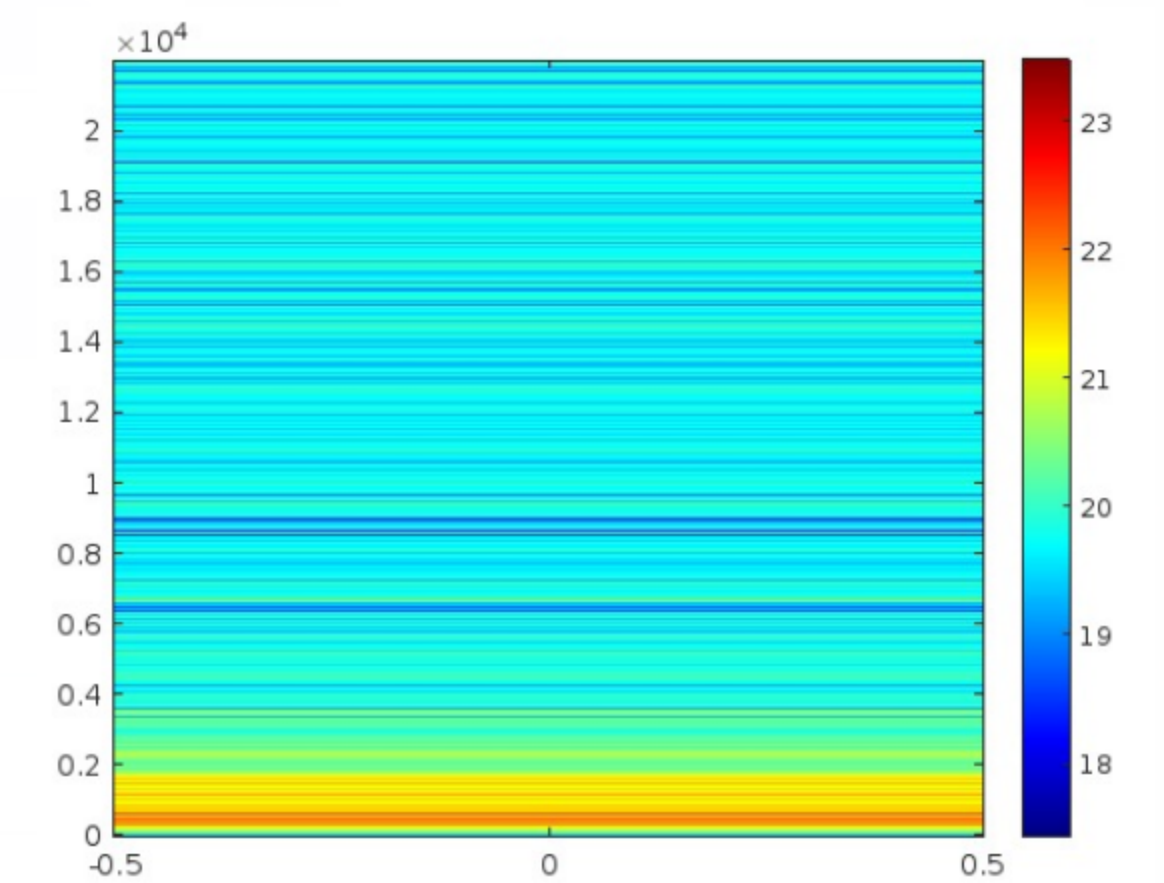


Figure4

question 2 plot



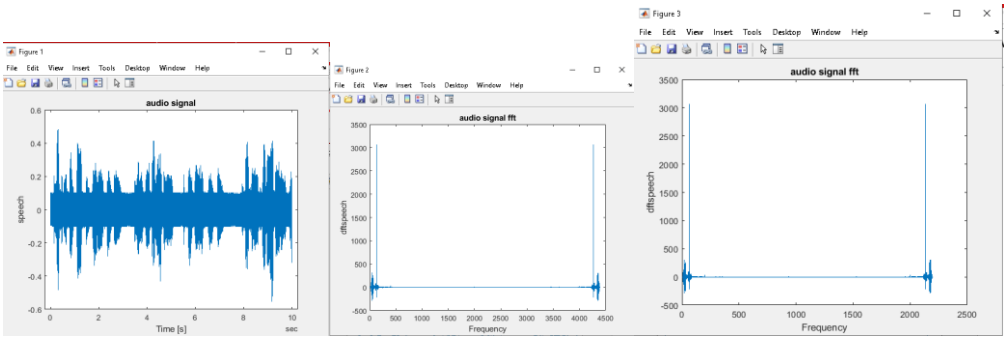
2-aiii

I also wrote a second script to print plots out separately for each portion for quick viewing for question 1 and sanity checking

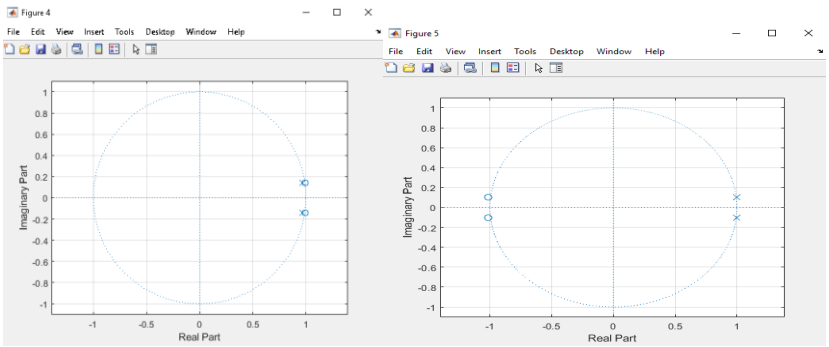
```
% separate script to print out 1-i through 1-iv print each separately
clc
clear all
close all
%(a)    Read the file TONE.wav into a vector called speech. Note its sample
rate sf.
%Plot speech versus time and play it using soundsc.
info = audioinfo('TONE.wav')
[speech, sf]=audioread('TONE.wav');
player = audioplayer(speech,sf);
play(player);
dt = 1/sf;  % Sampling time [s]
t = 0:seconds(1/sf):seconds(info.Duration);
t = t(1:end-1);
figure
plot(t,speech)
title('audio signal');
xlabel('Time [s]')
ylabel('speech')
hold on
%Note the length of speech as lengthspeech.
lengthspeech=length(speech);
```

```
nfft=2^16;
dftspeech = fft(speech,nfft); %fft of speech
f = (0:nfft-1)*(sf/nfft)/10;
figure
plot(f,dftspeech)
title('audio signal fft');
xlabel('Frequency')
ylabel('dftspeech')
hold on
f1 = (0:nfft-1)*(sf/nfft)/20;
figure
plot(f1,dftspeech)
title('audio signal fft');
xlabel('Frequency')
ylabel('dftspeech')
z=f(:,10);
t1=t(:,8);
%for wo
wo=2*pi.*z.*t1;
%for poles and zeros
NUM = [1 -1.98 1];
DEN=[1 -1.94 0.96];
TRANF=tf(NUM,DEN)      %transfer function
[ZEROS POLES]=tf2zp(NUM,DEN);
disp('Poles of transfer function are:');
disp(POLES);
disp('Zeros of transfer function are:');
disp(ZEROS);
figure
zplane(NUM,DEN);
grid on

hold on

%for filter signal
clean1 = medfilt1(speech,90);
Fn = sf/2;                              % Nyquist Frequency (Hz)
Wp = 1000/Fn;                           % Passband Frequency (Normalised)
Ws = 1010/Fn;                           % Stopband Frequency (Normalised)
Rp =   1;                               % Passband Ripple (dB)
Rs = 150;                               % Stopband Ripple (dB)
[n,Ws] = cheb2ord(Wp,Ws,Rp,Rs);
[z,p,k] = cheby2(n,Rs,Ws,'low');
[soslp,glp] = zp2sos(z,p,k);
filtered_sound2 = filtfilt(soslp, glp, clean1);
%_____________________________________________________________________________
___%

figure
plot(t,filtered_sound2);          % Filtered output
title('Filtered Signal ');
xlabel('Time (s)');
ylabel ('Amplitude');
```

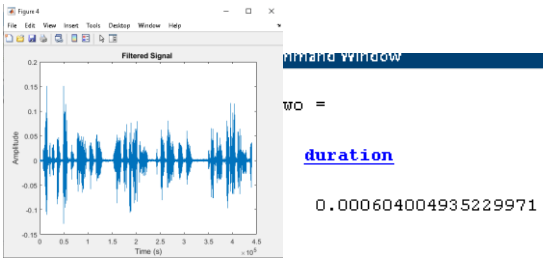
1-a-i &1-a-ii



1-a-iii



1-a-v

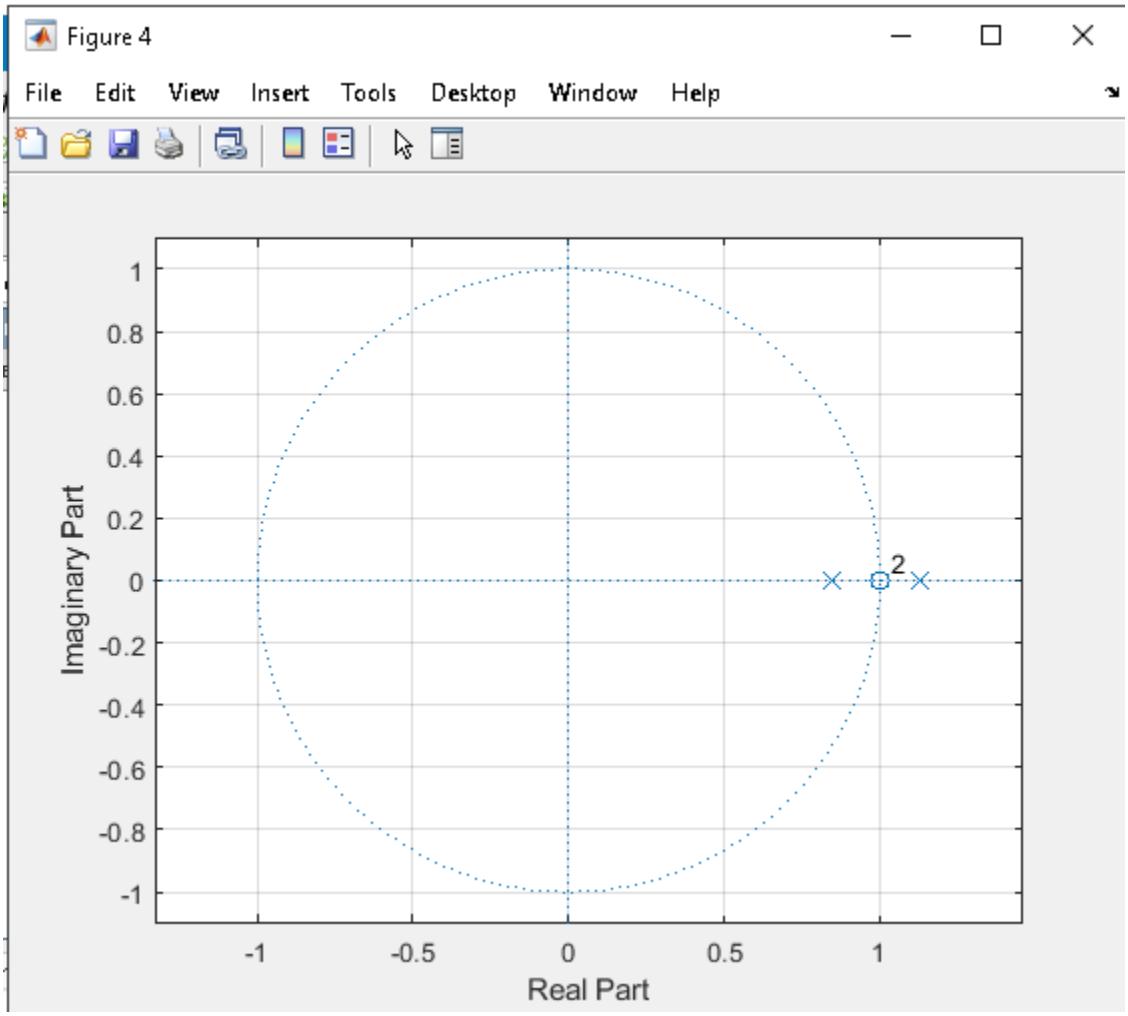


1-C plot below run separately from

```
clc
clear all
close all.
info = audioinfo('TONE.wav')
[speech, sf]=audioread('TONE.wav');
player = audioplayer(speech,sf);
play(player);
dt = 1/sf;  % Sampling time [s]
t = 0:seconds(1/sf):seconds(info.Duration);
t = t(1:end-1);
figure
plot(t,speech)
```

```
title('audio signal');
xlabel('Time [s]')
ylabel('speech')
hold on
lengthspeech=length(speech);
nfft=2^16;
dftspeech = fft(speech,nfft);
f = (0:nfft-1)*(sf/nfft)/10;
figure
plot(f,dftspeech)
title('audio signal fft');
xlabel('Frequency')
ylabel('dftspeech')
hold on
f1 = (0:nfft-1)*(sf/nfft)/20;
figure
plot(f1,dftspeech)
title('audio signal fft');
xlabel('Frequency')
ylabel('dftspeech')
z=4400;
t1=0.5;
%for wo
wo=2*pi.*z.*t1;
hh=sin(wo*t)
figure
plot(t,hh)
%for poles and zeros
gg=cos(wo)
NUM = [1 -2 1];
DEN=[1 -1.98 0.96];
TRANF=tf(NUM,DEN)
[ZEROS POLES]=tf2zp(NUM,DEN);
disp('Poles of transfer function are:');
disp(POLES);
disp('Zeros of transfer function are:');
disp(ZEROS);
figure
zplane(NUM,DEN);
grid on

hold on
%__%
%for filter signal
clean1 = medfilt1(speech,90);


Fn = sf/2;                                  % Nyquist Frequency (Hz)
Wp = 1000/Fn;                               % Passband Frequency (Normalised)
Ws = 1010/Fn;                               % Stopband Frequency (Normalised)
Rp =   1;                                   % Passband Ripple (dB)
Rs = 150;                                   % Stopband Ripple (dB)
[n,Ws] = cheb2ord(Wp,Ws,Rp,Rs);
[z,p,k] = cheby2(n,Rs,Ws,'low');
[soslp,glp] = zp2sos(z,p,k);
filtered_sound2 = filtfilt(soslp, glp, clean1);
```

```
%_____________________________________________________________________
____%

figure
plot(t,filtered_sound2);           % Filtered output
title('Filtered Signal ');
xlabel('Time (s)');
ylabel ('Amplitude');
```



Explanation of results

The output results of different blocks of code which are internally connected with each other in order to make the single unit name audio processing filter. The filter is usually responsible for performing tasks of filtering but here our focus is to perform and test the functionality of audio processing using simple filtering by using suitable inputs.

The inputs and outputs are shown by the simulation wave forms and these waveforms verify the results of the given conditions. These results verify that the value of output is updated only when the input is changing and according to the select operation. The results further prove the functionality of audio processing and removal of noise on different stages, as the output results come from the internal blocks and instructions. The result shows the execution of multiple instructions.

Conclusion

In this report, a basic design of the MATLAB was solved. In this, the basic understanding and knowledge about the audio processing along with filters and its details were presented. It was analyzed that outputs depend on the values of filter parameters as these are responsible for performing the filter operation of any data or input, and the value of the output changes as these values of filter parameters change. To make things clearer and more illustrative the input and output signals are represented by waveforms in the form of simulation graphs using the Simulator tool in MATLAB for plotting and visualization. The MATLAB code for the given Design was implemented in software, using the plots, to make the design more illustrative and clearer, and the screenshots of the outputs results were attached, along with the suitable explanations.