Speech Processing in Frequency domain Using Periodogram Assignment

Oluwabori Godsfavour

Biometrics and Intelligent Vision, Universite Paris-Est Creteil.

Contributing authors: godsfavour.oluwabori@etu.u-pec.fr;

Abstract

In this Speech processing and analysis assignment, I set out to analyze an audio signal data and gain insights into its frequency characteristics. The primary goal of this project was to gain hands-on experience with speech processing using Python and understand the basic principles of speech signal processing and frequency representation. I explored the impact of windowing functions and sampling frequency on speech signal analysis using periodogram techniques. By applying these techniques and peak detection, I was able to identify specific frequency components related to vocal characteristics like pitch and formants. The findings demonstrate the importance of using appropriate sampling rates and windowing methods to accurately capture the nuances of speech signals. This foundational knowledge will be useful for further studies in audio processing and its applications in fields such as telecommunications and linguistics.

1 INTRODUCTION

Signal processing and analysis are fundamental components of artificial intelligence (AI) with diverse applications in various fields, such as healthcare, telecommunications, and audio signal processing. In audio signal processing, understanding the frequency composition of speech signals is particularly valuable for applications in voice recognition, telecommunications, and linguistics.

In this project, I aim to explore the basic principles underlying speech signal processing, focusing on the analysis of audio signals to extract meaningful insights about vocal characteristics and frequency content. The speech signal consists of complex waveforms that convey essential information, and analyzing these signals requires a solid grasp of both time-domain and frequency-domain techniques.

The analysis commenced with the extraction of an audio dataset, followed by calculating the basic characteristics of the speech signal. Followed by applying various windowing functions to examine their effects on spectral representation and frequency resolution. Key aspects such as the impact of sampling frequency and window length on the resulting periodograms were also explored. By identifying and analyzing dominant frequencies within the speech signal, I correlated these components with vocal characteristics, including pitch and formants.

Ultimately, this project seeks to enhance the understanding of audio signal processing principles and their implications for effective speech analysis, laying the groundwork for further studies in this critical area of research.

2 METHOD

The project's methodology encompassed a series of steps, from data acquisition to comprehensive signal analysis. Below, I outlined the steps involved and provided the corresponding Python code that was implemented to achieve the desired results.

2.1 Data Acquisition and Description:

The project began with loading an audio recording of my voice and reading it using the read-wav function. I proceeded to visulaize the signal using matplotlib.

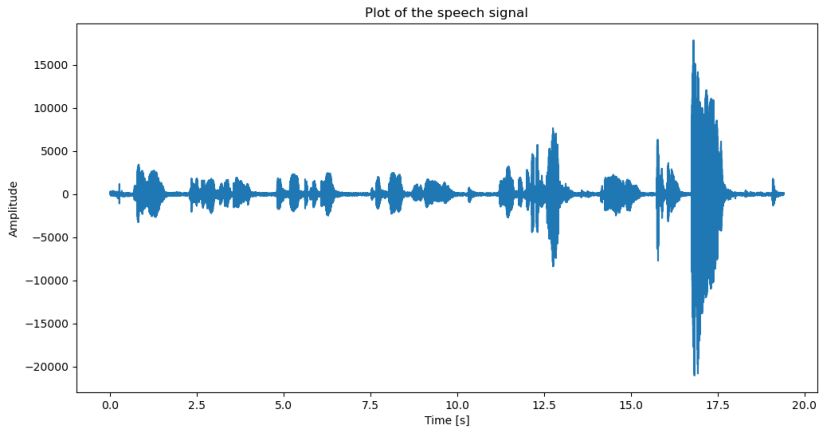

Fig 1: Visualisation of the speech signal

Below is the python script used for the above section

```
#Import neccessary libraries
from scipy.io.wavfile import read as read_wav
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import periodogram,find_peaks
from scipy.signal.windows import hamming,hann
#from scipy.special import title

#Load the recorded speech data from a WAV file.
```

```
sampling_rate, data = read_wav("C:\\Users\\user\\dev\\Speech Processing\\
                                Recording.wav")
# Print the audio data and its properties
print(data)
print(data.shape)
print(f"Number of channels = {data.shape[1]}")

Output: Number of channels = 2
#Plotting the speech signal
time = np.linspace(0,length,no_of_samples)
plt.figure(figsize=(12, 6))
plt.plot(time,data[:,0])
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.title('Plot of the speech signal')
plt.show()
```

Analysis of the Speech Signal:

The initial segment of the signal exhibits lower amplitude values, indicating the presence of softer sounds at the beginning of the recording. This aligns with the typical start of speech, where softer phonemes are often produced.

A significant increase in amplitude occurs around the 17-second mark, which corresponds to a louder section of the speech. This spike in amplitude reflects the raised voice and emphasis used towards the end of the recording.

2.2 Description of the Methods Used to Calculate Basic Characteristics and Frequency Representation:

In this analysis, I computed six basic characteristics of the speech signal: duration of the signal, sampling frequency, number of samples, mean, variance, and energy. These metrics offer valuable insights into the signal's overall behavior, strength, which are critical for understanding and analyzing audio data. I employed Python's numpy library to calculate these characteristics.

Summary of the Characteristics:

(a) Duration of the Signal: 19.4 seconds, representing the total time of the recording.
(b) Sampling Frequency: 48.0 kHz
(c) Number of Samples: 931,200
(d) Average Signal Value: -0.63
(e) Variance of the Signal: 2400225.66
(f) Energy of the Signal: -1584288051.00

```
#Basic Signal Characteristics:

#1a. Duration of the signal (in seconds).
length = data.shape[0] / sampling_frequency #total time
print(f"Duration of the signal (in seconds)= {length}s")


#1b. Sampling frequency and number of samples.
no_of_samples = data.shape[0]
print(f"Sampling frequency={sampling_frequency/1000} kHz and number of samples =
                                        {no_of_samples}")



signal = data[:,0]
```

```
# 1c. Mean value (average amplitude) and variance.
mean_signal = np.mean(signal)
variance = np.var(signal)
print(f'The average signal value is : {mean_signal:.2f} and variance of the
                                        signal is : {variance:.2f}')

#1d.Energy of the signal (sum of the squared values)
energy = np.sum(signal**2)
print(f'The energy of the signal is : {energy:.2f}')

Output: Duration of the signal (in seconds)= 19.4s
Sampling frequency=48.0 kHz and number of samples = 931200
The average signal value is : -0.63 and variance of the signal is : 2400225.66
The energy of the signal is : -1584288051.00
```

3 Frequency Representation (Periodogram):

A periodogram is used to identify the dominant periods (or frequencies) of a time series. The periodogram gives information about the relative strengths of the various frequencies for explaining the variation in the time series. For this section, I computed the periodogram and analyzed the impact of sampling frequency, windowing, and the length of the signal. I used the periodogram function from the scipy.signal module, and implemented the windowing functions (Hamming, Hanning) on the periodogram.

```
#Compute and plot the periodogram with different windowing functions
def plot_periodogram(signal, fs, window_func, window_name):
    # f contains the frequency components
    # S is the PSD
    f, S = periodogram(signal, fs, window=window_func, scaling='density')
    plt.plot(f, 10 * np.log10(S), label=f'Window: {window_name}')

plt.figure(figsize=(12, 6))
# No window (rectangular)
plot_periodogram(signal, sampling_rate, 'boxcar', 'Rectangular')
# Hamming window
plot_periodogram(signal, sampling_rate, hamming(len(signal)), 'Hamming')
# Hanning window
plot_periodogram(signal, sampling_rate, hann(len(signal)), 'Hanning')

plt.xlabel('Frequency (Hz)')
plt.ylabel('Power/Frequency (dB/Hz)')
plt.title('Periodogram of Speech Signal with Different Windowing Functions')
plt.legend()
plt.grid()
plt.show()
```

The first subplot below displays the frequency spectrum in Hertz (Hz) and the corresponding power in dB. The second and third subplots display the influence of the hamming, hanning windowing functions on the periodogram. The resulting plot is shown in Figure 2.

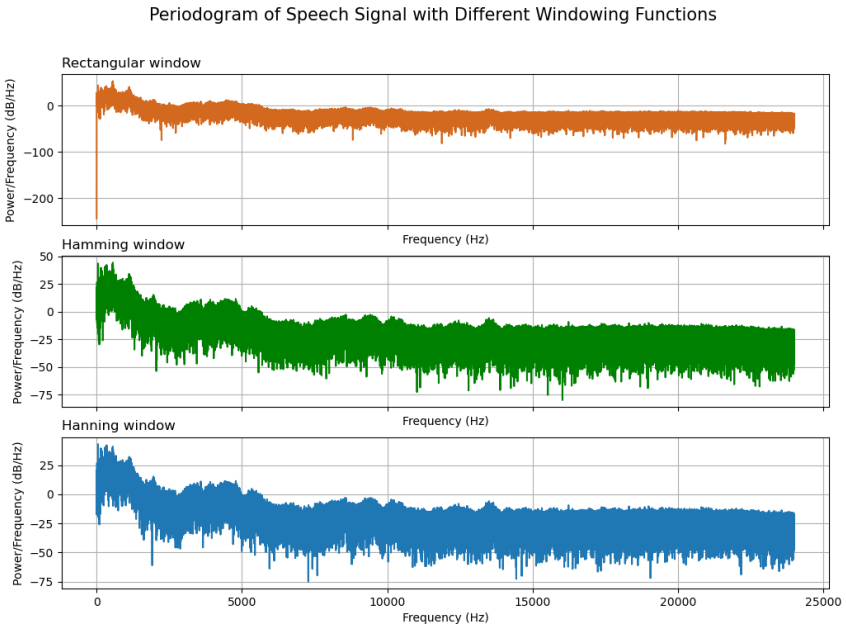

Fig:2 Plot of the periodogram speech signal with different windowing functions

Influence of the windowing functions (Hamming, Hanning) on the periodogram.

Windowing smooths the edges of a finite data segment, reducing spectral leakage in FFT analysis. The rectangular window (no windowing) shows more leakage, where power spreads to adjacent frequencies. Hamming and Hanning windows smooth the edges of the signal, reducing leakage but slightly broadening peaks. While the rectangular window gives sharper peaks, this introduces more leakage, where energy spreads into neighboring frequencies. Hamming and Hanning provide a clearer picture of the dominant frequencies in the Power Spectral Density (PSD) by smoothing the edges and preventing noise artifacts.

Rectangular Window (No Windowing) - Top Plot: The power spectral density (PSD) exhibits significant spectral leakage, visible as the energy spreads around dominant frequency components. Sharp transitions at the signal boundaries create artifacts, especially in lower frequencies, making it harder to identify clear peaks.

Hamming Window - Middle Plot: The Hamming window applies smoother transitions at the boundaries, reducing spectral leakage more effectively than the rectangular window. Although the main lobe is slightly wider, reducing peak sharpness, the overall energy distribution is more controlled, making the dominant frequencies easier to observe, particularly in the lower ranges.

Hanning Window - Bottom Plot: The Hanning window reduces spectral leakage effectively but not as well as the Hamming window. However, it offers a good

trade-off between leakage reduction and frequency resolution, with slightly less sharp peaks but cleaner suppression of side lobes, providing clear visibility of the frequency components.

Impact of Varying Window Lengths on Frequency Resolution

Frequency Resolution Definition: Frequency resolution refers to the ability to distinguish between two closely spaced frequency components in a signal. It is defined by the formula:

$$\Delta f = \frac{f_s}{N}$$

where Δf is the frequency resolution, f_s is the sampling frequency, and N is the number of samples in the window. A longer window length results in better frequency resolution because it increases N, allowing for finer distinctions between frequencies.

In this analysis, I considered three different window lengths:

- **Window Length: 232,800 samples** (one-quarter of the original signal length)
- **Window Length: 93,120 samples** (one-tenth of the original signal length)
- **Original Window Length: 931,200 samples**

Observations

1. **Short Windows (93,120 samples):** Shorter windows provide a more detailed view of what frequencies are present at each moment. This is particularly useful for capturing rapid changes in the signal. In speech analysis, this means that short windows help identify quick variations in pitch or tone within a sentence.
2. **Medium Windows (232,800 samples):** With a medium window length, the frequency resolution improves compared to the shortest window. It balances the ability to observe dynamic changes in frequency while also allowing for better resolution of distinct frequency components.
3. **Long Windows (931,200 samples):** Using the original length significantly enhances frequency resolution. The peaks in the Power Spectral Density (PSD) become sharper and more defined, allowing for easier identification of closely spaced frequency components. However, this comes at the cost of temporal resolution, making it more challenging to capture transient features of speech.

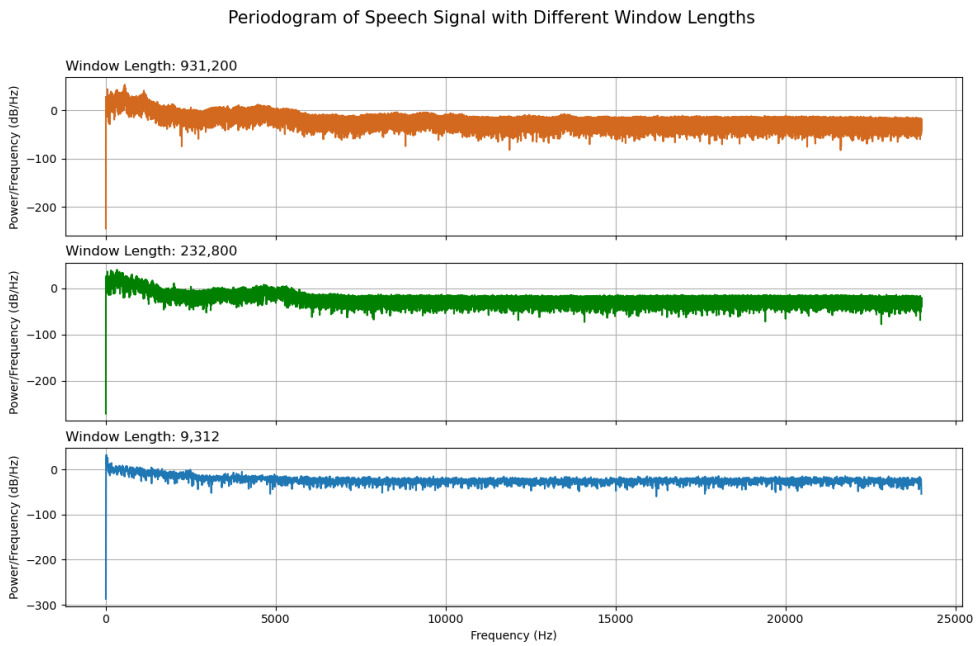

Fig:3 Plot of the periodogram speech signal with different windowing functions

The following Python script was used to implent and visualize the various window lengths on the periodogram:

```
    # Create subplots for different window lengths
fig, axs = plt.subplots(3, 1, sharex=True, figsize=(12, 10))

# Original signal with full length
f_full, Pxx_full = plot_periodogram(signal, sampling_frequency, boxcar(len(signal))
                                    )

# Quarter length window
window_length_quarter = int(len(signal) / 4)
f_quarter, Pxx_quarter = plot_periodogram(signal[:window_length_quarter],
                                          sampling_frequency, boxcar(
                                          window_length_quarter))

# One-tenth length window
window_length_tenth = int(len(signal) / 100)
f_tenth, Pxx_tenth = plot_periodogram(signal[:window_length_tenth],
                                      sampling_frequency, boxcar(
                                      window_length_tenth))

# Plot each periodogram
axs[0].plot(f_full, Pxx_full, label='Window Length: 931,200', color='chocolate')
axs[1].plot(f_quarter, Pxx_quarter, label='Window Length: 232,800', color='green')
axs[2].plot(f_tenth, Pxx_tenth, label='Window Length: 9,312')

# Setting labels and titles for each subplot
for ax in axs:
    ax.set_ylabel('Power/Frequency (dB/Hz)')
    ax.grid(True)
```

```
axs[0].set_title('Window Length: 931,200', loc='left')
axs[1].set_title('Window Length: 232,800', loc='left')
axs[2].set_title('Window Length: 9,312', loc='left')
axs[2].set_xlabel('Frequency (Hz)')  # Set x-label for the last subplot

# Main title for the figure
fig.suptitle('Periodogram of Speech Signal with Different Window Lengths', fontsize
                                        =15)

# Show the plot
plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust layout
plt.show()
```

In summary, the impact of varying window lengths on frequency resolution is profound. While shorter windows facilitate the observation of rapid frequency changes, longer windows enhance the ability to resolve closely spaced frequency components. The selected window lengths of 232,800 and 93,120 samples, being fractions of the original length, allowed for a comprehensive analysis of how different window sizes influence the interpretation of speech signals.

4 Dominant frequencies and the effect of varying window lengths on frequency resolution.

The term *dominant frequencies* identifies frequencies with the greatest power in the signal. I used the Pythons scipy.signal.find_peaks function to programmatically detect peaks within the periodogram revealing frequencies with the most energy.

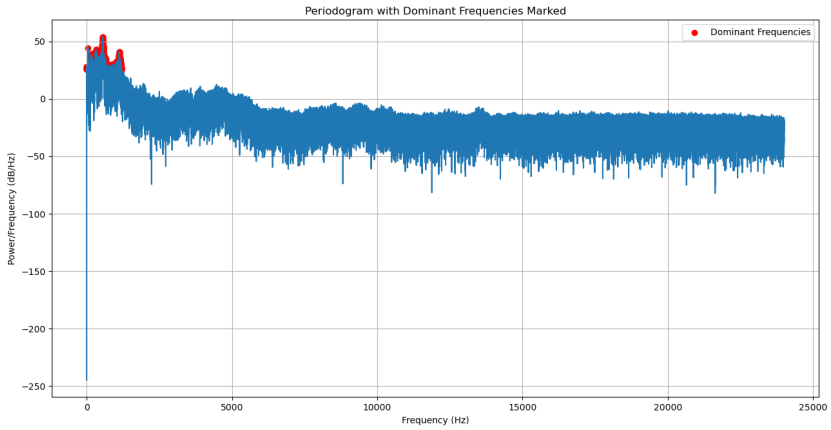

Fig:4 Periodogram with Dominant Frequencies Marked

The following Python script was used to identify the dominant frequencies in the speech signal:

```
    def find_and_plot_dominant_frequencies(frequencies, psd, height_threshold=25):
    """
```

```
    Find and plot dominant frequencies from a Power Spectral Density (PSD).

    Parameters:
    frequencies (array): Array of frequency values.
    psd (array): Array of Power Spectral Density values (in dB).
    height_threshold (float): Height threshold for peak detection (default is 25 dB
                                        ).

    Returns:
    tuple: Dominant frequencies and their corresponding powers.
    """
    # Find peaks with the specified height threshold
    peaks, _ = find_peaks(psd, height=height_threshold)

    # Extract dominant frequencies and their corresponding power levels
    dominant_frequencies = frequencies[peaks]
    peak_powers = psd[peaks]
    # Plot the PSD and mark the dominant frequencies
    plt.figure(figsize=(10, 6))
    plt.plot(frequencies, psd, label='Power Spectral Density')
    plt.scatter(dominant_frequencies, peak_powers, color='red', label='Dominant
                                            Frequencies')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Power/Frequency (dB/Hz)')
    plt.title('Periodogram with Dominant Frequencies Marked')
    plt.legend()
    plt.grid(True)
    plt.show()

    return dominant_frequencies, peak_powers
dominant_freqs, peak_powers = find_and_plot_dominant_frequencies(frequencies, psd)
```

4.1 Analysis and Interpretation:

Figure 4 shows the periodogram with marked peaks, highlighting prominent frequencies. In speech analysis, these frequencies correlate with formants, resonant frequencies of the vocal tract crucial for identifying vowel sounds. Formants are higher-frequency resonances that shape the vowel sounds. Formants are crucial for identifying vowels and are typically denoted as F1, F2, F3, etc. Dominant frequencies in different ranges suggest various aspects of speech:

- Frequency with the lowest peak ($\tilde{2}5$ Hz): Generally falls below the range of human vocalization and likely represents background noise or artifacts rather than phonetic content.
- Frequency with the highest peak ($\tilde{5}50$ Hz): Likely corresponds to the first formant (F1), typically ranging from 300 to 1000 Hz, indicating a vowel sound produced with an open mouth position.
- Frequency with the last peak ($\tilde{1}200$ Hz): Correlates with the second formant (F2), usually within 800 to 2500 Hz, suggesting front vowel characteristics.

Effect of Windowing While the dominant frequencies are consistent across windowing functions, windowing affects spectral leakage. The Hamming and Hanning windows reduce leakage, offering a clearer representation of formants without altering the fundamental frequency content. This is expected, as the windowing doesn't change the underlying frequency content; it only affects how that content is represented (especially around the edges).

5 Impact of Sampling Frequency on Spectrum Resolution and Quality of the Periodogram

The sampling frequency directly influences both the resolution of the periodogram and the highest frequency components observable in the spectrum. According to the Nyquist theorem, the highest frequency that can be accurately observed is half of the sampling rate, known as the Nyquist frequency. This impacts both the resolution and range of frequencies captured in the analysis.

The speech signal used for this analysis was sampled at 48kHz, this implies that the highest frequency that can be observed in the spectrum is 24kHz. This means that frequencies in your signal can be accurately represented up to 24,000 Hz.

Figure 5 displays the periodogram of the resampled speech signal at various sampling frequencies

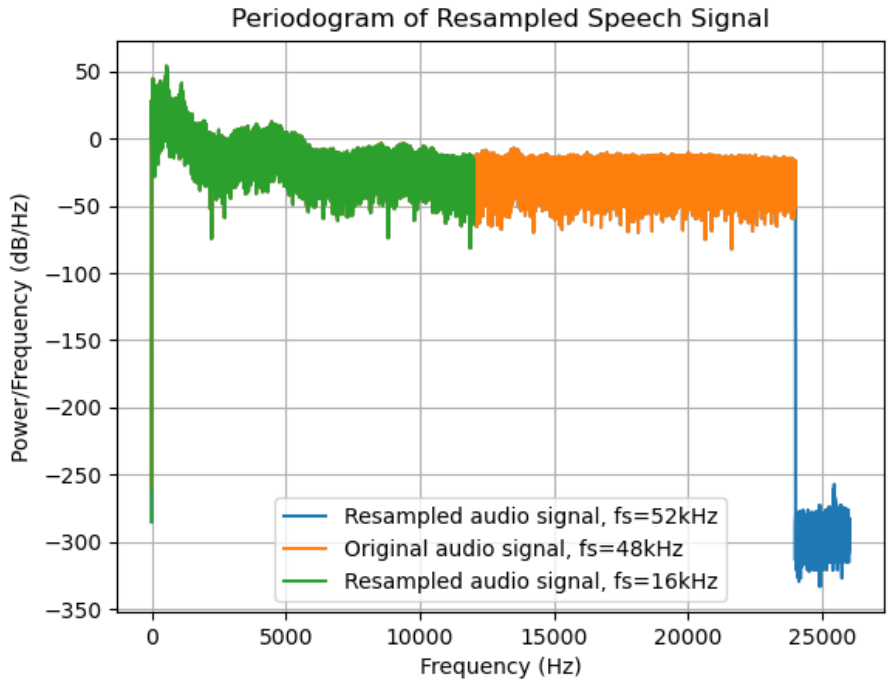

Fig 5: Periodogram of Resampled Speech Signal at Various Sampling Frequencies

The periodogram shown in Figure 5 presents the original audio signal sampled at 48 kHz, as well as two resampled signals at 16 kHz and 52 kHz. Observing these periodograms, several effects of sampling frequency on spectrum quality and resolution are evident:

- **Higher Sampling Rate (52 kHz):** Resampling to 52 kHz technically extends the frequency range up to 26 kHz. However, since the original signal was recorded

at 48 kHz, it contains no information beyond 24 kHz. This leads to a drop in the PSD above 24 kHz in the periodogram for the 52 kHz sample, as the additional range (24–26 kHz) contains no data from the original recording. The higher sampling rate simply extends the observable range without improving resolution within the existing 24 kHz bandwidth. Achieving better resolution within the existing range relies more on factors like windowing, FFT length, and signal duration rather than increased sampling rates alone.

- **Lower Sampling Rate (16 kHz):** Resampling to 16 kHz limits the observable frequencies to 8 kHz, which is sufficient for capturing the fundamental speech frequencies but filters out higher-frequency details. This reduction impacts the clarity and perceived quality, as seen with the green curve in the plot 5, which shows a more confined periodogram.

In summary, higher sampling rates allow for a broader frequency range but if those frequencies weren't in the original signal, it won't improve the resolution within the existing bandwidth. Conversely, lower sampling rates reduce clarity and can introduce aliasing if they fall below the Nyquist rate.

The following Python script was used to carry out the analysis detailed in the above section:

```
# Resample signal to different sampling frequencies
def plot_resampled_periodogram(signal,new_fs='new_fs'):
    """
        Resamples the signal to a new sampling frequency and computes the
                                                periodogram.

        Parameters:
        - signal: The input signal to resample and analyze.
        - new_fs: The new sampling frequency for resampling.

        Returns:
        - frequencies: Frequency components of the periodogram.
        - PSD: Power spectral density of the resampled signal.
        """
    # Resample the signal to the new sampling frequency
    # The original sampling frequency of the signal is 48kHZ
    resampled_signal = resample(signal, int(len(signal) * new_fs /
                                            sampling_frequency))

    # Compute the periodogram for the resampled signal
    frequencies, psd = plot_periodogram(resampled_signal, new_fs, 'boxcar', '
                                            Rectangular')
    return  frequencies, psd

# Define sampling frequencies to analyze
sampling_frequencies = [52000, 48000, 16000]

# Define a dictionary to store results for easy retrieval
periodogram_data = {}

# Compute periodograms for each sampling frequency
for new_fs in sampling_frequencies:
    frequencies, psd = plot_resampled_periodogram(signal,  new_fs)
    periodogram_data[new_fs] = (frequencies, psd)
# Plot the periodograms for each resampling rate
for new_fs, (frequencies, psd) in periodogram_data.items():
```

```
    label = f"Resampled audio signal, fs={new_fs//1000}kHz" if new_fs !=
                                                sampling_frequency else f"Original
                                                audio signal, fs={new_fs//1000}kHz"
    plt.plot(frequencies, psd, label=label)

# Customize plot

plt.xlabel('Frequency (Hz)')
plt.ylabel('Power/Frequency (dB/Hz)')
plt.title('Periodogram of Resampled Speech Signal')
plt.legend()
plt.grid()
plt.show()
```

6 SUMMARY

In this project, I carried out a comprehensive analysis of an audio signal, specifically focusing on speech processing techniques. The primary objectives were to understand the fundamental principles of audio signal processing and to gain insights into the frequency representation of speech signals.

Through the application of various methods, including periodogram analysis and peak detection, I identified key characteristics of the speech signal, including dominant frequencies and their correlation with vocal formants. The impact of sampling frequency on spectral resolution was also examined, revealing that while higher sampling rates can extend the observable frequency range, they do not inherently improve resolution for frequencies not present in the original signal.

Moreover, I explored the effect of varying window lengths on frequency resolution, demonstrating how shorter windows capture rapid changes in frequency, while longer windows provide finer distinctions between closely spaced frequencies.

The insights gained from this analysis not only deepen my understanding of speech signal processing but also highlight the importance of various factors in audio analysis, such as sampling frequency, window length, and peak detection algorithms. These findings are crucial for applications in fields like telecommunications, healthcare, and artificial intelligence, where effective speech and audio analysis can lead to significant advancements.