Boran Kılıç - 22103444

EEE 424 Coding Assignment 2

In this assignment, the aim was to design and implement a linear-phase FIR bandpass filter.

Q1) Bandpass Filter Design

In this part, the task was to design a linear-phase bandpass filter of length L = 10 that is selective to the human voice range that is f_{cl} = 300Hz to f_{ch} = 3 kHz. The filter is designed for audio recordings sampled at f_s = 20kHz.

Two methods have been implemented and compared for this task. First one is FIR filter design by windowing method and the other is Minimax FIR filter design more specifically Parks–McClellan algorithm also known as Chebyshev approximation.

FIR Filter Design by windowing

Since we are designing a digital filter we need to convert the cutoff frequencies from Hz to w randian/sample. Because an analog filter is usually continuous - aperiodic, therefore its Fourier transform should also be continuous-aperiodic. However, when the analog filter is sampled to get a digital filter it becomes discrete-aperiodic. Therefore, its Fourier transform should be periodic-continuous. So in order to fit into [-π,π] interval the cut-off frequencies calculated as:

$$w_{cl} = \frac{2\pi f_{cl}}{f_s}, \quad w_{ch} = \frac{2\pi f_{ch}}{f_s}$$

The required bandpass filter will realized with a difference of two ideal low pass filters with cutoffs w_{cl} and w_{ch}.

Ideal lowpass filter defined as:

$$h_{LPF}[n] = \begin{cases} \frac{\sin(w_c n)}{\pi n}, & n \neq 0 \\ \frac{w_c}{\pi}, & n = 0 \end{cases}$$

Therefore, ideal bandpass filter can be found as:

$$h_{\mathrm{BPF}}[n] = \frac{\sin(w_{\mathrm{ch}} n) - \sin(w_{\mathrm{cl}} n)}{\pi n}$$

This process is done in MATLAB with the following code snippet:

```
w_low = 2*pi*f_low/fs;
w_high = 2*pi*f_high/fs;

h_ideal = (sin(w_high*n) - sin(w_low*n)) ./ (pi*n);
```

Since length of the filter L = 10 is even; in order to filter to have linear phase, the filter should be a **Type II** filter with fractional delay (L-1)/2 = 4.5 and zero phase. For that reason the n is defined as:

```
n = -(L-1)/2 : (L-1)/2;
```

Impulse response values are calculated from the continuous version and all the points are shifted to right by (L-1)/2 so that the filters taps correspond to integers and midpoint will be at (L-1)/2. The impulse response of the filter can be written as:

$$h_{BPF}[n] = \begin{cases} \frac{\sin(w_{\text{high}}(n-\frac{L-1}{2}))-\sin(w_{\text{low}}(n-\frac{L-1}{2}))}{\pi(n-\frac{L-1}{2})}, & n = 0, 1, \ldots, L-1 \\ 0, & ow \end{cases}$$

For plotting reasons, to be able to shift the filter to [0,9] range in MATLAB simply a new index array is defined:

```
n1 = 0:L-1;
```

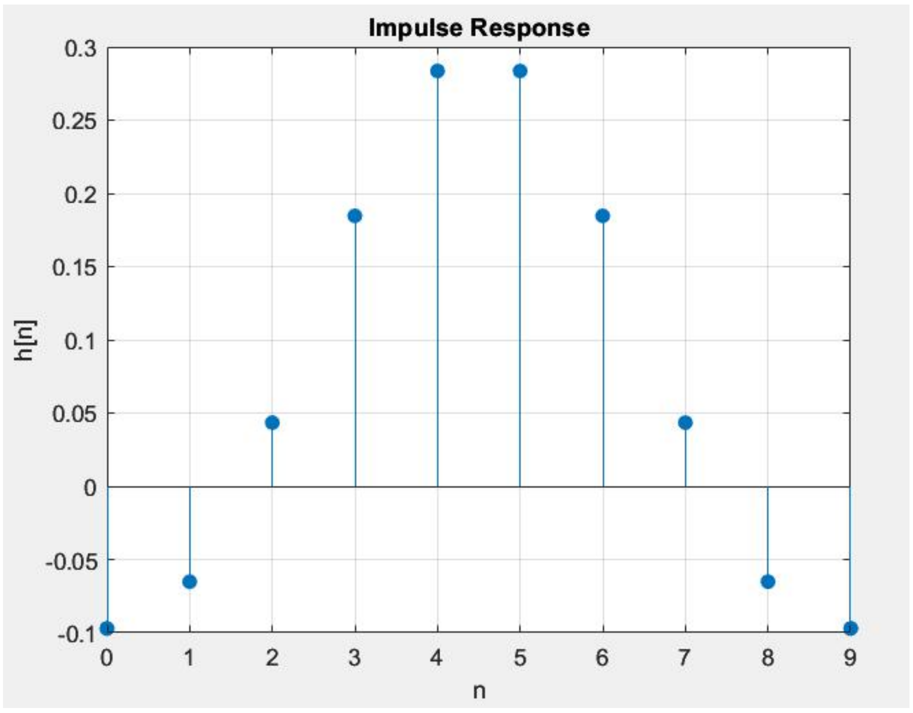


Figure 1: Impulse Response of the FIR BPF

The *Z*-Transform is calculated as follows:

$$H_{\text{BPF}}(z) = \sum_{n=0}^{9} h_{\text{BPF}}[n]\, z^{-n}$$

Taking the symmetry into account:

$$H_{\text{BPF}}(z) = z^{-4.5} \sum_{n=0}^{4} h[n] \left(z^{-(n-4.5)} + z^{n-4.5}\right)$$

Here $z^{-4.5}$ term corresponds to 4.5 tap delay.

Pole-Zero diagram is plotted in **Figure 2**, as expected, the filter has 9 zeros and 0 poles proving the FIR characteristic.

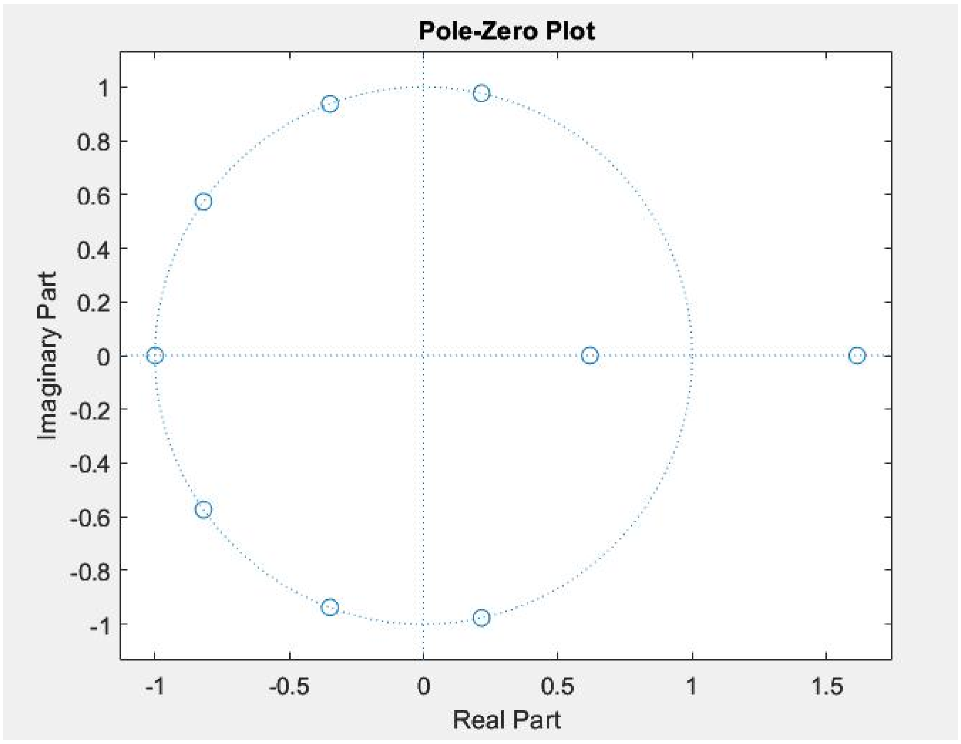


Figure 2: Pole-Zero Plot of the FIR BPF

The magnitude response it plotted in **Figure 3**, here -3dB line corresponding to 0.707x(maximum magnitude) is drawn to show the cutoff frequencies. Estimated cutoffs are:

$$f_{cl} = 341\ Hz \quad , \quad f_{hc} = 2998\ Hz$$

That are very close to the requirements, however this is achieved by slightly increasing the pass-band in the code by trial and error method (see Appendix A).

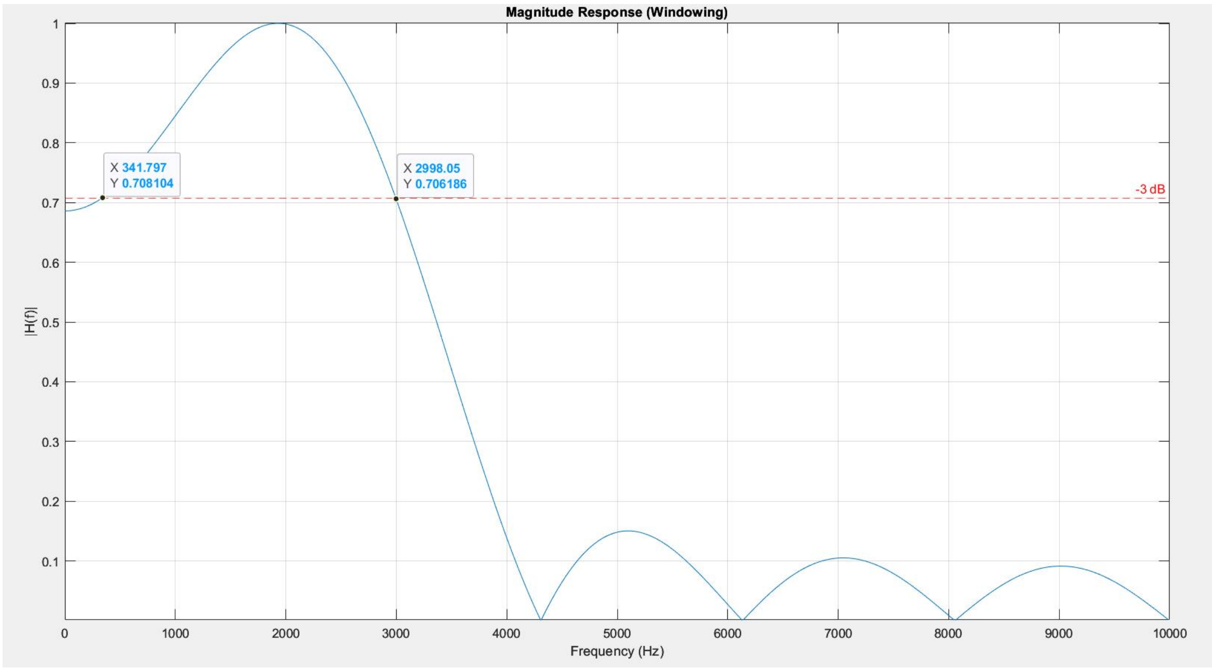


Figure 3: Magnitude Response of the FIR BPF

Phase Response is plotted in **Figure 4**, ignoring the jumps it is visible that the filter has a linear phase as intended. The jumps due to MATLAB fitting the phase between [0, -2π] by adding π whenever the phase exceeds the limit.

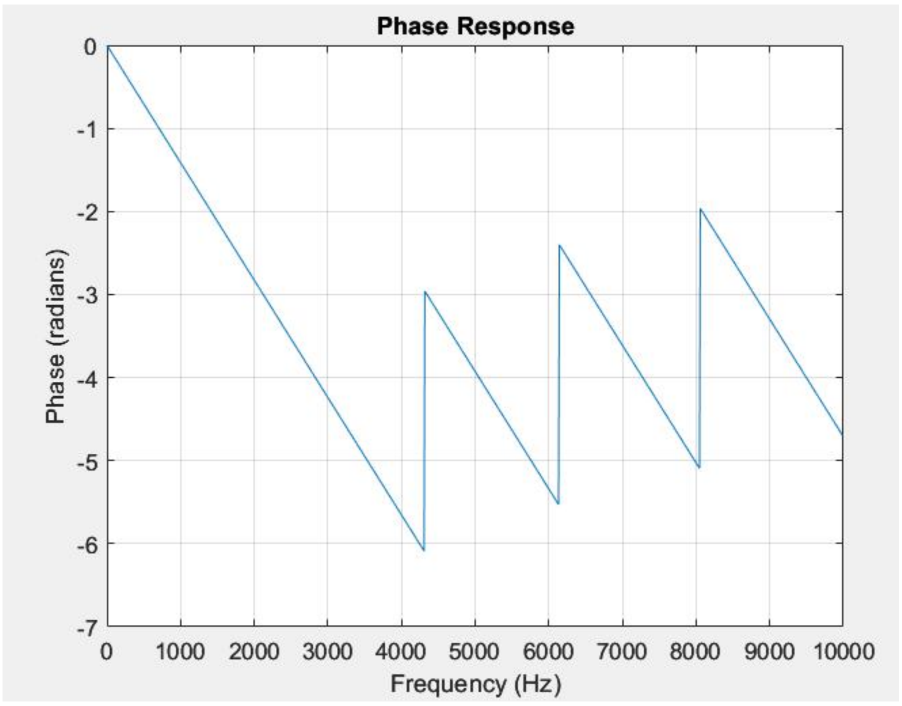


Figure 4: Phase Response of the FIR BPF

In order to show mathematically, writing the frequency response by substituting z = e^{jw}:

$$H_{\mathrm{BPF}}(e^{j\omega}) = e^{-j4.5\omega}\sum_{n=0}^{4} h[n]\left(\cos\left(\frac{\omega n - 4.5\omega}{2}\right)\right)$$

$$\angle H_{\mathrm{BPF}}(e^{j\omega}) = -4.5\,\omega$$

Therefore the phase is linear.

By far, we have used rectangular window to trim the impulse response of the ideal BPF. However, in **Figure 3** the ripples in the stopband is visible. In order to to reduce to stopband ripple one might use other windows such as Hamming, Blackman or Triangular. This windows reduce the ripples by smoothing the transitions at the edges of the window, effectively tapering the impulse response and thereby reducing the abrupt truncation effects that cause ripples in the frequency domain.

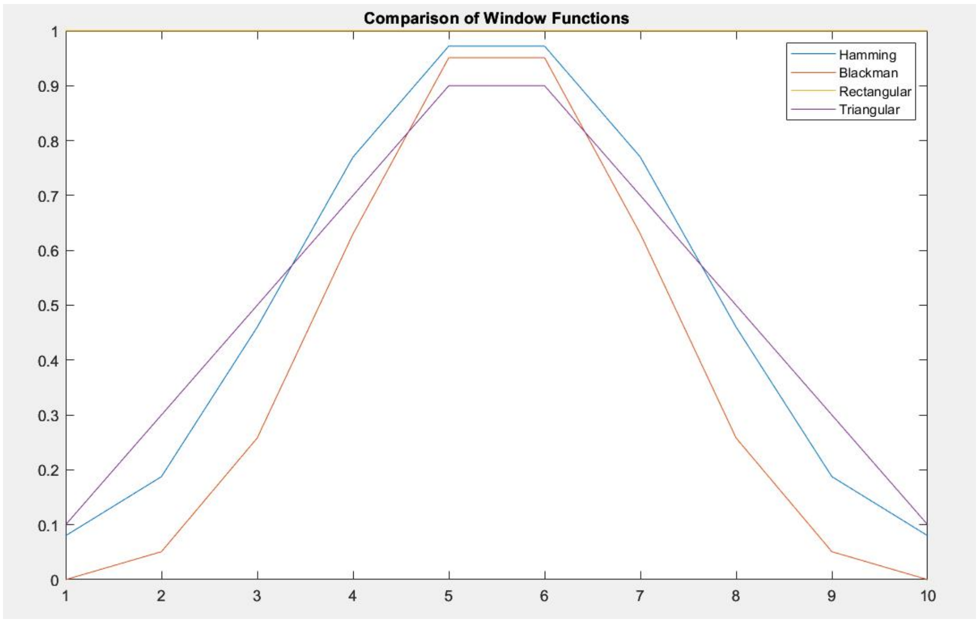


Figure 5: Different window functions L = 10

An example with Hamming window is applied in **Figure 6**, it is visible that Hamming window causes the filter to lose its band-pass characteristics. This is due to the side-lobe amplitude and main lobe width trade-off: windows like Hamming and Blackman reduce the side-lobe amplitudes, which lowers stopband ripples, but at the cost of widening the main lobe, resulting in a slightly poorer frequency resolution. This trade-off arises because tapering the window in time domain spreads the energy more in the frequency domain, leading to a wider main lobe.

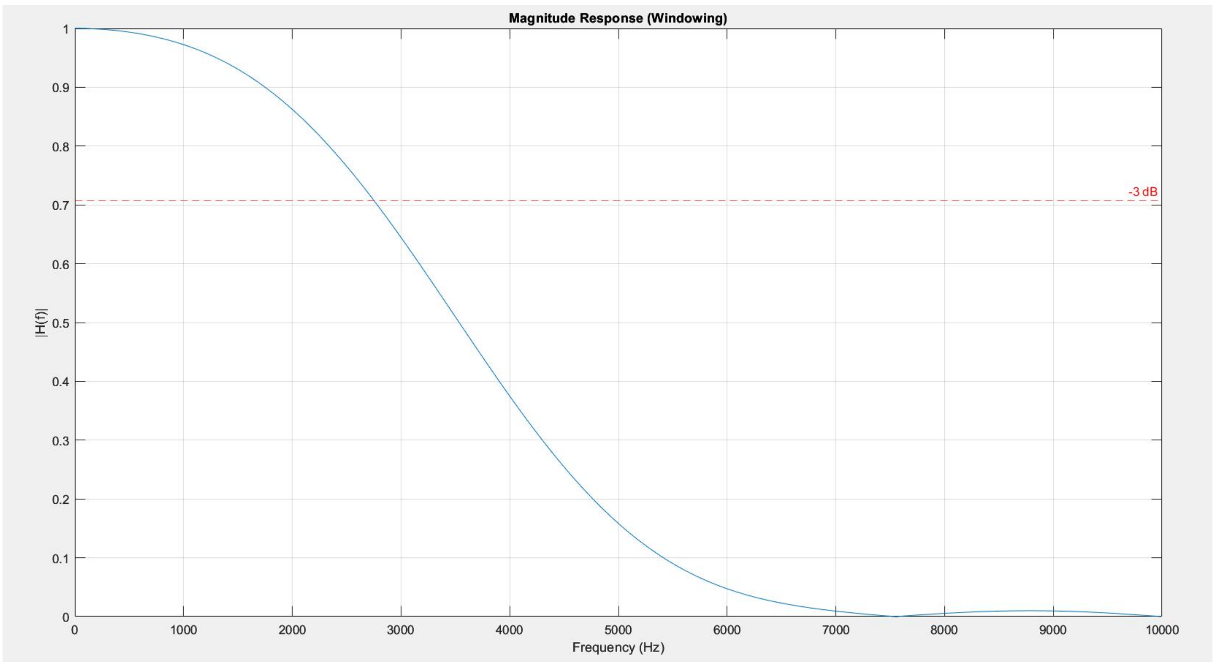


Figure 6: Magnitude Response of BPF with Hamming window L = 10

Another reason is the length of the window being to short. For a longer window, Hamming or Blackman window would have been a better choice, however for a short window like this rectangular is the best choice.

Parks–McClellan algorithm

The Parks–McClellan algorithm is an efficient method for designing optimal finite impulse response (FIR) filters using the minimax (or Chebyshev) criterion. It finds filter coefficients that minimize the maximum error between the desired and actual frequency responses, resulting in an equiripple behavior in the passband and stopband. The algorithm is implemented and impulse response is plotted in **Figure 7**.

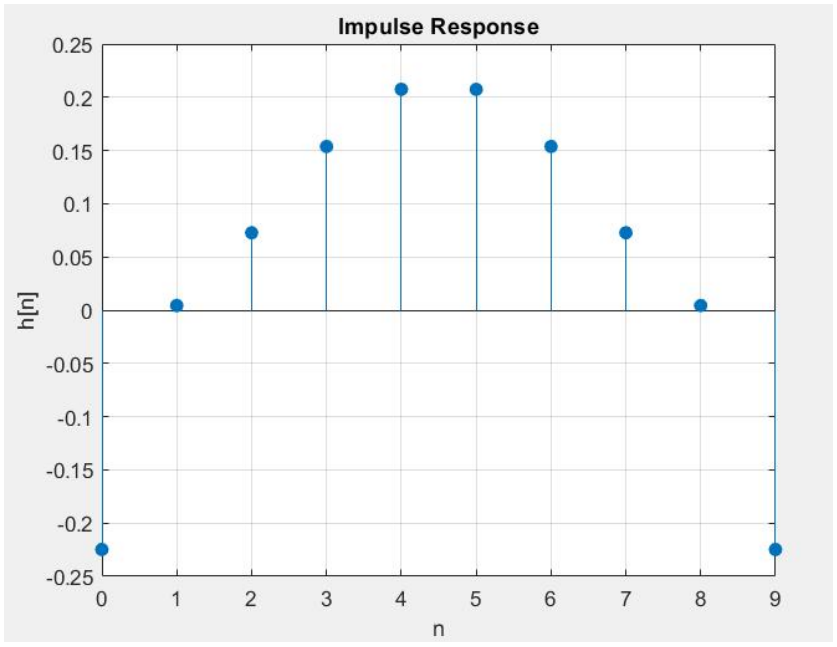


Figure 7: Impulse Response of the FIR BPF

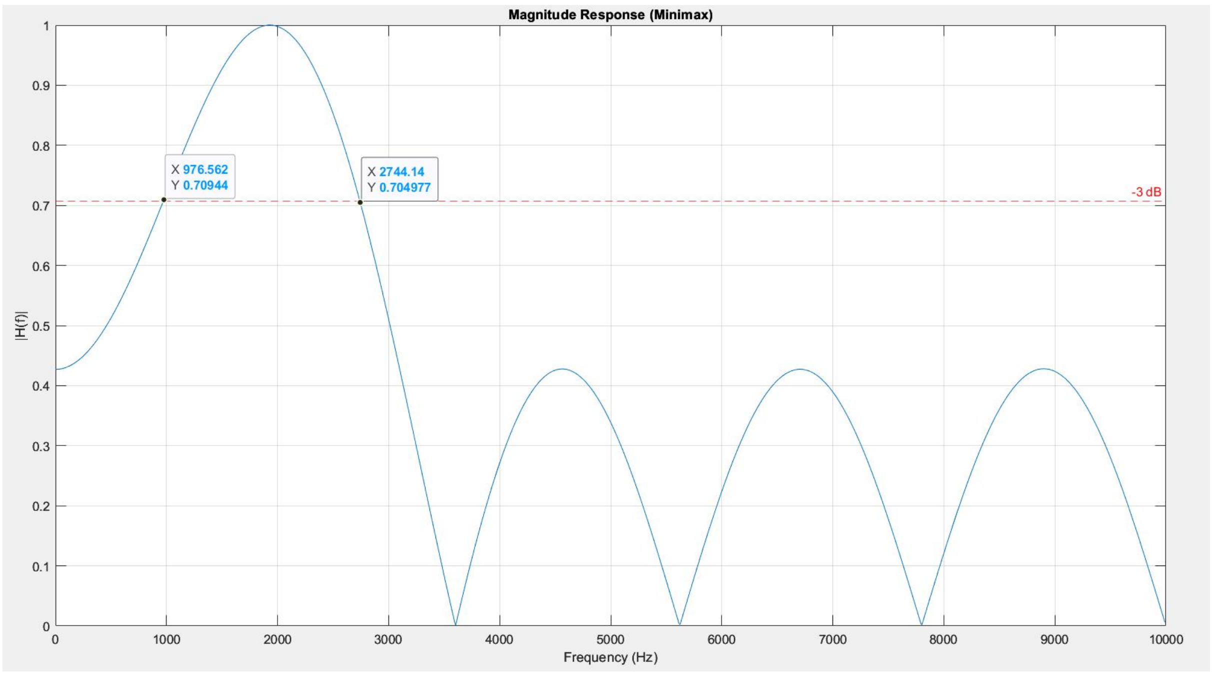


Figure 8: Magnitude Response of the FIR BPF

In **Figure 8** the magnitude response is plotted. The filter exhibits an equiripple behavior, but the cutoff frequencies are not accurately achieved due to the inherent trade-off between ripple magnitude and transition bandwidth.

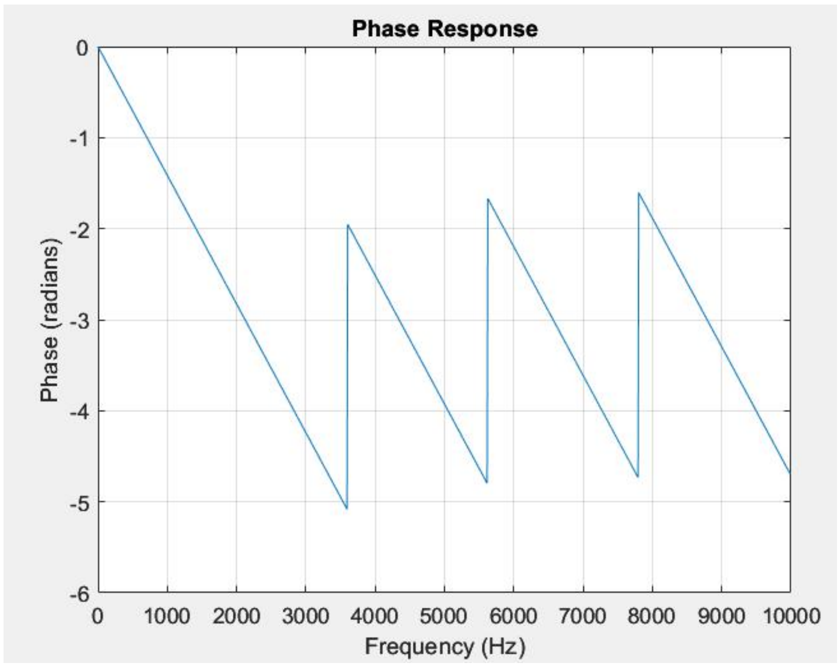


Figure 9: Magnitude Response of the FIR BPF

In **Figure 9** the magnitude response is plotted.Linear phase is visible

Given the constraints of the problem, particularly the limited filter length, it is unlikely that a significantly better design is possible. While the Parks–McClellan algorithm provides optimal equiripple filters in the minimax sense, it struggles to meet strict cutoff specifications with short filter lengths. Similarly, windowing methods (like Hamming or Blackman) reduce ripples but suffer from wider transition bands. In this case, the rectangular window, despite its higher ripples, yields the sharpest transitions, making it the most effective option. Ultimately, the main limitation is the filter length, which restricts how well the desired specifications can be met.

For the next part, FIR filter designed with windowing method is used.

Q2) Resampling and Filtering an Audio Signal

In this part, the task was to record an voice recording with 48 kHz sampling rate. Resample it to 20 kHz and filter the resampled signal with the previously designed FIR filter.

a) Audio Recording

The 10 second audio recording that is sampled at 48 kHz is recorded using MATLAB's built in '`audiorecorder`' function. The following code snippet is used for recording:

```
fs_orig = 48000;
recObj = audiorecorder(fs_orig, 16, 1);
duration = 11;
```

```
disp('Start speaking...');
recordblocking(recObj, duration);
disp('End of Recording.');

audio = getaudiodata(recObj);
audio = audio(fs_orig-1:fs_orig*11);
audiowrite('original_audio.wav', audio, fs_orig);
```

In this code, the sampling rate is manually set to desired value that is 48 kHz therefore, it was not necessary to resample it to 48 kHz. In the ‘`audiorecorder`’ function channel number and sample width are chosen to be mono channel and 16 bit respectively.

An 11 second recording is recorded and trimmed to the length of a 10 second recording with sampling rate 48 kHz that is 480k samples. During the process, it was observed that the first 1 second of the recording was always pure silence which due to a delay that is possibly imposed by a part of an audio recording process (hardware or software). For this reason, it was decided that to trim the first 1 second of the recording.

The recorded audio is saved in wav format to be able to loaded for the subsequent sections.

b) **Resampling**

It is not possible to directly downsample the audio signal 48 kHz to 20 kHz since the ratio 48k/20k = 12/5 is not an integer. Therefore, the idea is to first upsampling the signal to least common multiple (LCM) than downsampling to 20 kHz. That corresponds to upsampling the 48 kHz signal by 5 and then downsampling the resulting signal by 12.

$$\frac{48k \cdot 5}{12} = \frac{240k}{12} = 20k$$

During downsampling aliasing occurs because when a signal is sampled at a lower rate, frequency components above half the new sampling rate can fold back into the lower frequency spectrum. This results in distortion, as these higher-frequency components seen as lower-frequency ones, corrupting the original signal.

Therefore, during downsampling from 48 kHz to 20 kHz, the Nyquist frequency drops from 24 kHz to 10 kHz. Any frequency content between 10–24 kHz in the original signal will alias back into 0–10 kHz in the downsampled version, causing unwanted interference in the audible range.

To prevent aliasing, an anti-aliasing filter is applied before downsampling. This is a low-pass filter designed to eliminate frequency components above the new Nyquist limit 10 kHz.

An FIR filter is designed using a normalized cutoff frequency of 1/q = 1/12 ≈ 0.083, where q=12 is the downsampling factor. This normalized cutoff corresponds to the maximum frequency that should be preserved to avoid aliasing when the signal is later downsampled.

FIR filter:

```
filt = fir1(100, 1/q)
```

Since the original audio signal is upsampled by a factor of 5 (p = 5) before filtering, the intermediate signal has a higher sampling rate 100kHz, therefore the filter cut-off is determined proportionally.

The result is a clean signal, limited to the desired bandwidth, ready for safe and accurate resampling to 20 kHz.

c) **Filtering**

Resampled signal is filtered with the BPF as:

$$S_{filtered}[n] = h_{BPF}[n] * S[n]$$

When the recordings were listened, there was not much audible difference between recordings meaning that human voice was captured successfully, however FFT is applied for each signal for comparison.

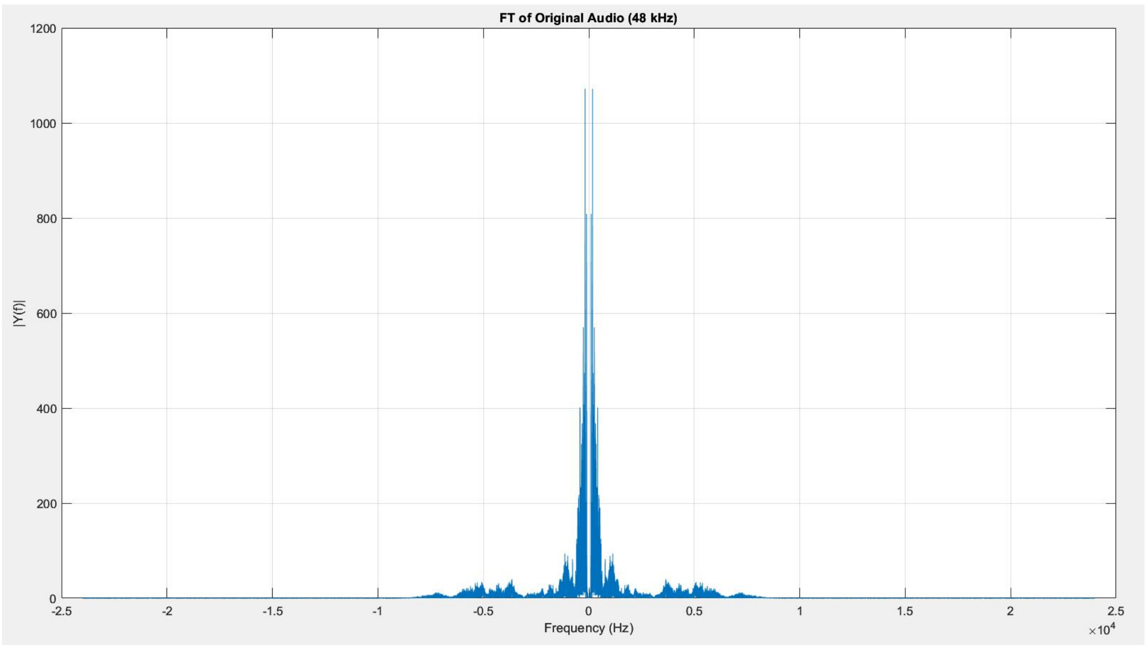


Figure 10: Frequency Domain Analysis of the Original Audio

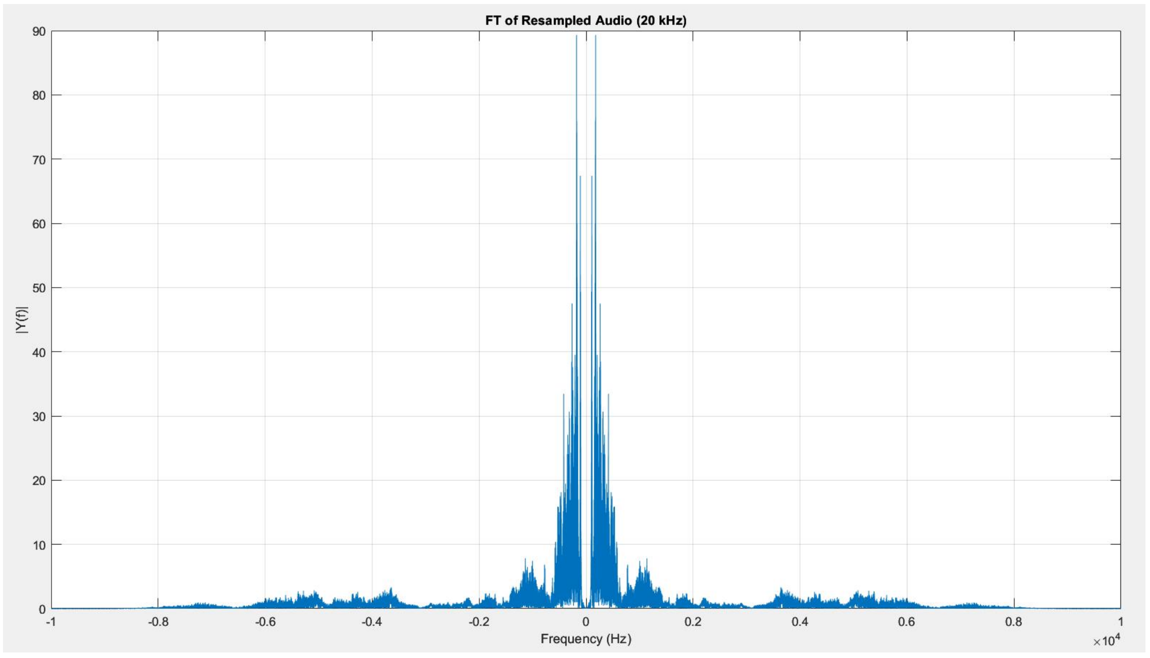


Figure 11: Frequency Domain Analysis of the Resampled Audio

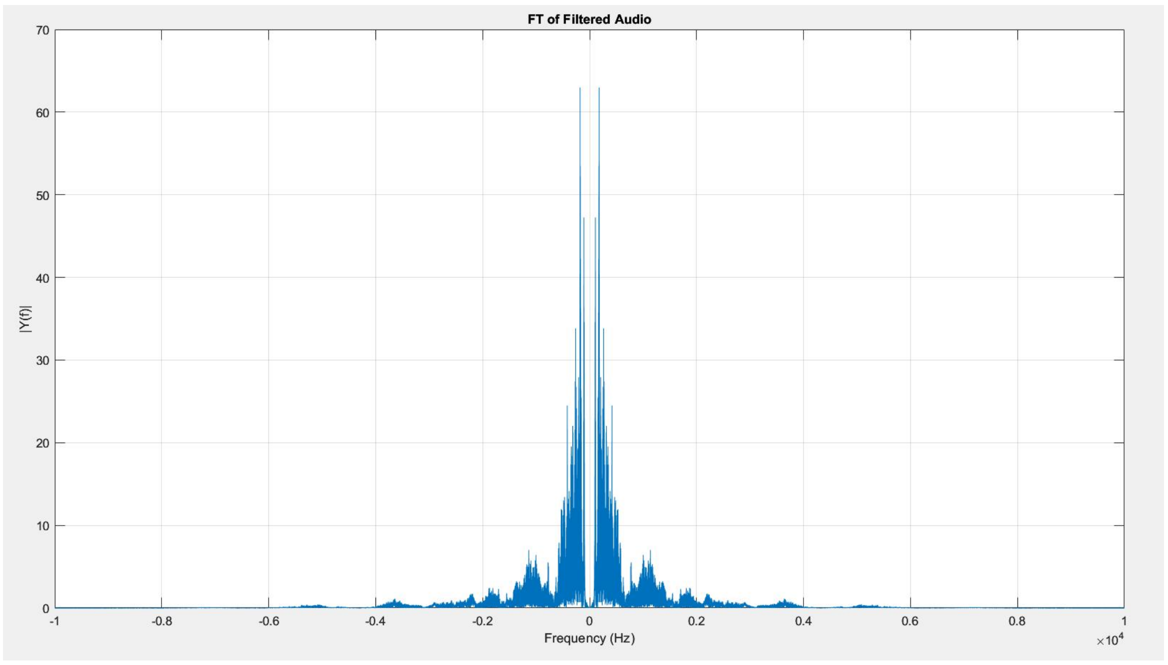


Figure 12: Frequency Domain Analysis of the Filtered Audio

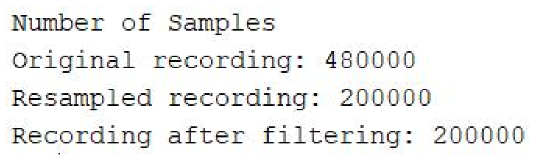
Figures 10, 11, and 12 show that in the frequency domain, the original audio (sampled at 48 kHz) has a much broader frequency spectrum, extending up to ±24 kHz, as seen in the first plot. After resampling to 20 kHz, the second plot confirms the expected spectral compression, with the signal now limited to ±10 kHz. The third plot, corresponding to the filtered audio, demonstrates that the filter effectively passes only the desired human voice frequency range (300 Hz to 3 kHz), significantly attenuating components outside this band.

These results are coherent with the findings from the previous question. Although no audible difference was noted during subjective listening, the frequency analysis through FFT validates that the resampling and filtering steps preserved the integrity of the voice signal while

discarding unnecessary frequencies. This confirms that the voice was captured correctly and the filter successfully met its design goals under the constraints provided.
Size of each file is calculated as follows:

$$\# \, of \, samples = T_d f_s$$

T_d : duration of the recording (10s)
f_s : sampling rate



```
Number of Samples
Original recording: 480000
Resampled recording: 200000
Recording after filtering: 200000
```

Appendicies

Appendix A - Q1 whole code

1. Windowing code

```
clear; close all; clc;
%%
fs = 20000;
L = 10;
n = -(L-1)/2 : (L-1)/2;
n1 = 0:L-1;

f_low = 275;
f_high = 3250;

w_low = 2*pi*f_low/fs;
w_high = 2*pi*f_high/fs;

h_ideal = (sin(w_high*n) - sin(w_low*n)) ./ (pi*n);

hamming_window = hamming(L);
blackman_window = blackman(L);
rect = ones(L,1);
triangle = triang(L);

figure
plot([hamming_window, blackman_window, rect, triangle])
legend('Hamming', 'Blackman', 'Rectangular', 'Triangular')
title('Comparison of Window Functions')
h = h_ideal .* rect';

figure;
stem(n1, h, 'filled');
title('Impulse Response');
xlabel('n'); ylabel('h[n]');
grid on;

% Frequency Response
[H, f] = freqz(h, 1, 1024, fs);

% Magnitude Response
figure;
plot(f, abs(H)/max(abs(H)));
title('Magnitude Response (Windowing)');
xlabel('Frequency (Hz)');
ylabel('|H(f)|');
grid on;
xlim([0 fs/2]);
yline(0.707, 'r--', '-3 dB');
hold off

% Phase Response
figure;
plot(f, unwrap(angle(H)));
title('Phase Response');
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
```

```
grid on;

% Pole-Zero Plot
figure;
zplane(roots(h),[]);
title('Pole-Zero Plot');

% Z-transform
syms z
H_z = 0;
for k = 1:L
H_z = H_z + h(k) * z^(-(k-1));
end
```

2. Parks–McClellan algorithm

```
clear; close all; clc;
%%
% Parameters
fs = 20000;
L = 10;
f_low =300;
f_high = 3000;
f = [0 0 f_low-100 f_high f_high+100 fs/2]/ (fs/2);
a = [0 0 1 1 0 0];

h = firpm(L-1, f, a);
[H_temp, ~] = freqz(h, 1, 1024);
h = h / max(abs(H_temp));
[H, freq] = freqz(h, 1, 1024, fs);

% Impulse Response
figure;
stem(0:L-1, h, 'filled');
xlabel('n'); ylabel('h[n]');
title('Impulse Response');
grid on;

% Magnitude Response
figure;
plot(freq, abs(H));
xlabel('Frequency (Hz)');
ylabel('|H(f)|');
title('Magnitude Response (Minimax)');
grid on; xlim([0 fs/2]);
hold on;
yline(0.707, 'r--', '-3 dB');
hold off

% Phase Response
figure;
plot(freq, unwrap(angle(H)));
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
title('Phase Response');
grid on;

% Pole-Zero Plot
```

```
figure;
zplane(roots(h),[]);
title('Pole-Zero Plot');

% Z-Transform
syms z
H_z = 0;
for k = 1:length(h)
H_z = H_z + h(k) * z^(-(k-1));
end
```

Appendix B - Q2 whole code

```
clear; close all; clc;
%% Downsampling

[audio, fs_orig] = audioread('original_audio.wav');

fs_target = 20000;
[p, q] = rat(fs_target/fs_orig); % 5/12
fprintf("Upsample: %d, Downsample: %d\n", p, q);

% Up-sample
audio_up = upsample(audio, p);

% Anti-aliasing filter before downsampling
filt = fir1(100, 1/q);
audio_filt = conv(audio_up, filt, 'same');

% Down-sample
audio_20k = downsample(audio_filt, q);
audiowrite('audio_20kHz.wav', audio_20k, fs_target);

%% FIR Bandpass Filter Windowing
L = 10;
n = -(L-1)/2 : (L-1)/2;
n1 = 0:L-1;

f_low = 275;
f_high = 3250;

w_low = 2*pi*f_low/fs_target;
w_high = 2*pi*f_high/fs_target;

h_ideal = (sin(w_high*n) - sin(w_low*n)) ./ (pi*n);
h = h_ideal;

%% Filtering
audio_filtered = conv(audio_20k, h, 'same');
audiowrite('filtered_audio_20kHz.wav', audio_filtered, fs_target);

%% Number of Samples
fprintf("Number of Samples\n");
fprintf('Original recording: %d\n', length(audio));
fprintf('Resampled recording: %d\n', length(audio_20k));
fprintf('Recording after filtering: %d\n', length(audio_filtered));

%% Frequency Domain Analysis
Fs1 = 48000;
n1 = length(audio);
f1 = linspace(-Fs1/2, Fs1/2, n1);
Y1 = fftshift(abs(fft(audio)));

figure;
plot(f1, Y1);
title('FT of Original Audio (48 kHz)');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
grid on;
```

```
Fs2 = 20000;
n2 = length(audio_20k);
f2 = linspace(-Fs2/2, Fs2/2, n2);
Y2 = fftshift(abs(fft(audio_20k)));

figure;
plot(f2, Y2);
title('FT of Resampled Audio (20 kHz)');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
grid on;

Fs3 = 20000;
n3 = length(audio_filtered);
f3 = linspace(-Fs3/2, Fs3/2, n3);
Y3 = fftshift(abs(fft(audio_filtered)));

figure;
plot(f3, Y3);
title('FT of Filtered Audio');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
grid on;
```

Appendix C - Audio Recording Code

```
clear; close all; clc;
%% Audio Recording
fs_orig = 48000;
recObj = audiorecorder(fs_orig, 16, 1);
duration = 11;

disp('Start speaking...');
recordblocking(recObj, duration);
disp('End of Recording.');

audio = getaudiodata(recObj);
audio = audio(fs_orig-1:fs_orig*11);
audiowrite('original_audio.wav', audio, fs_orig);
```