THE UNIVERSITY OF DANANG
UNIVERSITY OF SCIENCE AND TECHNOLOGY
Faculty of Advanced Science and Technology

LABORATORY
DIGITAL SIGNAL PROCESSING

Instructor:	PhD Ho Phuoc Tien
Class:	21ES
Student:	Tran Vu Hong Phuc
	Truong Phuoc Thinh

Da Nang, 19^{th} March 2025

Contents

LAB 1:

The goal of this assignment is to design a pre-emphasis digital filter in which the amplitude increases 6dB when the frequency doubles (6dB/octave). The input-output relationship of this filter is express by:

$$y[n] = x[n] - \alpha . x[n - 1]$$

Where:

- $y[n]$: value of output signal at discrete time n,
- $x[n]$: value of output signal at discrete time n,
- α: Pre-emphasis factor

Where:

$$\alpha = e^{-2\pi F \Delta t}$$

F: the frequency above which the spectral slope will increase 6dB/octave,

$\Delta t = \frac{1}{Fs}$: sampling period of the sound.

Design: a FIR pre-emphasis filter with F_s= 22050 Hz, $F = 1000Hz$

1. **Find α, h[n], numerator, and denominator coefficient vector of H[z]. Convert to normalize angular frequencies. Invoke 'freqz' command to plot magnitude response of the system. Is the magnitude response what you expect?**

First, in the Matlab script, we will declare F_s and F variable equal to 22050 and 1000 (Hz) respectively. Then we will calculate by using equation [1]:

```
Fs = 22050;
F = 1000;
alpha = exp(-2*pi*(F/Fs));
freqz([1 -alpha], 1);
% Plot magnitude response

xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Magnitude (dB)');
title('Magnitude Response');
grid on;
```

Then, we will get the result of α which is 0.751. Therefore, we could get $H(z) = \frac{1-0.751z^{-1}}{1}$, then we will use function $freqz([1 - 0.751],1)$ to plot Magnitude response of the system shows below:

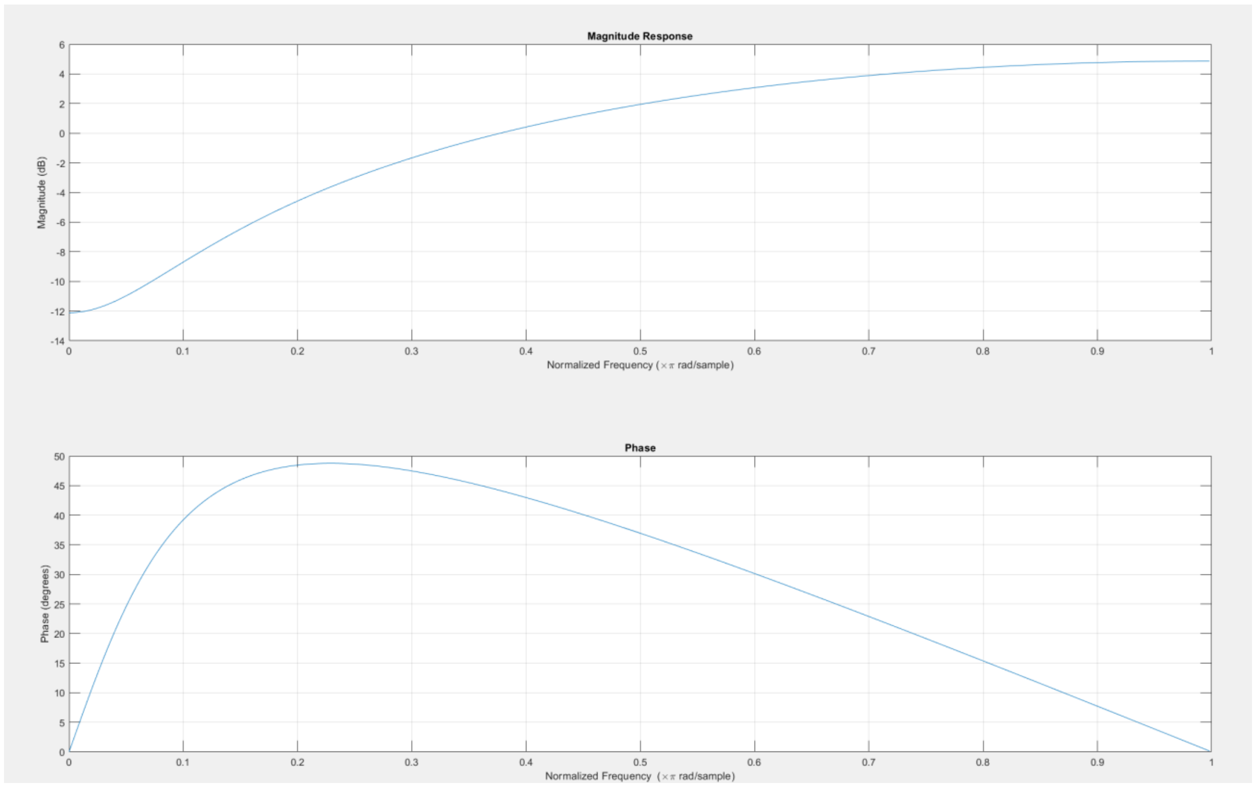


Figure 1: Magnitude Response

2. Invoke *wavrecord* to record your voice.

```
close all;
Fs = 22050;          % Sampling frequency (Hz)
Bits = 16;           % Bit depth
Channels = 1;        % Mono channel
recDuration = 5;     % Recording duration changed to 5 seconds
% Create an audio recorder object
r = audiorecorder(Fs, Bits, Channels);
% Start recording
disp("Begin Recording");
recordblocking(r, recDuration);
disp("End Recording");
% Play the recorded audio
play(r);
% Extract recorded data
y = getaudiodata(r);
% Save the audio file
filename = 'recorded_audio.wav';
audiowrite(filename, y, Fs);
% Plot the waveform
t = (0:length(y)-1)/Fs; % Time axis
figure;
plot(t, y, 'b');
```

```
xlabel('Time (s)');
ylabel('Amplitude');
title('Recorded Audio Waveform');
grid on;
% Read the saved audio and play it
[y, Fs] = audioread(filename);
sound(y, Fs);
```

After recording the voice, we extract the data from it using getaudiodata so that we could plot the recorded voice with “uint8” data type. Then we save it into a wav file using audiowrite function with respective sample rate and bits per sample. Finally, we playback the wav file using audioread and sound function.

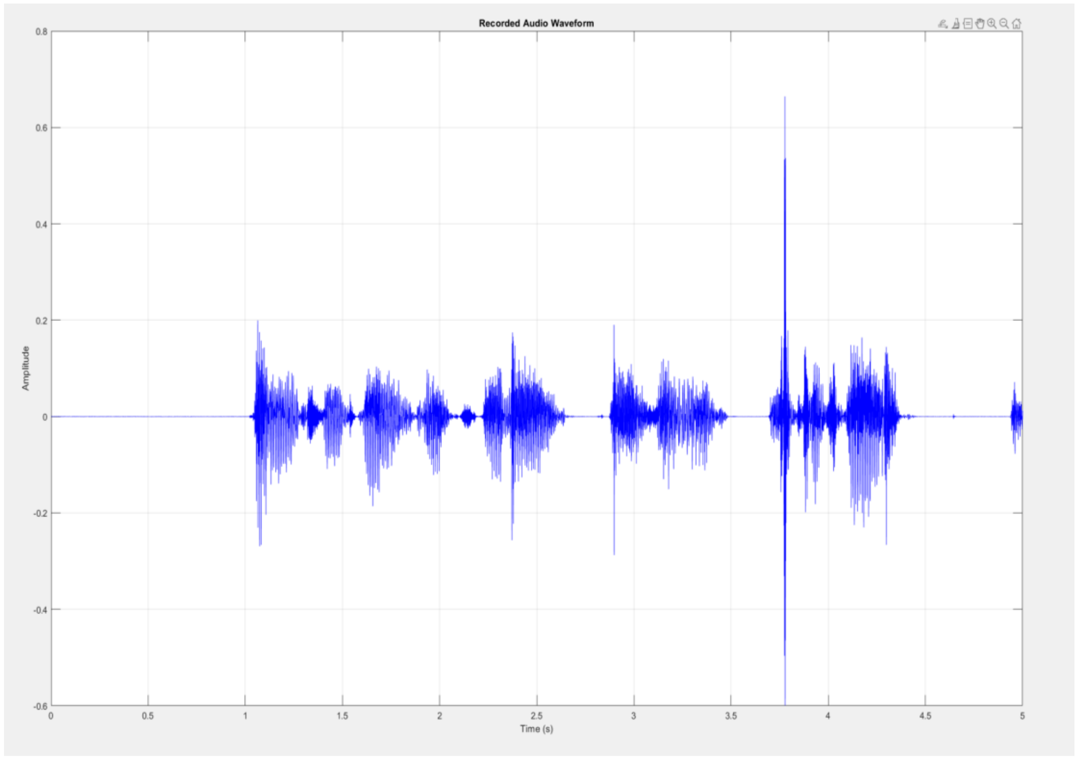
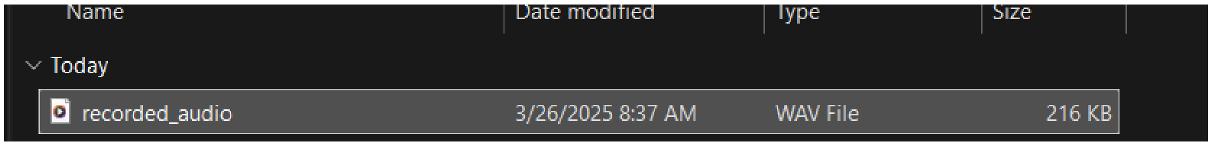


Figure 2: Plot the recording using audiowrite and audioread



Name	Date modified	Type	Size
Today			
recorded_audio	3/26/2025 8:37 AM	WAV File	216 KB

Figure 3: Wav file saved in the workspace folder

3. Performing filtering operation.

```
close all;
clear;
clc;
% Define alpha for pre-emphasis filtering
alpha = 0.751; % Given value in the document
```

```
% Frequency response of the filter
[H, W] = freqz([1 -alpha], 1);
% Read the first audio file
[y, Fy] = audioread("custom.wav");
% Plot the original "custom.wav" waveform
subplot(4,1,1);
plot([1:length(y)]/Fy, y);
title('Custom wave');
xlabel('Time (s)');
ylabel('Amplitude');
% Apply pre-emphasis filtering
y_preemph = filter([1 -alpha], 1, y);
% Plot the filtered "custom.wav" waveform
subplot(4,1,2);
plot([1:length(y_preemph)]/Fy, y_preemph);
title('Filtered custom wave');
xlabel('Time (s)');
ylabel('Amplitude');
axis([0 max(length(y_preemph)/Fy) -1 1]);
% Read the second audio file
[z, Fs] = audioread("Record.mp3");
% Plot the original "speech.wav" waveform
subplot(4,1,3);
plot([1:length(z)]/Fs, z);
title('Speech file');
xlabel('Time (s)');
ylabel('Amplitude');
% Apply pre-emphasis filtering
z_preemph = filter([1 -alpha], 1, z);
% Plot the filtered "speech.wav" waveform
subplot(4,1,4);
plot([1:length(z_preemph)]/Fs, z_preemph);
title('Filtered speech file');
xlabel('Time (s)');
ylabel('Amplitude');
```

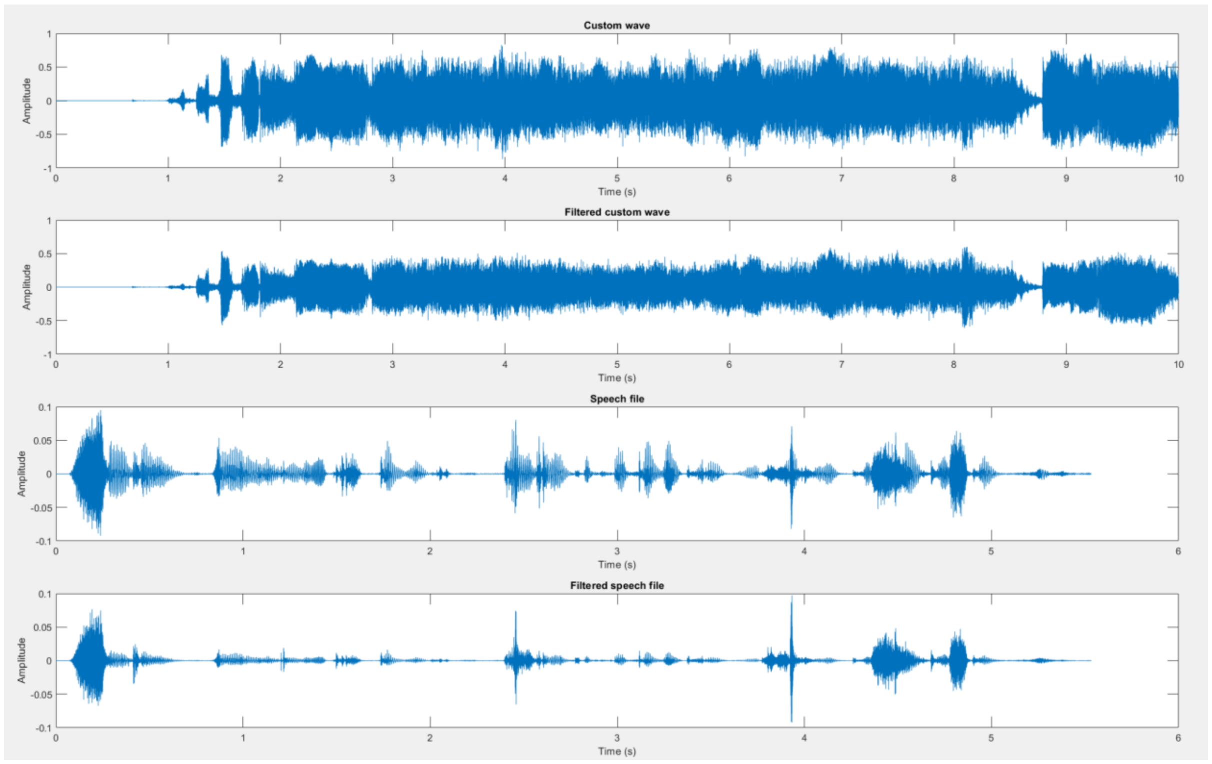


Figure 3: Comparison between the original and filtered version of signals.

4. **Using soundsc function to hear the signals.**

```
% Play the original "recorded_audio.wav"
soundsc(y, Fy);
pause(10); % Wait for 10 seconds
% Play the filtered "recorded_audio.wav"
soundsc(y_preemph, Fy);
pause(10); % Wait for 10 seconds
% Play the original "Record.mp3"
soundsc(z, Fs);
pause(10); % Wait for 10 seconds
% Play the filtered "Record.mp3"
soundsc(z_preemph, Fs);
```

Using this function, we can clearly hear that the lousy part of the signals has partly been eliminated hence the signal is much smoother than its original version

5. **Save our sounds using wavwrite**

```
% Save filtered audio files
audiowrite("y_preemph.wav", y_preemph, Fy);
audiowrite("z_preemph.wav", z_preemph, Fs);
```

Our filtered sounds are saved under the name of “y_preemph” for our custom wave while the file named “z_preemph” is the filtered version of the “speech” file.

Noise reduction using FIR filtering.

Our first task is to load the data set named “mtlb” and then we plot this data set in order to see its waveform

1. **Load and plot “mtlb” data set.**

```
clear;

subplot(111);

Fs = 7418;

load mtlb;

L = length(mtlb);

plot([1:L]/Fs, mtlb); axis([])

axis tight;

xlabel("Time (s)" ); title("mtlb");
```

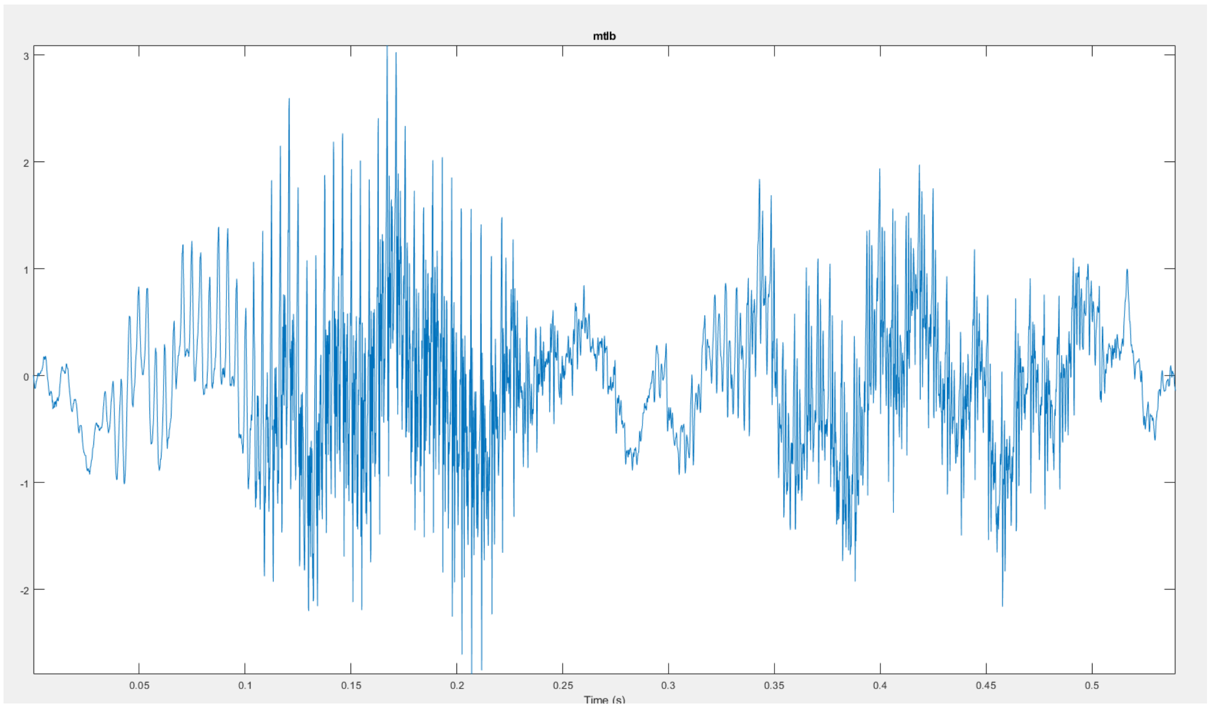


Figure 4: Waveform of dataset “mtlb”

2. **Load noisy version of “mtlb”**

```
loadnoisymtlb;
plot([1:L]/Fs, noisymtlb);
```

```
soundsc(noisymtlb, Fs);
axis tight;
xlabel("Time (s)" ); title("noisymtlb");
```

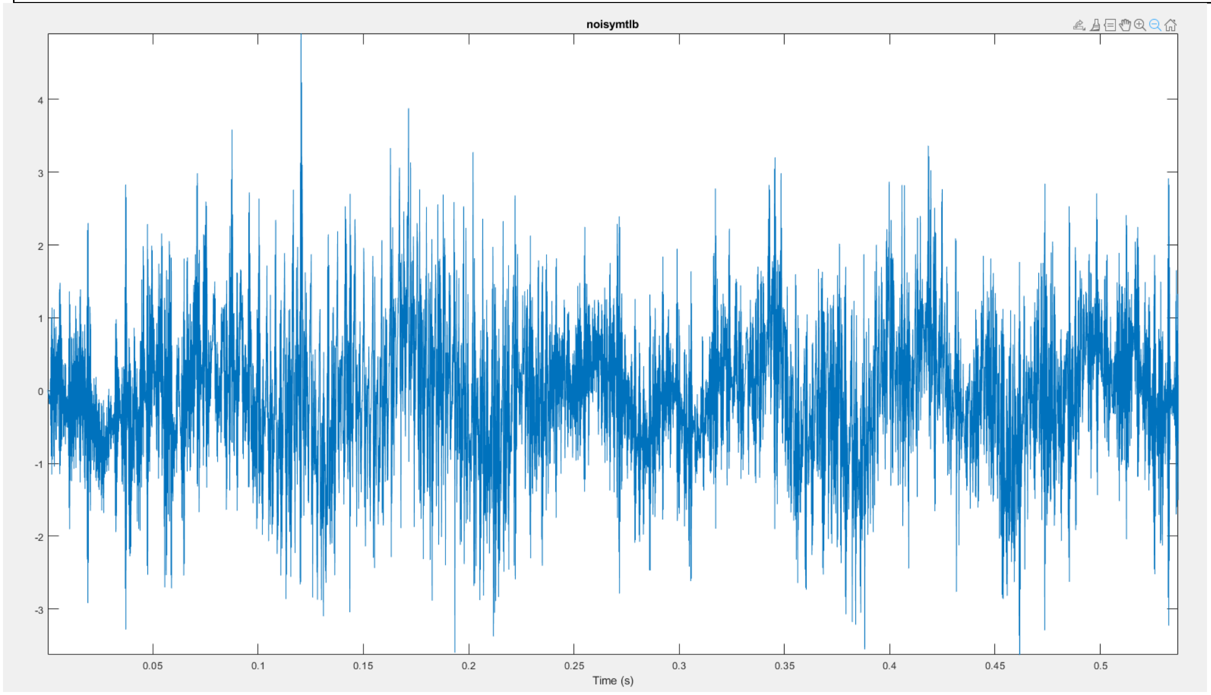


Figure 5: Load mtlb dataset and noisy

3. Magnitude spectrum code:

```
function [M,f] = magnitude_spectrum(x,Fs)
Nf = round(length(x)/2);
f = 0:Nf-1;
f = f/Nf*Fs/2;
M = fft(x);
M = abs(M(1:Nf));
```

Code

```
Fs = 7418;                      % Sampling frequency
N = length(mtlb);               % Number of samples in 'mtlb'

% Calculate FFT and center spectrum
M = abs(fftshift(fft(mtlb)));    % Removed normalization by N
f = (-N/2:N/2-1)*(Fs/N);         % Frequency vector centered at 0

% Plot the magnitude spectrum
subplot(211);
plot(f, M, 'b');                 % Plot the magnitude spectrum
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Spectrum of mtlb');
xlim([-4000 4000]);              % Symmetric frequency range
ylim([0 max(M)]);                % Dynamic y-axis if needed
```

Plot the dataset can help us have a peak of its waveform, however, as a normal person looking at this plot, there's nothing much can be learned from the figure. That's why we designed a function to plot its magnitude spectrum instead and with this, we can see the frequency components of this signal.
First, let's plot the magnitude spectrum of "mtlb" dataset.

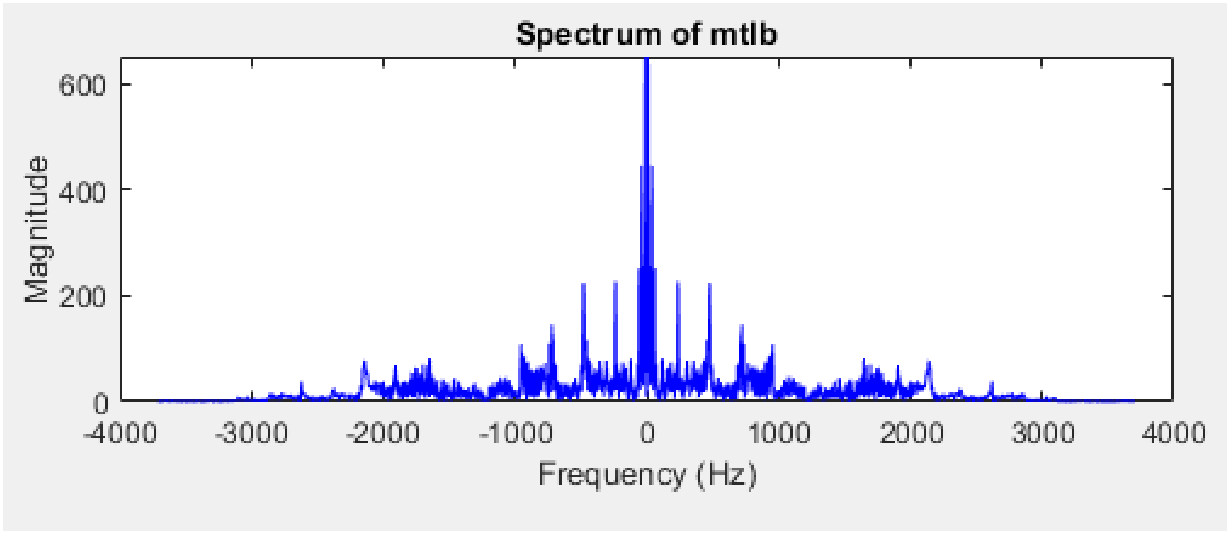


Figure 6: Spectrum of mtlb

We can see clearly from this figure that magnitude in this signal are in the range of 0 to 650 Hz
And the

4. Magnitude spectrum of noisy version of "mtlb"

Now, let's continue to the magnitude spectrum of noisy version of "mtlb" dataset.

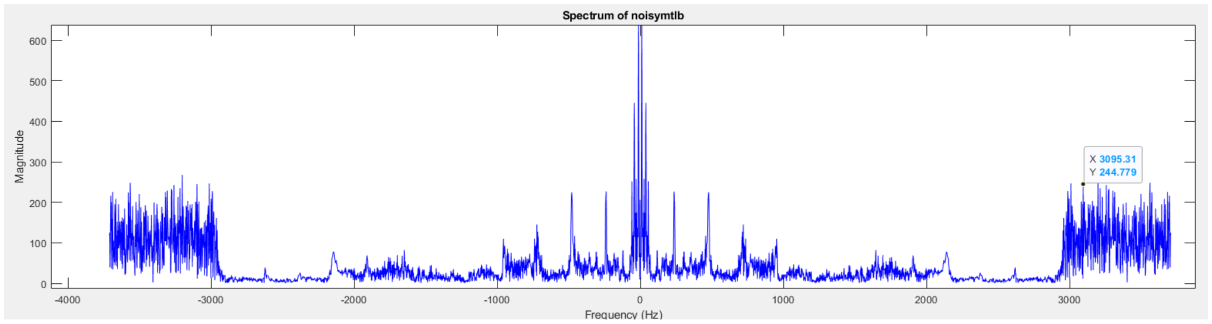


Figure 7: Spectrum of noisymtlb

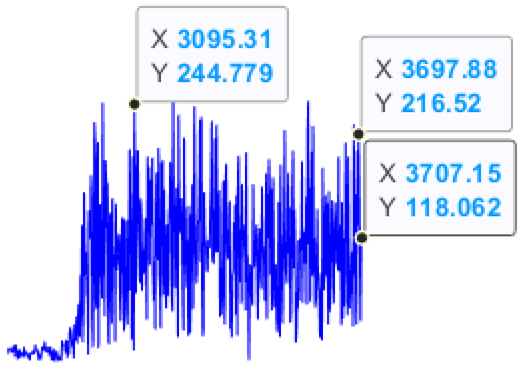


Figure 8: Noisy part of dataset “noisymtlb”

Here, we can clearly see that the “noise” from this dataset are frequencies from 2900 Hz onwards. Since the sample frequency of the later set is 7400Hz which means the maximum frequency of its signal is 3700Hz. Therefore, when converting the frequency into the range of - π to π (radian), frequency of 2900 is approximately

Code to print spectrum

```
Fs = 7418;  % Sampling frequency
N = length(noisymtlb);  % Number of samples in the noisy dataset


% Calculate the FFT and center the spectrum
M2 = abs(fftshift(fft(noisymtlb)));  % Magnitude spectrum (no
normalization by N)
f2 = (-N/2:N/2-1)*(Fs/N);            % Frequency vector centered at 0 Hz


% Plot the spectrum of noisy mtlb
subplot(212);
plot(f2, M2, 'b');  % Plot full spectrum from -Fs/2 to Fs/2
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Spectrum of noisymtlb');
xlim([-4000 4000]);  % Set symmetric limits for easier interpretation
ylim([0 max(M2)]);   % Adjust the y-axis dynamically to fit the full range
```

From that we calculate for the transition band with 0.1π

Sampling Frequency:

$$F_s = 7400\ Hz$$

Cutoff Frequency:

$$f_c = 2900\ Hz$$

Normalized Cutoff Frequency:

$$\omega_c = 2\pi \frac{f_c}{F_s} = 2\pi \frac{2900}{7400} \approx 0.783\pi$$

Transition Band:

Let’s assume a transition band of 0.1π, so the passband edge is around 0.75π, and the stopband starts after 0.85π.

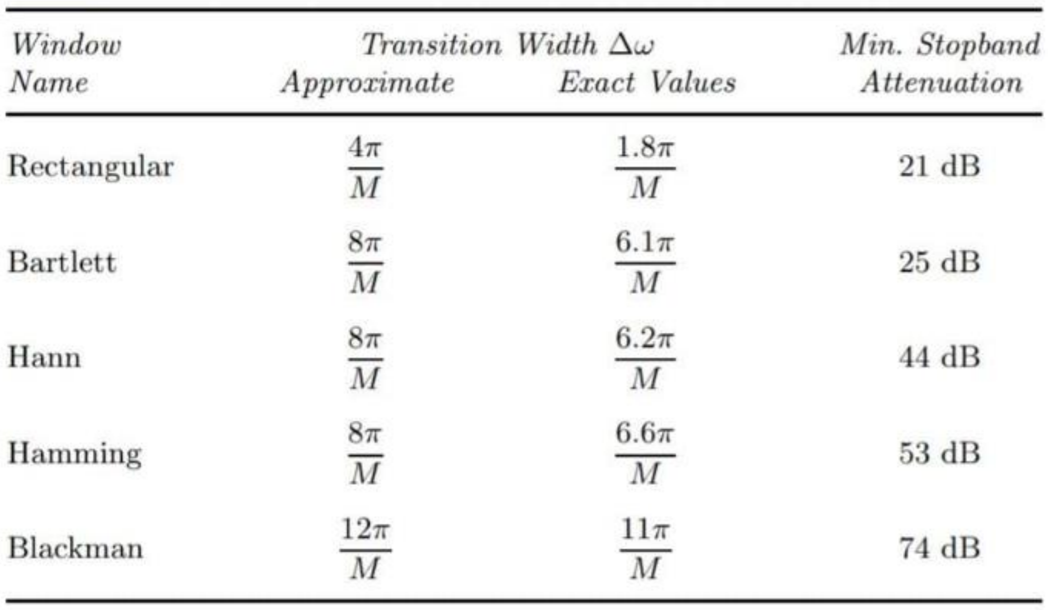
=> So the passband is around 0.75π and stopband starts after 0.85π

5. Design low-pass filter using window technique

As specified in the previous section, since we chose the cut-off frequency of 0.78π and transition band of 0.1π, the pass frequency and stop frequency will be 0.75π and 0.85π, respectively. In addition, we also chose R_p being 0.15dB, a typical value of ripple band.

Last but not least, the value of As being chosen by considering the magnitude of noise part in the signal. Choosing $A_s = 40\text{dB}$ of the filter

TABLE 7.1 *Summary of commonly used window function characteristics*



Window Name	*Transition Width Δω* *Approximate*	*Exact Values*	*Min. Stopband Attenuation*
Rectangular	$\frac{4\pi}{M}$	$\frac{1.8\pi}{M}$	21 dB
Bartlett	$\frac{8\pi}{M}$	$\frac{6.1\pi}{M}$	25 dB
Hann	$\frac{8\pi}{M}$	$\frac{6.2\pi}{M}$	44 dB
Hamming	$\frac{8\pi}{M}$	$\frac{6.6\pi}{M}$	53 dB
Blackman	$\frac{12\pi}{M}$	$\frac{11\pi}{M}$	74 dB

Looking at the table, we can see that, in order to achieve the desired value of A_s. We will choose the Hann windows, because it’s closet to out requirements

$$M = \frac{6.2\pi}{0.1\pi} + 1 = 63$$

From this point onwards, we continue to design the filter has been learned in the Class.

```
figure; % Create a new figure for the filter


% Filter parameters
wp = 0.75*pi;
ws = 0.85*pi;
```

```
tr_width = ws - wp;
M_filter = ceil(6.2*pi/tr_width) + 1;  % Filter length
n = 0:M_filter-1;
wc = (ws + wp)/2;  % Cutoff frequency


% Generate filter coefficients with Hann window
hd = ideal_lp(wc, M_filter);          % hd is a row vector from ideal_lp
w_hann = hann(M_filter)';             % Hann window (row vector)
h1 = hd .* w_hann;                     % Element-wise multiplication
% Frequency response
[db, ~, ~, ~, w] = freqz_m(h1, 1);
delta_w = 2*pi/1000;


% Calculate ripple/attenuation
Rp = -min(db(1:floor(wp/delta_w)+1));
As = -round(max(db(floor(ws/delta_w)+1:501)));


% Plot 1: Magnitude (dB)
subplot(211);
plot(w/pi, db, 'b', 'LineWidth', 1.5);
hold on;
% Horizontal lines (Rp/As)
plot([0, wp/pi], [-Rp, -Rp], 'r--', 'LineWidth', 1.5);
plot([ws/pi, 1], [-As, -As], 'g--', 'LineWidth', 1.5);
% Vertical lines (wp/ws)
plot([wp/pi, wp/pi], ylim, 'm:', 'LineWidth', 2);       % Passband edge
(magenta)
plot([ws/pi, ws/pi], ylim, 'c:', 'LineWidth', 2);       % Stopband edge
(cyan)
hold off;
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Magnitude (dB)');
title('Filter Frequency Response (dB)');
legend('Response', 'Rp', 'As', 'wp', 'ws', 'Location', 'Best');
grid on;


% Plot 2: Magnitude (Linear)
subplot(212);
plot(w/pi, abs(db2mag(db)), 'b', 'LineWidth', 1.5);
hold on;
% Horizontal lines (Rp/As in linear)
plot([0, wp/pi], db2mag(-Rp)*[1,1], 'r--', 'LineWidth', 1.5);
plot([ws/pi, 1], db2mag(-As)*[1,1], 'g--', 'LineWidth', 1.5);
% Vertical lines (wp/ws)
```

```
plot([wp/pi, wp/pi], ylim, 'm:', 'LineWidth', 2);       % Passband edge
plot([ws/pi, ws/pi], ylim, 'c:', 'LineWidth', 2);       % Stopband edge
hold off;
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Magnitude');
title('Filter Magnitude Response');
legend('Response', 'Rp', 'As', 'wp', 'ws', 'Location', 'Best');
grid on;


% Display results
disp(['Passband Ripple (Rp): ', num2str(Rp), ' dB']);
disp(['Stopband Attenuation (As): ', num2str(As), ' dB']);
```

Results:

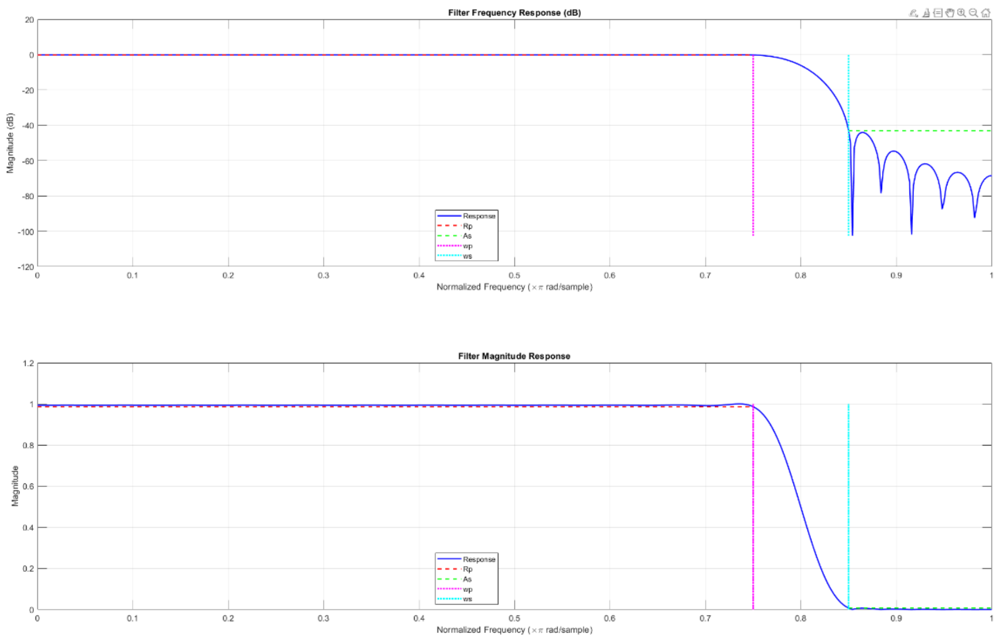


Figure 9. Low pass filter after using Hann Windows

The desired about R_p and A_s:

```
Passband Ripple (Rp): 0.11778 dB
Stopband Attenuation (As): 43 dB
```

6. **Perform similar steps as step 3&4 in previous section. Does the noise reduce?**
   Code:

```
figure(6)
Fs = 7418; % Sampling frequency


% --- Plot spectrum of noisymtlb ---
N = length(noisymtlb);
```

```
M = abs(fftshift(fft(noisymtlb))); % Unnormalized magnitude
f = (-N/2:N/2-1)*(Fs/N); % Frequency vector centered at 0 Hz


subplot(211);
plot(f, M, 'b');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Spectrum of noisymtlb');
xlim([-4000 4000]); % Focus on ±4 kHz range
ylim([0 max(M)]);


% --- Plot spectrum of filtered signal (z_preemph) ---
z_preemph = filter(h1, 1, noisymtlb); % Apply filter
N_filtered = length(z_preemph);
M2 = abs(fftshift(fft(z_preemph))); % Unnormalized magnitude
f2 = (-N_filtered/2:N_filtered/2-1)*(Fs/N_filtered); % Frequency vector


subplot(212);
plot(f2, M2, 'b');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Spectrum of filtered noisymtlb h1');
subtitle("Phuc-Thinh");
xlim([-4000 4000]); % Same frequency range as original
ylim([0 max(M2)]);


% Play sounds
soundsc(noisymtlb, Fs);
soundsc(z_preemph, Fs);
```

Result:

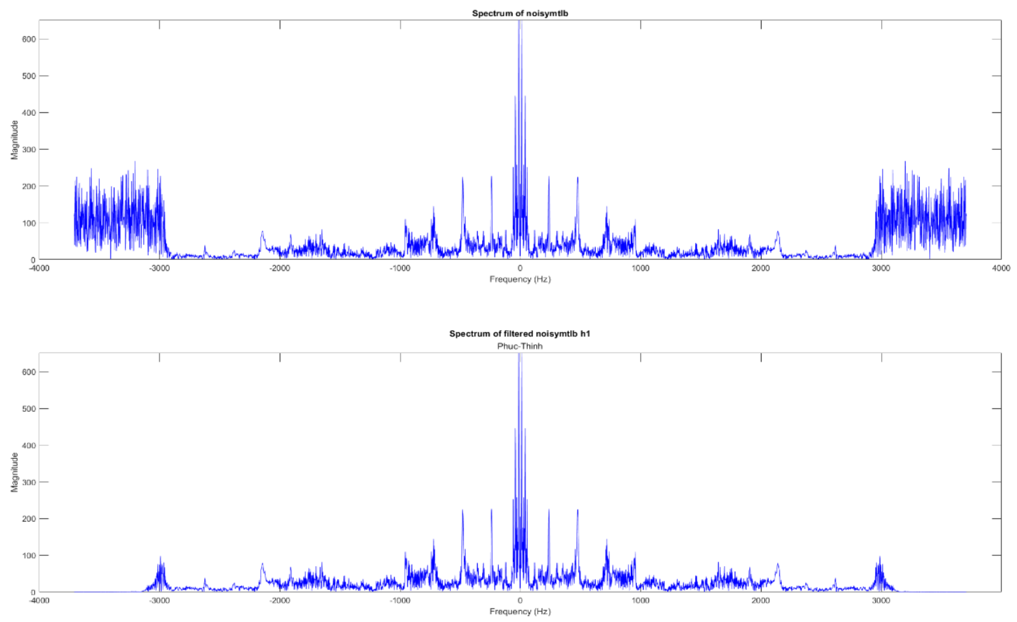


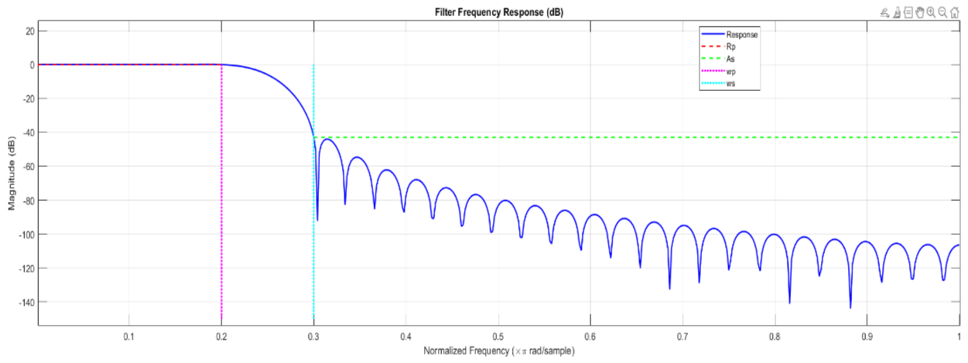
Figure 10. After apply $\omega_p = 0.75\pi$ and $\omega_s = 0.85\pi$

7. **Try different ω_p, ω_s , R_p, and A_s. Can you remove most of the noise from the noisy signal?**
   Code:

```
figure; % Create a new figure for the filter


% Filter parameters

wp = 0.2*pi;

ws = 0.3*pi;

tr_width = ws - wp;

M_filter = ceil(6.2*pi/tr_width) + 1;  % Filter length

n = 0:M_filter-1;

wc = (ws + wp)/2;  % Cutoff frequency


% Generate filter coefficients with Hann window

hd = ideal_lp(wc, M_filter);          % hd is a row vector from ideal_lp

w_hann = hann(M_filter)';             % Hann window (row vector)

h1 = hd .* w_hann;                     % Element-wise multiplication

% Frequency response

[db, ~, ~, ~, w] = freqz_m(h1, 1);
```

```
delta_w = 2*pi/1000;


% Calculate ripple/attenuation

Rp = -min(db(1:floor(wp/delta_w)+1));

As = -round(max(db(floor(ws/delta_w)+1:501)));


% Plot 1: Magnitude (dB)

subplot(211);

plot(w/pi, db, 'b', 'LineWidth', 1.5);

hold on;

% Horizontal lines (Rp/As)

plot([0, wp/pi], [-Rp, -Rp], 'r--', 'LineWidth', 1.5);

plot([ws/pi, 1], [-As, -As], 'g--', 'LineWidth', 1.5);

% Vertical lines (wp/ws)

plot([wp/pi, wp/pi], ylim, 'm:', 'LineWidth', 2);       % Passband edge
(magenta)

plot([ws/pi, ws/pi], ylim, 'c:', 'LineWidth', 2);       % Stopband edge
(cyan)

hold off;

xlabel('Normalized Frequency (\times\pi rad/sample)');

ylabel('Magnitude (dB)');

title('Filter Frequency Response (dB)');

legend('Response', 'Rp', 'As', 'wp', 'ws', 'Location', 'Best');

grid on;


% Plot 2: Magnitude (Linear)

subplot(212);

plot(w/pi, abs(db2mag(db)), 'b', 'LineWidth', 1.5);

hold on;

% Horizontal lines (Rp/As in linear)

plot([0, wp/pi], db2mag(-Rp)*[1,1], 'r--', 'LineWidth', 1.5);
plot([ws/pi, 1], db2mag(-As)*[1,1], 'g--', 'LineWidth', 1.5);
% Vertical lines (wp/ws)
plot([wp/pi, wp/pi], ylim, 'm:', 'LineWidth', 2);       % Passband edge
plot([ws/pi, ws/pi], ylim, 'c:', 'LineWidth', 2);       % Stopband edge
hold off;
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Magnitude');
title('Filter Magnitude Response');
legend('Response', 'Rp', 'As', 'wp', 'ws', 'Location', 'Best');
grid on;


% Display results
disp(['Passband Ripple (Rp): ', num2str(Rp), ' dB']);
disp(['Stopband Attenuation (As): ', num2str(As), ' dB']);
```

Result:



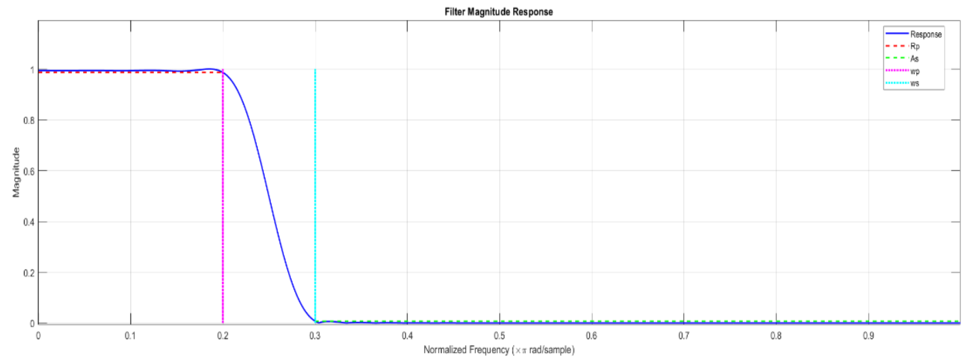


Figure 11. New low pass filter $\omega_p = 0.2\pi$, $\omega_s = 0.3\pi$

Apply with new filter $\omega_s = 0.2\pi$ and $\omega_p = 0.3\pi$, we got these results

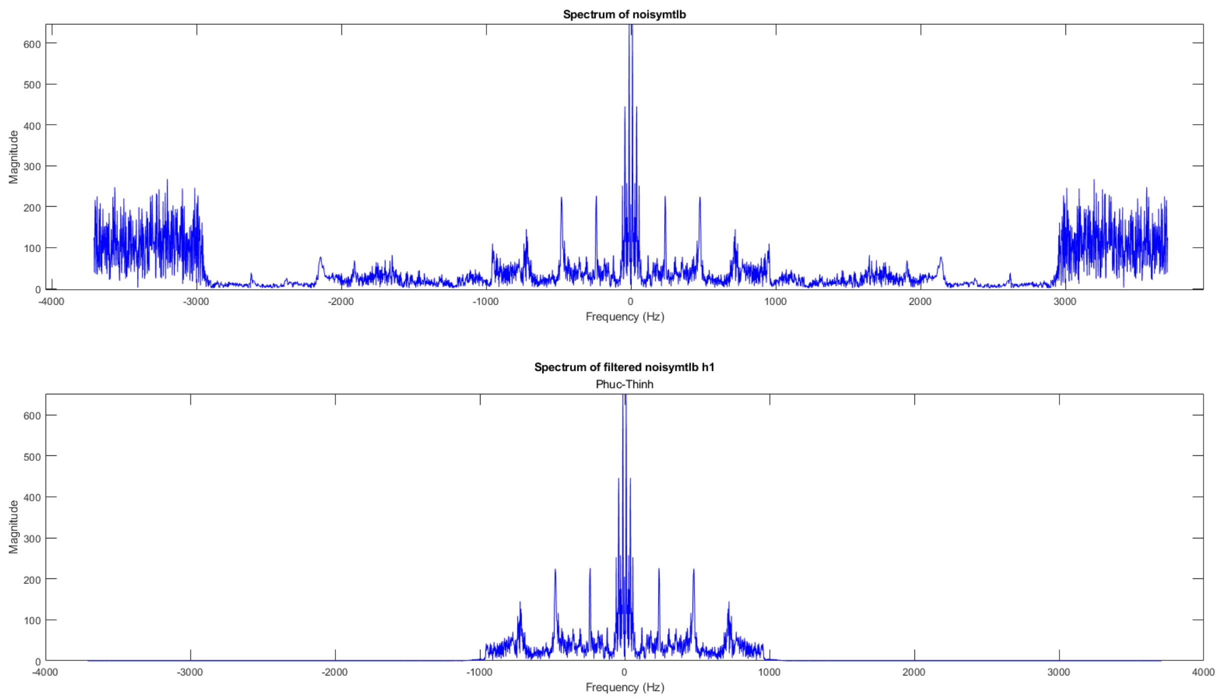
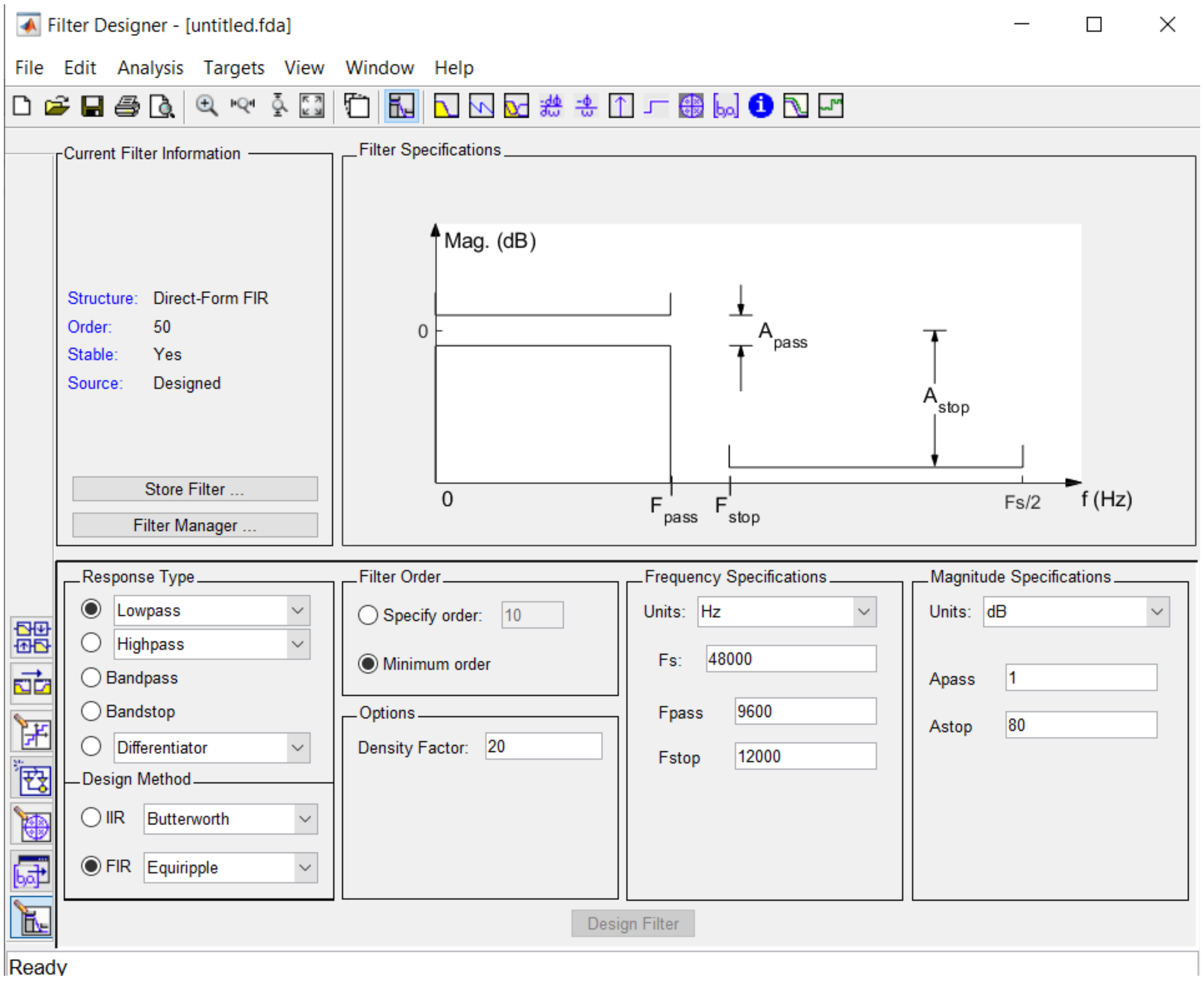


Figure 12. noisymtlb after using $\omega_p = 0.2\pi$, $\omega_s = 0.3\pi$ to reduce noise

Design filter using FDATool.

In this section, we will design the filter using an FDA tool. To open the FDA tool, we type command: fdatool in the Matlab command window.



Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR Equiripple]

Filter Order: Specify order: 10 [✓ Minimum order]

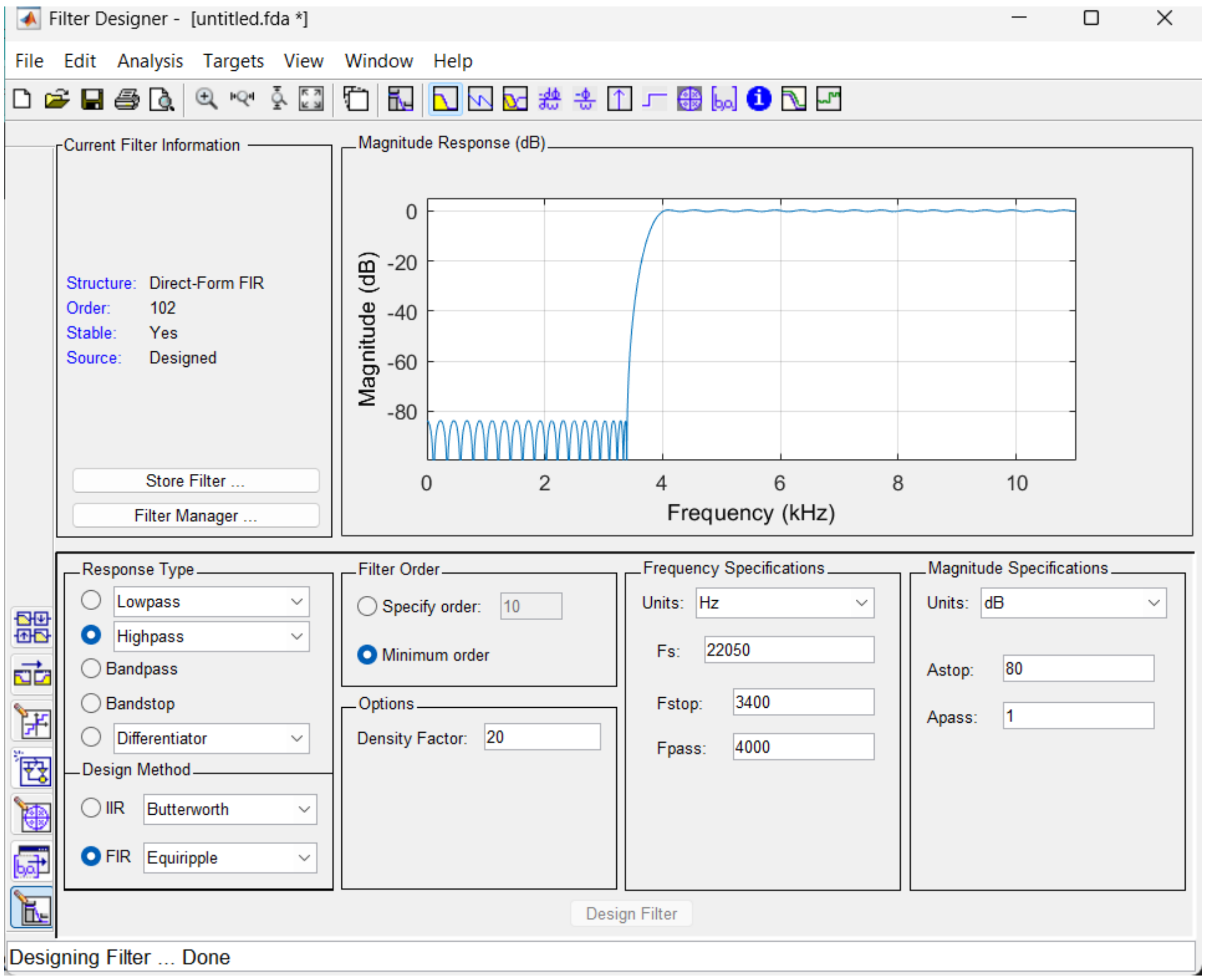
Options: Density Factor: 20

Frequency Specifications: Units: Hz; Fs: 48000; Fpass: 9600; Fstop: 12000

Magnitude Specifications: Units: dB; Apass: 1; Astop: 80

Figure 13. Filter Designer

1. **High pass filter:**
   We will design a FIR high pass filter using FDA tool with following configurations:
   - Sample rate: 22050 Hz.
   - Stopping frequency: 3400 Hz
   - Passing frequency: 4000 Hz
   - Apass: 80 dB
   - Astop: 1dB


Filter Designer - [untitled.fda *]
File Edit Analysis Targets View Window Help
Current Filter Information
Magnitude Response (dB)
Magnitude (dB)
Frequency (kHz)
Store Filter ...
Filter Manager ...
Design Filter

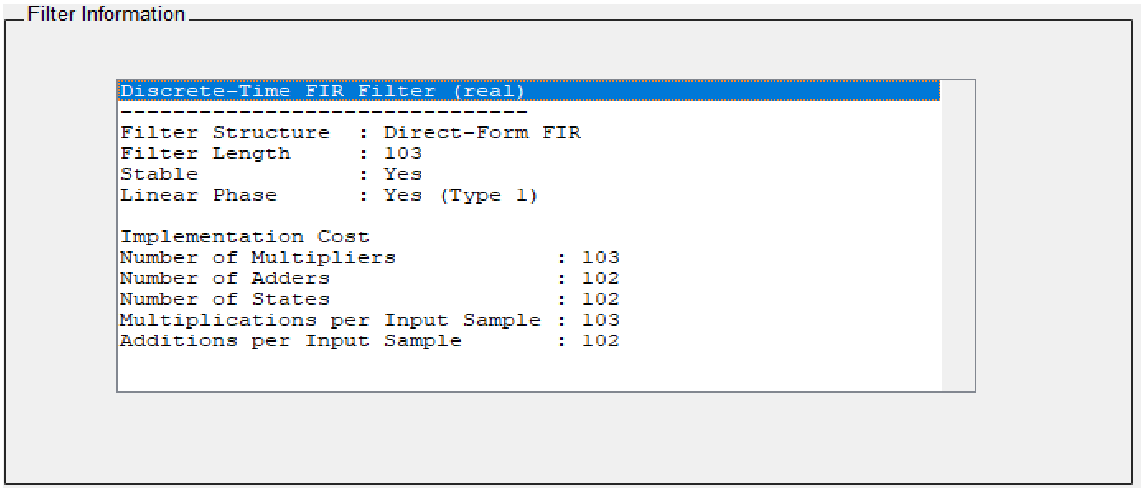
Structure: Direct-Form FIR
Order: 102
Stable: Yes
Source: Designed

Response Type: Lowpass, [✓ Highpass], Bandpass, Bandstop, Differentiator
Design Method: IIR Butterworth, [✓ FIR] Equiripple
Filter Order: Specify order: 10, [✓ Minimum order]
Options: Density Factor: 20
Frequency Specifications: Units: Hz; Fs: 22050; Fstop: 3400; Fpass: 4000
Magnitude Specifications: Units: dB; Astop: 80; Apass: 1

Designing Filter ... Done

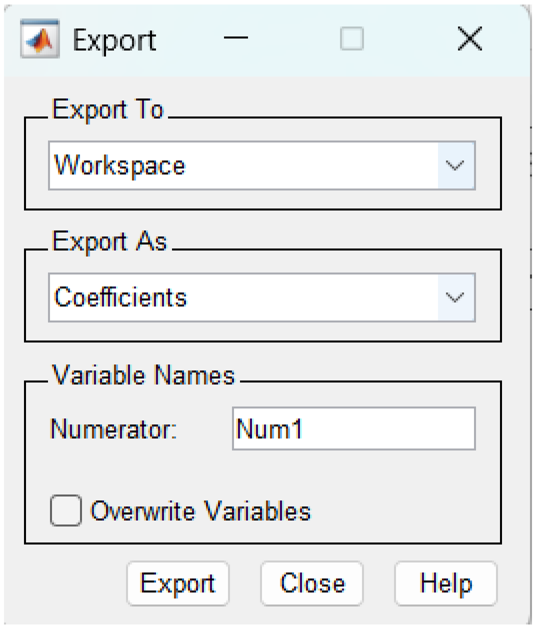
Figure 14. High-pass filter configuration in FDA tool

Filter information



```
Discrete-Time FIR Filter (real)
-------------------------------
Filter Structure  : Direct-Form FIR
Filter Length     : 103
Stable            : Yes
Linear Phase      : Yes (Type 1)

Implementation Cost
Number of Multipliers            : 103
Number of Adders                 : 102
Number of States                 : 102
Multiplications per Input Sample : 103
Additions per Input Sample       : 102
```

Next, we will export the filter coefficient to our workspace for implementation under a variable i called Num3.


Export

Export To: Workspace

Export As: Coefficients

Variable Names

Numerator: Num1

Overwrite Variables

Then, we will implement the filter to our wav file in the Matlab script. The script will play and plot the original sound and high pass filtered sound:

```
close all
[x, fs] = audioread('custom.wav');
% Apply high-pass filter
sound_filtered = filter(Num1, 1, x);
% Compute magnitude spectrum
[L, s] = magnitude_spectrum1(sound_filtered, 1/fs);
% Play filtered sound
soundsc(sound_filtered, fs);
```

And here is the respective result, showcasing the low frequency being filtered out

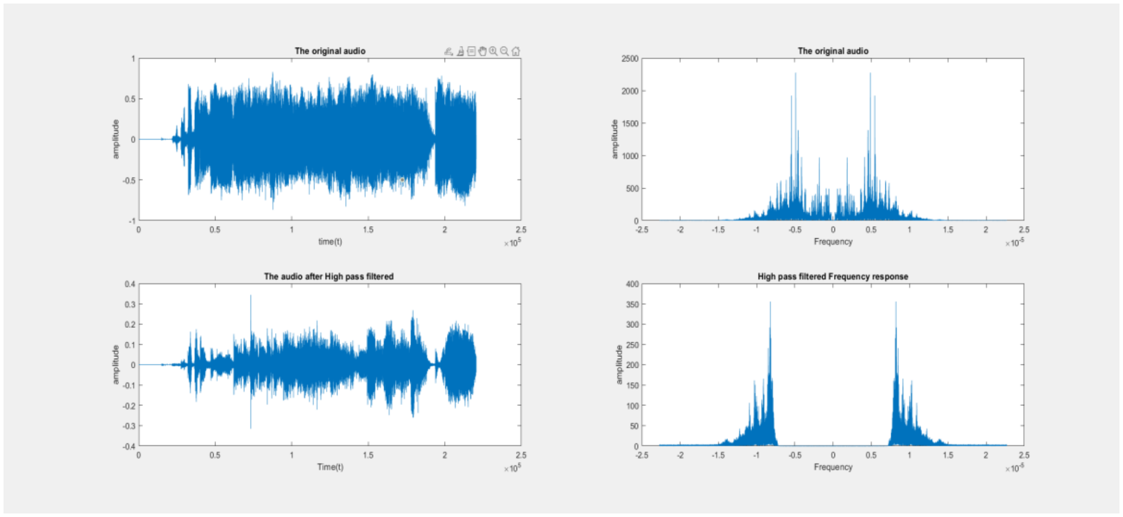


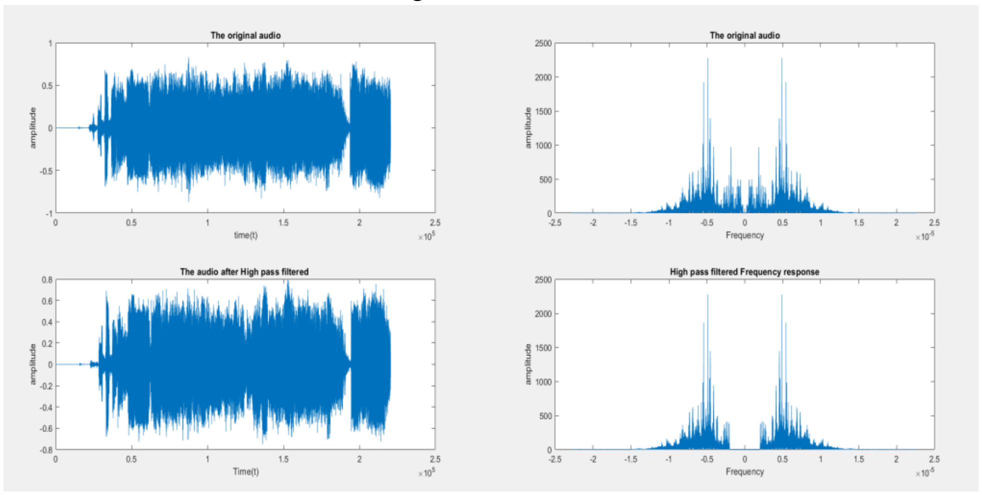
Figure 11. High-pass filter result

And after using soundsc to play the output, However, in this configuration, the passing frequency is too high therefore other low frequency has been filtered which makes the recording inaudible. Hence, we will make a custom high pass filter that makes the wav file more hearable.

We have alter the parameters into this configuration:

- **Sample rate:** 22050 Hz.
- **Stopping frequency:** 960 Hz
- **Passing frequency:** 1000 Hz
- **Apass:** 80 dB
- **Astop:** 1dB

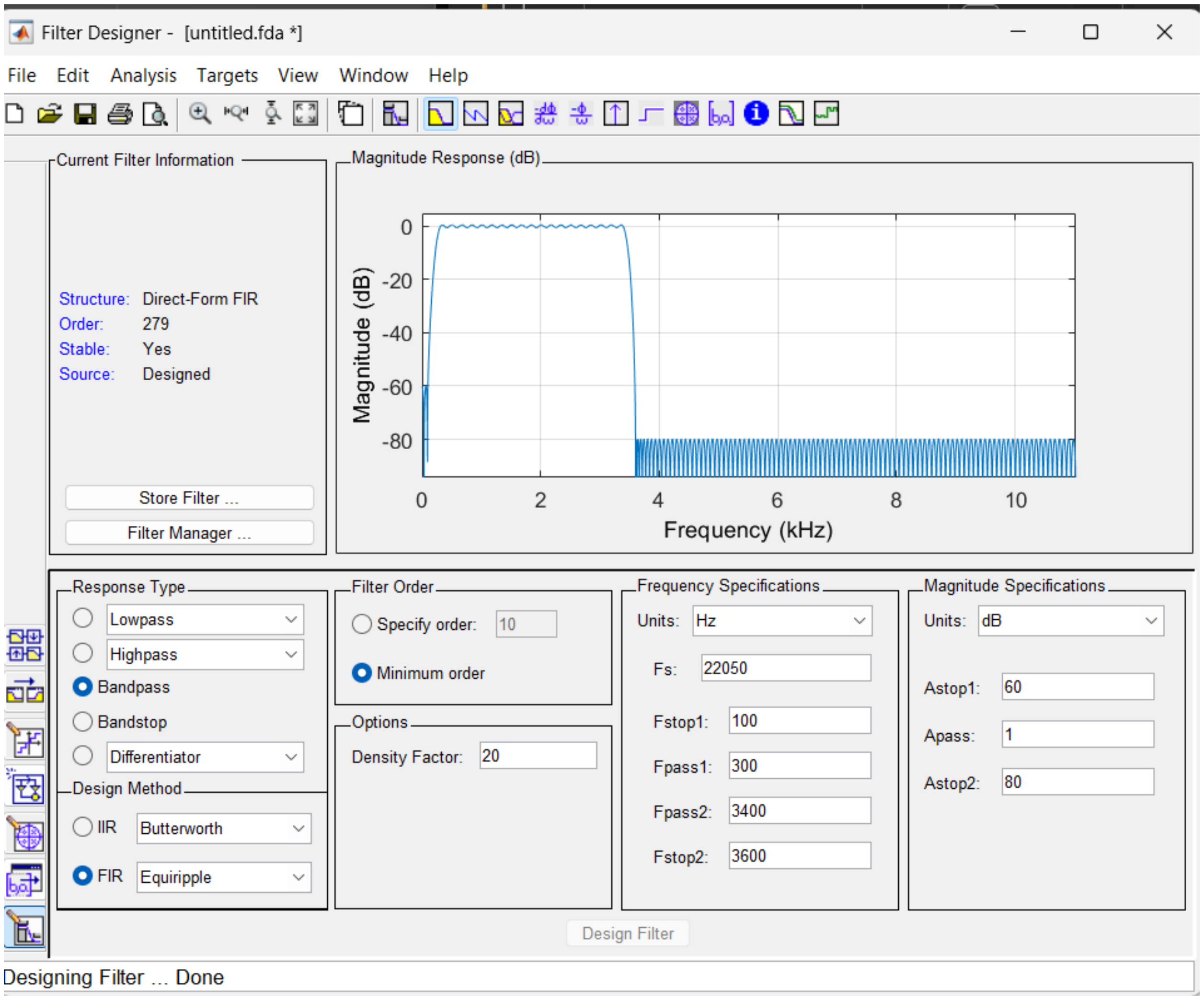
With the new filter, the recording is now more audible.



2. **Band pass filter:**

Now we will use the fdatool to design a FIR band-pass filter. Repeat the same procedure in high pass filter design but with different configurations:

- Sample rate: 22050 Hz.
- Fstop1: 100 Hz
- Fpass1: 300 Hz
- Fstop2: 3600 Hz
- Fpass2: 3400 Hz
- Apass: 60 dB
- Astop: 1dB


Filter Designer - [untitled.fda *]

File Edit Analysis Targets View Window Help

Current Filter Information

Structure: Direct-Form FIR
Order: 279
Stable: Yes
Source: Designed

Response Type: Lowpass, Highpass, [✓ Bandpass], Bandstop, Differentiator

Design Method: IIR Butterworth, [✓ FIR] Equiripple

Filter Order: Specify order: 10, [✓ Minimum order]

Options: Density Factor: 20

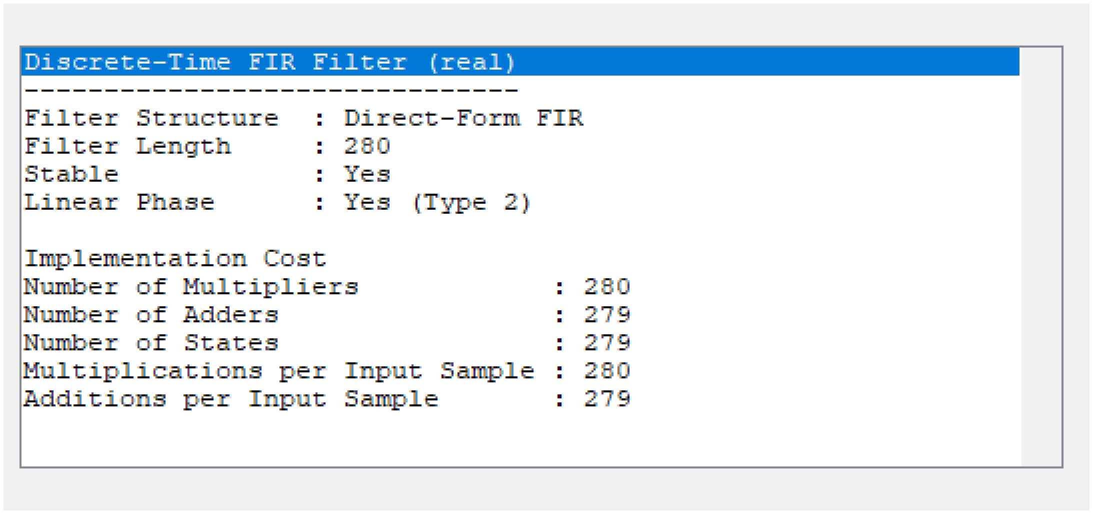
Frequency Specifications
Units: Hz
Fs: 22050
Fstop1: 100
Fpass1: 300
Fpass2: 3400
Fstop2: 3600

Magnitude Specifications
Units: dB
Astop1: 60
Apass: 1
Astop2: 80

Designing Filter ... Done

Figure 15. Band-pass filter configuration

The filter information includes:



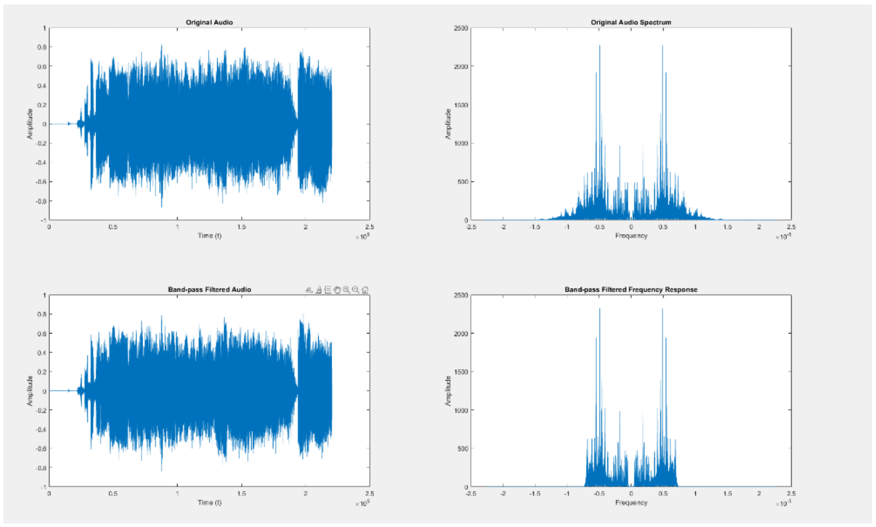
```
Discrete-Time FIR Filter (real)
-------------------------------
Filter Structure  : Direct-Form FIR
Filter Length     : 280
Stable            : Yes
Linear Phase      : Yes (Type 2)

Implementation Cost
Number of Multipliers            : 280
Number of Adders                 : 279
Number of States                 : 279
Multiplications per Input Sample : 280
Additions per Input Sample       : 279
```

The difference equation representation of FIR filter is:

$$y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_{M-1} x(n-M+1)$$

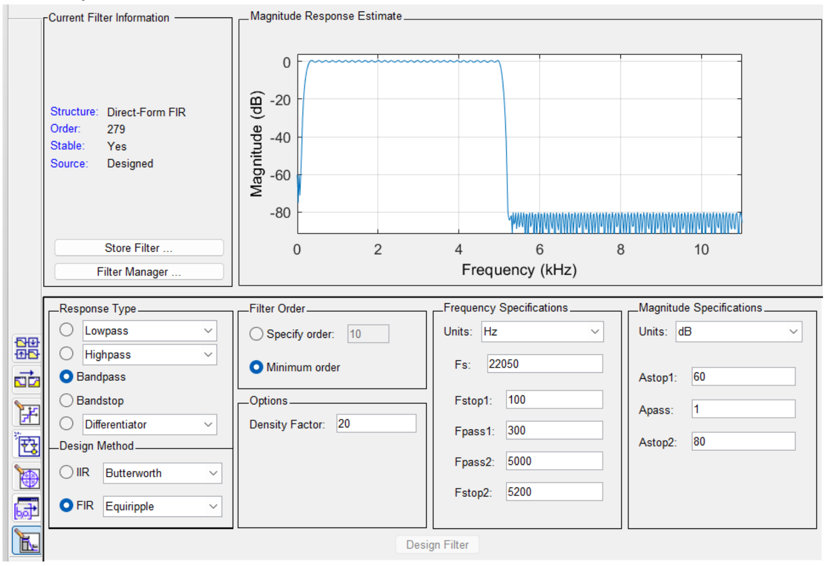
This means that this is a linear convolution of finite support. The order of the filter is M – 1 and the length of the filter is M. Hence, the FIR structures are always stable.

When we apply the filter to our audio we could hear that as if the song was sung from a far.



We alter the configuration a bit to make the higher sound passing through for clearer vocals, for that we use such variables:

- Sample rate: 22050 Hz.
- Fstop1: 100 Hz
- Fpass1: 300 Hz
- Fstop2: 5000 Hz
- Fpass2: 5200 Hz
- Apass: 60 dB
- Astop: 1dB



Current Filter Information

Structure: Direct-Form FIR
Order: 279
Stable: Yes
Source: Designed

Response Type: Lowpass, Highpass, [✓ Bandpass], Bandstop, Differentiator

Design Method: IIR Butterworth, [✓ FIR] Equiripple

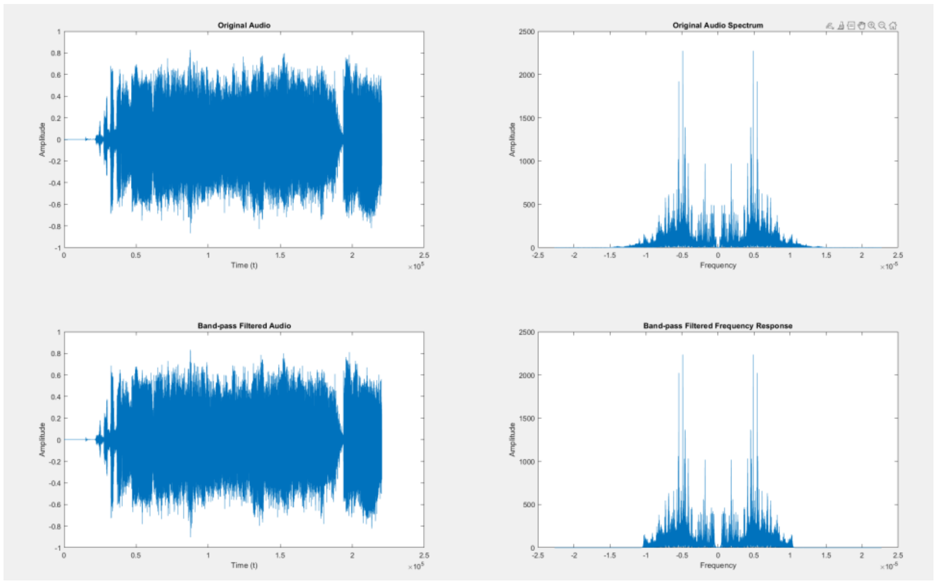
Filter Order: Specify order: 10, [✓ Minimum order]

Options: Density Factor: 20

Frequency Specifications
Units: Hz
Fs: 22050
Fstop1: 100
Fpass1: 300
Fpass2: 5000
Fstop2: 5200

Magnitude Specifications
Units: dB
Astop1: 60
Apass: 1
Astop2: 80

With this we have vividly sounded vocals, here the result graph



LAB 2:

The goal of this lab is to demonstrate a basic application of Notch IIR filter, which is rejecting hum noise. You will need to design several IIR filters to satisfy the given specifications.

Notch filter design

Notch filter is a filter that can reject a narrow frequency band and leave the rest of spectrum little changed. The frequency response of a single notch filter and its transfer function is shown below:

H(ejw)={o, w=w01, otherwise,*Hejw={1, otherwise,o, w=w0*

H(z)= b0(1−2cosw0z-1+z-21−2rcosw0z-1+r2z-2)*Hz=*
b01−2cos⁡w0z−1+z−21−2rcos⁡w0z−1+r2z−2

Where:

- b0*b0*

: Normalized notch angular frequency (the pole-zero angles on z plane)

- rr

: Distance between the pole and the origin

Code:

```
%% Parameters
notch_freq = 400; % Frequency to notch out (Hz)
Fs = 8000; % Sampling frequency (Hz)
w0 = 2*pi*notch_freq / Fs; % Normalized digital frequency
r = 0.6; % Pole radius (change this to test: e.g., 1, 0.95, 0.6)
%% Design Notch Filter
b = [1, -2*cos(w0), 1]; % Zeros on unit circle at ±w0
a = [1, -2*r*cos(w0), r^2]; % Poles inside unit circle at ±w0 with radius r
%% Plot Frequency Response of the Filter
figure;
freqz(b, a, 1024, Fs);
title(['Frequency Response of the Notch Filter (r = ', num2str(r), ')']);
%% Plot Pole-Zero Diagram
figure;
zplane(b, a);
title('Pole-Zero Plot of the Notch Filter');
hold on;
x_limits = xlim; y_limits = ylim;
plot(x_limits, [0 0], 'k--', 'LineWidth', 1.2); % Real axis
plot([0 0], y_limits, 'k--', 'LineWidth', 1.2); % Imag axis
hold off;
%% Load Signal with 400 Hz Hum
[y, Fs] = audioread('with_hum.wav');
%% Analyze Original Signal
[M_dB, f] = magnitude_response(y, Fs);
figure;
plot(f, M_dB, 'LineWidth', 1.5);
hold on;
plot([400 400], ylim, 'r--', 'LineWidth', 1.5);
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Magnitude Response of Original Signal (with 400 Hz Hum)');
legend('Original Signal', '400 Hz Hum');
grid on;
%% Apply Notch Filter to Remove 400 Hz Hum
y_filtered = filter(b, a, y);
%% Analyze Filtered Signal
[M_filtered_dB, f_filtered] = magnitude_response(y_filtered, Fs);
figure;
plot(f, M_dB, 'LineWidth', 1.5); % Original
hold on;
plot(f_filtered, M_filtered_dB, 'LineWidth', 1.5); % Filtered
plot([400 400], ylim, 'r--', 'LineWidth', 1.5); % 400 Hz line
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title(['Magnitude Response: Original vs. Filtered (r = ', num2str(r), ')']);
legend('Original', ['Filtered (r = ', num2str(r), ')'], '400 Hz');
grid on;
xlim([0 Fs/2]);
%% Play Audio Before and After Filtering
disp('Playing original sound (with hum)...');
soundsc(y, Fs);
pause(length(y)/Fs + 1);
disp(['Playing filtered sound (r = ', num2str(r), ')...']);
soundsc(y_filtered, Fs);
```

Magnitude response function:

```
%Code magnitude response using to plot for notch filter
%Must use magnitude response instead of magnitude spectrum because computed
%and plot for notch filter
%Use magnitude spectrum for FFT of a signal, just show how much frequency
%is present in signal
function [M_dB, f] = magnitude_response(x, Fs)
% Compute the single-sided magnitude response (in dB)
N = length(x); % Length of the signal
Nf = floor(N/2); % Nyquist index (robust for even/odd lengths)
% Frequency axis (0 Hz to Nyquist frequency)
f = (0:Nf-1) * (Fs/N); % Correct frequency resolution
% Compute FFT and convert to dB
X = fft(x); % FFT of the signal
M = abs(X(1:Nf)); % Magnitude (linear scale)
M_dB = 20*log10(M + eps); % Convert to dB (add eps to avoid log(0))
end
```

a) Determine ω_0 the notch filter that can eliminate 400 Hz hum in section 3.1. Let $b0$ be equal to 1 and r be equal to 0.6. Calculate numerator and denominator coefficient vector of H(z). Plot H(z) using freqz function.

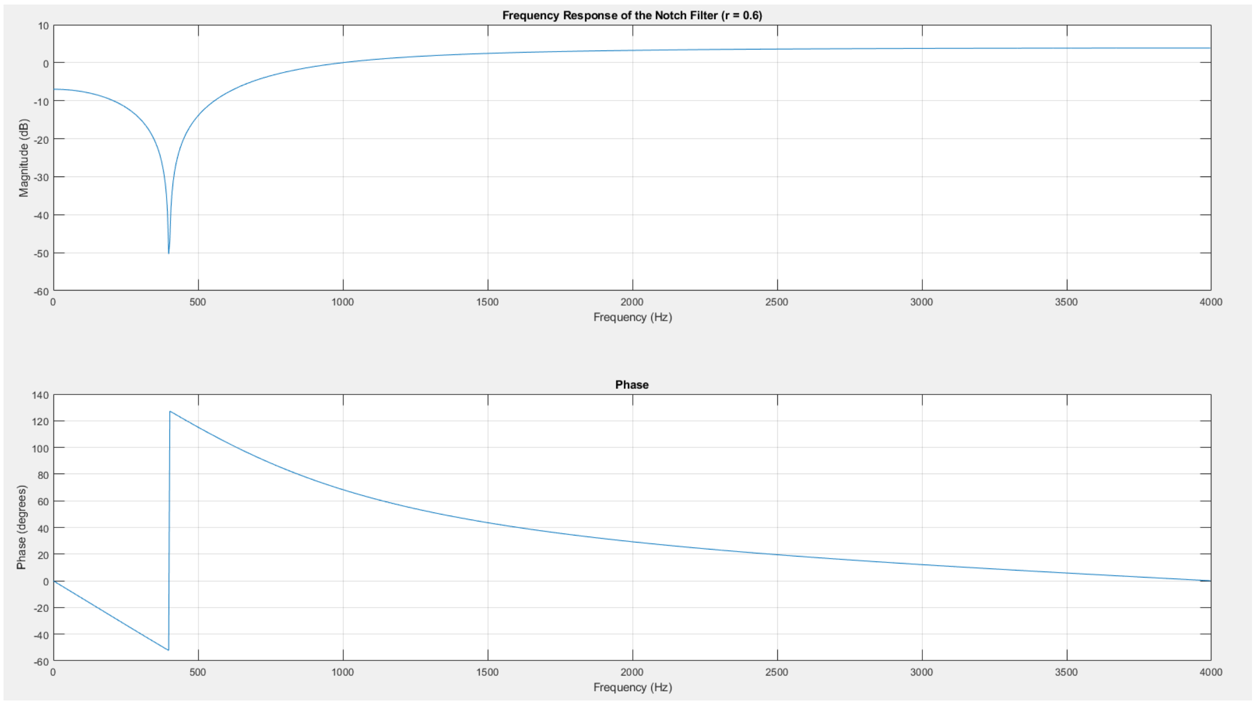


Figure 16 : Frequency Response

Comment: for r = 0.6 we can see that the narrow band filter did not just cut the 400 Hz but the frequency from 0 – 800 Hz are also affected, making this filter bad.

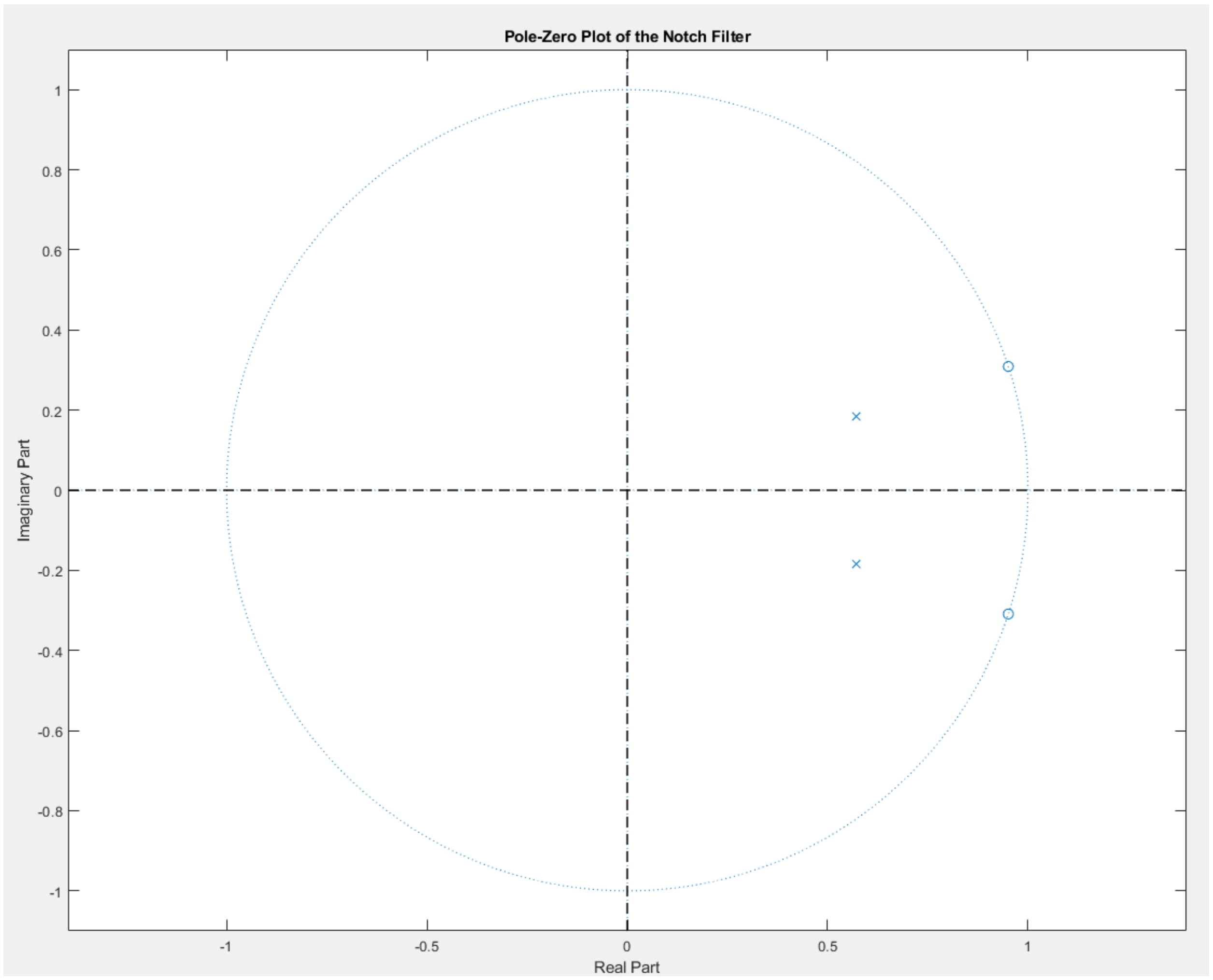


Figure 17: Pole-zero plot

We can see that with the pole is quite far from the zero, so the narrow band is not clear and sharp.

b) Load with_hum.wav and plot its magnitude response

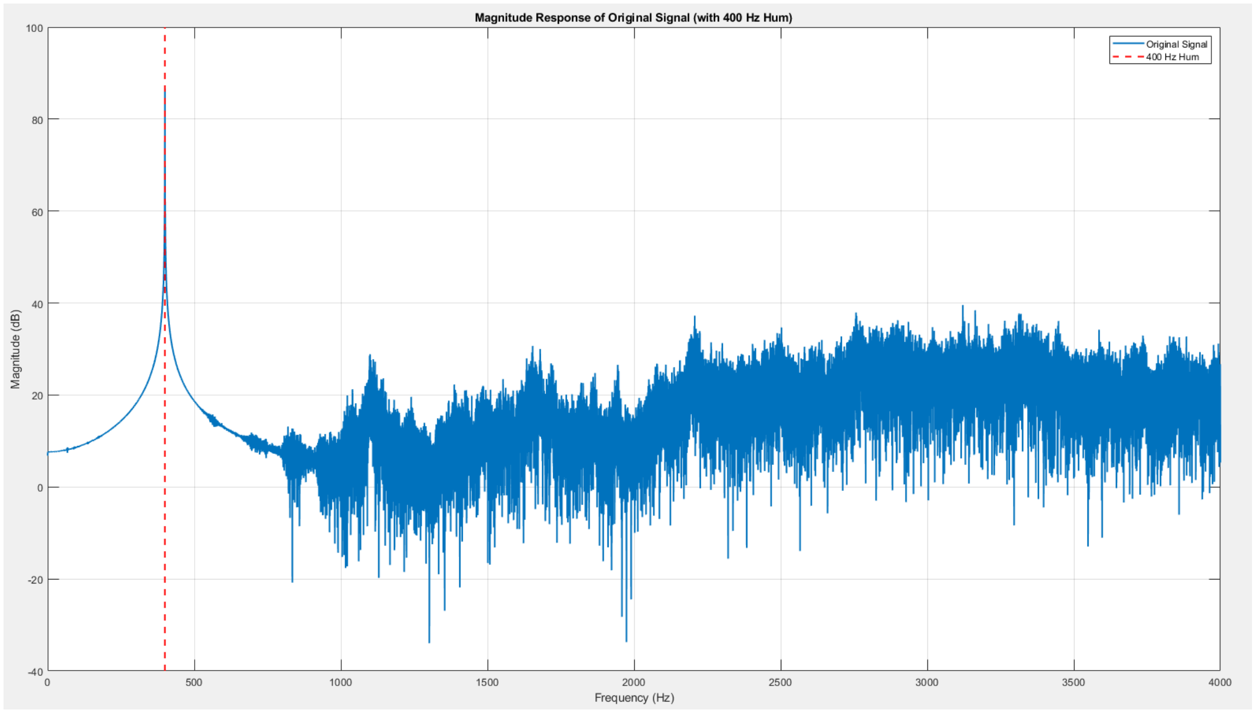


Figure 18: Magnitude Response of Original Signal

Comment: The sharp point is the noise that we introduce at 400 Hz

c) Eliminate the 400 Hz hum by using filter function with parameters determined in step 1. Play and comment about the sound? Does the noise decrease?

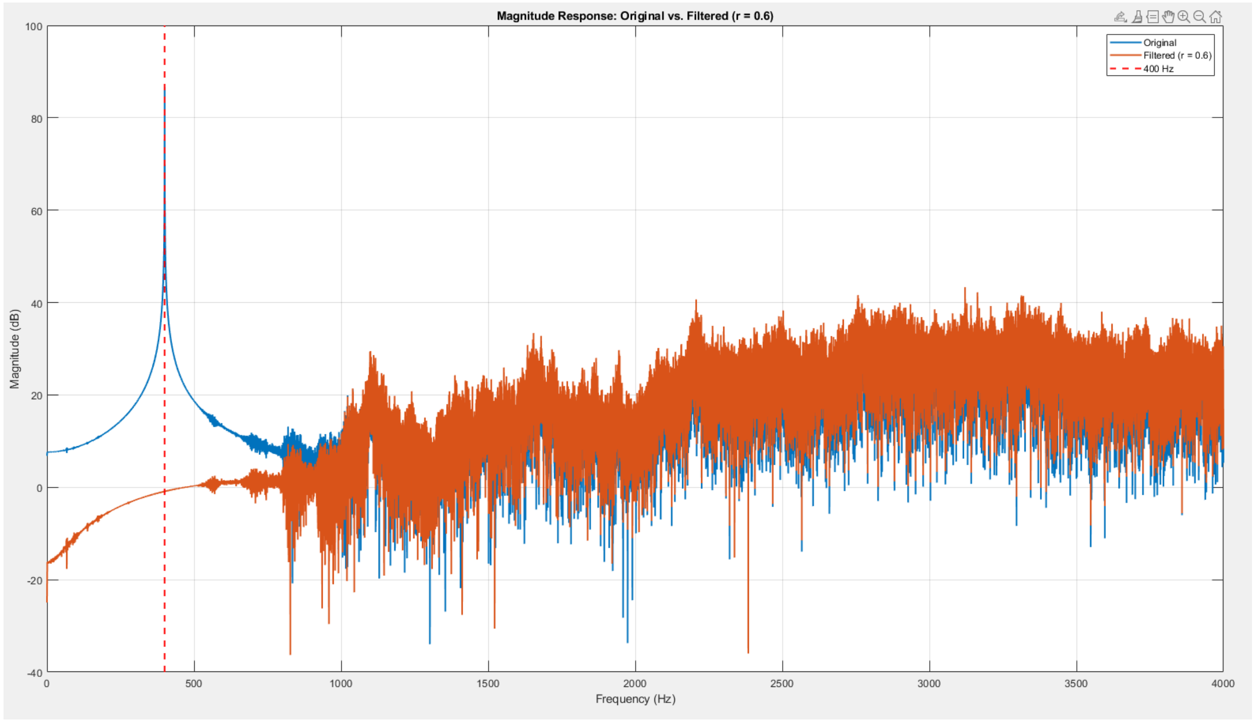


Figure 19: After apply r = 0.6 into the original sound

Comment: After we apply the filter in we see the noise has been filtered, so does the sound in the range of 0 to 800 Hz

d) Change r to 0.95, repeat the above steps. Does the output sound better? Can the coefficient r be greater or equal to 1? Explain why?

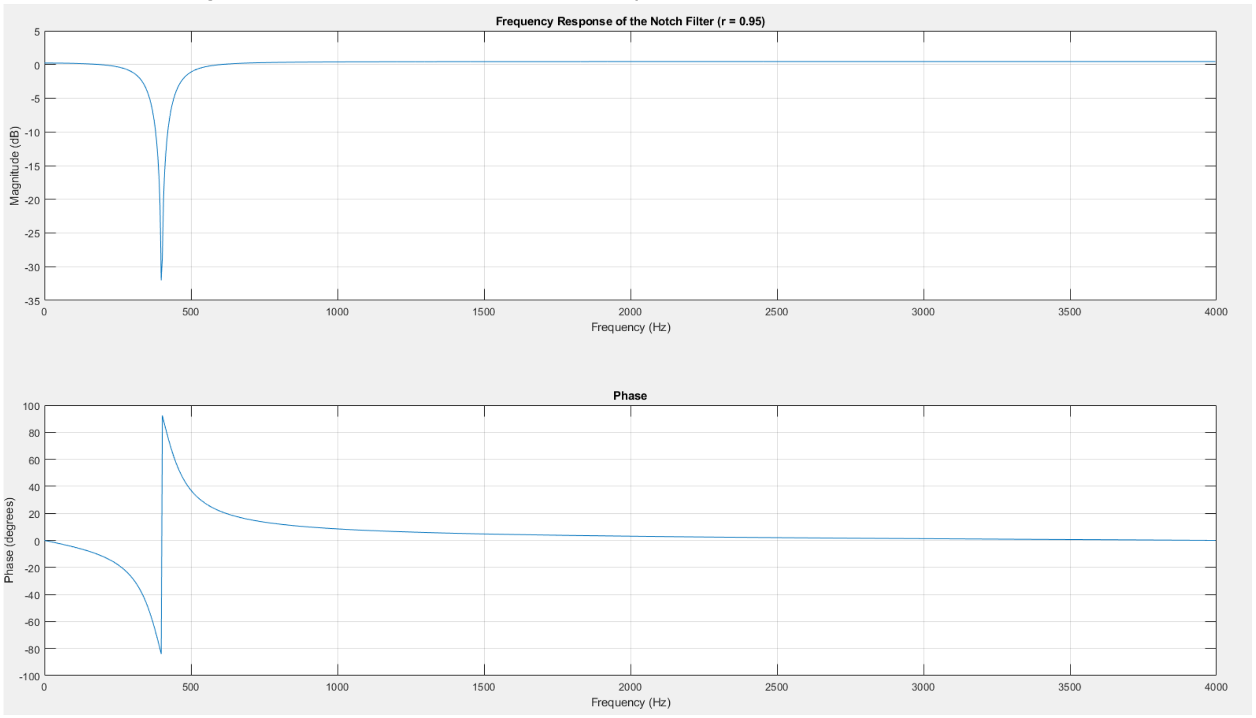


Figure 20: Frequency response while changing from

Comment: As r get close to 1, we see that the narrow band focus sharply at the 400Hz noise, making the effect clear and sharp.

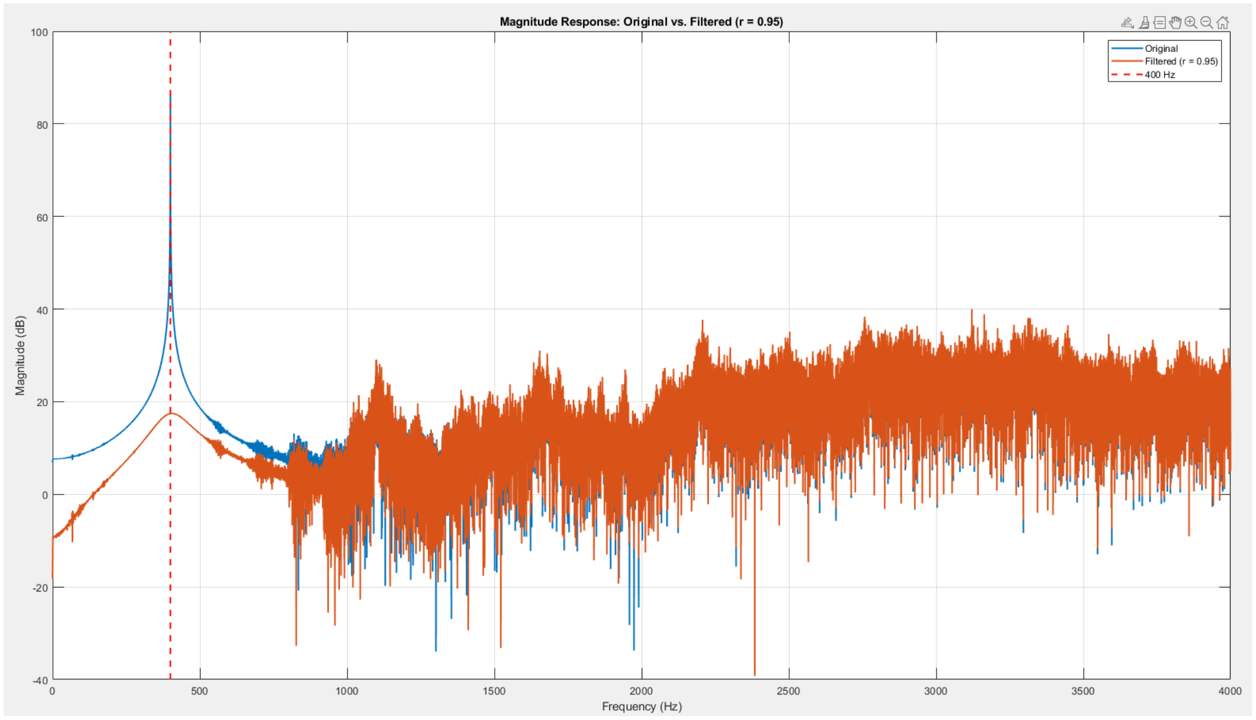
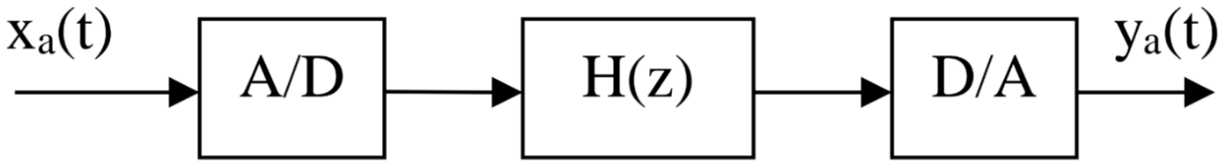


Figure 21: Filtered for original sound

Comment: We see that the noise has been filter and the near by frequency mostly stay unaffected.

IIR filter design using MATLAB

Our task is to design a digital high pass filter
H(z)$H(z)$
to be used in the following structure:



Along with these specifications:

- Sampling rate: $10kHz$
- Stopband edge: $1.5kHz$ with attenuation: $40dB$
- Passband edge: $2kHz$ with ripple of $3dB$
- Monotone passband and stopband, and impulse invariance transformation method

Since our sampling rate is $10kHz$, the stopband edge of $1.5kHz$ and passband of $2kHz$

```
clear; clc; close all;
Fs = 10000;             % Actual Sampling frequency (Hz)
T = 1/Fs;               % CORRECT Physical Sampling interval (s)
% Target Digital Highpass Filter Specifications (Used for w_p, w_s, R_p, A_s)
fp = 2000;              % Target Passband edge frequency (Hz)
fs = 1500;              % Target Stopband edge frequency (Hz)
Rp = 3;                 % Target Passband ripple (dB)
As = 40;                % Target Stopband attenuation (dB)

fprintf('Physical Sampling Frequency Fs = %.0f Hz\n', Fs);
fprintf('Physical Sampling Interval T = 1/Fs = %g s\n\n', T);
fprintf('Target Digital Highpass Filter Specs (Used for frequency points):\n');
fprintf('  fp = %.0f Hz, fs = %.0f Hz, Rp = %.1f dB, As = %.1f dB\n', fp, fs, Rp,
As);

% Target Digital Frequencies (rad/sample)
omega_p = 2*pi*fp / Fs;   % Target digital passband edge for HPF (0.4*pi
rad/sample)
omega_s = 2*pi*fs / Fs;   % Target digital stopband edge for HPF (0.3*pi
rad/sample)
fprintf('Corresponding Target Digital Frequencies (rad/sample):\n');
fprintf('  omega_p = %.4f (%.3f pi), omega_s = %.4f (%.3f pi)\n\n', omega_p,
omega_p/pi, omega_s, omega_s/pi);

% Intermediate Digital Lowpass Prototype Specs
% Use the user-specified passband edge for the digital LPF prototype
wplp = 0.306 * pi; % SPECIFIED digital LPF Passband edge frequency (rad/sample)
```

```
fprintf('Intermediate Digital LPF Prototype Specs:\n');
fprintf('  Chosen Passband Edge wplp = %.4f (%.3f pi) rad/sample\n', wplp,
wplp/pi);
% Calculate transformation parameter alpha and corresponding digital LPF stopband
edge
alpha = -(cos((wplp+omega_p)/2))/(cos((wplp-omega_p)/2));
wslp = angle(-(exp(-j*omega_s)+alpha)/(1+alpha*exp(-j*omega_s))); % Calculate
digital LPF stopband
fprintf('  Calculated alpha = %.4f\n', alpha);
fprintf('  >>> Calculated Stopband Edge wslp = %.4f (%.3f pi) rad/sample
<<<\n\n', wslp, wslp/pi);
% --- Analog Prototype Specifications (Mapping from Digital LPF using T=1/Fs) ---
fprintf('Mapping Digital LPF Specs to Analog LPF Specs using T = 1/Fs = %g:\n',
T);
OmegaP = wplp / T; % Analog Prototype Passband edge (rad/s)
OmegaS = wslp / T; % Analog Prototype Stopband edge (rad/s)
fprintf('  Analog Prototype Passband Edge OmegaP = %.4f rad/s\n', OmegaP);
fprintf('  Analog Prototype Stopband Edge OmegaS = %.4f rad/s\n\n', OmegaS);

% Design Analog Lowpass Butterworth Prototype Filter
fprintf('Designing Analog LPF Prototype (cs, ds) using afd_butt...\n');
% Uses user's afd_butt (needs u_buttap) with calculated OmegaP, OmegaS
[cs,ds] = afd_butt(OmegaP, OmegaS, Rp, As);
fprintf('  Analog LPF Numerator (cs): '); disp(cs);
fprintf('  Analog LPF Denominator (ds): '); disp(ds); fprintf('\n');

% Convert Analog LPF to Digital LPF Prototype using Impulse Invariance (T=1/Fs)
fprintf('Converting Analog LPF to Digital LPF (blp, alp) using imp_invr with
T=1/Fs');
% Uses user's imp_invr function with the correct physical T
[blp,alp] = imp_invr(cs, ds, T);
fprintf('  Digital LPF Numerator (blp): '); disp(blp);
fprintf('  Digital LPF Denominator (alp): '); disp(alp); fprintf('\n');

% Convert Digital LPF to Final Digital HPF using Z-Mapping
fprintf('Converting Digital LPF to Digital HPF (b, a) using zmapping...\n');
Nz = -[alpha, 1]; % Transformation numerator polynomial (using calculated alpha)
Dz = [1, alpha];  % Transformation denominator polynomial (using calculated
alpha)
% Uses user's zmapping function
[b, a] = zmapping(blp, alp, Nz, Dz);
fprintf('Final Digital HPF Numerator (b): '); disp(b);
fprintf('Final Digital HPF Denominator (a): '); disp(a); fprintf('\n');
```

```
% Plotting
%(a): Analog Lowpass Prototype
fprintf('Plotting (a): Analog Lowpass Prototype Response...\n');
figure('Name', 'Plot (a): Analog LPF Prototype', 'NumberTitle', 'off');
try
    wmax_plot_a = OmegaS * 1.5; % Plot range in rad/s
    [db_a, mag_a, ~, w_a] = freqs_m(cs, ds, wmax_plot_a); % Use freqs_m
    f_a = w_a / (2*pi); % Convert rad/s to Hz
    plot(f_a, db_a); % Plot vs Frequency (Hz)
    title({'a) Relative Mag. Response of Analog LPF Prototype (cs, ds)',
'[freqs\_m]'});
    xlabel('Frequency (Hz)');
    ylabel('Relative Magnitude (dB)'); grid on;
    ylim([-As - 10, 5]);
    hold on;
    % Add spec lines for the ANALOG LP prototype (OmegaP, OmegaS)
    lp_pass_gain = max(db_a(f_a <= OmegaP/(2*pi))); % Estimate passband gain near
DC
    plot([OmegaP/(2*pi), OmegaP/(2*pi)], [lp_pass_gain-Rp, lp_pass_gain+5], 'r--
', 'LineWidth', 1.2, 'DisplayName', sprintf('f_{P}=%.1f Hz',OmegaP/(2*pi))); %
Adjust precision
    plot([0, OmegaP/(2*pi)], [lp_pass_gain-Rp, lp_pass_gain-Rp], 'r--',
'LineWidth', 1.2, 'HandleVisibility','off');
    plot([OmegaS/(2*pi), OmegaS/(2*pi)], [get(gca,'YLim')*[1;0]+5, lp_pass_gain-
As], 'm--', 'LineWidth', 1.2, 'DisplayName', sprintf('f_{S}=%.1f
Hz',OmegaS/(2*pi))); % Adjust precision
    plot([OmegaS/(2*pi), wmax_plot_a/(2*pi)], [lp_pass_gain-As, lp_pass_gain-As],
'm--', 'LineWidth', 1.2, 'HandleVisibility','off');
    legend('show','Location','southwest');
    hold off;
catch ME
    title({'a) Plotting Failed for Analog LP Prototype', ME.message});
    grid on;
end

%(b): Digital Lowpass Prototype
fprintf('Plotting (b): Digital Lowpass Prototype Response...\n');
figure('Name', 'Plot (b): Digital LPF Prototype', 'NumberTitle', 'off');
try
    [db_b, mag_b, ~, ~, w_b] = freqz_m(blp, alp); % Use freqz_m
    plot(w_b/pi, db_b); % Plot vs Normalized Frequency (x pi rad/sample)
    title({'b) Relative Mag. Response of Digital LPF Prototype (blp, alp)',
'[imp\_invr(T=1/Fs) + freqz\_m]'});
    xlabel('Normalized Frequency (\times\pi rad/sample)');
    ylabel('Relative Magnitude (dB)'); grid on;
    ylim([-As - 10, 5]); xlim([0 1]);
    hold on;
    % Add spec lines for the intermediate DIGITAL LPF (wplp, wslp)
    dlp_pass_gain = max(db_b(w_b <= wplp)); % Estimate passband gain near DC
    plot([wplp/pi, wplp/pi], [dlp_pass_gain-Rp, dlp_pass_gain+5], 'r--',
'LineWidth', 1.2, 'DisplayName', sprintf('\\omega_{plp}=%.3f\\pi',wplp/pi));
    plot([0, wplp/pi], [dlp_pass_gain-Rp, dlp_pass_gain-Rp], 'r--', 'LineWidth',
1.2, 'HandleVisibility','off');
    plot([wslp/pi, wslp/pi], [get(gca,'YLim')*[1;0]+5, dlp_pass_gain-As], 'm--',
'LineWidth', 1.2, 'DisplayName', sprintf('\\omega_{slp}=%.3f\\pi',wslp/pi));
    plot([wslp/pi, 1], [dlp_pass_gain-As, dlp_pass_gain-As], 'm--', 'LineWidth',
1.2, 'HandleVisibility','off');
    legend('show','Location','southwest');
    hold off;
catch ME
    title({'b) Plotting Failed for Digital LP Prototype', ME.message});
    grid on; xlim([0 1]); ylim([-50 5]);
end

%(c): Overall Digital Highpass Filter
fprintf('Plotting (c): Final Digital Highpass Filter Response...\n');
figure('Name', 'Plot (c): Final Digital HPF', 'NumberTitle', 'off');
try
    [db_c, mag_c, ~, ~, w_c] = freqz_m(b, a); % Use freqz_m for final filter
(b,a)
    plot(w_c/pi * Fs/2000, db_c); % Plot vs Frequency in kHz (w/pi * (Fs/2)/1000)
    title({'c) Relative Mag. Response of Final Digital HPF (b, a)', '[zmapping +
freqz\_m]'});
    xlabel('Frequency (kHz)');
    ylabel('Relative Magnitude (dB)'); grid on;
    ylim([-As - 10, 5]);
    xlim([0 Fs/2000]); % Plot up to Nyquist frequency in kHz
    hold on;
    % Add spec lines for the TARGET DIGITAL HPF (fp, fs in kHz)
    passband_indices = w_c >= omega_p;
    if any(passband_indices)
        dhp_pass_gain = max(db_c(passband_indices));
    else % Fallback if passband starts exactly at Nyquist or filter fails
        dhp_pass_gain = db_c(end);
        if isnan(dhp_pass_gain) || isinf(dhp_pass_gain), dhp_pass_gain = 0; end %
Further fallback
    end
    plot([fp/1000, fp/1000], [dhp_pass_gain-Rp, dhp_pass_gain+5], 'r--',
'LineWidth', 1.2, 'DisplayName', sprintf('Pass Edge (%.1f kHz)', fp/1000));
```

```
    plot([fp/1000, Fs/2000], [dhp_pass_gain-Rp, dhp_pass_gain-Rp], 'r--',
'LineWidth', 1.2, 'HandleVisibility','off');
    plot([fs/1000, fs/1000], [get(gca,'YLim')*[1;0]+5, dhp_pass_gain-As], 'm--',
'LineWidth', 1.2, 'DisplayName', sprintf('Stop Edge (%.1f kHz)', fs/1000));
    plot([0, fs/1000], [dhp_pass_gain-As, dhp_pass_gain-As], 'm--', 'LineWidth',
1.2, 'HandleVisibility','off');
    legend show;
    hold off;
catch ME
    title({'c) Plotting Failed for Final Digital HPF', ME.message});
    grid on; xlim([0 Fs/2000]); ylim([-50 5]);
end
```

a) Plot the magnitude response of the analog lowpass prototype

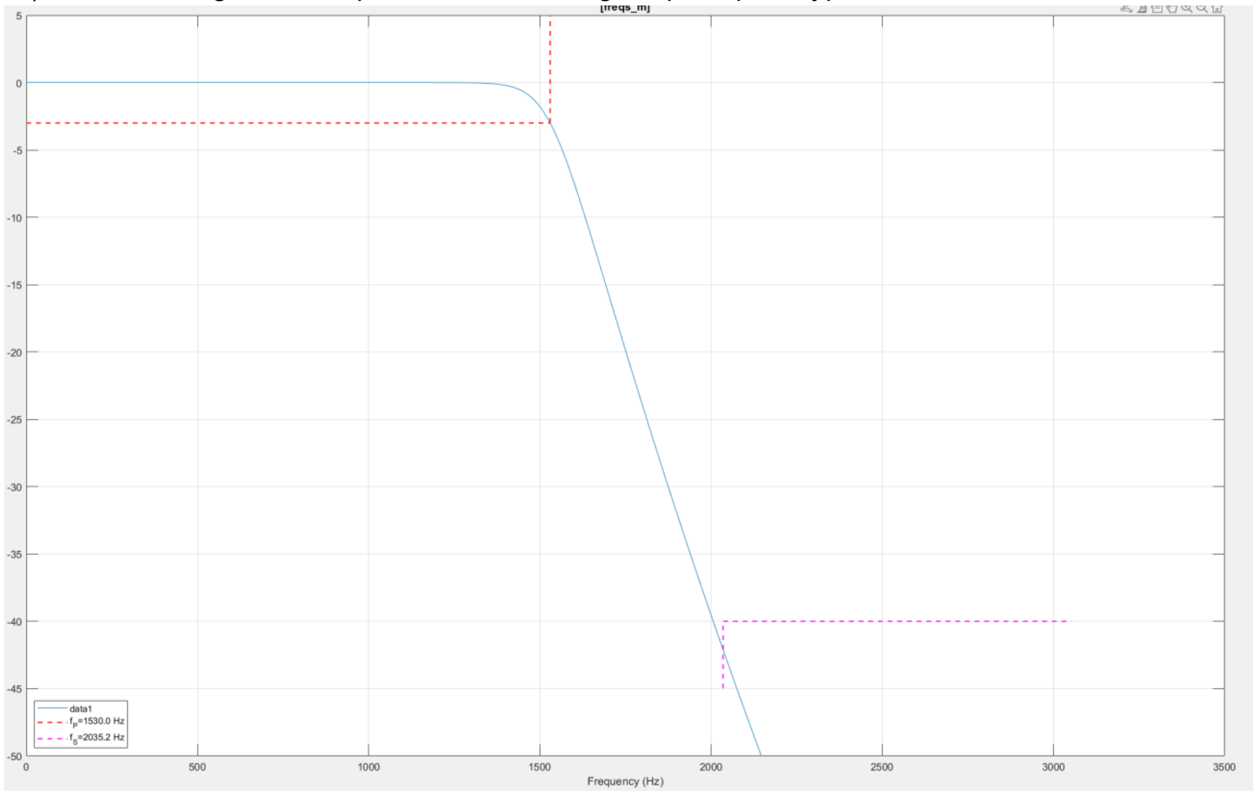


Figure 22:Magnitude response of the analog lowpass prototype

Comment: The Analog filter did indeed meet all the demand of the design, it is monotone, with Rp = 3dB and do not exceed the stop band of 40dB, and the cutoff band is in he range of 1.5 kHz to 2 kHz like intended

b) Plot the magnitude response of the digital lowpass prototype

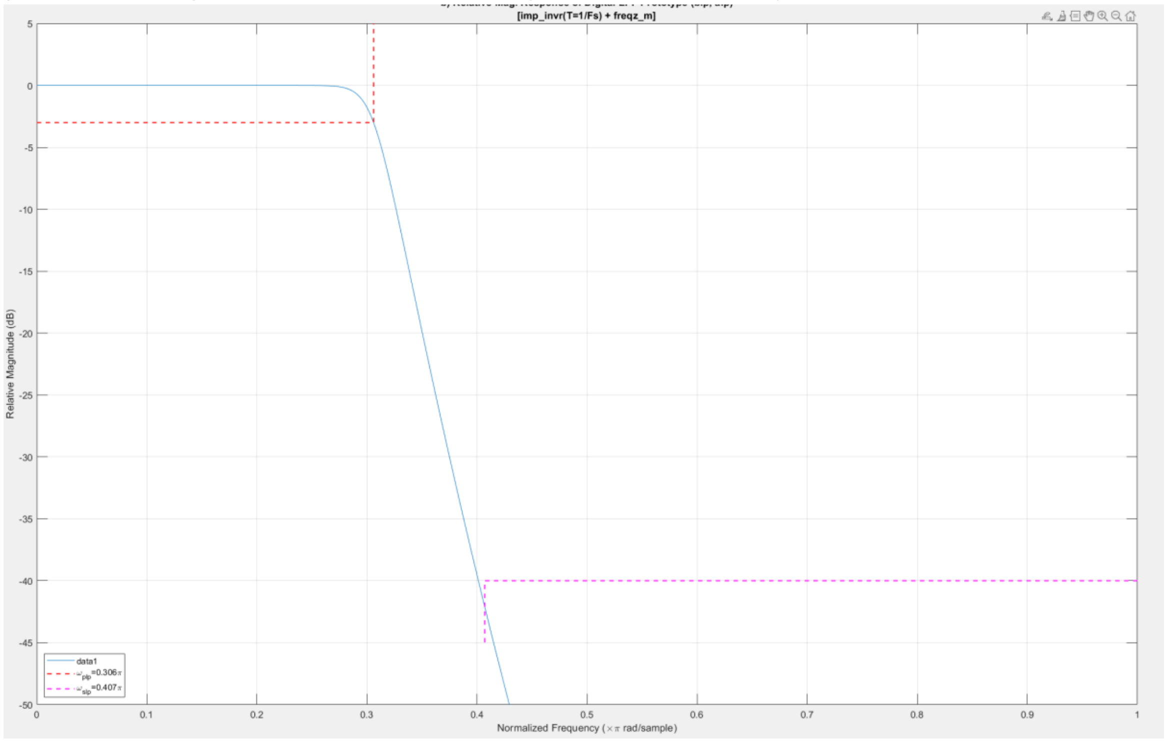


Figure 23: Magnitude response of the digital lowpass prototype

Comment: By using impulse invariance, we change from analog to digital lowpass filter without changing the specification.

c) Plot magnitude response of the overall Digital filter over the [0, 5Kh] interval

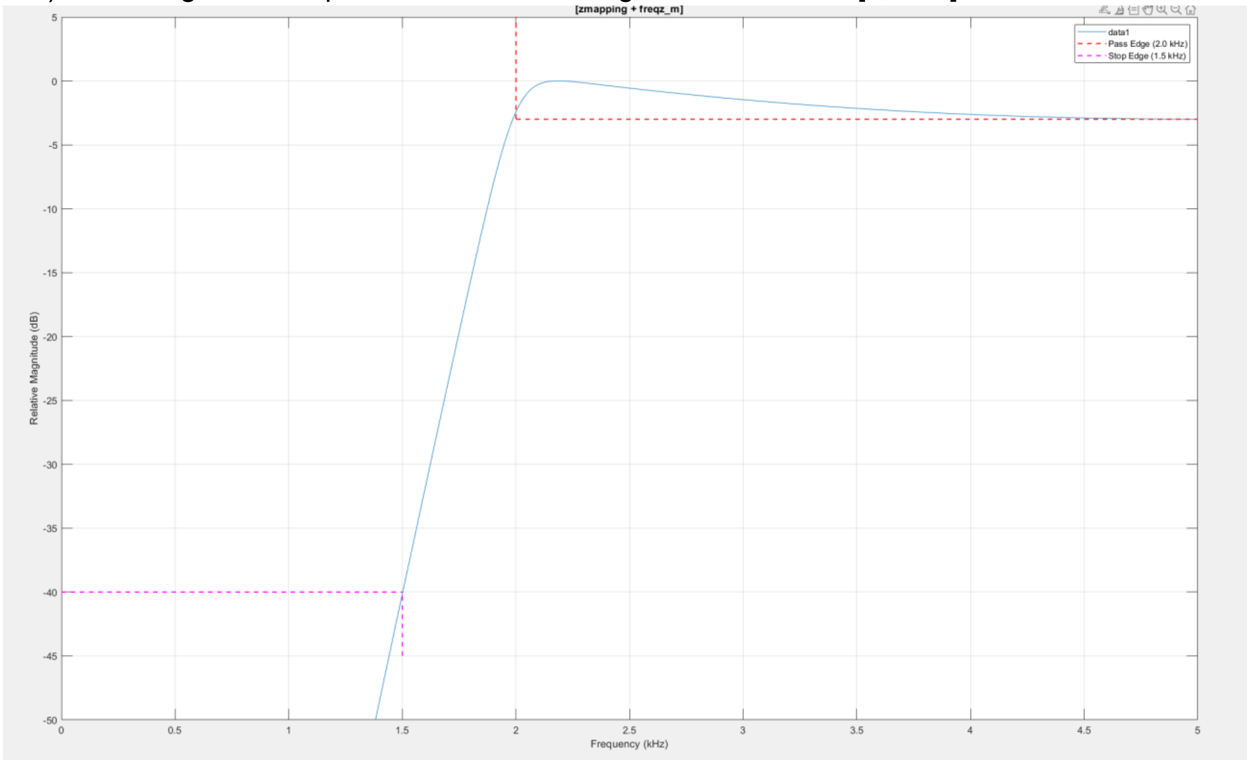


Figure 24: The overall digital highpass filter

Comment: By using z-mapping we successfully design a digital highpass filter as intended for this laboratory.

d) What limitations must be placed on the input signals so that the above structure truly acts like a high pass filter to them?

The input signal xa(t) must be band-limited to frequencies below the Nyquist frequency ($\frac{Fs}{2}$=5 kHz, with Fs=10kHz) to avoid aliasing. If the signal contains frequencies above 5 kHz, an anti-aliasing lowpass filter must be applied before sampling to ensure the system acts as a high pass filter.