Spectral Analysis

Shrey Mehta (200580)

***Abstract*—Given an audio file, use python to plot:**

- **the waveform x(t)**
- **spectrogram —X(f,t)—, as a function of window size (in s), hop length (in s), N, etc.**
- **spectrum —X(f)—, as a function of start time, N, etc.**

We use Python language to take as input the .wav file of the audio signal and output the required plots using the following code:

- Import the required libraries

```
import pyaudio
import wave
import matplotlib.pyplot as plt
import numpy as np
from scipy.fftpack import fft
```

- Process the audio signal using pyaudio

```
audio_file = 'audio_file.wav'
p = pyaudio.PyAudio()
wf = wave.open(audio_file, 'rb')
sample_width = wf.getsampwidth()
sample_rate = wf.getframerate()
audio_data = wf.readframes(-1)
audio_data = np.frombuffer(audio_data, dtype=np.
    int16)
print(sample_rate)
print(len(audio_data))
wf.close()
p.terminate()
```

- Plotting the waveform x(t)

```
duration = len(audio_data) / sample_rate  #
    Calculate audio duration
time = np.linspace(0, duration, num=len(
    audio_data))  # Create time axis
plt.figure(figsize=(10, 4))
plt.plot(time, audio_data)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title("Audio Waveform")
plt.grid()
plt.show()
```

- Plotting the spectrogram X(f,t) for the given waveform

```
# Define the window size and overlap
window_size = 1024  # Adjust this as needed
overlap = int(window_size * 0.5)

# Calculate the number of FFTs
num_ffts = int((len(audio_data) - overlap) / (
    window_size - overlap))

# Create an empty array to store the results
spectrogram = np.empty((num_ffts, window_size //
     2))

# Perform FFT on overlapping segments
for i in range(num_ffts):
    start = i * (window_size - overlap)
    end = start + window_size
    segment = audio_data[start:end]
    fft_result = np.fft.fft(segment)
    frequencies = np.fft.fftfreq(len(fft_result)
    , 1.0 / sample_rate)
    magnitude = 20 * np.log10(np.abs(fft_result
    [:len(fft_result) // 2]))
    spectrogram[i, :] = magnitude

# Create a time axis
time_axis = np.arange(0, num_ffts) * (
    window_size - overlap) / sample_rate

# Create a frequency axis
frequency_axis = frequencies[:window_size // 2]

# Plot the spectrogram
plt.figure(figsize=(10, 4))
plt.pcolormesh(time_axis, frequency_axis,
    spectrogram.T, shading='auto', cmap='viridis
    ')
plt.colorbar(label='Magnitude (dB)')
plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)')
plt.title('Spectrogram (Time vs. Frequency)')
plt.show()
```

- Plotting the spectrum X(t) for the given waveform (using decibel scale)

```
# Perform FFT on the entire audio signal
fft_result = np.fft.fft(audio_data)
frequencies = np.fft.fftfreq(len(fft_result),
    1.0 / sample_rate)

# Take the absolute value and scale to dB
magnitude = 20 * np.log10(np.abs(fft_result[:len
    (fft_result) // 2]))

# Plot the frequency spectrum
plt.figure(figsize=(10, 4))
plt.plot(frequencies[:len(frequencies) // 2],
    magnitude)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.title('Frequency Spectrum (FFT)')
plt.grid()
plt.show()
```

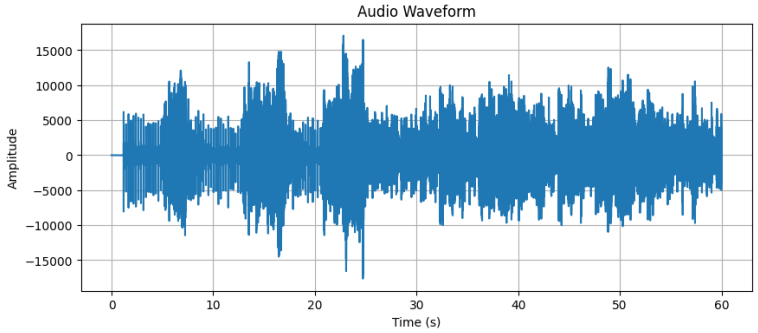
- Plotting the spectrogram for the given waveform (using mel scale)

```
# Create a function to compute Mel filterbanks
def mel_filterbank(num_filters, fft_size,
    sample_rate):
    mel_min = 0
    mel_max = 2595 * np.log10(1 + (sample_rate /
     2) / 700)
    mel_points = np.linspace(mel_min, mel_max,
    num_filters + 2)
    hz_points = 700 * (10**(mel_points / 2595) -
     1)
    bin_points = np.floor((fft_size + 1) *
    hz_points / sample_rate).astype(int)
    filters = np.zeros((num_filters, fft_size //
     2 + 1))
    for i in range(1, num_filters + 1):
        filters[i - 1, bin_points[i - 1]:
    bin_points[i]] = (np.arange(bin_points[i -
    1], bin_points[i]) - bin_points[i - 1]) / (
    bin_points[i] - bin_points[i - 1])
```

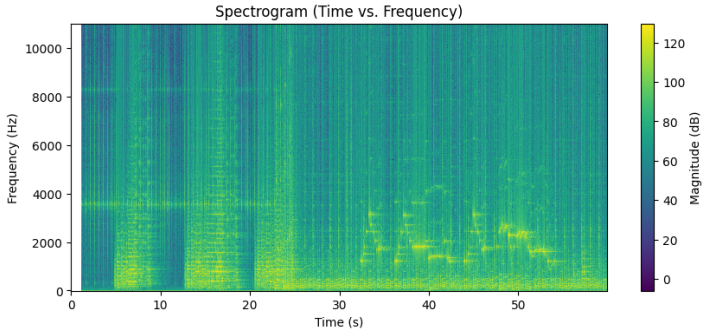
```
        filters[i - 1, bin_points[i]:bin_points[i + 1]] = 1 - (np.arange(bin_points[i], bin_points[i + 1]) - bin_points[i]) / (bin_points[i + 1] - bin_points[i])
    return filters

# Define the parameters for the spectrogram
window_size = 512  # Adjust this as needed
overlap = window_size // 2  # Adjust this as needed
hop_size = window_size - overlap

# Calculate the FFT frequencies
fft_frequencies = np.fft.fftfreq(window_size, 1.0 / sample_rate)

# Create an empty list to store the spectrogram frames
spectrogram_frames = []

# Create the Mel filterbank
num_filters = 40  # Adjust this as needed
mel_filters = mel_filterbank(num_filters, window_size, sample_rate)

# Iterate through the audio data with overlapping windows
for i in range(0, len(audio_data) - window_size, hop_size):
    audio_segment = audio_data[i:i + window_size]
    fft_result = fft(audio_segment)
    mel_spectrum = np.dot(mel_filters, np.abs(fft_result[:window_size // 2 + 1]))
    spectrogram_frames.append(mel_spectrum)

# Convert the list of frames to a numpy array
spectrogram = np.array(spectrogram_frames).T

# Plot the spectrogram
plt.figure(figsize=(10, 4))
plt.imshow(spectrogram, aspect='auto', cmap='viridis', origin='lower')
plt.colorbar(label='Magnitude')
plt.xlabel('Time Frame')
plt.ylabel('Mel Filterbank Index')
plt.title('Mel Spectrogram')
plt.show()
```

Example: We take a sample audio file as an example and plot the corresponding four plots for the given audio signal. The plots obtained are as shown:

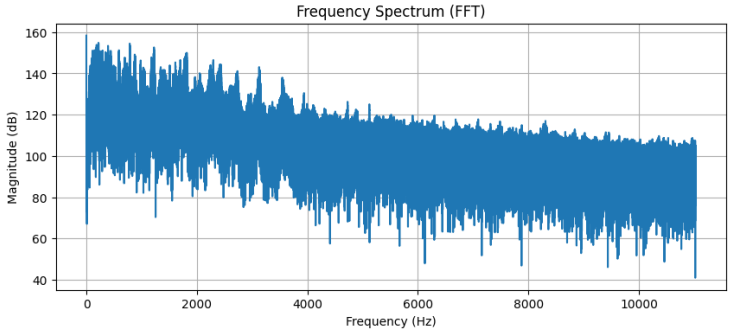
- Waveform



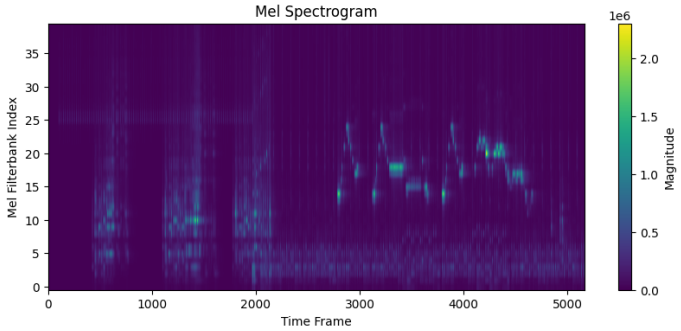
- Spectrogram (using dB scale)



- Spectrum



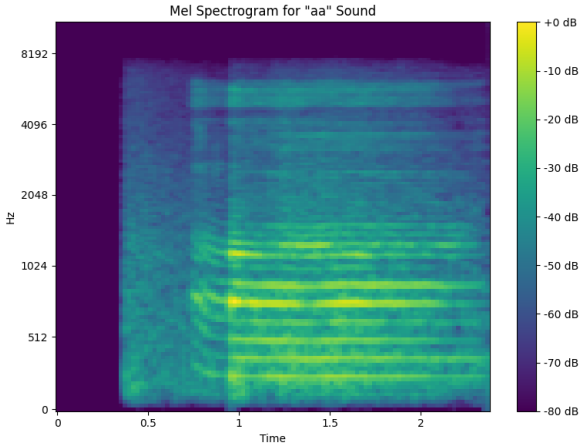
- Spectrogram (using mel scale)



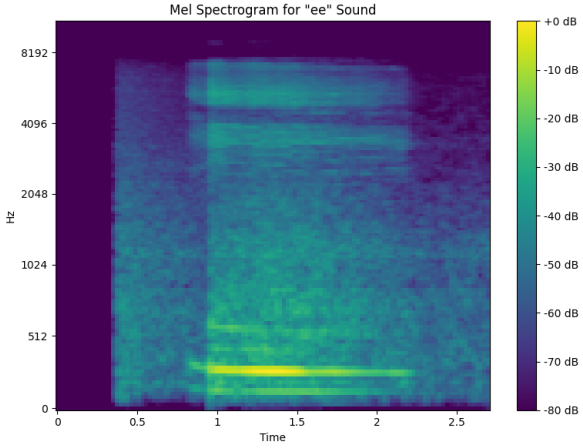
Now, we will analyse these plots for various waveforms and draw conclusions from them.

Analysis of vowels: We will analyse the sounds such as aa, ee, o, u, etc., and study the characteristic features of their spectra/spectrograms. On plotting the spectrograms of these sounds, we obtain the following plots:

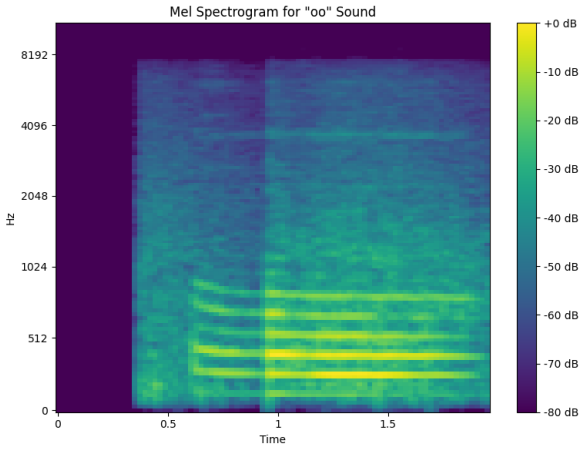
- ’**aa**’ sound



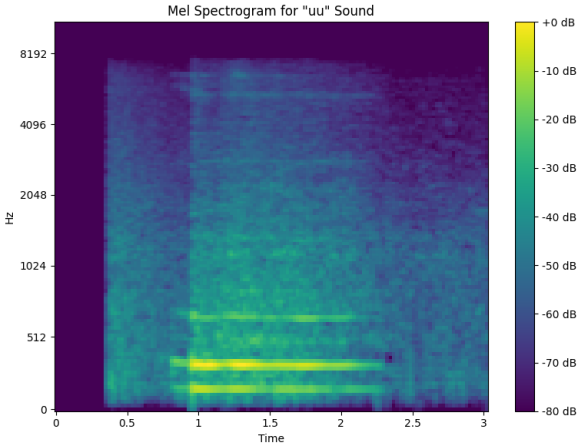
- ’**ee**’ sound



- ’**o**’ sound



- ’**u**’ sound



Observations:

- ’**aa**’ sound is a low, open vowel sound. The spectrogram shows strong fundamental frequency (the lowest frequency component) and several harmonics (integer multiples of the fundamental). The formants (resonant frequencies) are relatively evenly spaced and typically low in frequency.
- ’**ee**’ sound is a high, front vowel sound. The spectrogram shows strong fundamental frequency and harmonics, similar to ”aa” but the formants are higher in frequency. The first formant is relatively high, and the second formant is even higher, making it a front vowel.
- ’**o**’ sound is a mid-to-high, back vowel sound. The spectrogram shows strong fundamental frequency and harmonics, but the formants are shifted towards the lower-mid frequencies. The first formant is relatively low, and the second formant is moderate in frequency, placing it in the back vowel category.
- ’**u**’ sound is a high, back vowel sound. The spectrogram shows strong fundamental frequency and harmonics like the others, but the formants are positioned at relatively high frequencies.The first formant is low, and the second formant is high, making it a high, back vowel.

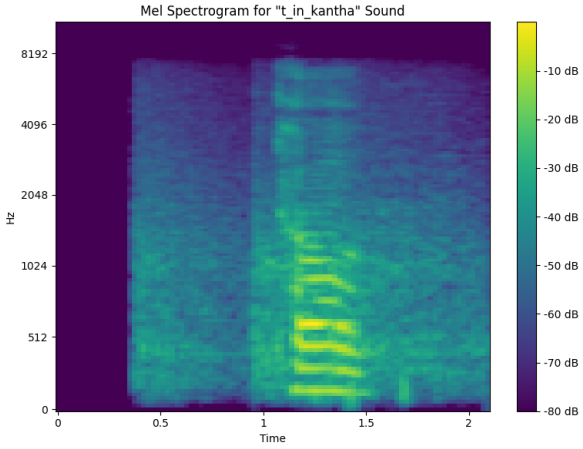
In spectrograms of these vowel sounds:

- ”aa” will show a strong, dark band in the low-frequency range.
- ”ee” will have formants concentrated in the high-frequency range.
- ”o” will show formants in the mid-frequency range.
- ”u” will exhibit formants in the high-frequency range but shifted towards the back.

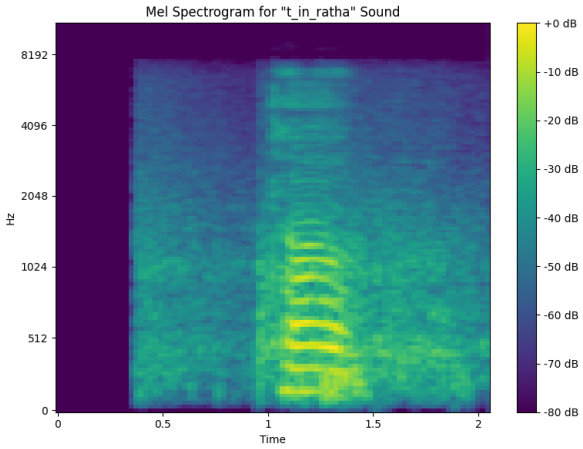
These spectral and spectrographic characteristics are essential in phonetics and speech analysis as they help differentiate between different vowel sounds and contribute to our understanding of speech production and perception.

Analysis of 't' sounds: We will analyse the sounds such as t (as in tiwari), th (as in ratha), t (as in topi), th (as in neelkantha), and study the characteristic features of their spectra/spectrograms. On plotting the spectrograms of these sounds, we obtain the following plots:

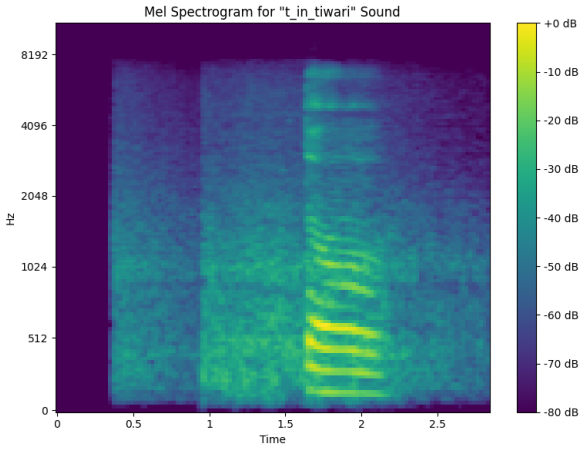
- '**t (as in tiwari)**' sound



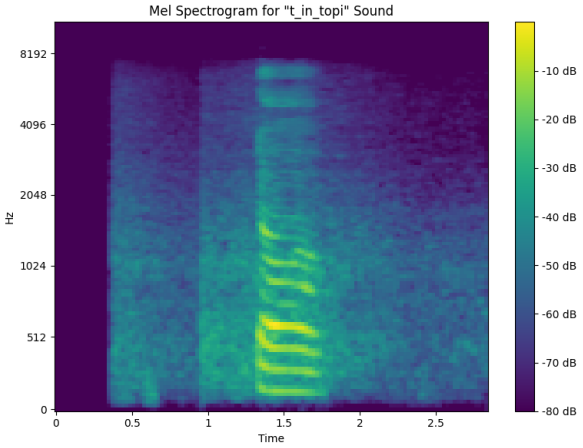
- '**th (as in ratha)**' sound



- '**t (as in topi)**' sound



- '**th (as in neelkantha)**' sound



Observations:

- '**t (as in tiwari)**': The spectrogram shows burst of energy at a specific frequency corresponding to the release of the closure (the moment when the tongue is released).
- '**th (as in ratha)**': The spectrogram shows a similar burst of energy as in the above "t," but with a longer duration.
- '**t (as in topi)**': The spectrogram shows a burst of energy at a specific frequency corresponding to the release of the closure.
- '**th (as in neelkantha)**': The spectrogram shows a similar burst of energy as in the above "t," but with a longer duration.

As seen from the plots on Page 2, we can also analyse the differences between the spectrograms plotted using the decibel scale and the mel scale:

- Using the dB (Decibel) Scale:
  - The dB scale is a logarithmic scale that measures the intensity or amplitude of a signal relative to a reference level. It represents power or energy ratios in a more human-perceptible way.
  - When the dB scale is used for spectra or spectrograms, the y-axis (intensity) is represented in decibels instead of linear amplitude.
  - The dB scale is particularly useful because it compresses the range of intensity values. In speech processing, this helps highlight the signal's relevant features, making it easier to identify and analyze formants, harmonics, and other spectral components.
- Using the Mel Scale:
  - Designed to mimic the way humans perceive differences in pitch. It's based on the fact that our perception of pitch is not linear but rather logarithmic.
  - In the Mel scale in spectrograms, the frequency axis is transformed so that it's no longer linear but follows the Mel scale. This means that the spacing between frequency bins is wider at lower frequencies and narrower at higher frequencies.
  - By using the Mel scale, we can represent the spectrogram in a way that's more relevant to how our ears perceive speech sounds.
  - It helps in capturing important acoustic features for speech recognition and synthesis.