Task 1.1: plot voice audio signal in the time domain (linear axis: normalised amplitudes vs time)

The recorded sentence was loaded in Python using scipy.io.wavfile. Since the audio was mono the left channel was used since there wasn't anything on the right channel as its mono. The amplitude values were normalised to the range of −1 to +1 using the integer bit depth. The time axis was generated from the sampling rate (44.1 kHz). The resulting plot shows the speech waveform in the time domain with linear axes (normalised amplitude versus time).

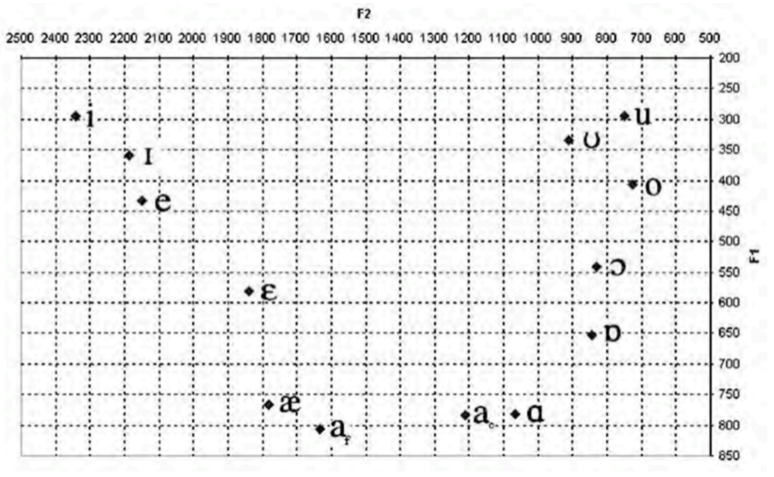
Peaks of higher amplitude correspond to voiced vowel regions, while low-amplitude or rapidly changing sections correspond to unvoiced consonants or silences.

Task 1.2: plot the frequency domain: logarithmic axis for both frequency and amplitude where the amplitude should be in dB with proper axis labels

The time domain signal was transformed into the frequency domain using Fast Fourier Transform (FFT). fd_mag_raw = np.fft.fft(td_normalized) This converts the waveform into its frequency components, producing complex coefficients that represent both the amplitude and phase at each frequency. The corresponding frequency values for each FFT were obtained using: freq_raw = np.fft.fftfreq(num_samples, 1/sample_rate) which returns the frequency in Hz associated with every FFT coefficient. Because the input signal is real, its spectrum is symmetric. Therefore, only the positive frequencies were kept: pos_mask = freq_raw >= 0. The magnitude of the complex FFT coefficients was then converted to decibels (dB) using: fd_db = 20 * np.log10(2/num_samples * np.abs(fd_mag_raw))[pos_mask]. Where np.abs(fd_mag_raw) gives the magnitude of each complex FFT value. 2/num_samples scales the amplitude correctly for a positive sided spectrum. The logarithmic function converts the linear magnitude to a logarithmic dB scale and [pos_mask] keeps only the positive half of the spectrum. Finally, the frequency domain plot was created with a logarithmic frequency axis.

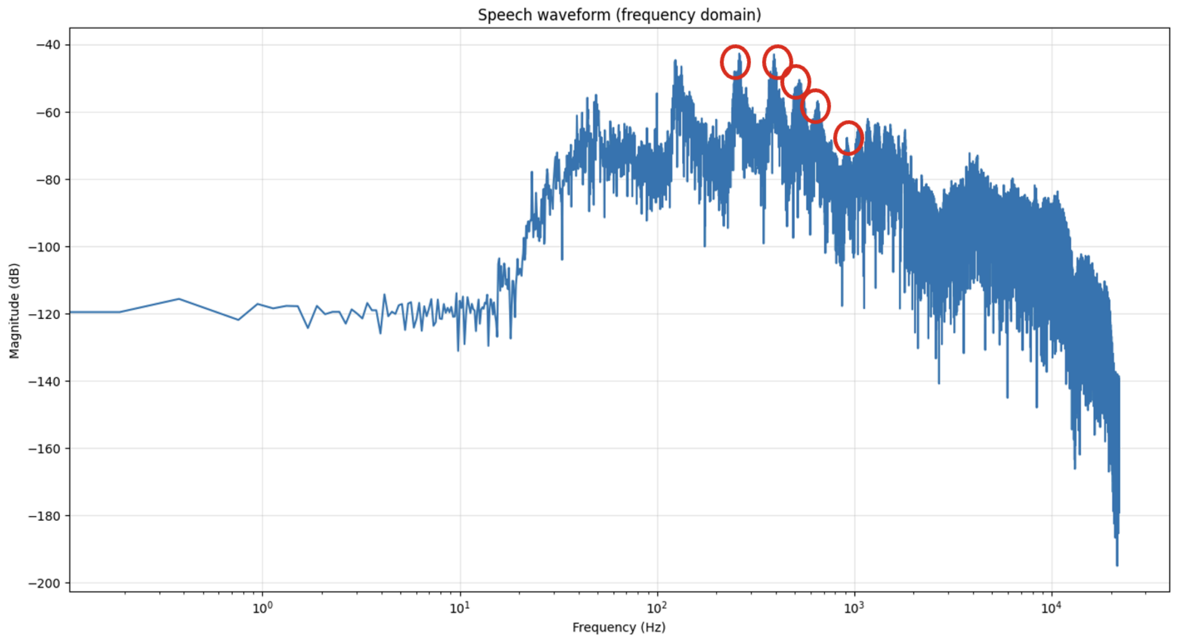
Task 1.3: Mark the peaks in the spectrum which correspond to the fundamental frequencies of the vowels spoken.

Plot 1: Phonetic Pronunciation Vowels and their Frequency Ranges

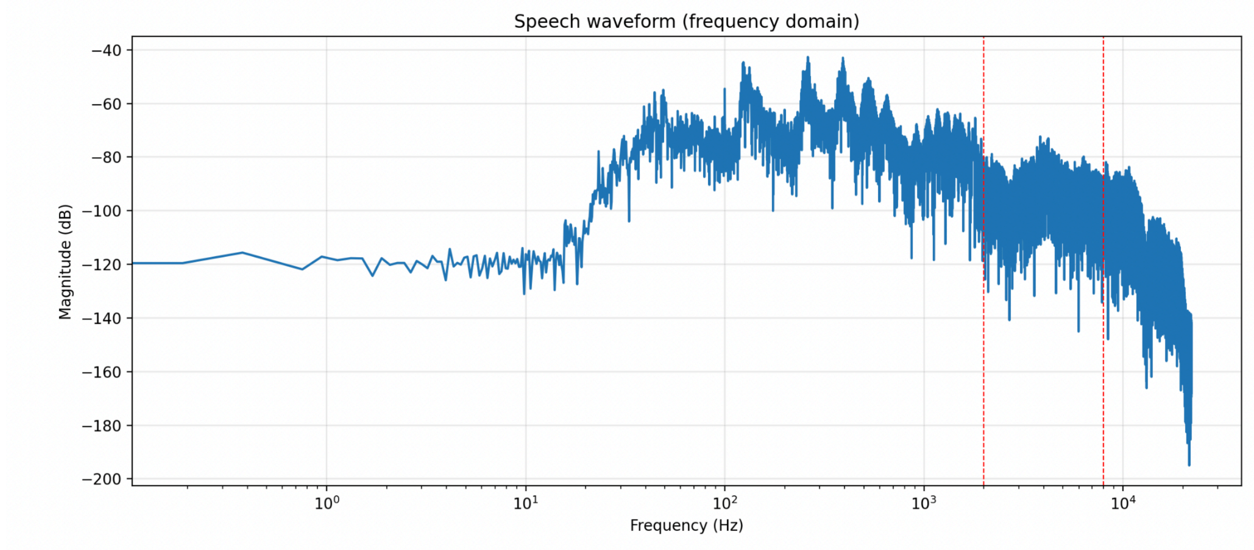


Our sentence was “Our group is Veer and Arda”. According to Wiktionary, these are the phonetic pronunciations for all the words in the sentence.
“Our” : aʊɚ “Group”: ɡɹuːp “is”: ɪz “Veer”: vɪɹ “and”: ænd
As can be seen on the plot, F1 for all of these vowels are between 250 Hz – 850 Hz. The peaks on the frequency domain plot to which these vowels correspond are shown below.

Plot 2: Frequency domain representation of the speech signal showing vowel formant peaks

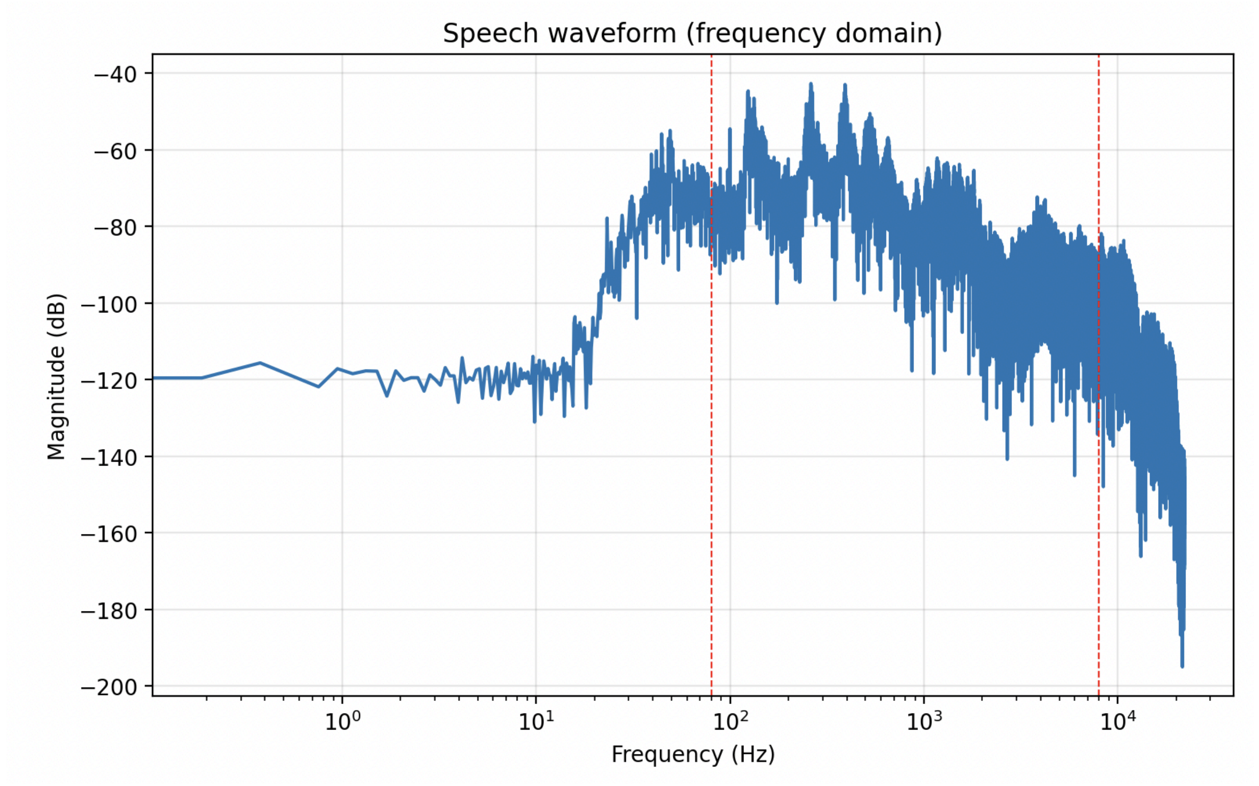


Task 1.4: Mark up the frequency range which mainly contains the consonants up to the highest frequencies containing them.



The range containing consonants (2000 Hz – 8000 Hz) is marked

Task 1.5: Mark up the whole speech spectrum containing the vowels, consonants and harmonics.



The range containing the vocal spectrum (80 Hz – 8000 Hz) is marked

Task 2: Low pass filtering and downsampling of the speech signal

In telephony, the standard sampling rate is 8kHz and the standard speech bandwidth is approximately from 300Hz to 3400Hz. To meet this standard the signal was low pass filtered at 3.4kHz to remove higher frequencies and then downsampled to 8kHz, satisfying the Nyquist criterion ($2 \times 3.4kHz = 6.8kHz < 8kHz$).

The downsampling was done using interpolation. After applying the low pass filter a new set of time points were generated at the target sampling rate covering the same duration as the original signal. The amplitudes of the signal at these new time points were obtained by estimating them with NumPy's linear interpolation.

During testing, we also experimented with lower sampling rates such as 4kHz and found that the speech was still recognizable but with significant loss of clarity particularly in high frequency consonants. The 8kHz version represented the best compromise between intelligibility and bandwidth reduction. Therefore we used a sampling rate of 8kHz and bandwidth of 3.4kHz.

Task 3: Touch Tone Detection

In this task, a DTMF (Dual Tone Multi Frequency) detection algorithm was implemented to identify which keys were pressed on a telephone keypad.

First, the DTMF row and column frequencies were defined according to the standard frequency table. Each combination of one row and one column frequency corresponds to a specific button on the phone dial, and these pairs were stored in a DTMF mapping table.

The process begins by loading the WAV file using wavfile.read() and converting the audio data into a NumPy array. The signal was then normalized to the range between -1 and 1 to standardize amplitude values. Next, the button_press_detector() function was used to detect button presses in the time domain. In this function, the amplitude values along the y-axis were compared against a threshold value. When the signal exceeded the threshold, it indicated the start of a button press, and when the signal stayed below the threshold (between 0 and 0.1) for 820 samples, it was considered the end of that press. Each detected button press segment was stored as a tuple containing the start and end indices of the original signals NumPy array, and since the WAV file represented a phone number, the result was an array containing 11 such tuples.

After the segments were identified, the detect_which_button_is_pressed() function was called. This function takes the list of segments, the sample rate, and the full time domain signal as input. For each segment, it extracts the corresponding end and start indexes of the signal and performs a Fast Fourier Transform (FFT) to convert it from the time domain to the frequency domain. The resulting frequencies and magnitudes are stored in separate arrays. Only positive frequencies and magnitudes are kept, as the negative side of the spectrum is redundant for real valued signals.

Boolean values are then created to separate the frequency spectrum into two regions which are the low frequency row band (600–1000 Hz) and the high frequency column band (1150–1700 Hz). Within each band, the frequency with the highest magnitude is selected. The detected highest magnitude frequencies are then compared with the DTMF row and column frequencies, and the closest matching values are chosen. Finally, the matching (row, column) pair is looked up in the DTMF map to identify the corresponding button that has been pressed. Each detected key is appended to an array of digits, and the function outputs the full phone number detected from the WAV file as a sequence of digits.