Faculty of Engineering and Technology
Electrical and Computer Engineering Department

Spoken Language Processing
ENCS5433
Assignment 1

Prepared by:

Dana Assad	**1211451**
Jack Ghunaim	**1190876**

Instructor:
Dr. Abualseoud Hanani

Section: 1

BIRZEIT
December – 2024

Abstract

The aim of this assignment in Spoken Language Processing is to explore the analysis, synthesis, and processing of speech signals through eight interconnected experiments. It highlights components of speech production, such as spectrogram evaluation, digital filtering, formant description, and concatenative speech synthesis. Participants will use the Speech Filing System (SFS) and MATLAB or Python to conduct activities like analyzing spectrograms, measuring formant frequencies, identifying phonemes, and experimenting with synthesis techniques. Supplementary exercises include bandwidth analysis, digital signal filtering, measurement of vowel formants, and vowel generation using source-filter modeling. This thorough method enhances the comprehension of speech mechanics, integrating acoustic physics, signal processing, and linguistic phonetics for a detailed investigation of spoken language processing methods.

Contents

Table of Figures

List of Tables

Part1: Basic sound analysis

In order to comprehend the bandwidth and variations in frequency resolution of audio signals, this section focuses on spectrogram analysis. Spectrograms which show how a signal's frequency content varies over time, are crucial tools in sound analysis. The way that wide-band and narrow-band spectrograms resolve frequency and time information varies. Narrow-band spectrograms get better in frequency resolution, whereas wide-band spectrograms prioritize time resolution.

a. Bandwidth

To examine the bandwidth of the audio signals in sound1.wav and sound2.wav.

⇨ We followed these steps:

- Audio files can be imported into SFS software through [Item | Import | Speech (copy)].
- Wide-band spectrograms produced using [Tools | 1.Speech | Display | Wide Spectrogram].
- Spectrograms analyzed for lower and upper frequency thresholds with visible energy, ±500 Hz buffers included for precision, and frequency ranges recorded for each file.

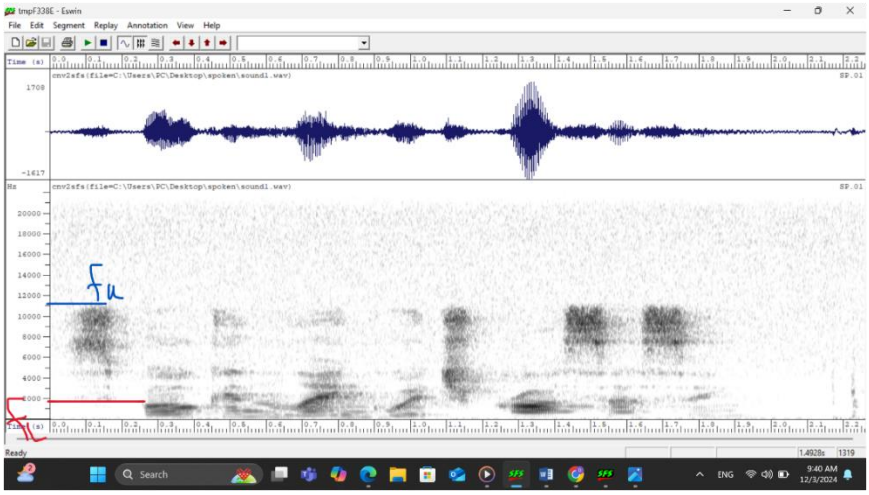


Figure 1: Wide spectrogram of sound1.wav

As shown in figure 1 above, the wide-band spectrogram of `sound1.wav` illustrates a lower frequency threshold just under 2,000 Hz and an upper threshold near 11,000 Hz, indicating the primary energy distribution. The energy distribution across the spectrum is inconsistent, with specific frequency bands exhibiting concentrated energy, affected by the characteristics of the sound source, its pitch, harmonics,

And the acoustic qualities of the environment. The majority of energy is concentrated in mid-range frequencies, whereas higher frequencies show diminished energy, probably because of attenuation or the inherent characteristics of sound waves. A ±500 Hz buffer allows for minor spectral variations to ensure thorough frequency range assessment.

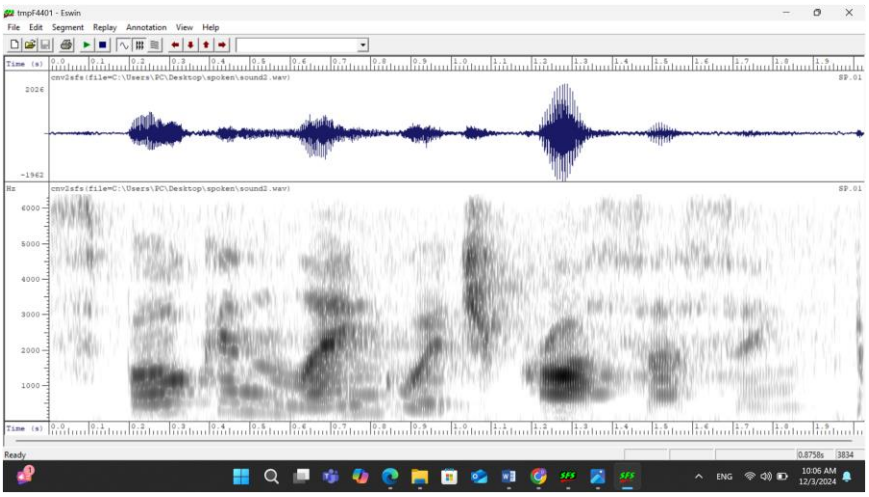


Figure 2: Wide spectrogram of sound2.wav

In the wide-band spectrogram of `sound2.wav`, a notable frequency range can be observed between around 500 Hz and 6,000 Hz. The minimum frequency is approximately 500 Hz, with energy gradually increasing over time. The majority of energy is found within this range, but some higher frequencies go beyond 6,000 Hz. This suggests that there is focused energy in the lower and mid-range frequencies, while higher frequencies are less prominent. The spectrogram displays temporary spikes and changes in frequency content, indicating a complexity in the sound signal and a more dispersed energy distribution in comparison to `sound1.wav`, probably because of its intricate harmonic structures.

➔ The table below includes the necessary values for lower and upper frequency boundaries (+/- 500Hz) of these sounds.

Table 1: Frequency Values

	SOUND1.	**SOUND2.**
LOWER FREQUENCY	1000 Hz	500 Hz
UPPER FREQUENCY	11000 Hz	6000 Hz

B. Spectrogram

This part examines the spectrograms of "sound3.wav" and "sound4.wav" utilizing wide-band and narrow-band methods. It shows the variations in frequency and time features of the audio signals, emphasizing how each type of spectrogram influences the depiction of spoken sentences.

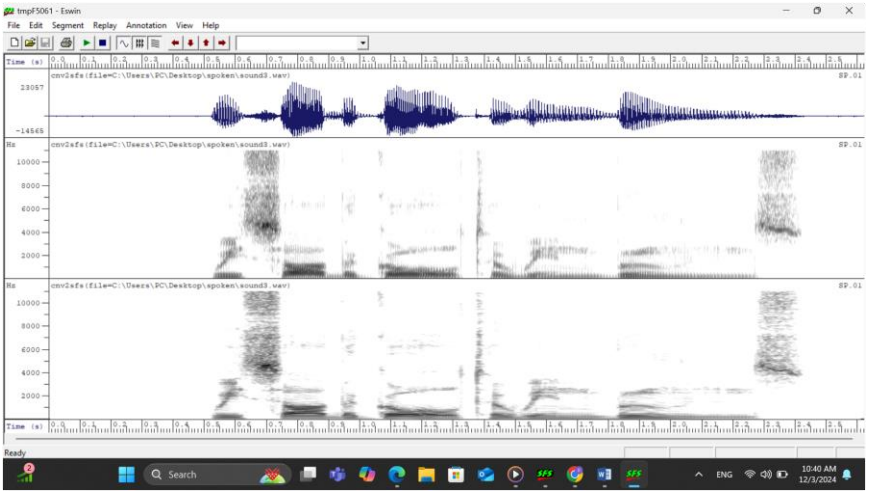


Figure 3: Wide and Narrow Spectrogram of sound3.wav

⇔ The **wide-band spectrogram** of Sound3 emphasizes clear shifts between phonemes. The consonant "s" in "saw" and the stops in "goal" show bursts of intense energy over a broad frequency range, seen as vertical bands. The vowels, like "saw" and "goal," exhibit consistent harmonic patterns, showcasing their quasi-stationary characteristics. In comparison, **the narrow-band spectrogram** displays more distinct harmonic structures, featuring noticeable horizontal lines that represent the vocal pitch (fundamental frequency) and its harmonics. These harmonics stay fairly stable throughout vowel sounds and show some variation during transitions.

The **wide-band spectrogram** offers outstanding temporal resolution, accurately recording the exact timing of shifts between speech sounds such as consonants and vowels. Nevertheless, the frequency resolution is lower, causing individual harmonics to be less discernible. On the other hand, **the narrow-band spectrogram** offers superior frequency resolution, highlighting the harmonic structure of vowels while diminishing the clarity of temporal transitions. This trade-off highlights temporal characteristics in the wide-band and frequency characteristics in the narrow-band representation.

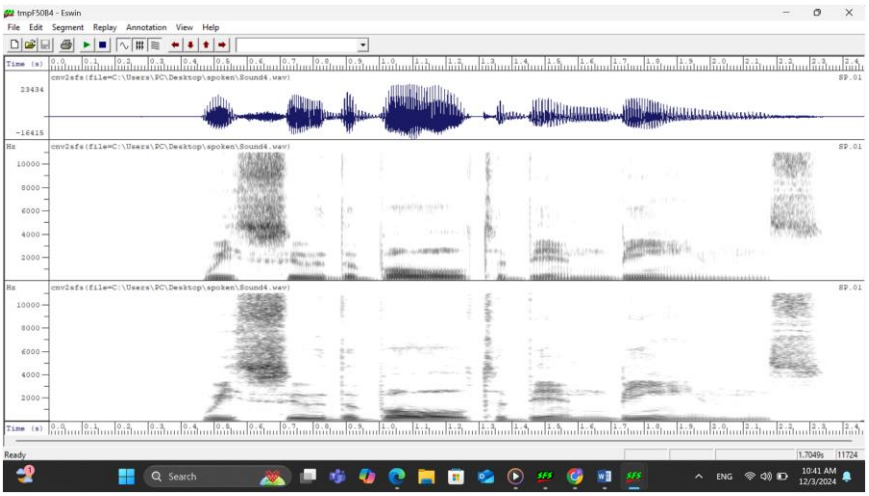


Figure 4: Wide and Narrow Spectrogram of sound4.wav

⇔The **wide-band spectrogram** for Sound4 also depicts quick shifts, like the fricative "s" in "sue" and the stop "b" in "bin," shown as brief, high-energy bursts. The vowel sounds in "sue" and "bowl" display more fluid energy patterns. In the **narrow-band spectrogram**, the harmonic arrangement is clearer, featuring noticeable horizontal lines that indicate the fundamental frequency and its harmonics while the vowel sounds occur. In the narrow-band view, the consonant transitions are not as clearly defined as in the wide-band representation.

The **wide-band spectrogram** is outstanding at capturing quick occurrences, like the brief, explosive surge of the "b" sound in "bin." Nonetheless, its frequency resolution is lower, because the harmonic elements merge into wider ranges. Conversely, the narrow-band spectrogram shows the intricate harmonic composition of vowels, facilitating the analysis of pitch and formant frequencies. This leads to reduced temporal accuracy but facilitates an in-depth examination of stationary sounds, such as vowels.

➔ In addition, the spectrograms of Sound3 ("We saw the goal to win boons") and Sound4 ("We sue the bowl to bin beans") mainly vary in the articulation of phonemes. For instance, the fricative "s" in "saw" and "sue" exhibits unique energy patterns, while the vowel transitions reveal slight variations in harmonic structures. Moreover, consonants such as "b" in "bin" in Sound4 show more distinct bursts, differing from stops like "g" in "goal" in Sound.

⇔ The two types of spectrogram enhance each other in examining various elements of the speech signals. Wide-band spectrograms are optimal for analyzing timing and transitions, whereas narrow-band spectrograms focus on frequency specifics such as pitch and harmonics.

Part2: Filters

This part involves generating a 1 kHz sine wave signal and a white noise signal sampled at 16 kHz. These signals will be merged and processed through IIR and FIR digital filters. The aim is to evaluate and contrast the performance of each filter, specifically in terms of their effects on the sinusoidal and noise elements.

→ To reach that goal we follow these steps:

- Using SFS [Tools->Generate->Test signals] , generate a 1 KHz sinusoidal wave and a white noise signal with a 16 KHz sampling rate for a duration of 3 seconds. And save each in sinusoidal_signal.wav and Wnoise_signal.wav files.

- In MATLAB, utilize `audioread()` to import the sinusoidal and white noise `.wav` files, then sum the two signals with the `+` operator to produce a merged signal. As shown in figure 5 below:

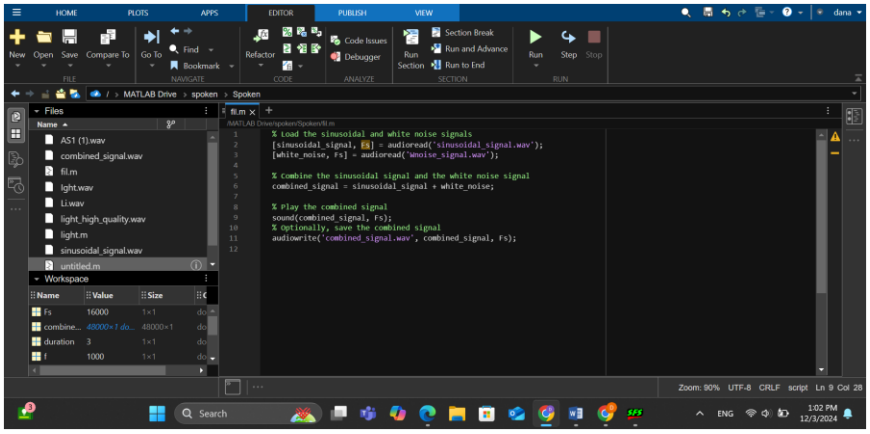


Figure 5: Matlab code to add the two signals

⇨ The output save in file named: **'combined_signal.wav'**

- Then, implement and design the IIR & FIR:

→ Design the IIR Filter:

Design an IIR low-pass filter in MATLAB utilizing the equation:

$$y(n) = 0.99y(n-1) + x(n)$$

For a 1 kHz sinusoidal waveform that contains noise. Employ a loop or `filter()` to implement the filter, then use `sound()` to listen to both signals.

Moreover, determine the magnitude frequency response of the IIR filter by utilizing the `freqz` function, which shows the filter's reaction across different frequencies. The magnitude is subsequently transformed into decibels using the formula **20 * log10(magnitude_iir)** to enhance visualization. This will be accomplished effectively; refer to the code below:

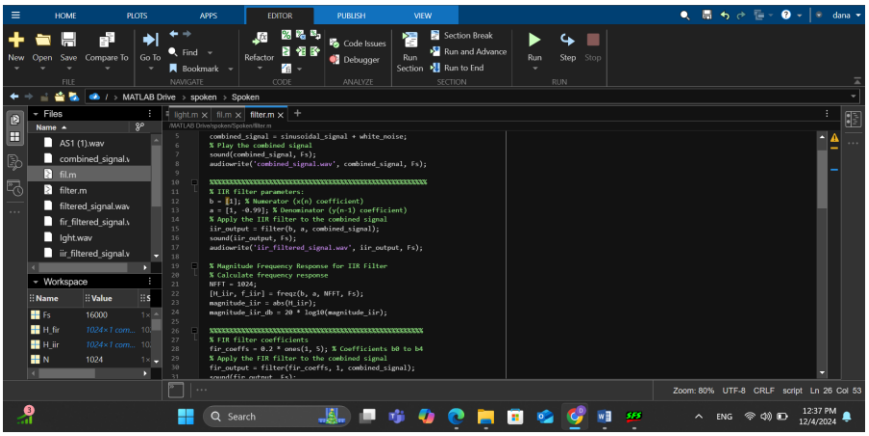


Figure 6: Matlab Code of IIR Filter design

→ Design the FIR Filter:

Construct the FIR filter utilizing the averaging filter equation:

$$y(n) = 0.2x(n) + 0.2x(n-1) + 0.2x(n-2) + 0.2x(n-3) + 0.2x(n-4)$$

Which averages five input samples to help smooth out and diminish high-frequency noise in the output signal.

Additionally, calculate the frequency response of the FIR filter with freqz, which provides the complex response H_fir. The size of this response is determined using abs(H_fir) and is subsequently transformed into decibels (dB) with 20 * log10(magnitude_fir) to visualize the filter's magnitude frequency response. This will be accomplished successfully; refer to the code provided below:

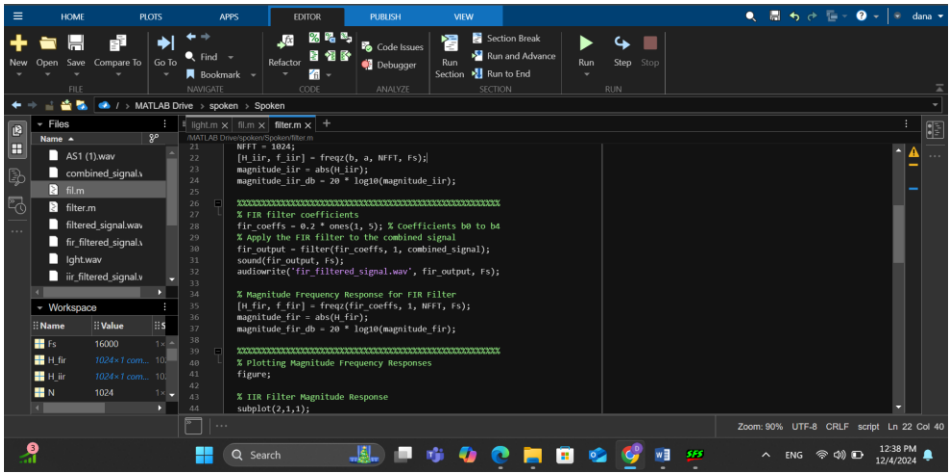


Figure 7: Matlab Code of FIR Filter design

→ The code below, to create a figure containing two subplots: the magnitude frequency responses in dB of the IIR and FIR filters, both appropriately labeled.

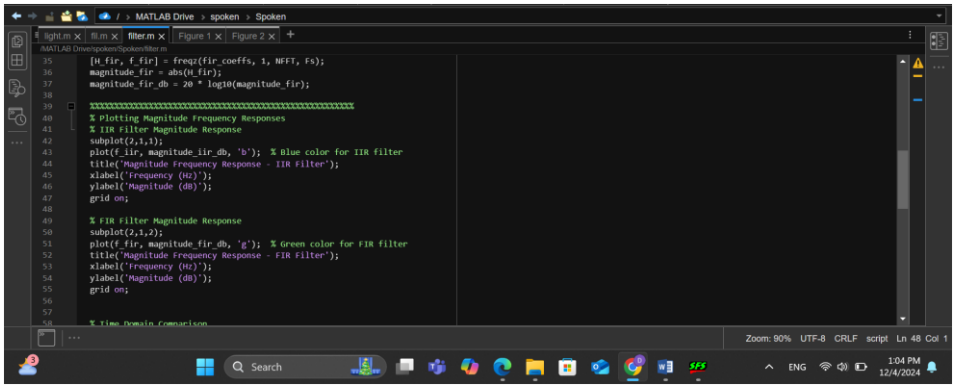


Figure 8: Matlab code plot the magnitude frequency characteristic

The result:

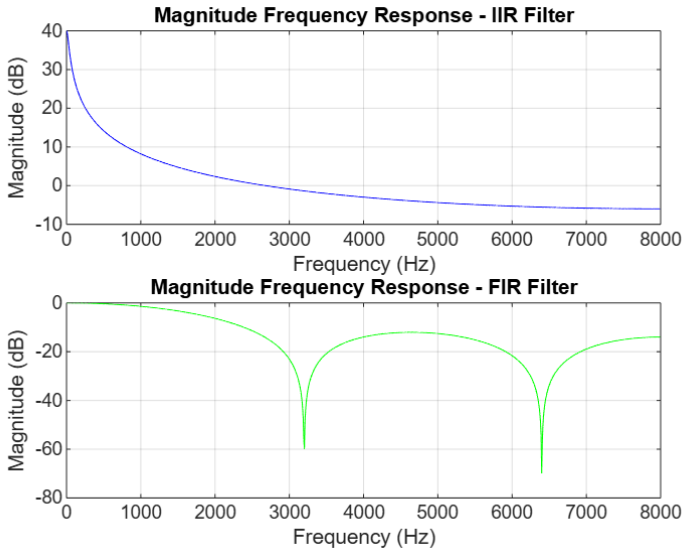
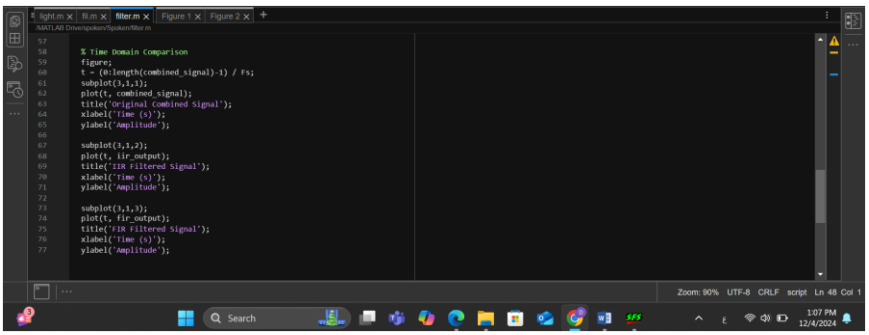


Figure 9: Magnitude frequency characteristic of each filter

→ This code creates three subplots that compare the original signal with the IIR and FIR filtered signals over time.



```
% Time Domain Comparison
figure;
t = (0:length(combined_signal)-1) / Fs;
subplot(3,1,1);
plot(t, combined_signal);
title('Original Combined Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,2);
plot(t, iir_output);
title('IIR Filtered Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,3);
plot(t, fir_output);
title('FIR Filtered Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```

Figure 10: plot the original & filtered signals of filter over time

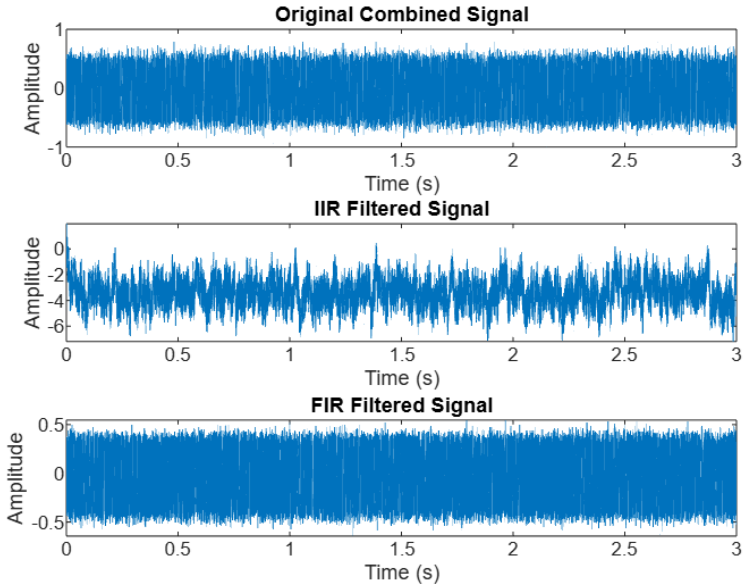


Figure 11: The signal's output of filters

➔ The signal filtered by IIR (Infinite Impulse Response), depicted in the middle plot, exhibits more gradual amplitude fluctuations compared to the original combined signal above. This smoothing characteristic is typical of IIR filtering, which employs extended memory and a recursive method, considering both present inputs and past outputs. As a result, IIR filters efficiently reduce quick variations and noise. The slow alterations in the IIR filtered signal result from averaging quick fluctuations, which reduces the impact of white noise. This quality is advantageous for uses that seek to remove high-frequency noise while maintaining the signal's overall form and pattern. The precise filter settings, which encompass coefficients, determine the degree of smoothing and frequency response; in this scenario, the IIR filter effectively minimizes high-frequency noise while preserving the sinusoidal characteristics of the original signal.

➔ The effects of filtering on a combined signal made up of a sinusoidal wave and white noise are demonstrated by the waveforms shown. With the irregular amplitude fluctuations of white noise superimposed on the obvious periodic oscillations of the sinusoidal component, the first "Combined Input Signal" appears a little chaotic and uneven. A cleaner and more constant waveform results from the significant decrease in noise level that occurs after filtering, as demonstrated in the "Output Signal after Filtering." Because the sinusoidal wave's periodic structure is mostly unaltered, it appears that the filter mostly reduced the high-frequency noise components while maintaining the signal's basic frequency. The ability of the filter to selectively lower noise without appreciably changing the essential characteristics of the original signal is demonstrated by this comparison.

The main differences between FIR and IIR filters are that the first (FIR) have limited memory that is suitable for immediate filtering tasks and guarantee constant stability, while the latter have extended memory that permits gradual smoothing and may introduce instability due to recursion. The selection of filters in signal processing is impacted by these variations.

Part 3: Speech Analysis

The source-filter model of speech production explains how speech is generated by passing a source signal, such as the glottal waveform, through a filter created by the vocal tract. Formants, which are the resonant frequencies of the vocal tract, define the speech spectrum. The [Tools | 1.Speech | Display | Cross Section] tool utilizes linear predictive analysis to illustrate the source and filter spectra as well as the complete speech spectrum.

→ S & F & O Spectrum:

- **Source Spectrum (S):**

The source spectrum represents the raw sound before it passes through the vocal tract filter. Typically, for human speech, this will either be periodic (for voiced sounds) or aperiodic (for unvoiced sounds). If the spectrum has a lot of irregularities or noise across a wide frequency range, it indicates that the source is more aperiodic, like in fricative sounds (e.g., "s," "sh"). And if the spectrum is relatively flat across frequencies, it suggests a non-tonal sound.

- **Filter Spectrum (F):**

The filter spectrum shows the resonant frequencies of the vocal tract, which are typically observed as sharp peaks. These peaks correspond to formants and identify the formants, explaining the configuration of the vocal tract and how it shapes the speech sound.

- **Overall Spectrum (O):**

The overall spectrum clearly shows the influence of the filter on the source signal. The harmonics of the source are present, but they are modified by the peaks corresponding to the formants at (F1) and (F2). The resulting spectrum is characteristic of a clear front vowel sound, with a tonal quality due to the periodic source signal and a well-defined spectral envelope.

❖ **S & F & O Spectrum for AS.wave file:**

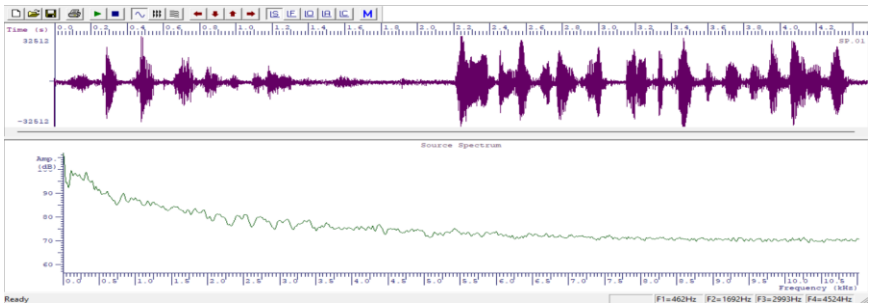


Figure 12: source Spectrum of AS.wav

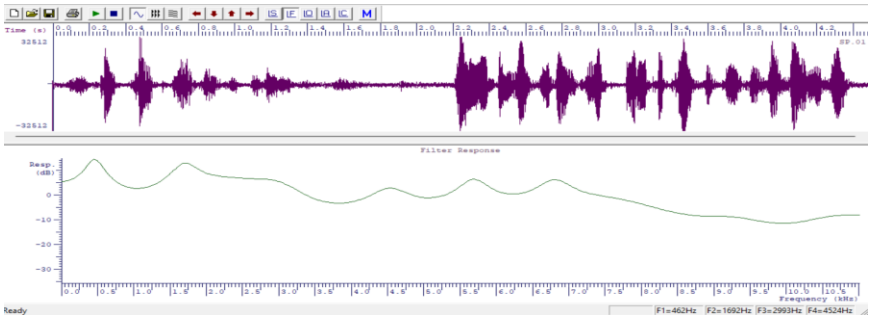


Figure 13: filter Spectrum of AS1.wav

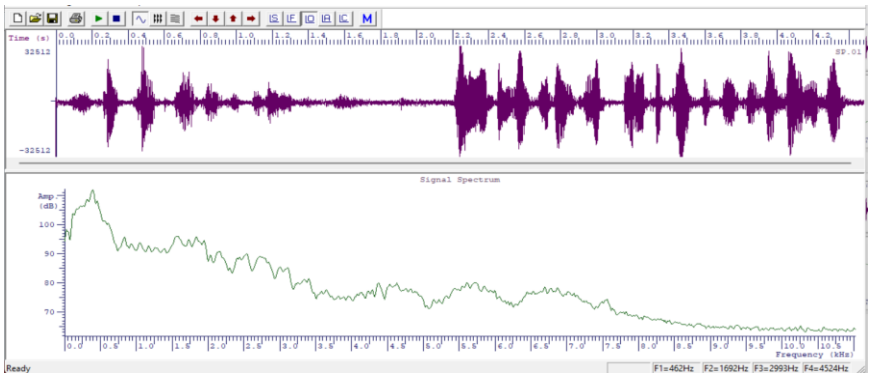


Figure 14: overall Spectrum of AS1.wav

❖ **S & F & O Spectrum for Sample.wave file:**

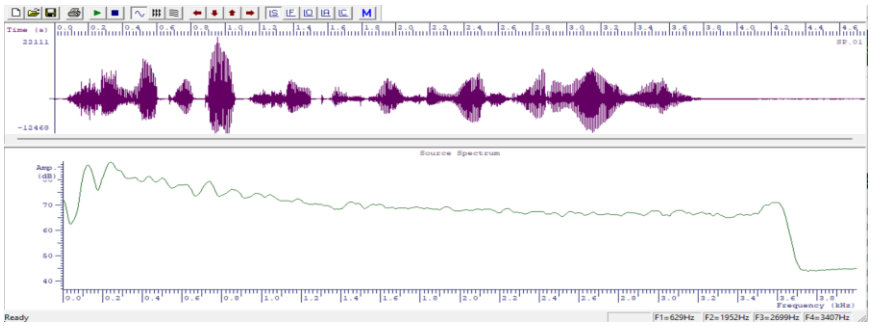


Figure 15: Source Spectrum of sample.wave

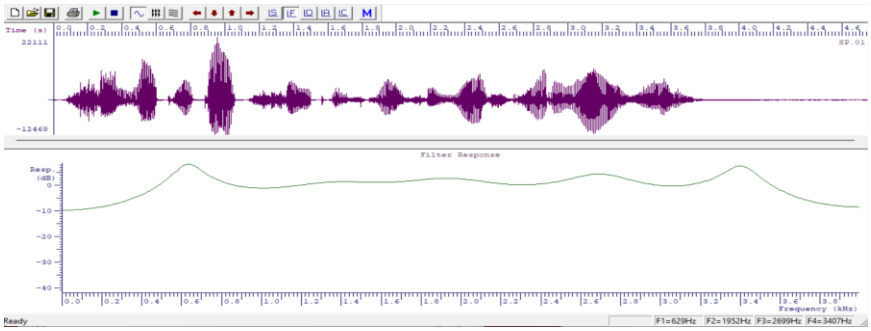


Figure 16: filter Spectrum of sample.wave

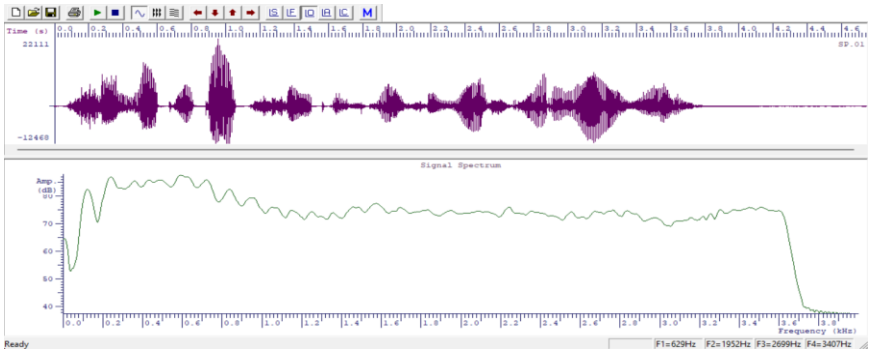


Figure 17: Overall Spectrum of sample.wave

❖ S & F & O Spectrum Sound1.wave file:

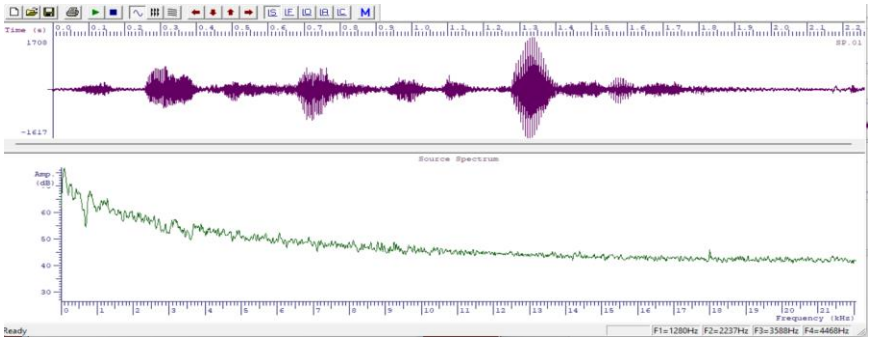


Figure 18: Source Spectrum of sound1.wave

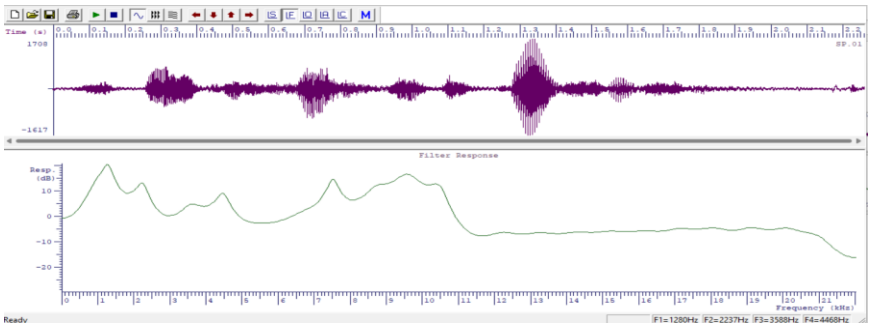


Figure 19: filter Spectrum of sound1.wave

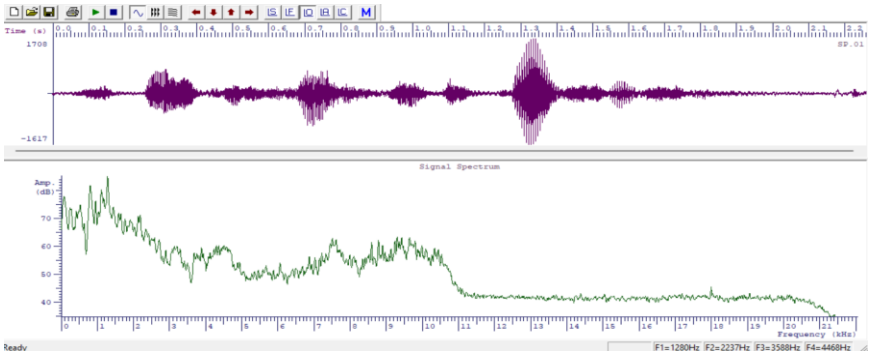


Figure 20: Overall Spectrum of sound1.wave

❖ S & F & O Spectrum Sound2.wave file:

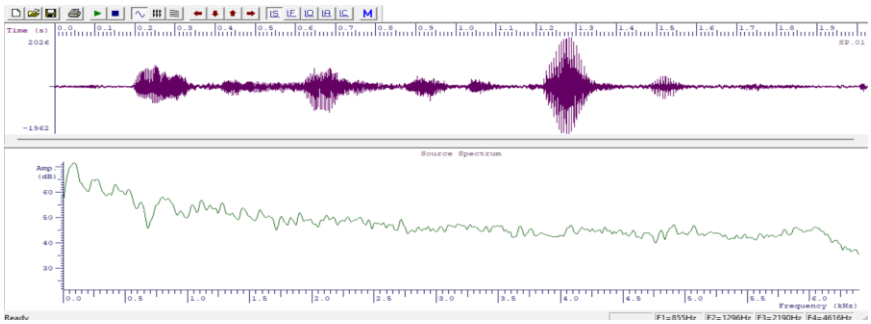


Figure 21: Source Spectrum of sound2.wave

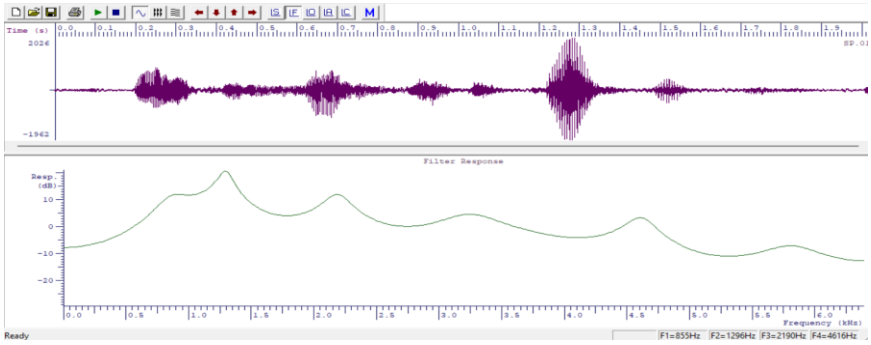


Figure 22: filter Spectrum of sound2.wave

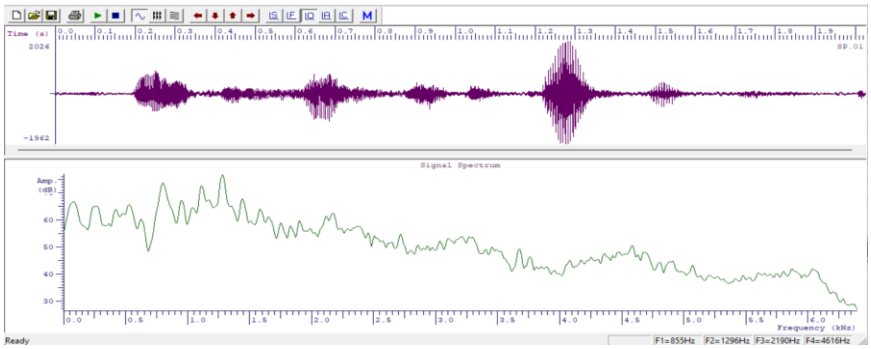


Figure 23: overall Spectrum of sound2.wave

❖ S & F & O Spectrum Sound3.wave file:

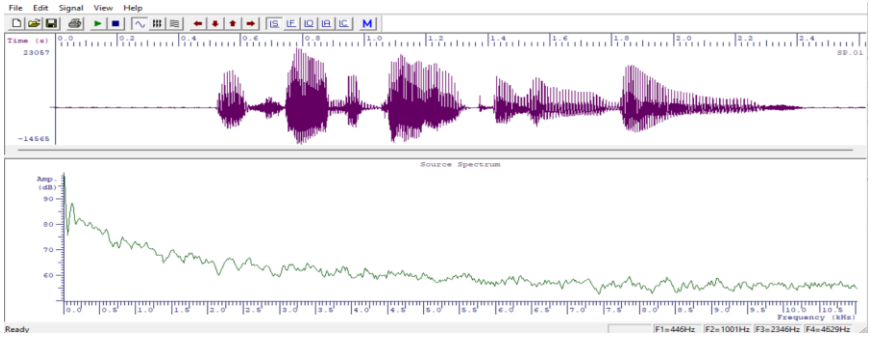


Figure 24: Source Spectrum of sound3.wave

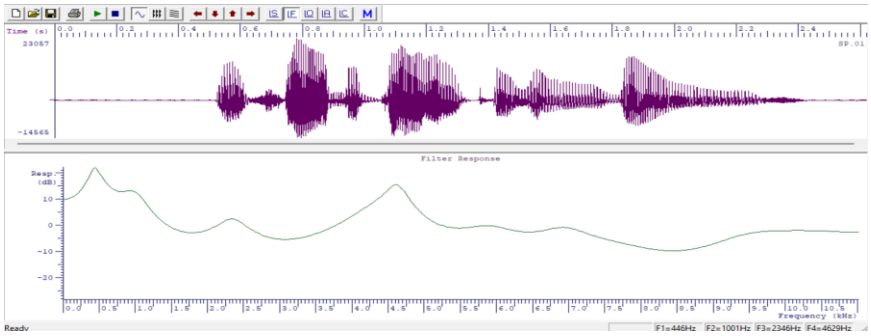


Figure 25: filter Spectrum of sound3.wave

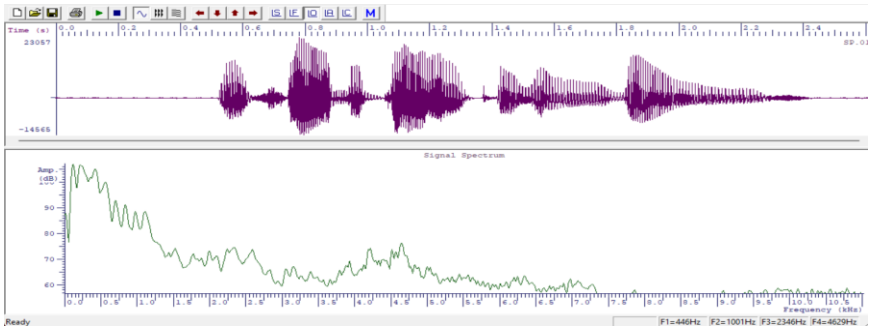


Figure 26: overall Spectrum of sound3.wave

❖ S & F & O Spectrum Sound4.wave file:

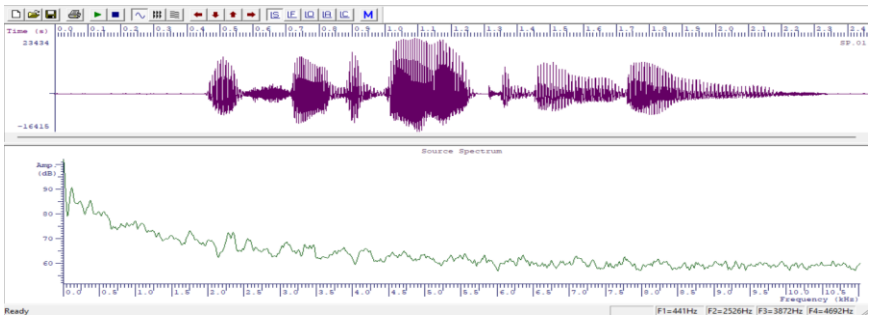


Figure 27: Source Spectrum of sound4.wave

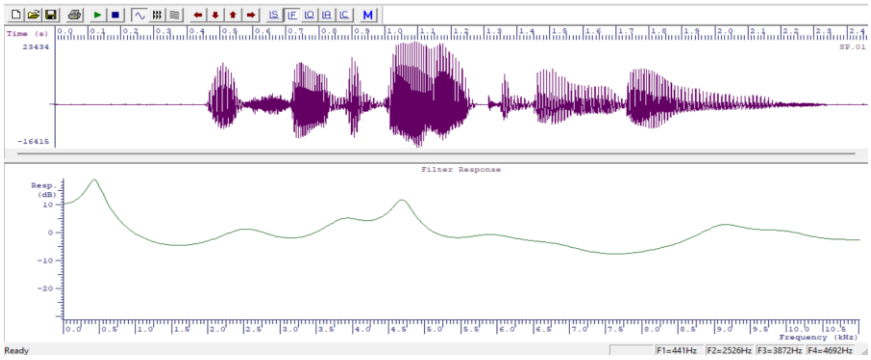


Figure 28: filter Spectrum of sound4.wave

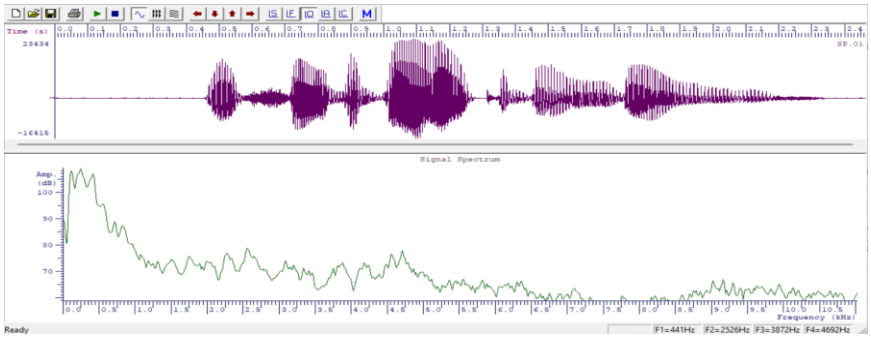


Figure 29: overall Spectrum of sound4.wave

a. Formants for vowels

This task analyzes the first three formant frequencies (F1, F2, F3) of vowels to understand their differences across speakers. Using the (F) display in the tool, formant frequencies are estimated from the filter spectrum, focusing on the middle segment (minimum 20ms) of vowels :/ i: / in 'She'; / { / in 'had'; / u: / in 'suit'; / A: / in 'dark'. From the sample file `sample1.wav`. The analysis includes plotting F1 and F2-F1 values to compare results with theoretical benchmarks, highlighting speaker variability and vowel articulation.

1. Examine the formant frequencies F1, F2, and F3 in the audio file `**sample1.wav**` for the Vowels concentrating on sections lasting a minimum of 20ms as shown in table below:

Table 2: freq of vowels for sample1.wav

Vowel	F1 (Hz)	F2 (Hz)	F3 (Hz)	F2 - F1 (Hz)
/i:/	398	2307	2798	1909
/{/	448	838	2622	390
/u:/	395	1416	2536	1021
/A:/	637	1505	1993	868

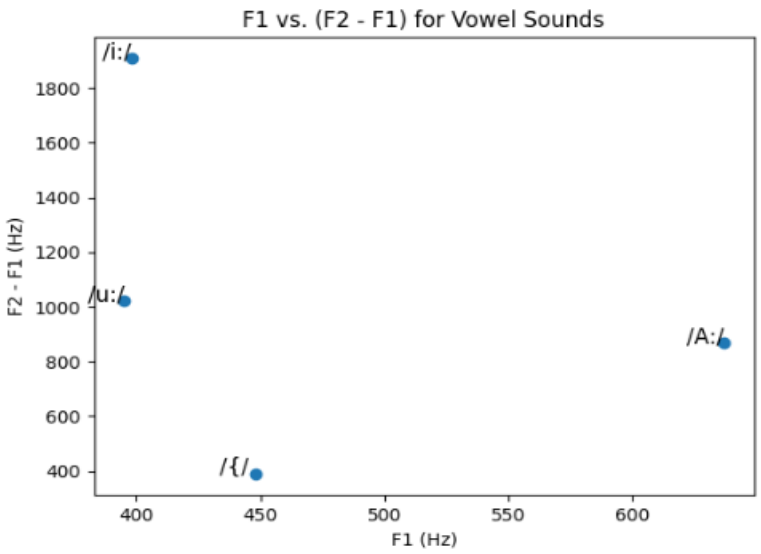


Figure 30: F1 vs (F2-F1) for Sample

2. **For student 1 → (JACK):**

Table 3: freq vowels for student 1

Vowel	F1 (Hz)	F2 (Hz)	F3 (Hz)	F2 - F1 (Hz)
/i:/	321	1879	3438	1558
/{/	521	1458	3481	937
/u:/	533	1961	3145	1428
/A:/	484	1297	2282	813

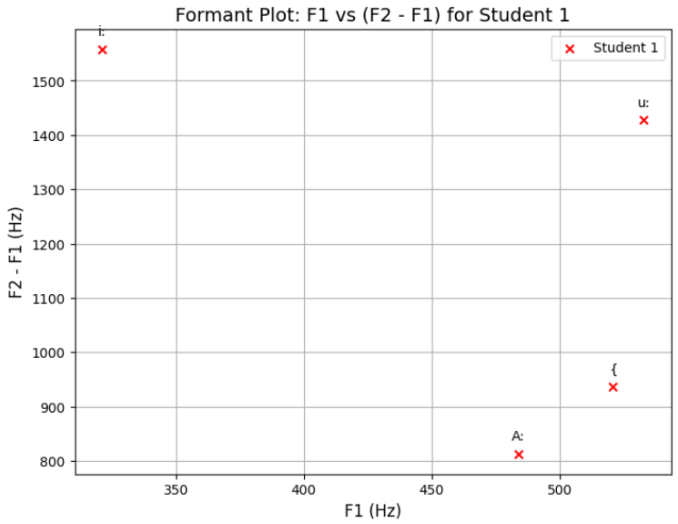


Figure 31: F1 vs (F2-F1) for Student1

3. **For student 2 → (DANA)**

Table 4: freq vowels for student 2

Vowel	F1 (Hz)	F2 (Hz)	F3 (Hz)	F2 - F1 (Hz)
/i:/	355	2050	2750	1695
/{/	475	1220	2800	745
/u:/	440	1550	2700	1130
/A:/	600	1400	2050	800

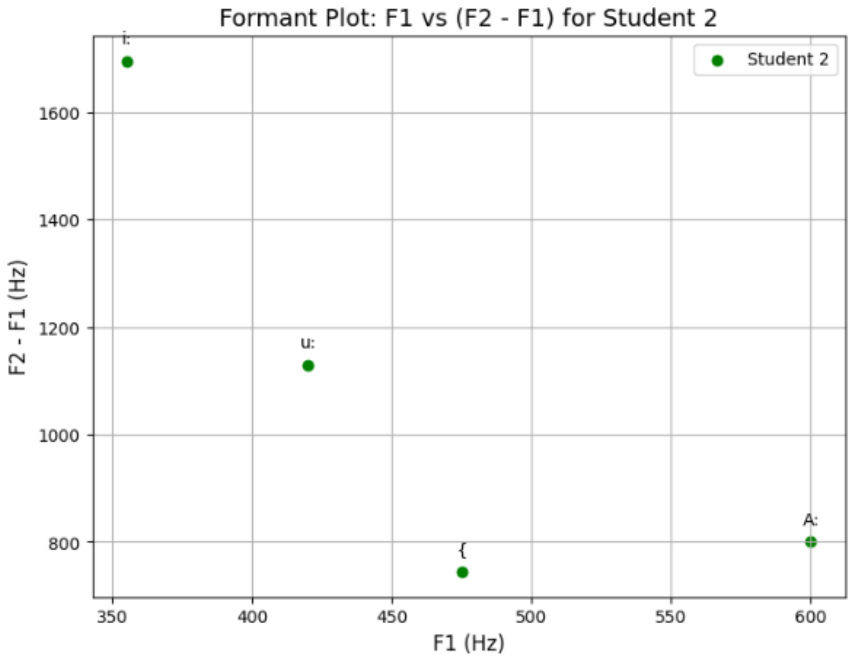


Figure 32: F1 vs (F2-F1) for Student2

➔ All results align with the expected vowel quadrilateral pattern, where front vowels like **/i:/** have high F2 and low F1, and back vowels like **/u:/** show low F2 and low F1.

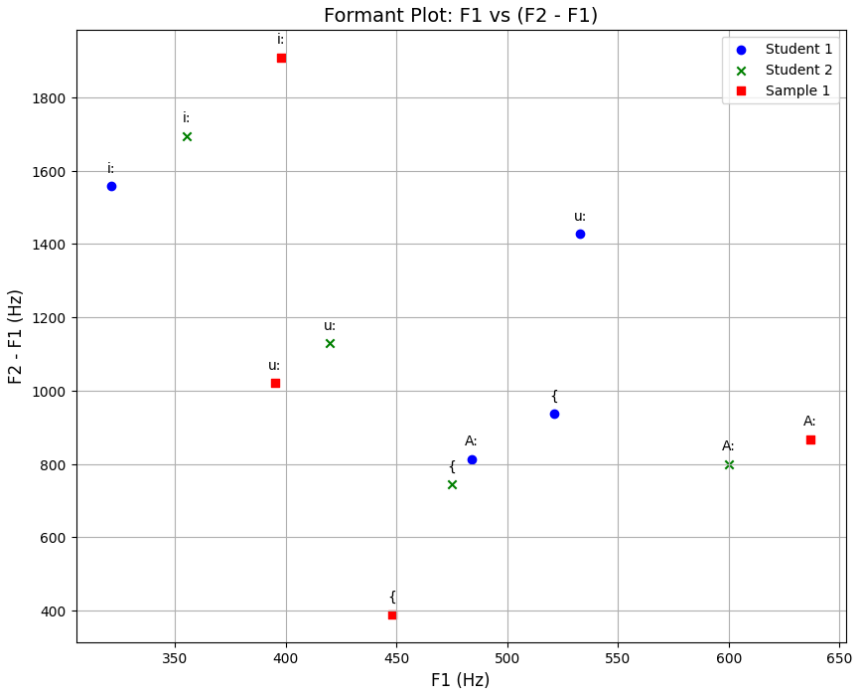


Figure 33: Formant Plot

/i:/ (as in "She"):

Student 1 and **Sample 1** have similar F2 values, with **Student 1** showing a slightly lower F2, suggesting a more centralized tongue position. **Student 2** has a higher F2 closer to **Sample 1**, indicating a more typical production of the vowel.

/{/ (as in "had"):

The formant values for **Student 1** and **Student 2** are consistent, with **Student 2** showing a slightly lower F2, which is typical for central vowels. Both are close to **Sample 1**, indicating similar articulation.

/u:/ (as in "suit"):

Student 2 has a higher F2 than **Student 1**, but both are within the expected range for **/u:/**. The overall formant structure is similar to **Sample 1**, with a lower F1 and F2.

/A:/ (as in "dark"):

Both students show similar F1 and F2 values to **Sample 1**, with **Student 1** having slightly lower F2, aligning with the expected back vowel characteristics.

b. Source and Filter analysis:

Examine the phoneme /s/ from "suit" and /ɑ:/ from "dark" in `sample1.wav`, presenting their source-filter spectra and exploring the differences in their production.

→ Source-filter analysis for phoneme / s /'suit'

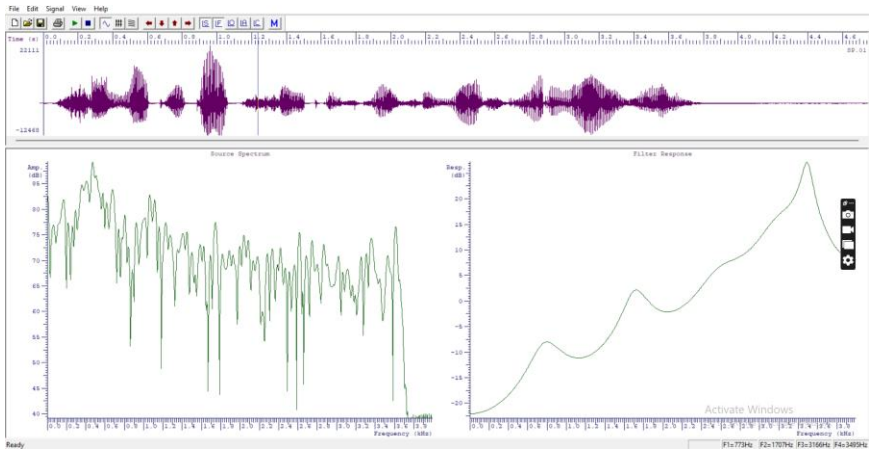


Figure 34: source-filter analysis for phoneme / s /'suit'

For /s/: The source spectrum of /s/ appears noisy (non-periodic), indicating that the sound is produced by turbulent airflow through a constriction (fricative). The filter spectrum shows some resonant peaks, but they are less pronounced, and less harmonic compared to vowel sounds. The **/s/** phoneme has a high-frequency, noisy spectrum due to turbulent airflow, with less structure in the filter spectrum

→ **Source-filter analysis for phoneme / A: /'dark':**

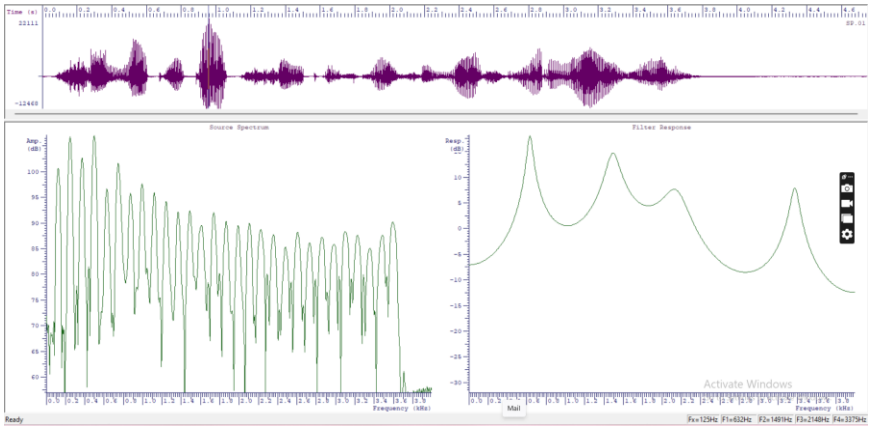


Figure 35: source-filter analysis for phoneme / A: /'dark':

For /A:/: The source spectrum of /A:/ shows a periodic waveform with clear harmonic peaks, indicating a voiced sound. The filter spectrum have distinct, regularly spaced formant peaks that correspond to the resonant frequencies of the vocal tract during the production of the vowel. The **/A:/** vowel has a harmonic, tonal source spectrum with clear peaks at the formant frequencies, corresponding to the resonances of the vocal tract.

C. Fundamental Frequency:

"Had" (vowel /a:/ / { /)

➔ For Vowel Wide Spectrogram was used to clearly see the harmonic structure and F0.

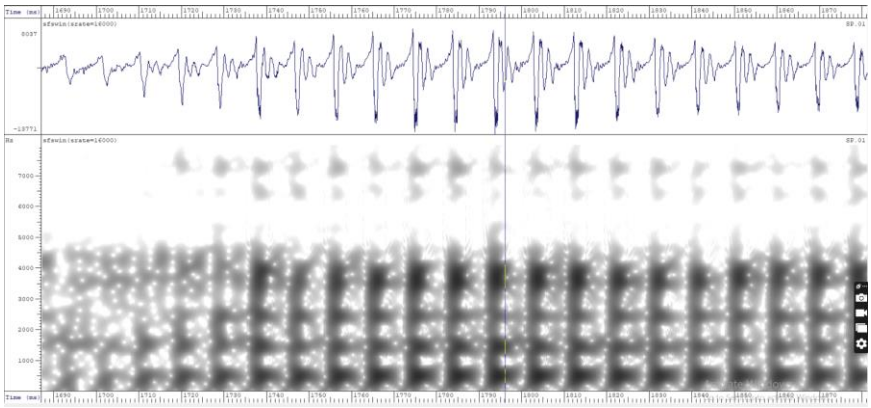


Figure 36: Wide Spectrogram of vowel /a/

- The time interval between two consecutive peaks is 0.01 seconds (10 milliseconds)
- Fundamental Frequency (F0) is calculated using the formula: F0=1/T = 1/0.01=100Hz.
- The waveform shows repeating peaks that represent the periodicity of the voice signal
- The fundamental frequency (F0) was calculated based on the periodicity of the waveform, which shows regular peaks indicative of periodic vibration of the vocal cords during the vowel /a:/ in 'had'. The calculated F0 is 100 Hz, representing the primary pitch of this vowel sound

Part 4: Speech analysis

This part requires analyzing the term **"capacity"** between 3290 and 3800 milliseconds in the audio file AS1.wav. Start and end times must be determined for each phoneme using the SAMPA convention as: (/k/, /@/, /p/, /{/, /s/, /I/, /t/, /i/), with the beginning time for /k/ set to 0 ms.

SAMPA which stands for Speech Assessment Methods Phonetic Alphabet, is a phonetic alphabet created to transcribe speech sounds using machine-readable ASCII characters, based on the International Phonetic Alphabet (IPA). It substitutes IPA symbols with ASCII characters, making it appropriate for digital systems. SAMPA accurately transcribes phonemes, not letters, to improve accuracy in language and speech-processing activities. It is tailored for various languages, offering distinct phoneme mappings. Possible paraphrase: Possible uses are speech synthesis, linguistic research, and instructing phonetics.

➔ To apply SAMPA to the word "capacity," we followed these steps:

1. Import AS1.wav file into SFS software.
2. View the waveform of the speech and Select the word “capacity” from 3290 to 3800 milliseconds, then copy it using “[Segment | Copy].”

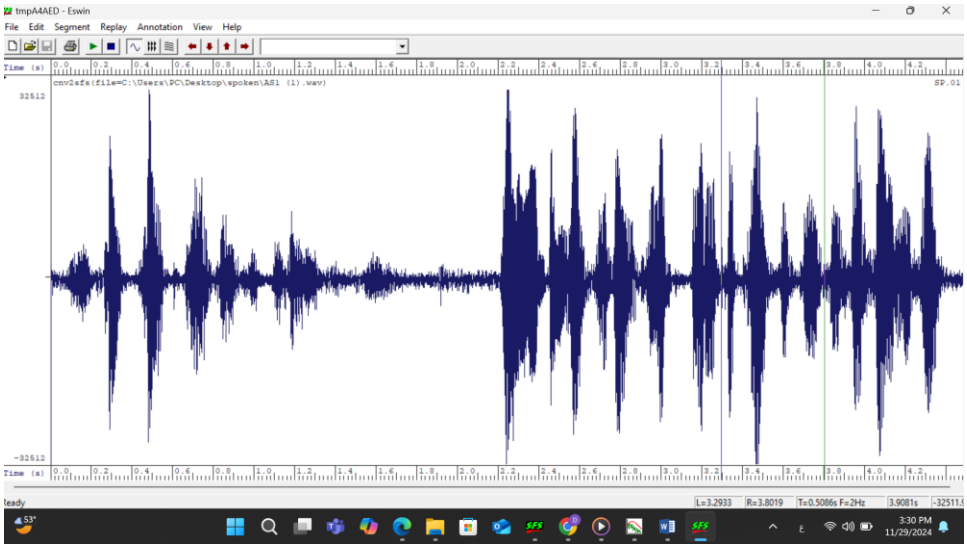


Figure 37: AS1.wav waveform

Next, the extracted waveform corresponding to the word "capacity," as illustrated in the figure below.

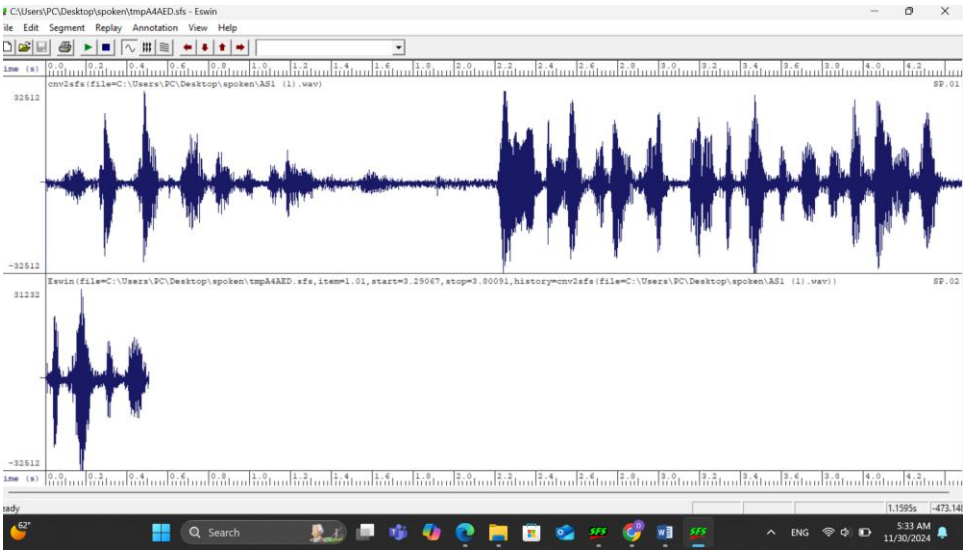


Figure 38: "capacity" waveform

➔ Here is the Narrow - wide Spectrogram for the waveforms:

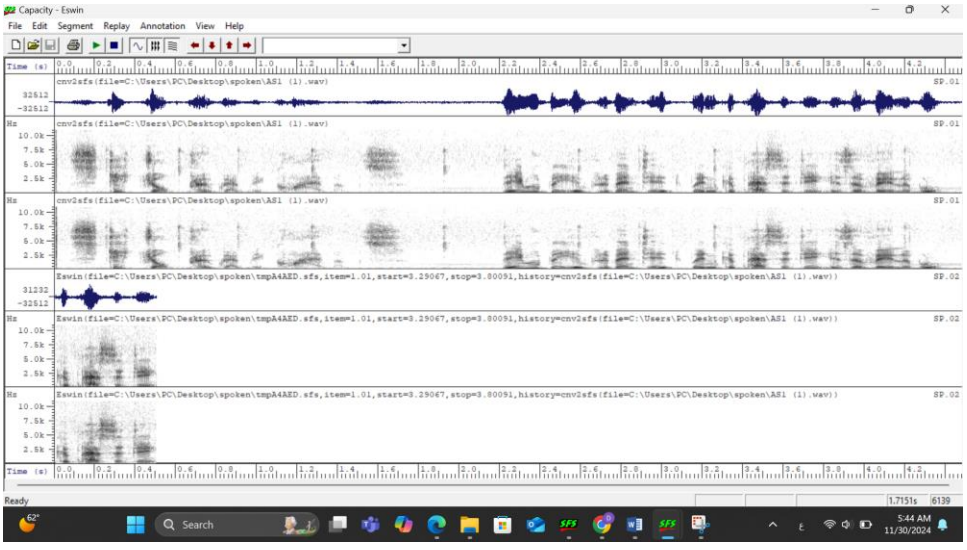


Figure 39: Narrow - wide Spectrogram for AS1 and "Capacity"

"Note: The "capacity" waveform will be from 0 – 510 ms"

3. Identify visually the beginning and ending points of phonemes.
4. Then, Approximate the beginning and ending times with a margin of ±10 ms.

This means: By listening to the audio signal, we utilized the waveform of the term "capacity" to recognize the beginning and ending of every phoneme. The start of a phoneme was identified by the sound heard, while the conclusion was indicated by the shift to the following phoneme.

➔ The phonetic transcription of "capacity" is /k @ p { s I t I/. It consists of the following phonemes:

- /k/ - "k"
- /@/ - "uh"
- /p/ - "p"
- /{/ - "a"
- /s/ - "s"
- /I/ - short "i"
- /t/ - "t"
- /i/ - "ee”

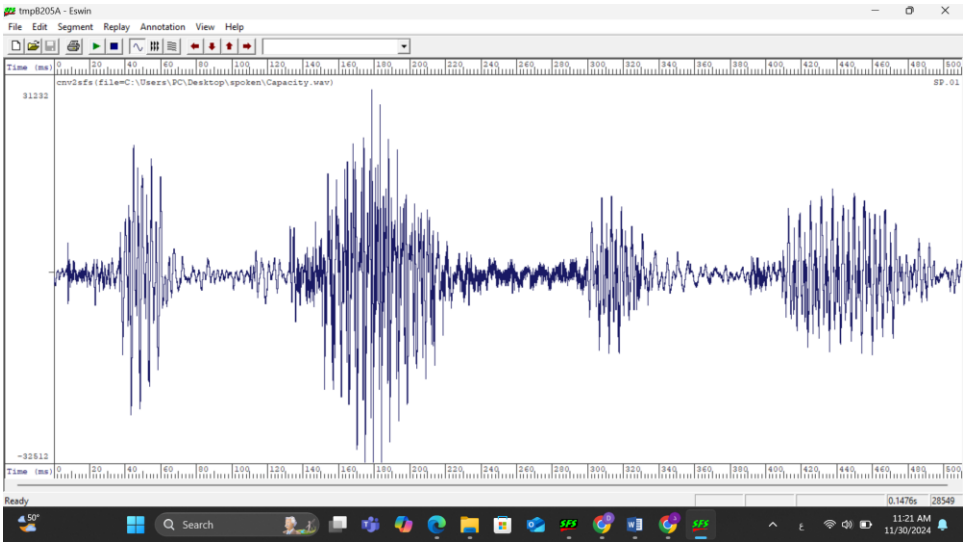


Figure 40: "Capacity" waveform

The Result: After that, the start and end times of each of the eight phonemes in table below:

Phonemes	*Start time(ms)*	*End time(ms)*
/k/ - "k"	**0**	**70**
/@/ - "uh"	**70**	**120**
/p/ - "p"	**120**	**200**
/{/ - "a"	**200**	**280**
/s/ - "s"	**280**	**355.8**
/I/ - short "i"	**360**	**410**
/t/ - "t"	**410**	**475**
/i/ - "ee"	**475**	**510**

It is important to identify phoneme boundaries and timing values, although this can be affected by factors such as background noise, subjectivity in manual marking, coarticulation effects, speaker differences, and limitations of tools. Excessive background noise and distortion may mask clear changes between phonemes. Manual marking depends on individual assessments based on what is heard and seen, resulting in potential bias and inaccuracies. Coarticulation has the potential to influence the beginning and ending points of phonemes. Differences in speakers and limitations of tools may result in slight variations in phoneme characteristics that impact accuracy. Timing values should be seen as close estimations with a degree of uncertainty, rather than exact. Future research could see improvement with automated tools which could reduce mistakes and enhance accuracy when identifying phoneme boundaries and determining timing values.

Note: The project directory holds a file titled "capacity.sfs." You are able to examine the waveform, Spectrogram, and additional information in this file.

Part 5: Sub-word level concatenation

This part includes capturing the sound "ah" in the words "life" and "night" by saying "Say again" at a 16 kHz sampling frequency. The isolated recordings are used to extract sections and combine them to form a new word "light" (/laIt/) through concatenative synthesis, seamlessly merging parts of "life" and "night". Precise alignment and editing are essential to prevent errors in the fused audio, blending parts of various speech signals to create a new word.

➔ To achieve this, we followed these steps:

1. The expressions "Say life again" and "say night again" were captured at a frequency of 16 kHz, along with the single vowel sound "ah" (/a/) for potential later use. And save in files called: “life.wav” and “ night.wav”

2. The terms "life" and "night" were taken from the initial recordings and save in files named "Life_word.wav" and "night_word.wav". Then, Parts of the recordings were copy to create the term "light."

3. We cut the segment "Li" from the "life" recording and the segment "it" from the "night" recording, and saved them as separate files named “Li.wav” and “ight.wav”

4. The segments "Li.wav" and "light.wav" were combined using MATLAB to generate a new segment, as shown in the provided code snippet

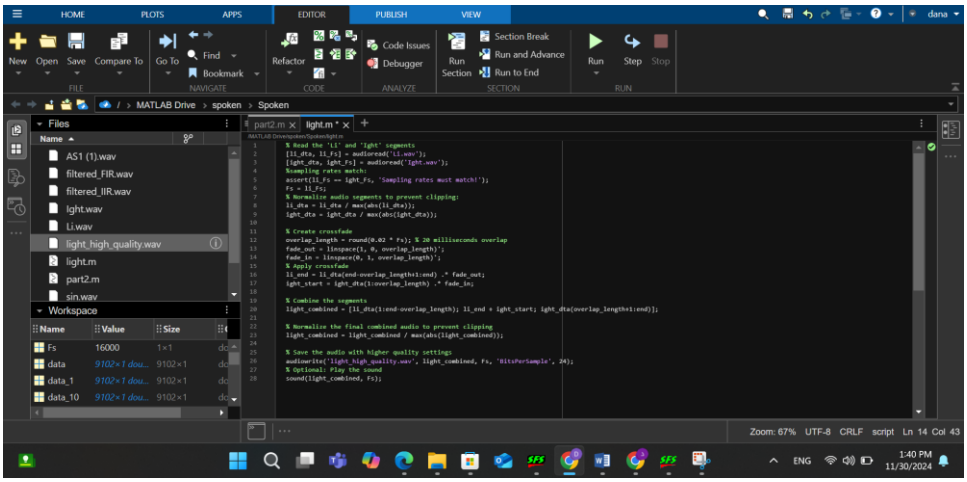


Figure 41: Matlab code for combined signal

➔ The program processes two audio parts ('Li' and 'Ight'), creates a gradual transition between them for a seamless blending effect, and exports the merged audio as a WAV file with adjusted volume to avoid any distortion. It guarantees that the audio sections have consistent sampling rates and employs a 20-millisecond overlap to generate a smooth transition between the two segments.

➔ **The result** of this code will be the new word "Light", which is saved in a file named "light.wav."

Note: “The clarity of the sound from the file "light.wav" was insufficient after being downloaded from MATLAB. This may result from compression or conversion artifacts that occurred while exporting the file. In order to present the outcome in a clearer way, there is a video recording named "record light" that displays the result more precisely.”

Part 6: Phone-level concatenation

This part involves forming a new term by combining phonemes from "capacity" (/k/, /ə/, /p/, /æ/, /s/, /ɪ/, /t/, /i/). Select a target word with a phonetic transcription tool, obtain its phonemes , store them as separate `.wav` files, and combine them. Present the compiled file and assess i ts quality, identifying any artifacts and proposing enhancements.

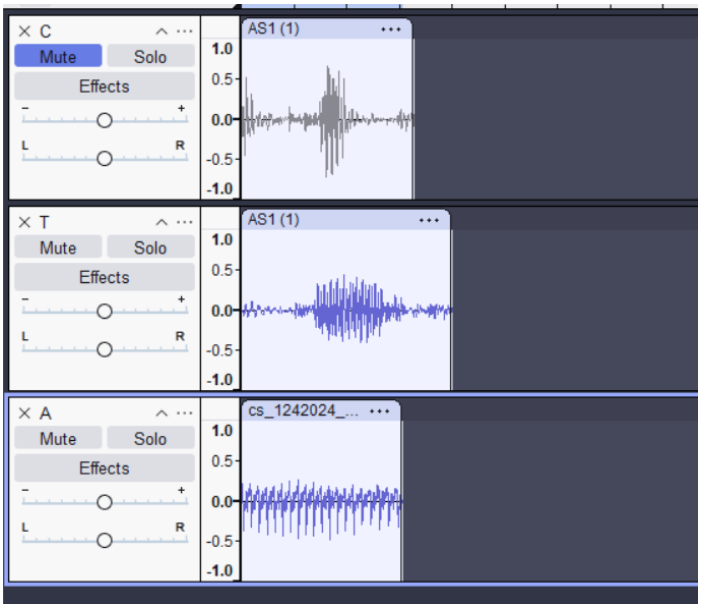


Figure 42: Waveform of phoneme segments /C/, /T/, and /A/ for speech synthesis.

➔ To create the word "cat," we began by extracting the necessary phonemes. The phonemes **/c/** and **/t** were sourced from the word "capacity," while **/æ/** was extracted from the word "hand." Using an audio editing tool, we imported the phonemes and arranged them sequentially: **/c/** → **/æ/** → **/t/**. Careful adjustments were made to the timing between the phonemes to ensure smooth transitions, was applied as needed to eliminate clicks or unnatural gaps. Once the arrangement was complete, the new word "cat" was exported as a .wav file. The figure above shows the phonemes being cut C and T from capacity, and the last sound was cut from other word ‘hand’ shown in figure below.

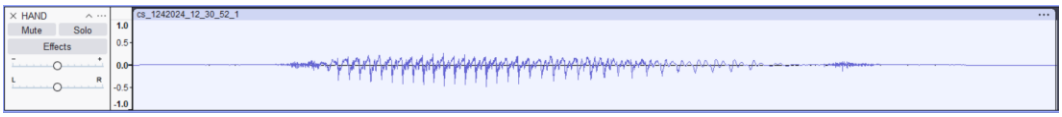


Figure 43: 'hand.wav' sound

→ The figure below shows the last result for the word cat merged from separated phonemes. We have experienced some trouble doing this phase due to the extremely short duration of each phoneme. So, the final result was not as clear as wanted. To improve the synthesized word "cat," adjust timing, apply cross-fades, match pitch and intonation, and normalize volume for smooth transitions and natural flow. We can also use noise reduction and equalization for clarity, and export in high-quality .wav format.

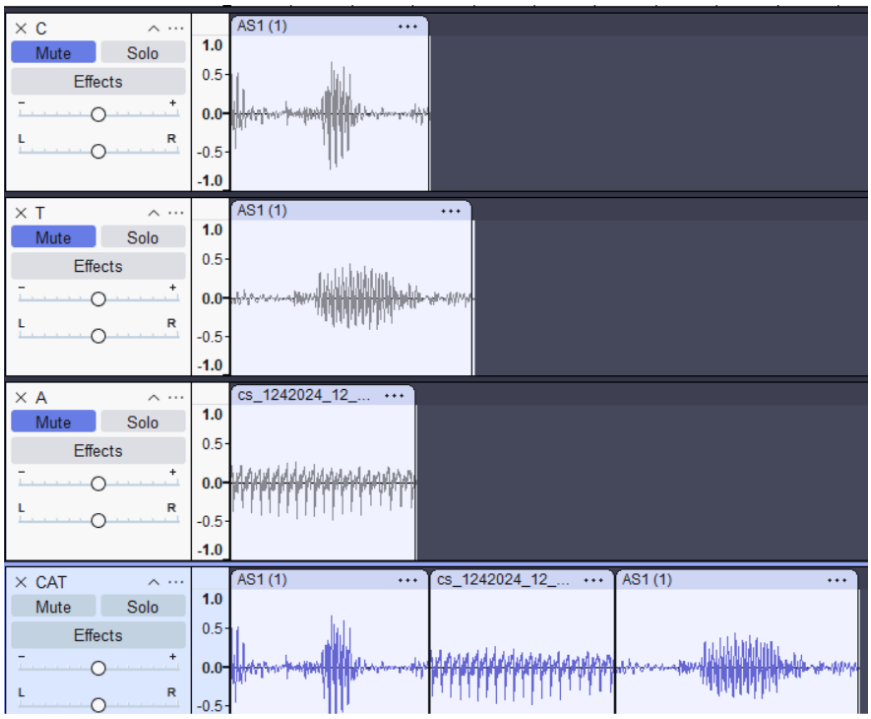


Figure 44: 'Cat' creation

Part 7 – Generation of vowel sounds using the source-filter model:

In this part, we focus on generating vowel sounds using the source-filter model with a parallel formant synthesizer. The source signal, a periodic pulse-train at 110 Hz and 8000 Hz sampling, mimics vocal cord vibrations during voiced sounds. For the vowels (/A:/ in “hard,” /u:/ in “who'd,” and /ə/ in “heard”), formant frequencies (F1 and F2) define the vocal-tract filter. Band-pass filters are created with edges at F-200 Hz and F+200 Hz. The source signal is separately processed through these filters, summed in MATLAB, and exported as WAV files. We can also use white noise for simulating whispered speech.

→ Steps for Generating the Vowel /A:/ ("hard")

- First we generated the Source Signal and selected Pulse-train as the signal type and Set the pulse frequency to 110 Hz and sampling frequency to 8000 Hz. And then generated a periodic pulse-train signal representing:

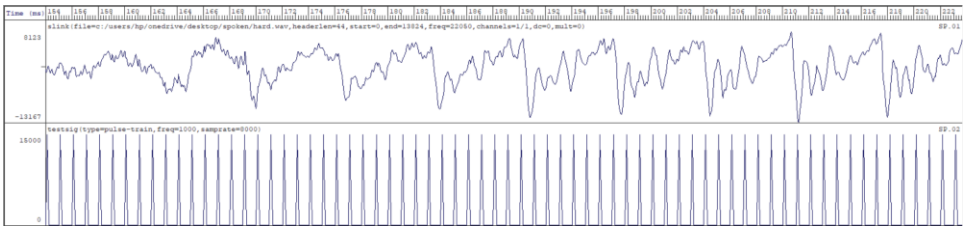


Figure 45: 'hard' word

  - ⇨ The formant values for the vowel /A:/ are:
    - F1 = 700 Hz
    - F2 = 1100 Hz.

- Then we Applied band-pass filtering for F1 (700 Hz). Set the frequency range to 500 Hz (F1 - 200 Hz) to 900 Hz (F1 + 200 Hz). And exported the filtered signal as a .wav file.

- Then the filtering process was repeated for F2 (1100 Hz). Set the frequency range to 900 Hz (F2 - 200 Hz) to 1300 Hz (F2 + 200 Hz) and exported the filtered signal as a .wav file.

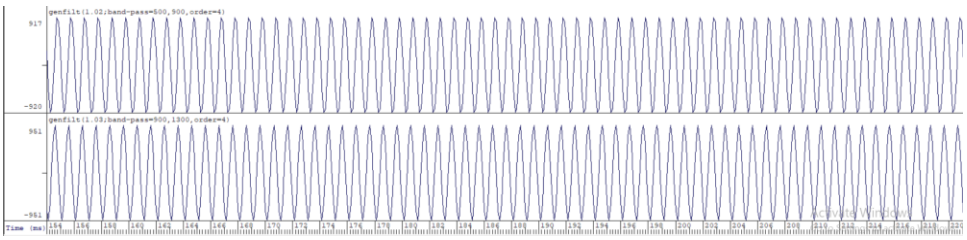


Figure 46: band-pass filter

- Finally, we combined the Filtered Signals on Google Colab. Imported both .wav files (F1 and F2 filtered signals) into Google Colab using the audioread function and combined the signals sample-wise using the formula:

  y(n) = sig1(n) + sig2(n) (where sig1 is the F1 signal and sig2 is the F2 signal).

- Then we exported the combined signal as a .wav file using the audiowrite function.

```
[1]  from scipy.io.wavfile import read, write
     import numpy as np

     # Load the two filtered signals
     sample_rate1, sig1 = read('F11.wav')
     sample_rate2, sig2 = read('F12.wav')

     # Ensure both signals have the same sample rate
     assert sample_rate1 == sample_rate2, "Sample rates must match!"
```

```
     # Ensure both signals are of the same length
     min_length = min(len(sig1), len(sig2))
     sig1 = sig1[:min_length]
     sig2 = sig2[:min_length]

     # Combine the two signals
     combined_signal = sig1 + sig2
```

```
[3]  # Normalize the combined signal
     max_val = np.max(np.abs(combined_signal))
     combined_signal = (combined_signal / max_val) * 32767  # For 16-bit audio
     combined_signal = combined_signal.astype(np.int16)
```

```
[4]  # Save the combined signal to a new WAV file
     write('combined_signal.wav', sample_rate1, combined_signal)
     print("Combined signal saved as 'combined_signal.wav'")
```

```
Combined signal saved as 'combined_signal.wav'
```

```
     from IPython.display import Audio

     # Play the audio
     Audio('combined_signal.wav', rate=sample_rate1)
```


Figure 47: Python Code using colab

The process was repeated for the other two vowels.

→ /u:/ as in “who'd”:

- The formant values for the vowel /u:/ are:
  - F1 = 300 Hz
  - F2 = 800 Hz.

- Then we Applied band-pass filtering for F1 (300 Hz). Set the frequency range to 100 Hz (F1 - 200Hz) to 500 Hz (F1 + 200 Hz). And exported the filtered signal as a .wav file.

- Then the filtering process was repeated for F2 (1100 Hz). Set the frequency range to 600 Hz (F2 - 200 Hz) to 1000 Hz (F2 + 200 Hz) and exported the filtered signal as a .wav file.

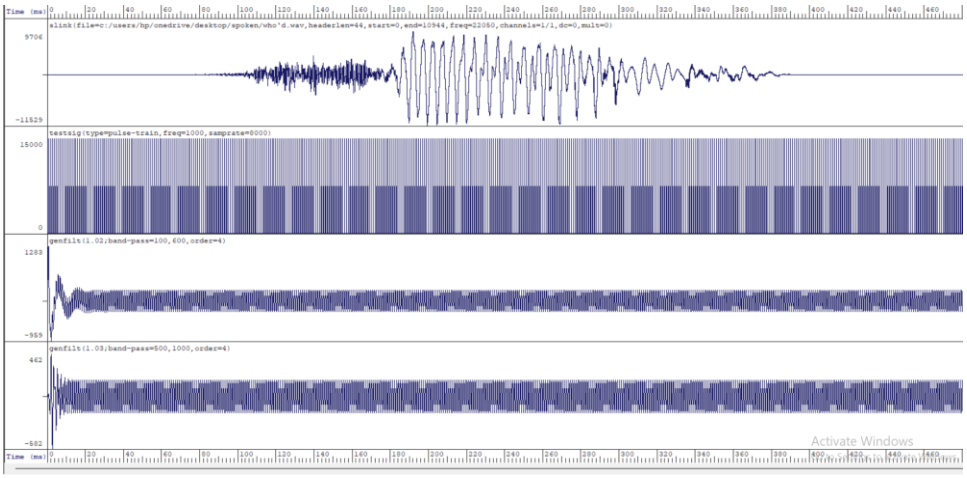


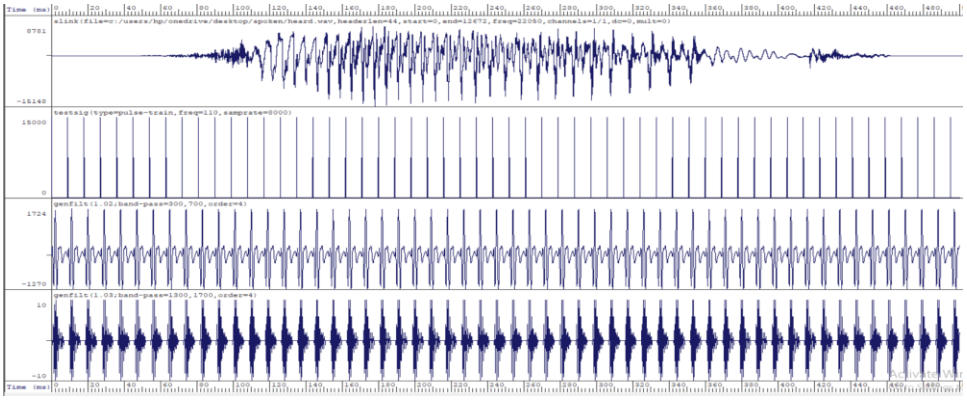
Figure 48: band-pass filter

→ /ə/ as in “heard”:

- The formant values for the vowel / ə:/ are:
  - F1 = 500 Hz
  - F2 = 1500 Hz.

- Then we Applied band-pass filtering for F1 (500 Hz). Set the frequency range to 300 Hz (F1 - 200 Hz) to 700 Hz (F1 + 200 Hz). And exported the filtered signal as a .wav file.

- Then the filtering process was repeated for F2 (1500 Hz). Set the frequency range to 1300 Hz (F2 - 200 Hz) to 1700 Hz (F2 + 200 Hz) and exported the filtered signal as a .wav file.

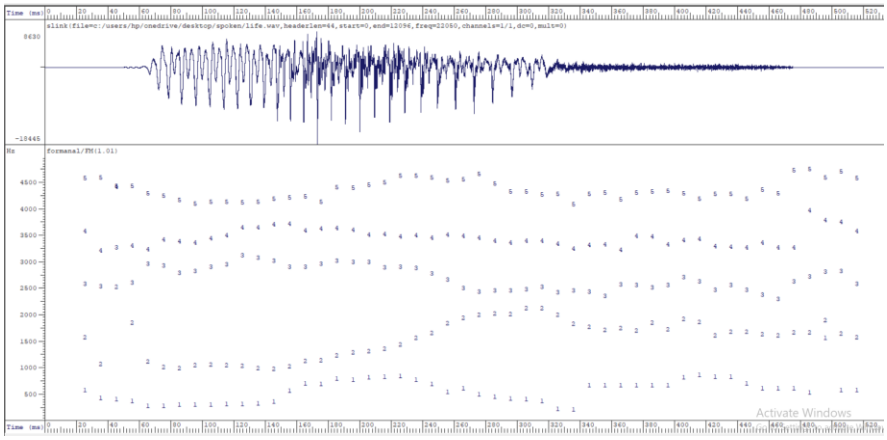


The audio for the 3 vowel sounds did not resemble the vowel sounds as needed. Reasons for this lack of clarity is the accuracy of the formant frequencies used to design the vocal-traction, the sampling frequency which could have limited the resolution or bandwidth.

Part 8: Formant synthesis in SFS

In Part 8, we explore formant synthesis utilizing the Speech Filing System (SFS) with a r ecorded sound of "life" (/laIf/). We conduct formant analysis to obtain essential parameters, which produce synthesizer control data for a synthetic speech signal. The research alters the formant frequencies of the vowel /a/ from "ah" within a diphthong (/aI/) to create new speech. Outcomes feature audio files of both original and altered speech, examined through waveforms and spectrograms, demonstrating the impact of formant modifications on voice traits in synthesis.

→ Formant Analysis for the word “life” (/laIf/)



Formant Data:

formants_life.txt

File Edit View

```
! item=12.01 file=C:\Users\hp\AppData\Local\Temp\tmp1E24A.sfs
! posn size v   gain;   F1   B1 A1;   F2   B2 A2;   F3   B3 A3;   F4   B4 A4;   F5   B5 A5;
!
  24.5 49.0 0    1.0;  500 1000  0; 1500 1000  0; 2500 1000  0; 3500 1000  0; 4500 1000  0;
  34.5 49.0 0    0.0;  355  175  0;  990  573  0; 2455  270  0; 3132  227  0; 4514  864  0;
  44.5 49.0 0    1.2;  321   70 24; 2451  290  6; 3189  349  5; 4340  748  1; 4357  595 14;
  54.5 49.0 0    5.3;  296   52 39; 1775  913 10; 2532  567 16; 3233  359 19; 4357  595 14;
  64.5 49.0 0   42.2;  204   68 61; 1041  799 31; 2884  332 44; 3159  282 46; 4216  854 27;
  74.5 49.0 0  150.6;  199   35 77;  929  352 47; 2844  163 55; 3339  145 57; 4172  191 48;
  84.5 49.0 0  185.5;  227   40 76;  910  386 48; 2716  143 55; 3306  115 60; 4088  204 52;
  94.5 49.0 0  195.5;  221   25 79;  965  218 51; 2748  145 55; 3281  157 58; 4022  267 50;
 104.5 49.0 0  206.5;  226   23 80;  975  115 57; 2824  149 55; 3368  167 58; 4057  168 55;
 114.5 49.0 0  222.8;  235   29 80;  968  125 58; 2877  247 52; 3426  193 59; 4048  181 56;
 124.5 49.0 0  225.0;  239   49 77;  958  201 56; 3042  321 52; 3570  337 57; 4045  348 55;
 134.5 49.0 0  219.4;  243   66 74;  916  176 57; 3004  461 48; 3577  335 57; 4051  258 57;
 144.5 49.0 0  246.0;  276  113 72;  898  138 61; 2943  511 45; 3628  397 55; 4108  130 64;
 154.5 49.0 0  257.6;  482  251 66;  949  102 66; 2828   96 56; 3646  527 50; 4134  148 60;
 164.5 49.0 0  287.7;  623  238 65; 1045   66 70; 2826  201 52; 3510  199 57; 4158  437 50;
 174.5 49.0 0  343.7;  609  378 63; 1058   76 71; 2885  148 59; 3544  293 58; 4052  737 50;
 184.5 49.0 0  246.2;  714  373 58; 1157  192 59; 2936  119 57; 3559  167 57; 4325  329 48;
 194.5 49.0 0  257.5;  699  359 57; 1209   79 64; 2914  115 58; 3524  303 53; 4321  299 51;
 204.5 49.0 0  282.1;  737  534 56; 1235   67 68; 2920  189 56; 3436  325 53; 4379  242 51;
 214.5 49.0 0  262.6;  751  568 55; 1275   88 66; 2815  296 51; 3442  397 50; 4424  449 46;
 224.5 49.0 0  288.3;  764  461 56; 1362   82 66; 2830  199 57; 3396  438 51; 4551  497 43;
 234.5 49.0 0  225.8;  699  552 53; 1480  169 57; 2804  183 55; 3419  380 48; 4551  497 43;
 244.5 49.0 0  207.8;  609  518 52; 1577  113 59; 2701  192 53; 3381  368 47; 4514  276 46;
 254.5 49.0 0  173.0;  465  771 49; 1764  211 53; 2587  259 51; 3432  294 46; 4458  172 48;
 264.5 49.0 0  195.0;  531  328 52; 1868  131 57; 2430  214 53; 3412  238 48; 4483  145 52;
 274.5 49.0 0  195.9;  416  299 55; 1920  137 59; 2353  225 56; 3379  420 44; 4580  380 47;
 284.5 49.0 0  150.1;  374  308 53; 1938  246 51; 2374  264 51; 3318  212 49; 4393  545 42;
 294.5 49.0 0  109.9;  336  209 55; 1926  298 46; 2376  256 48; 3290  186 47; 4248  564 38;
```

Figure 49: data format

Synthesizer Control Parameters:

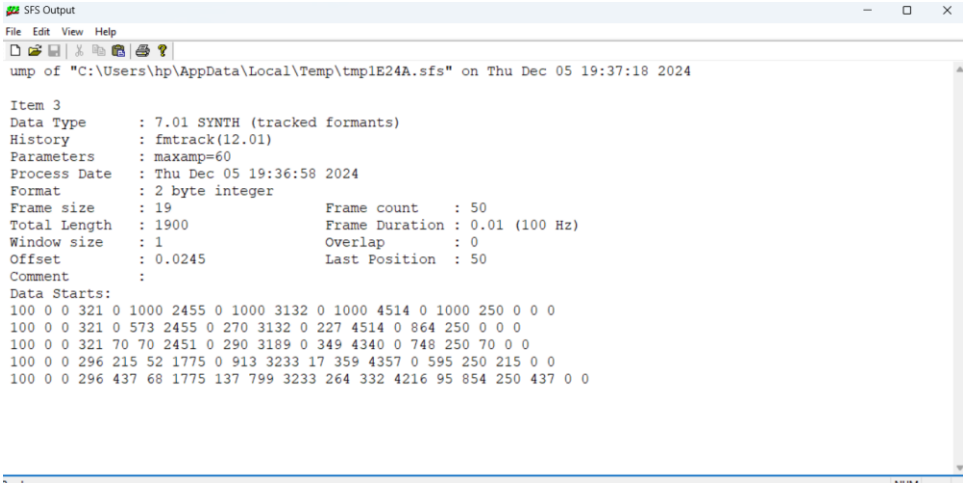


Figure 50: Synthesizer Control Parameters:

→ Then we modified Formant Data for the Vowel /a/

- The formant values for /a/ (vowel "ah"): F1 = 700 Hz, F2 = 1100 Hz. And changed the values in the txt file for F1, and F2 as the vowel /a/.

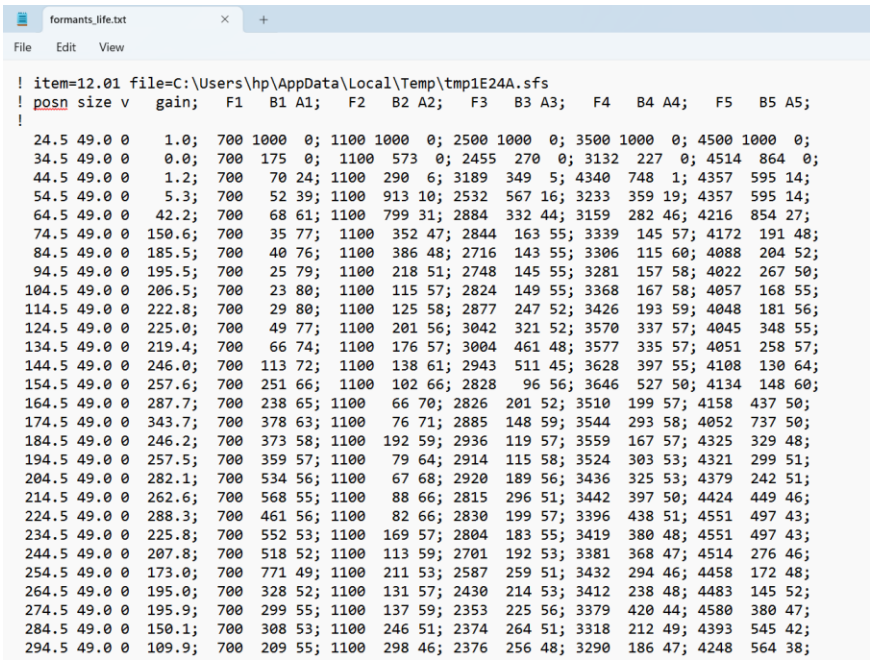


Figure 51: new data format

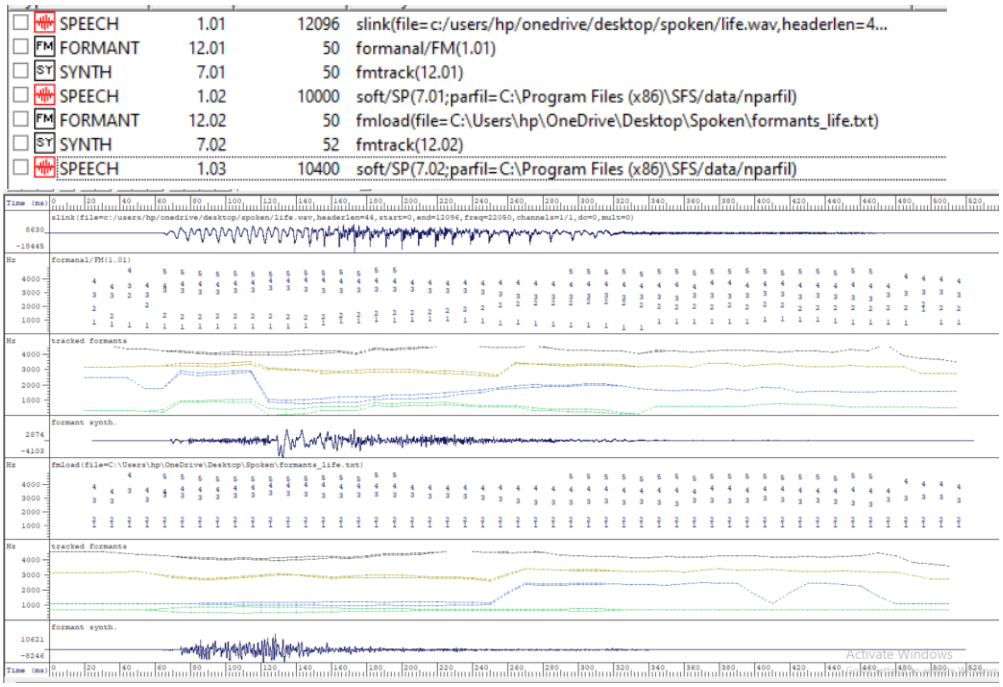


Figure 52: Formant

We synthesized speech using formant data from the word "life" (/laɪf/) to explore the impact of formant modifications on speech perception. The original synthesized speech maintained clarity and intonation, closely resembling the original phonetic features. However, when we modified the vowel /a/ in the diphthong /aɪ/ by applying formant data from the vowel "Ah," the result sounded more like a stretched /a/ rather than "life."

Our research showed clear variations in spectral properties, especially in the F1 and F2 formants, which are key to identifying vowels. The modified speech sounded less intelligible, demonstrating the importance of formants in speech clarity. We attempted to integrate the "Ah" sound into "life" by changing its formant structure, but due to issues with audio file cropping, the result was closer to "laugh" than "life." Despite these challenges, we focused on accurately adjusting the F1 and F2 formants within the diphthong, based on detailed waveform analysis.

Part8. B.

In a formant-based speech synthesizer, changing a speaker's voice can be achieved through several procedures:

- **Formant Frequency Adjustment:** Each speaker's unique vocal tract shape determines their distinct formant pattern. By modifying the formant frequencies in the synthesizer settings, various vocal qualities can be simulated.
- **Pitch and Duration:** Adjusting the pitch and duration helps match the speech rhythm and pitch of the new speaker, including modifying phoneme delivery times and fundamental frequencies.
- **Timbre and Intonation:** Fine-tuning the harmonic-to-noise ratio and other tonal characteristics can alter the voice's warmth or brightness, allowing for subtle changes in timbre.
- **Dynamic Range:** Modifying the amplitude envelope throughout a speech sequence helps replicate the vocal dynamics of the new speaker.

These adjustments make formant synthesis a powerful tool for voice morphing and character development in applications such as virtual assistants, animation, and telecommunications, enabling the synthesizer to convincingly mimic the voice of another individual.

Appendix

Matlab Code1, Part2:

```
% Load the sinusoidal and white noise signals
[sinusoidal_signal, Fs] = audioread('sinusoidal_signal.wav');
[white_noise, Fs] = audioread('Wnoise_signal.wav');
% Combine the sinusoidal signal and the white noise signal
combined_signal = sinusoidal_signal + white_noise;
% Play the combined signal
sound(combined_signal, Fs);
audiowrite('combined_signal.wav', combined_signal, Fs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% IIR filter parameters:
b = [1]; % Numerator (x(n) coefficient)
a = [1, -0.99]; % Denominator (y(n-1) coefficient)
% Apply the IIR filter to the combined signal
iir_output = filter(b, a, combined_signal);
sound(iir_output, Fs);
audiowrite('iir_filtered_signal.wav', iir_output, Fs);

% Magnitude Frequency Response for IIR Filter
% Calculate frequency response
NFFT = 1024;
[H_iir, f_iir] = freqz(b, a, NFFT, Fs);
magnitude_iir = abs(H_iir);
magnitude_iir_db = 20 * log10(magnitude_iir);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FIR filter coefficients
fir_coeffs = 0.2 * ones(1, 5); % Coefficients b0 to b4
% Apply the FIR filter to the combined signal
fir_output = filter(fir_coeffs, 1, combined_signal);
sound(fir_output, Fs);
audiowrite('fir_filtered_signal.wav', fir_output, Fs);

% Magnitude Frequency Response for FIR Filter
[H_fir, f_fir] = freqz(fir_coeffs, 1, NFFT, Fs);
magnitude_fir = abs(H_fir);
magnitude_fir_db = 20 * log10(magnitude_fir);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plotting Magnitude Frequency Responses
% IIR Filter Magnitude Response
subplot(2,1,1);
plot(f_iir, magnitude_iir_db, 'b');  % Blue color for IIR filter
title('Magnitude Frequency Response - IIR Filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;
```

```
% FIR Filter Magnitude Response
subplot(2,1,2);
plot(f_fir, magnitude_fir_db, 'g');  % Green color for FIR filter
title('Magnitude Frequency Response - FIR Filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;


% Time Domain Comparison
figure;
t = (0:length(combined_signal)-1) / Fs;
subplot(3,1,1);
plot(t, combined_signal);
title('Original Combined Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,2);
plot(t, iir_output);
title('IIR Filtered Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(3,1,3);
plot(t, fir_output);
title('FIR Filtered Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```

Part 5/ Light code

```
% Read the 'Li' and 'Ight' segments
[li_dta, li_Fs] = audioread('Li.wav');
[ight_dta, ight_Fs] = audioread('Ight.wav');
%sampling rates match:
assert(li_Fs == ight_Fs, 'Sampling rates must match!');
Fs = li_Fs;
% Normalize audio segments to prevent clipping:
li_dta = li_dta / max(abs(li_dta));
ight_dta = ight_dta / max(abs(ight_dta));

% Create crossfade
overlap_length = round(0.02 * Fs); % 20 milliseconds overlap
fade_out = linspace(1, 0, overlap_length)';
fade_in = linspace(0, 1, overlap_length)';
% Apply crossfade
li_end = li_dta(end-overlap_length+1:end) .* fade_out;
ight_start = ight_dta(1:overlap_length) .* fade_in;
```

```
% Combine the segments
light_combined = [li_dta(1:end-overlap_length); li_end + ight_start;
ight_dta(overlap_length+1:end)];

% Normalize the final combined audio to prevent clipping
light_combined = light_combined / max(abs(light_combined));

% Save the audio with higher quality settings
audiowrite('light_high_quality.wav', light_combined, Fs, 'BitsPerSample', 24);
% Optional: Play the sound
sound(light_combined, Fs);
```

---

https://colab.research.google.com/drive/18wrexxoOokSdVupIewoYOw5Qota4qncs?usp=sharing&fbclid=IwY2xjawG-4r9leHRuA2FlbQIxMAABHWxmw53se4kdA-AE71ULNej8xyP7gYhun3PdSh-zKZT_EWdCaou_3o9C1A_aem_OR2gwR7a8rSfSrlGOcEz9Q

Colab Link, U can find codes for part 7 & part 3

Python code, Part 3. 1 Frequency for vowels for sample1.wav

```
import matplotlib.pyplot as plt

# Data: F1 and F2 - F1 for each vowel
vowels = ['/i:/', '/{/ ', '/u:/', '/A:/']
F1 = [398, 448, 395, 637]
F2_minus_F1 = [1909, 390, 1021, 868]

# Create the plot
plt.scatter(F1, F2_minus_F1)

# Label the points with the vowel names
for i, vowel in enumerate(vowels):
    plt.text(F1[i], F2_minus_F1[i], vowel, fontsize=12, ha='right')

# Label the axes
plt.xlabel('F1 (Hz)')
plt.ylabel('F2 - F1 (Hz)')
plt.title('F1 vs. (F2 - F1) for Vowel Sounds')

# Show the plot
plt.show()
```

Formant plot for sample 1 and student 1

```
import matplotlib.pyplot as plt

# Formant data for Sample 1
sample_vowels = ['i:', '{', 'u:', 'A:']
sample_f1 = [398, 448, 395, 637]
sample_f2 = [2307, 838, 1416, 1505]
sample_f3 = [2798, 2622, 2536, 1993]
sample_f2_minus_f1 = [f2 - f1 for f1, f2 in zip(sample_f1, sample_f2)]

# Formant data for Student 1
student_f1 = [321, 521, 533, 484]
student_f2 = [1879, 1458, 1961, 1297]
student_f3 = [3438, 3481, 3145, 2282]
student_f2_minus_f1 = [f2 - f1 for f1, f2 in zip(student_f1, student_f2)]

# Plotting
plt.figure(figsize=(8, 6))

# Plot Sample 1 data
plt.scatter(sample_f1, sample_f2_minus_f1, color='blue', label='Sample 1',
marker='o')

# Plot Student 1 data
plt.scatter(student_f1, student_f2_minus_f1, color='red', label='Student
1', marker='x')

# Adding labels and title
plt.title('Formant Plot: F1 vs (F2 - F1)', fontsize=14)
plt.xlabel('F1 (Hz)', fontsize=12)
plt.ylabel('F2 - F1 (Hz)', fontsize=12)

# Adding a legend
plt.legend()

# Display the plot
plt.grid(True)
plt.show()
```

Formant plot for student 1

```
import matplotlib.pyplot as plt

# Formant data for Student 1
student_vowels = ['i:', '{', 'u:', 'A:']
student_f1 = [321, 521, 533, 484]
student_f2 = [1879, 1458, 1961, 1297]
student_f3 = [3438, 3481, 3145, 2282]
student_f2_minus_f1 = [f2 - f1 for f1, f2 in zip(student_f1, student_f2)]

# Plotting
plt.figure(figsize=(8, 6))

# Plot Student 1 data
plt.scatter(student_f1, student_f2_minus_f1, color='red', label='Student
1', marker='x')

# Adding labels and title
plt.title('Formant Plot: F1 vs (F2 - F1) for Student 1', fontsize=14)
plt.xlabel('F1 (Hz)', fontsize=12)
plt.ylabel('F2 - F1 (Hz)', fontsize=12)

# Adding grid for easier readability
plt.grid(True)

# Annotating the points with vowel names
for i, vowel in enumerate(student_vowels):
    plt.annotate(vowel, (student_f1[i], student_f2_minus_f1[i]),
textcoords="offset points", xytext=(0, 10), ha='center')

# Show the plot
plt.legend()
plt.show()
```

Formant plot for sample 1 and st1 & st2

```
import matplotlib.pyplot as plt

# Formant data for Student 1
student_1_vowels = ['i:', '{', 'u:', 'A:']
student_1_f1 = [321, 521, 533, 484]
student_1_f2 = [1879, 1458, 1961, 1297]
student_1_f3 = [3438, 3481, 3145, 2282]
student_1_f2_minus_f1 = [f2 - f1 for f1, f2 in zip(student_1_f1,
student_1_f2)]

# Formant data for Student 2
student_2_vowels = ['i:', '{', 'u:', 'A:']
student_2_f1 = [355, 475, 420, 600]
student_2_f2 = [2050, 1220, 1550, 1400]
student_2_f3 = [2750, 2800, 2700, 2050]
student_2_f2_minus_f1 = [f2 - f1 for f1, f2 in zip(student_2_f1,
student_2_f2)]

# Formant data for Sample 1
sample_1_f1 = [398, 448, 395, 637]
sample_1_f2 = [2307, 838, 1416, 1505]
sample_1_f3 = [2798, 2622, 2536, 1993]
sample_1_f2_minus_f1 = [f2 - f1 for f1, f2 in zip(sample_1_f1,
sample_1_f2)]

# Plotting
plt.figure(figsize=(10, 8))

# Plot Student 1 data
plt.scatter(student_1_f1, student_1_f2_minus_f1, color='blue',
label='Student 1', marker='o')

# Plot Student 2 data
plt.scatter(student_2_f1, student_2_f2_minus_f1, color='green',
label='Student 2', marker='x')

# Plot Sample 1 data
plt.scatter(sample_1_f1, sample_1_f2_minus_f1, color='red', label='Sample
1', marker='s')

# Adding labels and title
```

```
plt.title('Formant Plot: F1 vs (F2 - F1)', fontsize=14)
plt.xlabel('F1 (Hz)', fontsize=12)
plt.ylabel('F2 - F1 (Hz)', fontsize=12)

# Adding grid for readability
plt.grid(True)

# Annotate the points with vowel names
for i, vowel in enumerate(student_1_vowels):
    plt.annotate(vowel, (student_1_f1[i], student_1_f2_minus_f1[i]),
textcoords="offset points", xytext=(0, 10), ha='center')

for i, vowel in enumerate(student_2_vowels):
    plt.annotate(vowel, (student_2_f1[i], student_2_f2_minus_f1[i]),
textcoords="offset points", xytext=(0, 10), ha='center')

for i, vowel in enumerate(student_1_vowels):
    plt.annotate(vowel, (sample_1_f1[i], sample_1_f2_minus_f1[i]),
textcoords="offset points", xytext=(0, 10), ha='center')

# Show the plot
plt.legend()
plt.show()
```

Formant plot student 2

```
import matplotlib.pyplot as plt

# Formant data for Student 2
student_2_vowels = ['i:', '{', 'u:', 'A:']
student_2_f1 = [355, 475, 420, 600]
student_2_f2 = [2050, 1220, 1550, 1400]
student_2_f3 = [2750, 2800, 2700, 2050]
student_2_f2_minus_f1 = [f2 - f1 for f1, f2 in zip(student_2_f1,
student_2_f2)]

# Plotting
plt.figure(figsize=(8, 6))

# Plot Student 2 data
plt.scatter(student_2_f1, student_2_f2_minus_f1, color='green',
label='Student 2', marker='o')

# Adding labels and title
plt.title('Formant Plot: F1 vs (F2 - F1) for Student 2', fontsize=14)
plt.xlabel('F1 (Hz)', fontsize=12)
plt.ylabel('F2 - F1 (Hz)', fontsize=12)

# Adding grid for easier readability
plt.grid(True)

# Annotating the points with vowel names
for i, vowel in enumerate(student_2_vowels):
    plt.annotate(vowel, (student_2_f1[i], student_2_f2_minus_f1[i]),
textcoords="offset points", xytext=(0, 10), ha='center')

# Show the plot
plt.legend()
plt.show()
```