Mini Project

Time and Frequency Domain Analysis of Your Recorded Voice Signal

Objective:

- **Capture one word in your voice**
- **Capture 10-15 second of your voice.**

Time Domain Analysis:

- **Objective 1: Visual Analysis of phonetics**
- **Objective 2: Effect of LPF filtering in Time Domain**
- **Objective 3: Effect of HPF filtering in Time Domain**
- **Objective 4: Segregate Voice and Un-voice part**

Frequency Domain Analysis:

- **Objective 1: Visual Analysis about the frequency Details**
- **Objective 2: Effect of LPF filtering in Frequency Domain**
- **Objective 3: Effect of HPF filtering in Frequency Domain**
- **Objective 4: Segregate Voice and Un-voice part**

Theory:

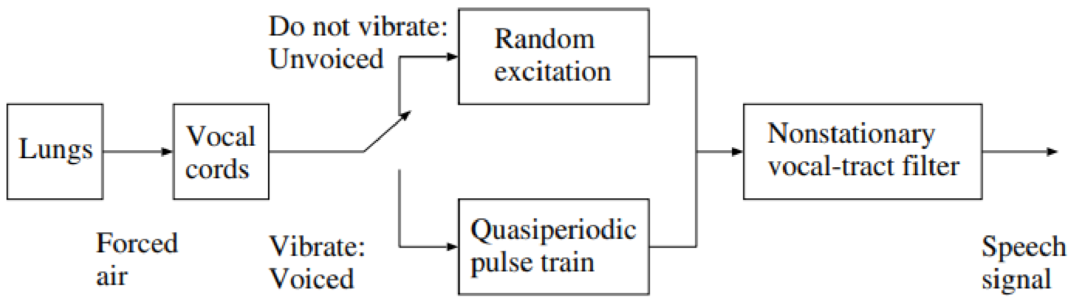


Figure 1.49 Schematic representation of the production of voiced and unvoiced speech.

quasiperiodic pulses of air which is passed through the vocal tract; see Figure 1.49. The input to the vocal tract may be treated as a train of impulses or pulses that is almost periodic. The vocal tract acts as a filter: Upon convolution with the impulse response of the vocal tract, which is held steady in a certain configuration for the duration of the voiced sound desired, a quasiperiodic signal is produced with a characteristic waveshape that is repeated.

A voice signal is an analog signal produced by human vocal cords. To process it using digital systems like computers, we need to sample it. **Sampling involves discretizing the continuous voice signal at regular intervals in time**. The rate at which this is done is known as the sampling frequency (Fs).

The *audiorecorder* function in MATLAB allows you to record audio using a microphone connected to your computer. It provides a simple interface to capture audio data for processing within MATLAB.

Procedure:

1. **Capture 10-15 sec of your voice signal.**
2. **Convert the recorded signal to .wav type if applicable, and save the recorded file in a folder.**
3. **In order to visually analyse the phonetics of your voice signal, use "audioread" command in MATLAB; for example, [y]=audioread('airtel_flute.wav');**
4. **Analyse the time domain representation, and frequency domain representation of audio vector "y" using DTFT.**
5. **Study the effect of "low-pass", and "high-pass" filtering of the audioread file using MATLAB.**
6. **Explore the phonetic variation by using time, and frequency domain representation.**

1. Capture 10-15 sec of your voice signal and Analyse the time domain representation, and frequency domain representation of audio vector "y" using DTFT.

In the time domain, a signal is represented as a sequence of samples, each corresponding to a specific point in time using audioread function. Audioread is a MATLAB command used to read audio files. In this step, we use audioread to load the recorded .wav file into MATLAB for further processing. This representation provides information about the amplitude and variation of the signal over time The Discrete Time Fourier Transform (DTFT) is a mathematical tool used to represent discrete signals in terms of their frequency content. It provides information about the spectral composition of the signal, showing which frequencies are present and their respective magnitudes and phases. FFT is an algorithm for efficiently computing the Discrete Fourier Transform (DFT), which is a sampled version of the DTFT. It is much faster than direct computation of the DTFT.

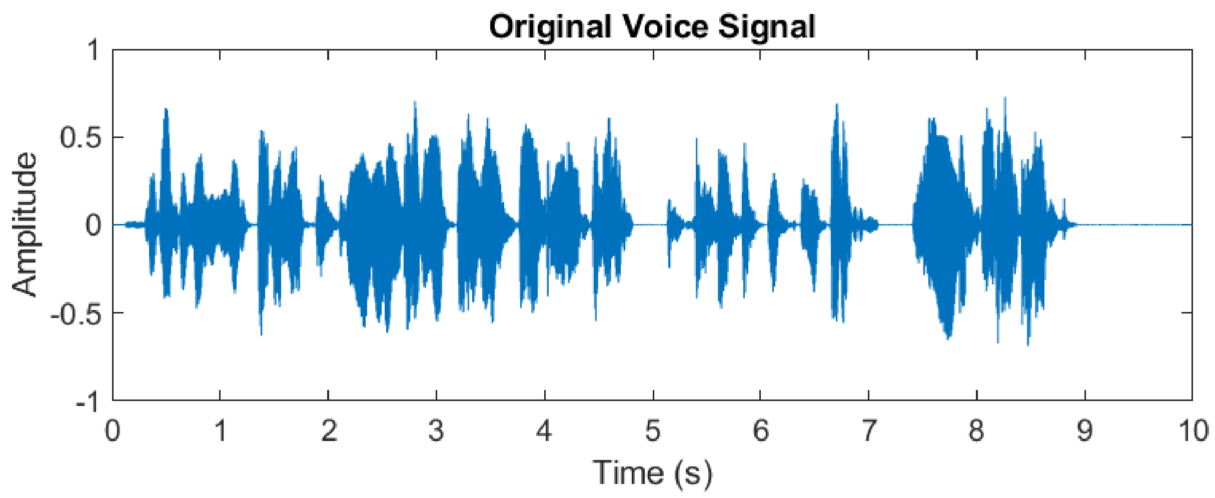
<u>MATLAB Code:</u>

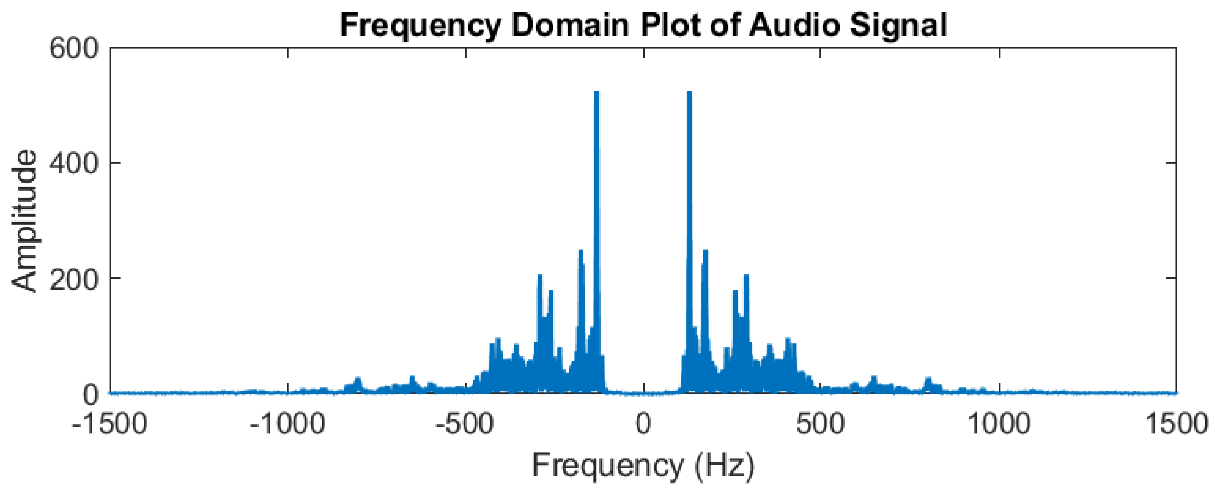
```
Fs = 3000; Channels = 1; bits = 16;
r = audiorecorder(Fs,bits,Channels);
duration = 10 ; disp('Recording Started');
recordblocking(r, duration);
disp('Recording stoppped');
X = getaudiodata(r);
% or either we can use [X, Fs]=audioread('myvoice_10_second.wav');
% sound(X,Fs,bits); to listen to sound you have entered
t = 0:1/Fs:(length(X)-1)/Fs;
subplot(2,1,1); plot(t,X,"Linewidth",1.5);
xlabel('time (sec)'); ylabel('Amplitude');
title('Time Domain plot of the Recorded signal');
n = length(X); F = 0:(n-1)*Fs/n;
Y = fft(X,n);
F_0 = (-n/2:n/2-1).*(Fs/n);
Y_0 = fftshift(Y);
AY_0 = abs(Y_0);
subplot(2,1,2);
plot(F_0,AY_0,"Linewidth",1.5);
xlabel('Frequency (Hz)');  ylabel("Amplitude");
title('Frequency Domain Plot of Audio Signal');
```

PART II. the record file in a folder *USING AUDIOWRITE*:

```
filename = 'myvoice_10_second.wav';
audiowrite(filename,X,Fs);
```

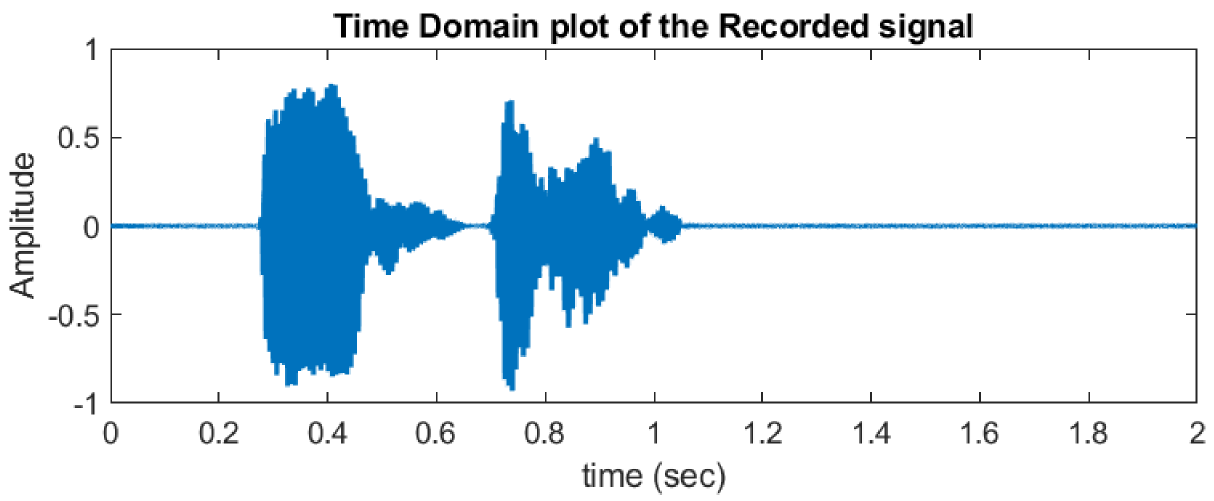
Output graphs:

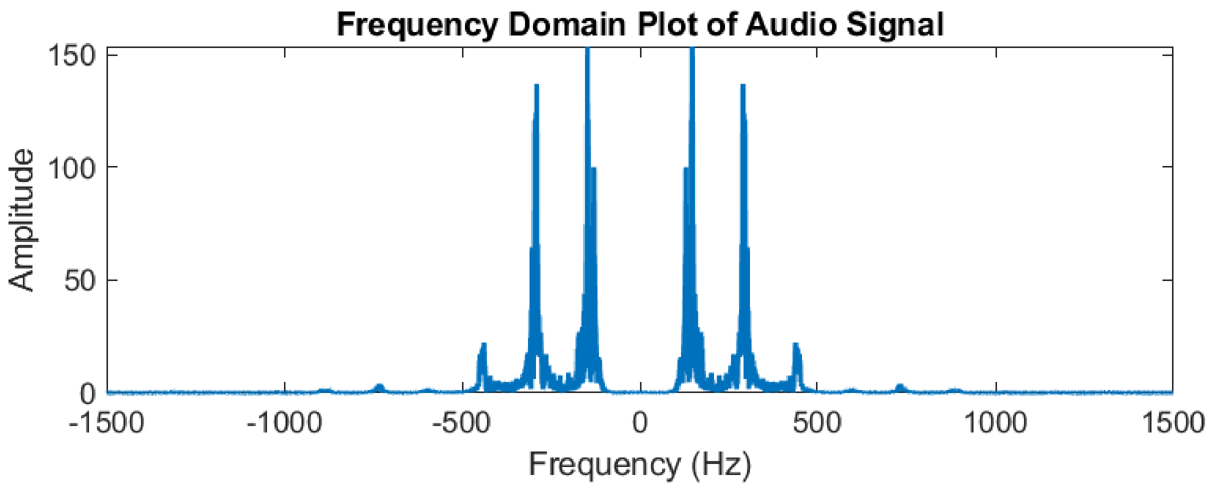




<u>OUTPUT GRAPHS</u>:

For one word signal:





DISCUSSION: we imported and played back the recorded audio using the audioread function, verifying the fidelity of the digital recording. Subsequently, we embarked on the analysis phase, starting with the time domain.

FREQUENCY DOMAIN: Moving into the frequency domain, we performed a Discrete-Time Fourier Transform (DTFT) to examine the frequency components. This mathematical representation unveiled the spectral composition of the voice signal, revealing dominant frequencies and their respective magnitudes.

CONCLUSION: We visualized the waveform, obtaining insights into the signal's temporal characteristics. This allowed us to assess properties like signal duration and amplitude range and frequency variaiton.

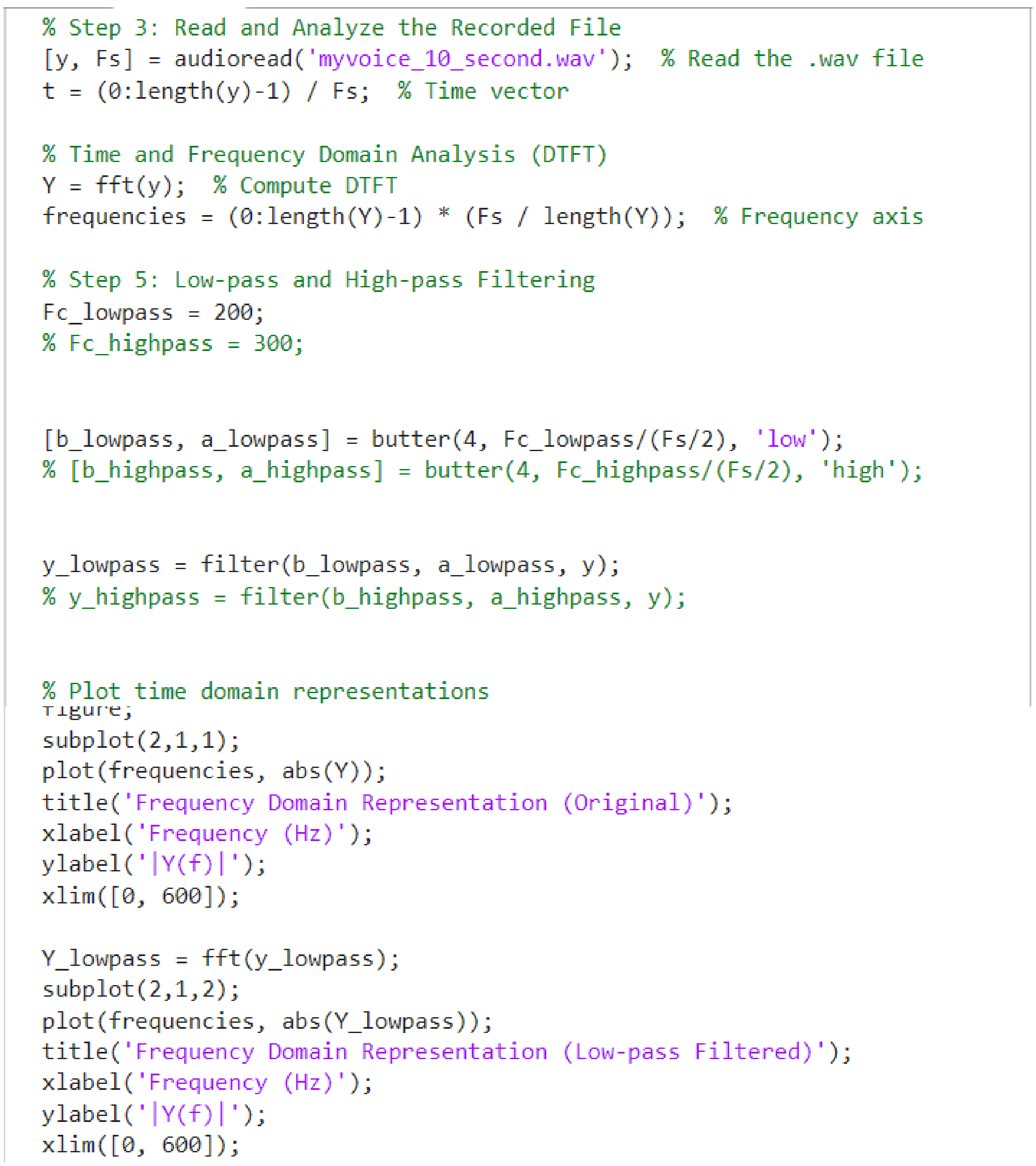
II Study the effect of "low-pass", and "high-pass" filtering of the audioread file using MATLAB:

Filtering techniques are used to modify the frequency content of a signal. Low-pass filters allow low-frequency components to pass through, while high-pass filters allow high-frequency components to pass through. This can be useful for isolating specific frequency ranges of interest or removing unwanted noise.

The Butterworth filter is characterized by its maximally flat frequency response in the passband. This means that it allows for a smooth transition from the passband (where the filter allows certain frequencies to pass) to the stopband (where it attenuates other frequencies). Cutoff Frequency (fc): This is the frequency at which the filter starts to attenuate the signal. It's the point where the filter's gain drops to -3 dB (half of the passband power). *Order (N): The order of the filter determines the roll-off rate.*

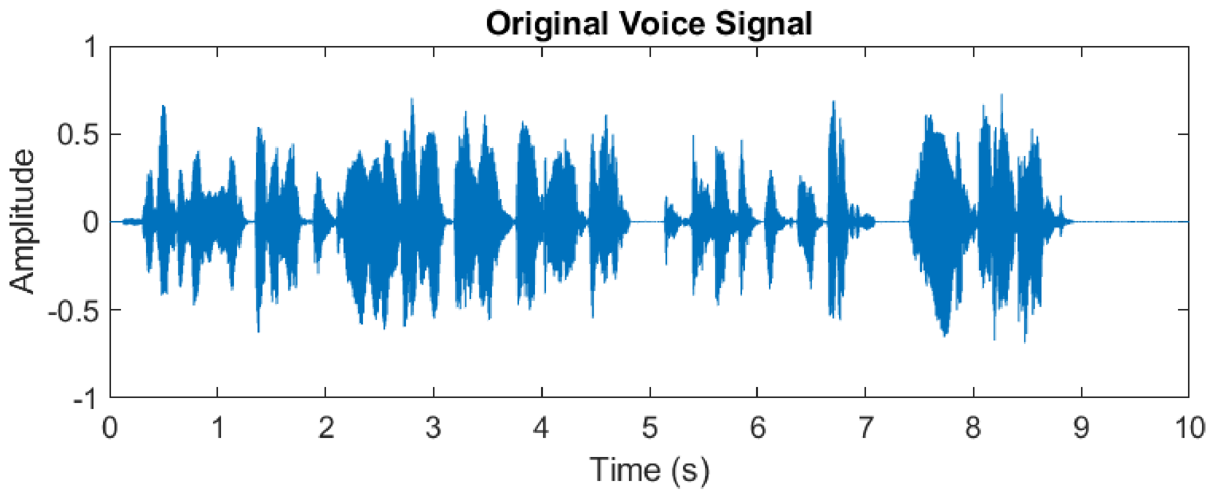
Higher-order filters have steeper roll-off characteristics.

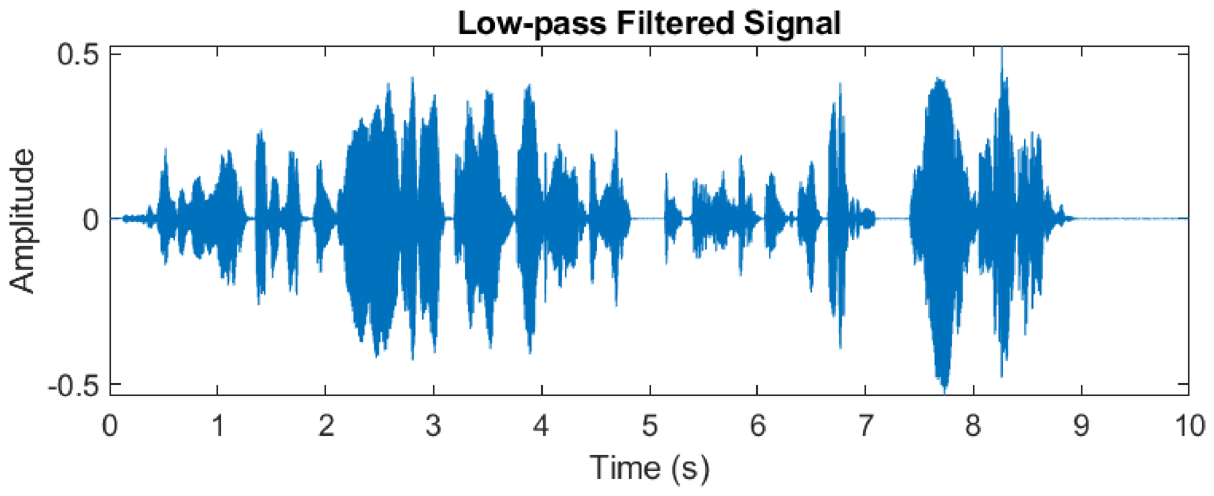
(a) LOW PASS :

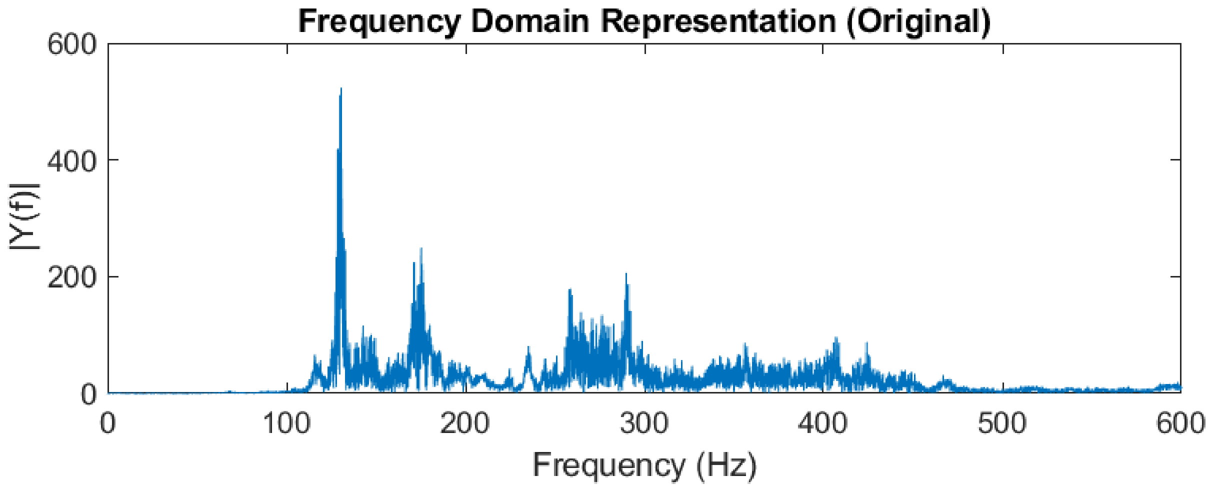


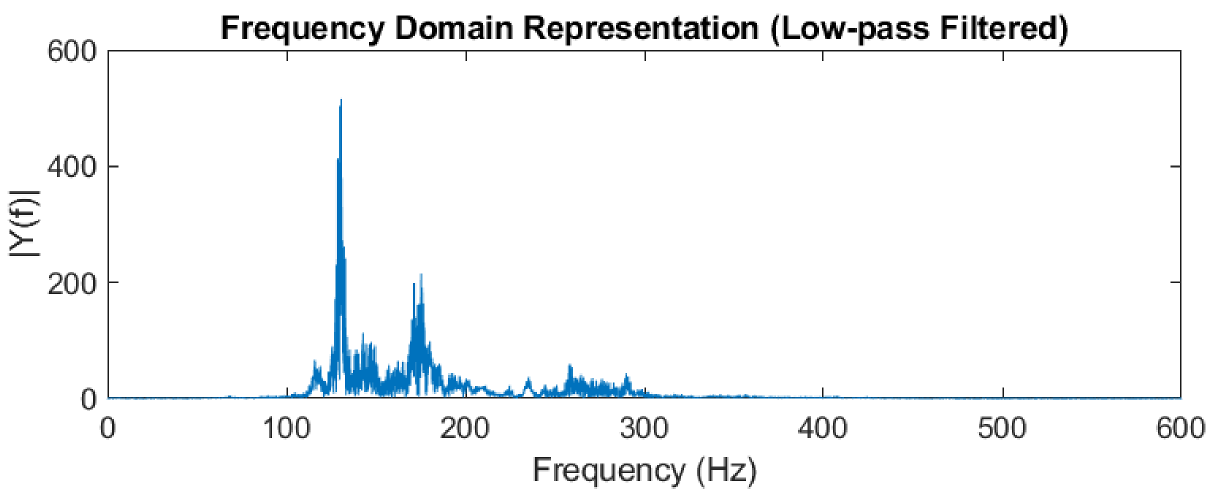
```
% Step 3: Read and Analyze the Recorded File
[y, Fs] = audioread('myvoice_10_second.wav');  % Read the .wav file
t = (0:length(y)-1) / Fs;  % Time vector

% Time and Frequency Domain Analysis (DTFT)
Y = fft(y);  % Compute DTFT
frequencies = (0:length(Y)-1) * (Fs / length(Y));  % Frequency axis

% Step 5: Low-pass and High-pass Filtering
Fc_lowpass = 200;
% Fc_highpass = 300;


[b_lowpass, a_lowpass] = butter(4, Fc_lowpass/(Fs/2), 'low');
% [b_highpass, a_highpass] = butter(4, Fc_highpass/(Fs/2), 'high');


y_lowpass = filter(b_lowpass, a_lowpass, y);
% y_highpass = filter(b_highpass, a_highpass, y);


% Plot time domain representations
figure;
subplot(2,1,1);
plot(frequencies, abs(Y));
title('Frequency Domain Representation (Original)');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
xlim([0, 600]);

Y_lowpass = fft(y_lowpass);
subplot(2,1,2);
plot(frequencies, abs(Y_lowpass));
title('Frequency Domain Representation (Low-pass Filtered)');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
xlim([0, 600]);
```

OUTPUT GRAPHS:

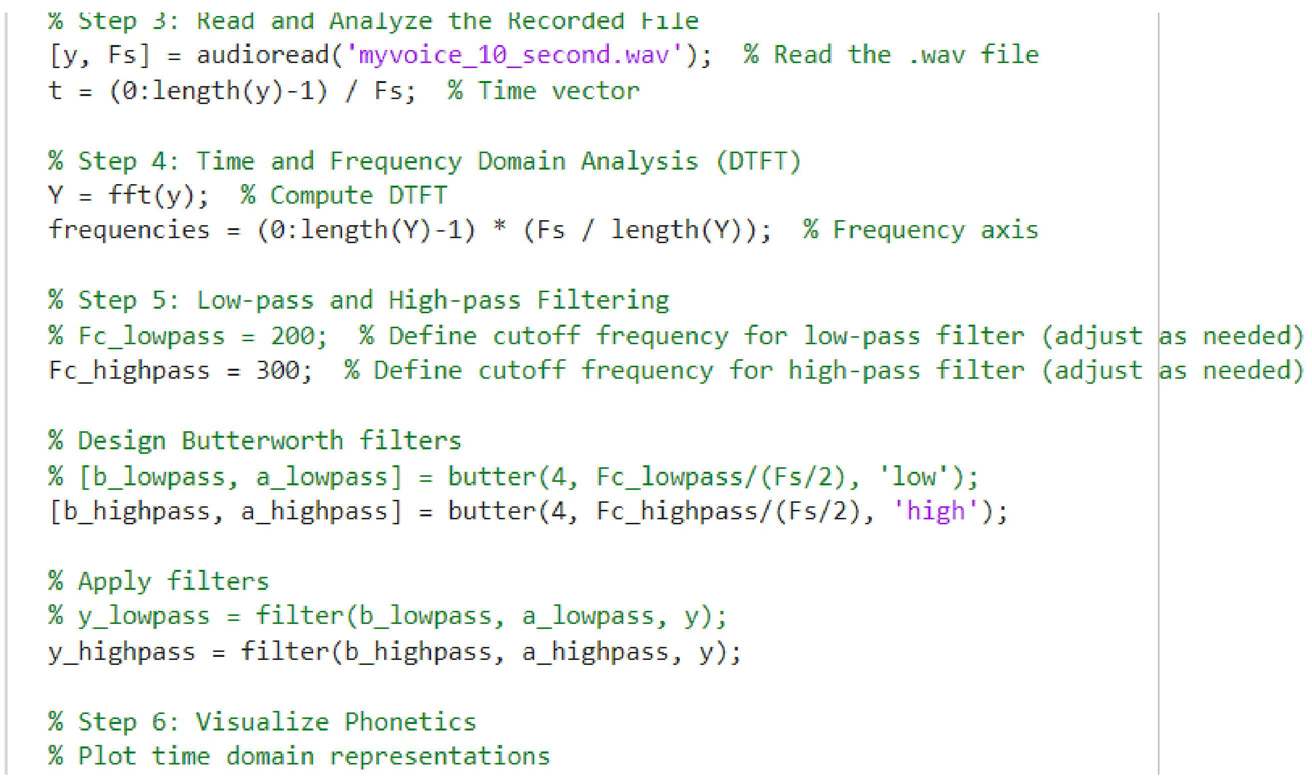

Original Voice Signal
Amplitude
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
8
9
10
Time (s)


Low-pass Filtered Signal
Amplitude
0.5
0
-0.5
0
1
2
3
4
5
6
7
8
9
10
Time (s)

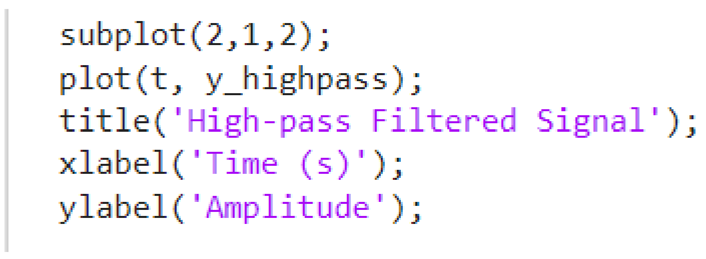

Frequency Domain Representation (Original)
|Y(f)|
600
400
200
0
0
100
200
300
400
500
600
Frequency (Hz)



(b) **HIGH PASS:**



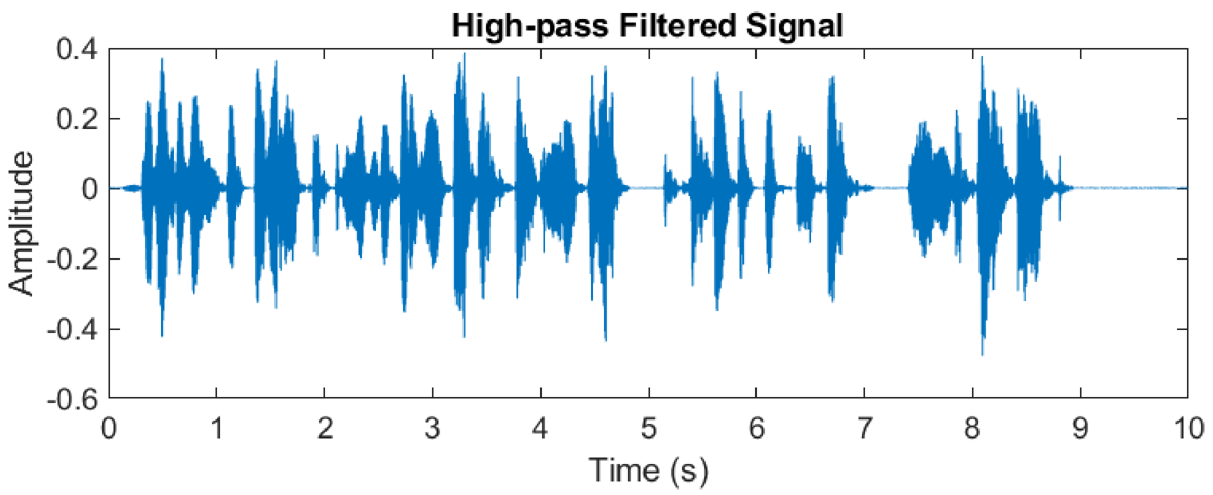
```
% Step 3: Read and Analyze the Recorded File
[y, Fs] = audioread('myvoice_10_second.wav');  % Read the .wav file
t = (0:length(y)-1) / Fs;  % Time vector

% Step 4: Time and Frequency Domain Analysis (DTFT)
Y = fft(y);  % Compute DTFT
frequencies = (0:length(Y)-1) * (Fs / length(Y));  % Frequency axis

% Step 5: Low-pass and High-pass Filtering
% Fc_lowpass = 200;  % Define cutoff frequency for low-pass filter (adjust as needed)
Fc_highpass = 300;  % Define cutoff frequency for high-pass filter (adjust as needed)

% Design Butterworth filters
% [b_lowpass, a_lowpass] = butter(4, Fc_lowpass/(Fs/2), 'low');
[b_highpass, a_highpass] = butter(4, Fc_highpass/(Fs/2), 'high');

% Apply filters
% y_lowpass = filter(b_lowpass, a_lowpass, y);
y_highpass = filter(b_highpass, a_highpass, y);

% Step 6: Visualize Phonetics
% Plot time domain representation
```

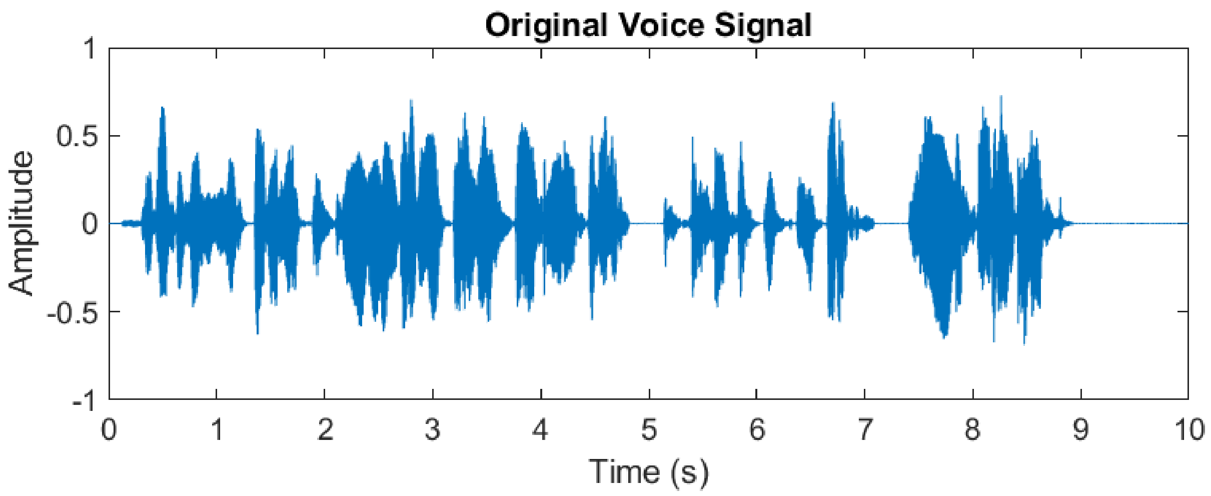


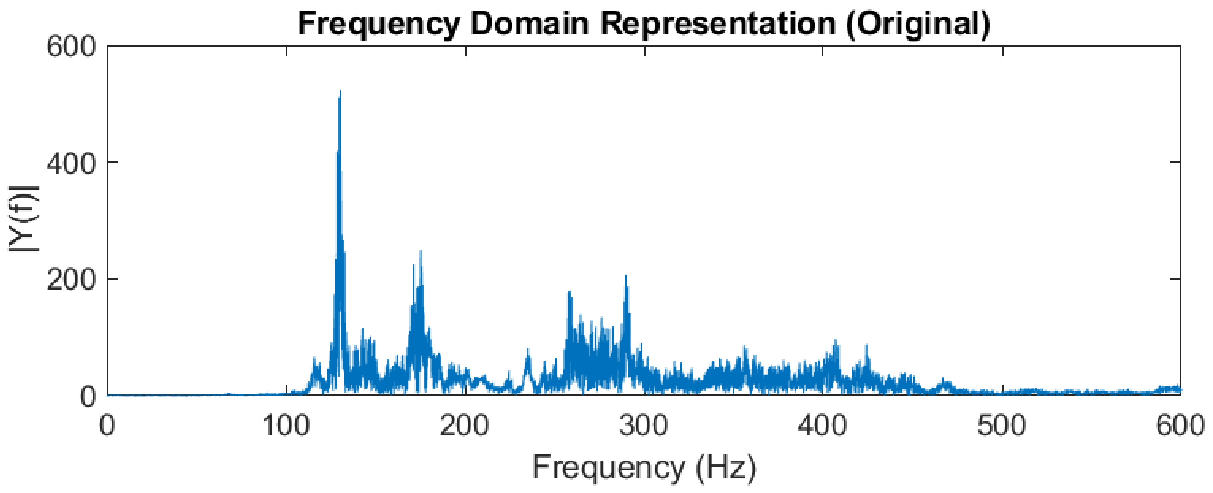
```
subplot(2,1,2);
plot(t, y_highpass);
title('High-pass Filtered Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```

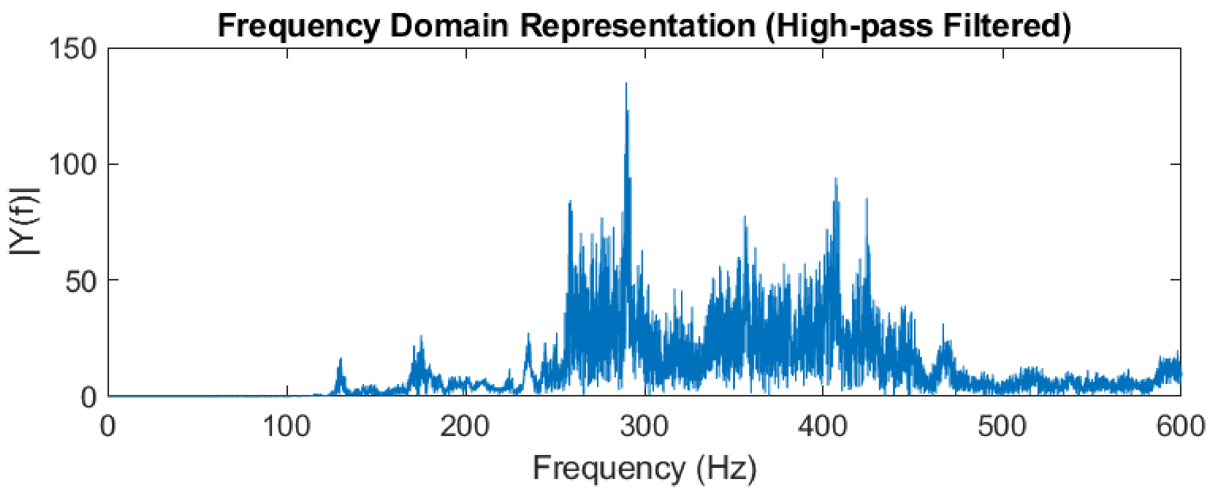
```
Y_highpass = fft(y_highpass);
subplot(2,1,2);
plot(frequencies, abs(Y_highpass));
title('Frequency Domain Representation (High-pass Filtered)');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
xlim([0, 600]);
```

OUTPUT GRAPHS:







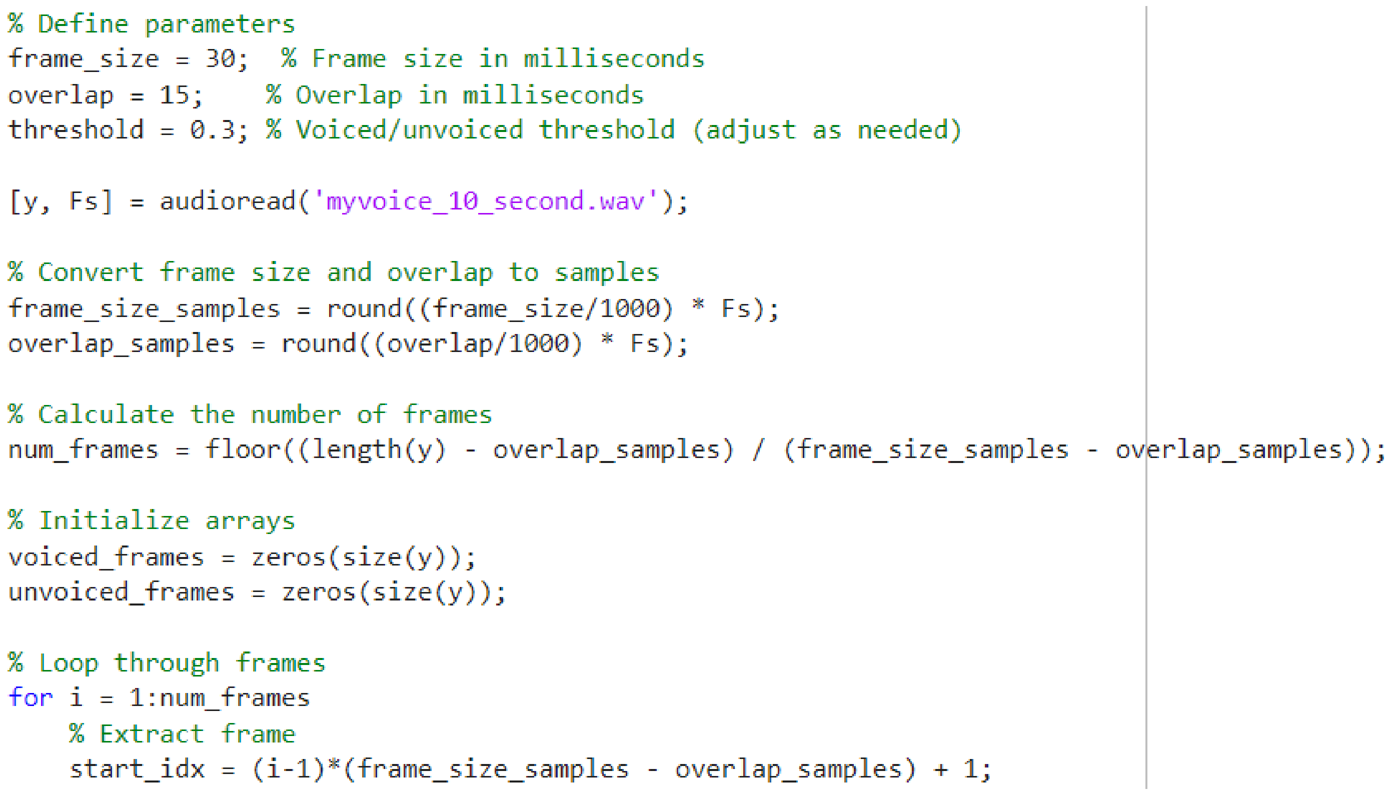


4. Segregate Voice and Un-voice part for time domain:

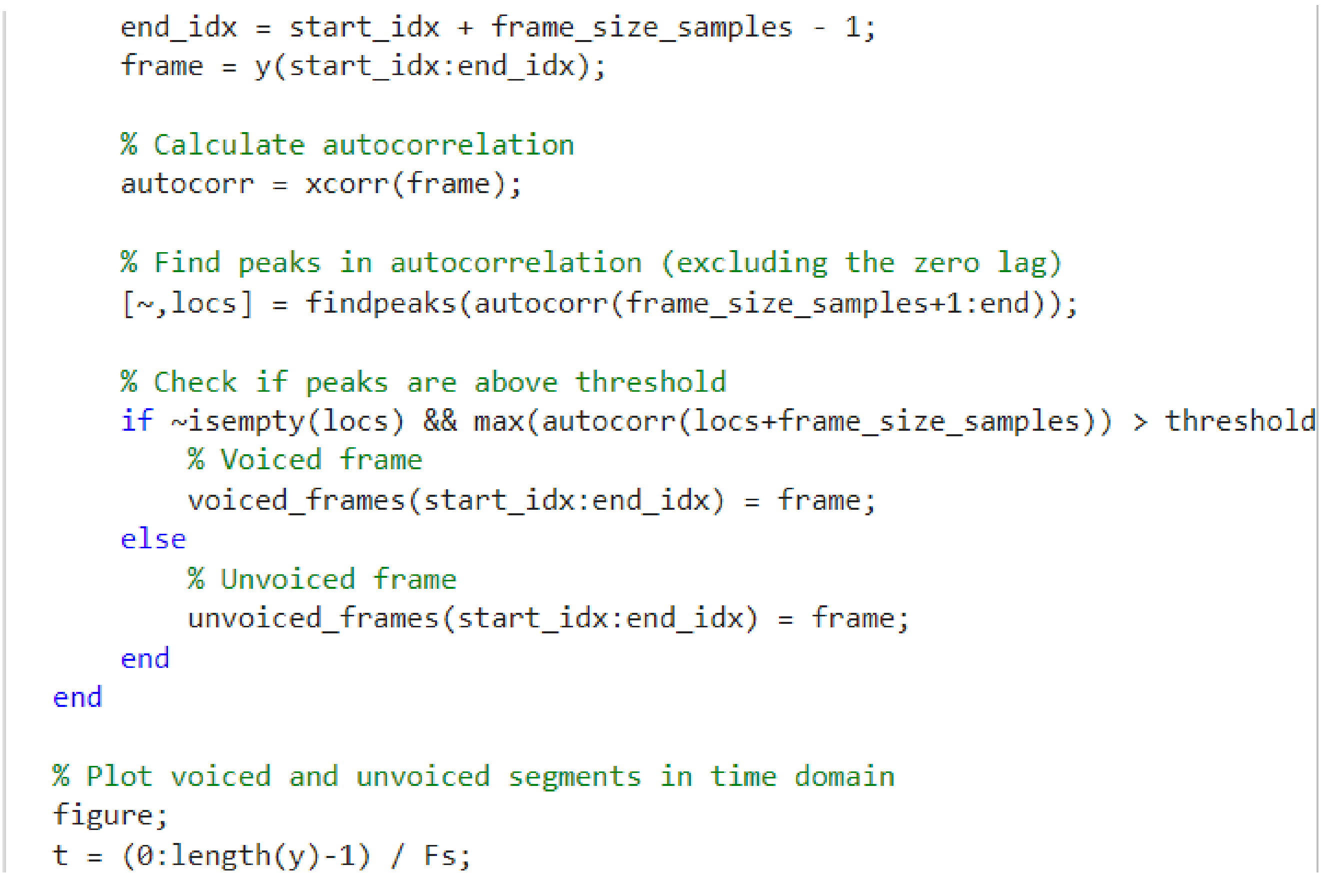
THEORY:

One way to do segregation of voiced and unvoiced signal is to use autocorrelation to estimate the pitch of the speech signal and use it as a criterion to separate voiced and unvoiced parts. Voiced speech segments have a periodic structure and a well-defined pitch, while *unvoiced speech segments have a random structure and no pitch*.

Pitch Detection Algorithm: A pitch detection algorithm analyzes the waveform of the audio signal to estimate the fundamental frequency (F0) at each time frame. This is often done using techniques like **autocorrelation**, *cepstral analysis*, or methods based on the Fast Fourier Transform (FFT). The fundamental frequency represents the perceived pitch of the sound.



```
% Define parameters
frame_size = 30;  % Frame size in milliseconds
overlap = 15;     % Overlap in milliseconds
threshold = 0.3; % Voiced/unvoiced threshold (adjust as needed)

[y, Fs] = audioread('myvoice_10_second.wav');

% Convert frame size and overlap to samples
frame_size_samples = round((frame_size/1000) * Fs);
overlap_samples = round((overlap/1000) * Fs);

% Calculate the number of frames
num_frames = floor((length(y) - overlap_samples) / (frame_size_samples - overlap_samples));

% Initialize arrays
voiced_frames = zeros(size(y));
unvoiced_frames = zeros(size(y));

% Loop through frames
for i = 1:num_frames
    % Extract frame
    start_idx = (i-1)*(frame_size_samples - overlap_samples) + 1;
```



```
    end_idx = start_idx + frame_size_samples - 1;
    frame = y(start_idx:end_idx);

    % Calculate autocorrelation
    autocorr = xcorr(frame);

    % Find peaks in autocorrelation (excluding the zero lag)
    [~,locs] = findpeaks(autocorr(frame_size_samples+1:end));

    % Check if peaks are above threshold
    if ~isempty(locs) && max(autocorr(locs+frame_size_samples)) > threshold
        % Voiced frame
        voiced_frames(start_idx:end_idx) = frame;
    else
        % Unvoiced frame
        unvoiced_frames(start_idx:end_idx) = frame;
    end
end

% Plot voiced and unvoiced segments in time domain
figure;
t = (0:length(y)-1) / Fs;
```

```
subplot(2,1,1);
plot(t, y, 'b', t, voiced_frames, 'r');
title('Voiced (Red) and Unvoiced (Blue) Segments in Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Original', 'Voiced');

subplot(2,1,2);
plot(t, y, 'b', t, unvoiced_frames, 'g');
title('Voiced (Red) and Unvoiced (Green) Segments in Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Original', 'Unvoiced');

% Plot voiced and unvoiced segments in frequency domain
Y = fft(y);
Voiced = fft(voiced_frames);
Unvoiced = fft(unvoiced_frames);

frequencies = (0:length(Y)-1) * (Fs / length(Y));
figure;
subplot(2,1,1);
```

5. Segregate Voice and Un-voice part for frequency domain:

```
plot(frequencies, abs(Y), 'b', frequencies, abs(Voiced), 'r');
title('Voiced (Red) and Unvoiced (Blue) Segments in Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
legend('Original', 'Voiced');
xlim([0, 600]);

subplot(2,1,2);
plot(frequencies, abs(Y), 'b', frequencies, abs(Unvoiced), 'g');
title('Voiced (Red) and Unvoiced (Green) Segments in Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('|Y(f)|');
legend('Original', 'Unvoiced');
xlim([0, 600]);
```

Discussion: Here to distinguish voiced and unvoiced signal autocorrelation is employed to capture periodic patterns within each audio frame. Voiced speech typically exhibits periodicity, and autocorrelation helps identify this periodic behavior by detecting peaks at the corresponding time lags. The thresholding step ensures that only frames with a significant autocorrelation peak are classified as "voiced," while others are considered "unvoiced."

Final Discussion:

In this MATLAB experiment, we delved into the Time and Frequency Domain Analysis of a recorded voice signal. First, we recorded a voice sample, converting the analog signal into a digital representation using a microphone. We emphasized the significance of sampling frequency, ensuring it adheres to the Nyquist-Shannon theorem for accurate signal reconstruction.

Final Conclusion:

Overall, this experiment equipped us with valuable skills in processing and analyzing audio data, showcasing the interplay between time and frequency domains. Understanding these domains is crucial for various applications, including speech processing, audio compression, and voice recognition systems.