GROUPWORK

Coursework Declaration and Feedback Form

Course Code:	**ENG 5027**	Course Name:	**Digital Signal Processing**		
Course Co-Ordinator:	**Scott Watson**	Mentor:		Group Number Lab Group Tutorial Group	**36**
Title of Assignment:	**Assignment 1 (Fourier Transform – FFT)**				
Date of Submission:	**20-10-2024**				
Declaration of Originality and Submission Information		*I affirm that this submission is my own / the groups original work in accordance with the University of Glasgow Regulations and the School of Engineering Requirements* ***All students should sign this form***			
Student Name:	**Basav Prasad (21961731)**	Student Name:	**Cem Cetinkaya (2961798)**	Student Name:	**Mingxuan Sun (3025864)**
Student Name:	**Syafiq Fathullah (2830739)**	Student Name:	**Kabir Sani Muhammad (2973267)**	Student Name:	**Abdulrazaq Muhammad Sani (2962438)**

Feedback from Lecturer to Student – to be completed by Lecturer or Demonstrator	
Grade Awarded: Feedback (as appropriate to the coursework which was assessed)	
Lecturer/Demonstrator	Date returned to the Teaching Office

University of Glasgow | School of Engineering

Assignment 1 (Fourier Transform – FFT)

October 20, 2024

Assignment report submitted in partial fulfilment of the requirements for the credit of course -

Digital Signal Processing

Introduction

This report presents the analysis and enhancement of an audio signal using Python. The main focus was to examine the signal in both the time and frequency domains, identify key features such as fundamental frequencies, harmonics, and noise bands, and enhance the audio by improving vocal quality. The script was implemented to make the audio perceptually more pleasant, with clearer and more vibrant vocal content.

Objectives

1. **Load and Plot the Original Audio Signal**: Visualize the audio signal in both the time and frequency domains with proper labeling and logarithmic axes for the frequency plot.
2. **Identify Key Features**: Identify fundamental frequencies, harmonics, and noise bands in the audio spectrum.
3. **Improve Voice Quality**: Enhance the quality of the voice by manipulating the frequency bands above 3 kHz.
4. **Enhance the Voice**: Apply an aural exciter to enhance the vocal clarity and richness.

Methods

Task 1: Loading and Plotting the Original Audio

The audio file, “orig.wav”, was loaded using the Python `wave` module and converted into a NumPy array. The signal, originally in 16-bit format, was normalized to the range –1 to 1 by dividing each sample by $\mathbf{2^n/_2 - 1 = 2^{16}/_2 - 1}$ or by $\mathbf{2^{(n-1)} - 1 = 2^{15} - 1}$, ensuring the values fell within the range **-32768 to +32767** (since 16-bit signed integers can represent 65536 different values). for further processing. Alternatively, normalization can also be done by dividing each sample by the maximum absolute value in the data, as shown in the code:

```
# Normalize the audio data to be between -1 and +1
audio_data = audio_data / np.max(np.abs(audio_data), axis=0)
```

This ensures that the highest amplitude in the audio data is mapped to 1, and the lowest to -1, which helps in standardizing the signal for further analysis and prevents clipping during processing.

- **Time Domain Plot**: The time domain plot was generated by creating a time axis from 0 to the total duration of the audio, calculated as the number of frames divided by the frame rate.

```
# Create a time axis
time = np.linspace(0, len(audio_data) / sampling_rate, num=len(audio_data))
```

This line creates an evenly spaced array of time values corresponding to each audio sample, allowing us to plot the amplitude of the audio signal over time. The normalized audio signal was then plotted against this time axis to visually observe amplitude variations.

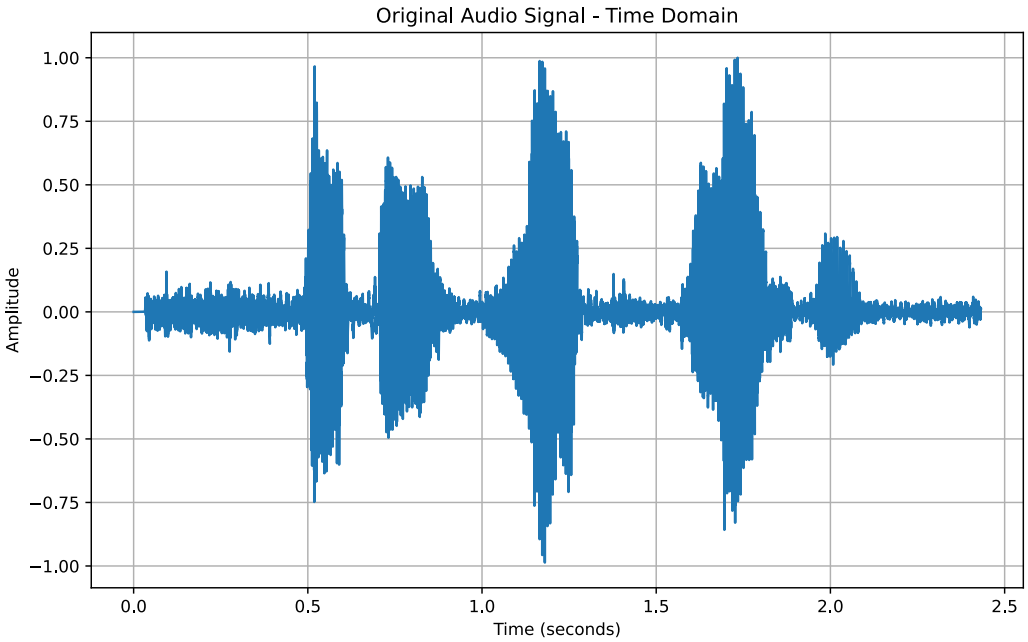

Figure 1 - Time domain plot of orig.wav

- **Frequency Domain Plot**: A Fourier Transform (FFT) was applied to convert the audio signal from the time domain to the frequency domain. Only the positive frequencies were extracted and plotted. To visualize the wide range of frequency components effectively, both the frequency and amplitude axes were plotted on a logarithmic scale using `plt.xscale('log')` and `plt.yscale('log')`. This helped to emphasize the lower and higher frequency components that are important for audio analysis.

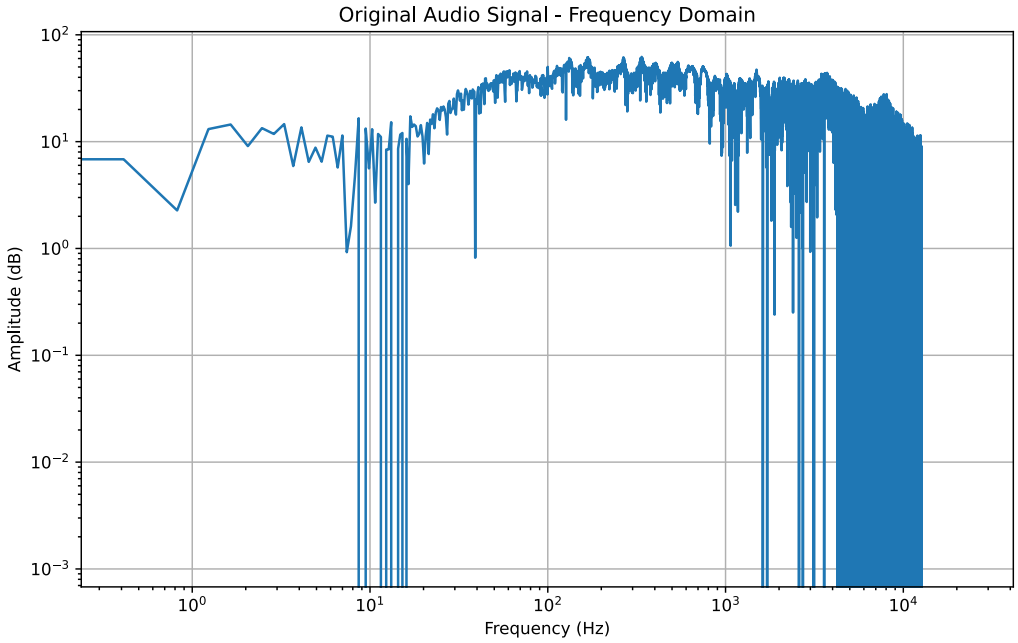

Figure 2 - Frequency domain plot of orig.wav

Task 2: Identifying Key Features

- **Fundamental Frequency**: The fundamental frequency is the lowest frequency present in a sound and is often associated with the pitch of the sound. For human speech, particularly vowels, the fundamental frequency is the primary component that defines the perceived pitch. For human speech, this typically ranges between **85 Hz and 255 Hz**, male voices lies in lower side of the range while female voices generally have higher fundamental frequencies. In this analysis, the fundamental frequency was identified at **168.174 Hz**, which is within the typical range for male speech.

- **Harmonics:** Harmonics are frequencies that are integer multiples of the fundamental frequency. They contribute to the richness and timbre of the sound, giving it a fuller and more complex quality. For example, if the fundamental frequency is $\boldsymbol{f_0}$, then the harmonics are $\boldsymbol{2f_0}$, $\boldsymbol{3f_0}$ and so on. In this analysis, harmonic frequencies ranging from **2 to 5 times** the fundamental frequency were marked on the frequency domain plot to observe their contributions. These harmonics help distinguish different vowels and add color to the voice.

- **Noise:** Noise refers to unwanted components in the audio signal that do not contribute meaningfully to the vocal content. It is typically present in both lower (**below 85 Hz**) and higher (**above 8 kHz**) frequency ranges. In the context of human sound production, low-frequency noise may come from background hum or body movements, while high-frequency noise can be from equipment or environmental factors. Humans perceive sound by focusing on the important frequency components (such as the fundamental and harmonics), while noise is often ignored or filtered out by our auditory system. Reducing these noise bands is crucial for improving the clarity and intelligibility of the audio. In this analysis, harmonic frequencies ranging from 2 to 5 times the fundamental frequency were marked on the frequency domain plot to observe their contributions.

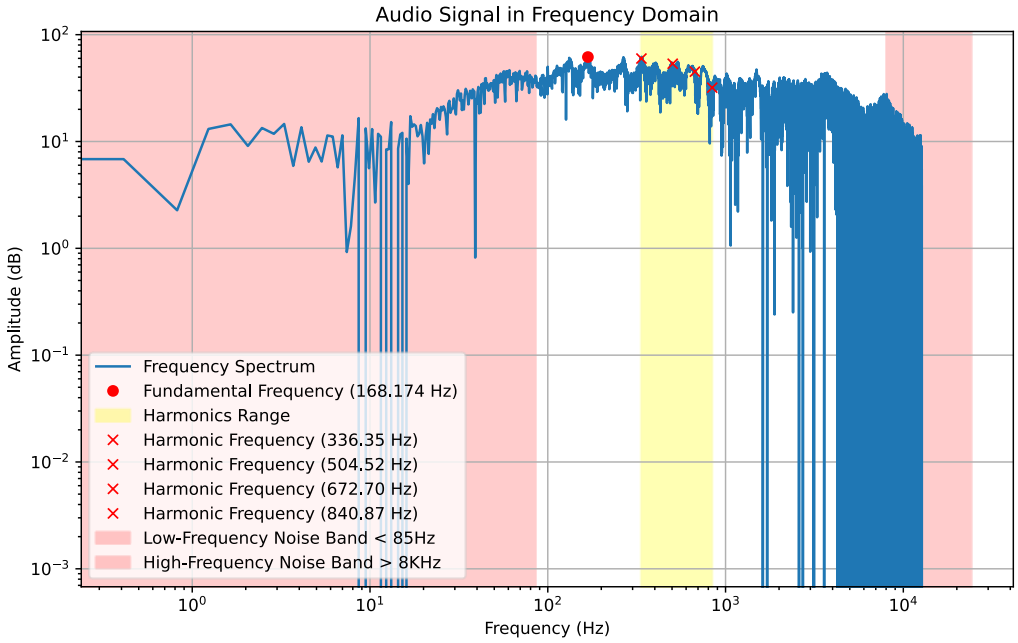

Figure 3 - Frequency domain plot with marked key elements

- **Vowel and Consonant Ranges:** Vowel and consonant sounds occupy different parts of the frequency spectrum, which played a crucial role in our analysis. For this, each letter sound was recorded separately from the phrase **“People Make Glasgow”** and compared the spectrograms to the original signal. The aim of the comparison was to help identify which parts of the signal contain important information, which then would help particularly in choosing the appropriate filters and enhancement techniques.
- **Vowels:** The spectrograms of vowels, such as "**E**" in "**People**" (Figure 3) and "**A**" in "**Glasgow**" (Figure 4), clearly demonstrate that their fundamental frequencies and first harmonics generally reside between **100 Hz and 500 Hz**. This is especially evident on the lower end of the graphs, where bright yellow bands indicate the stronger energy in this range. Vowels are characterized by distinct formants, which appear as horizontal bands that extend up to around 5 kHz. These features confirm that vowels primarily occupy the low to mid-frequency range, where most of the vocal energy is concentrated. The harmonics of these vowels are visible, with decreasing intensity beyond 500 Hz, aligning with expected vowel behavior.

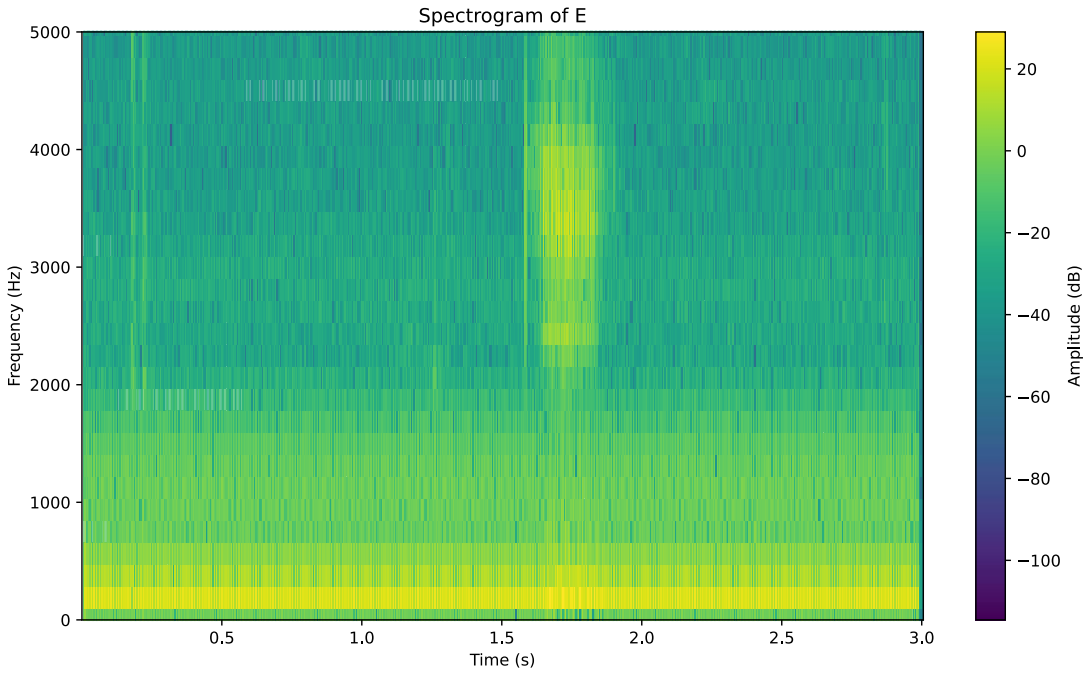

Figure 4 - Spectrogram of E spoken at 1.7th second

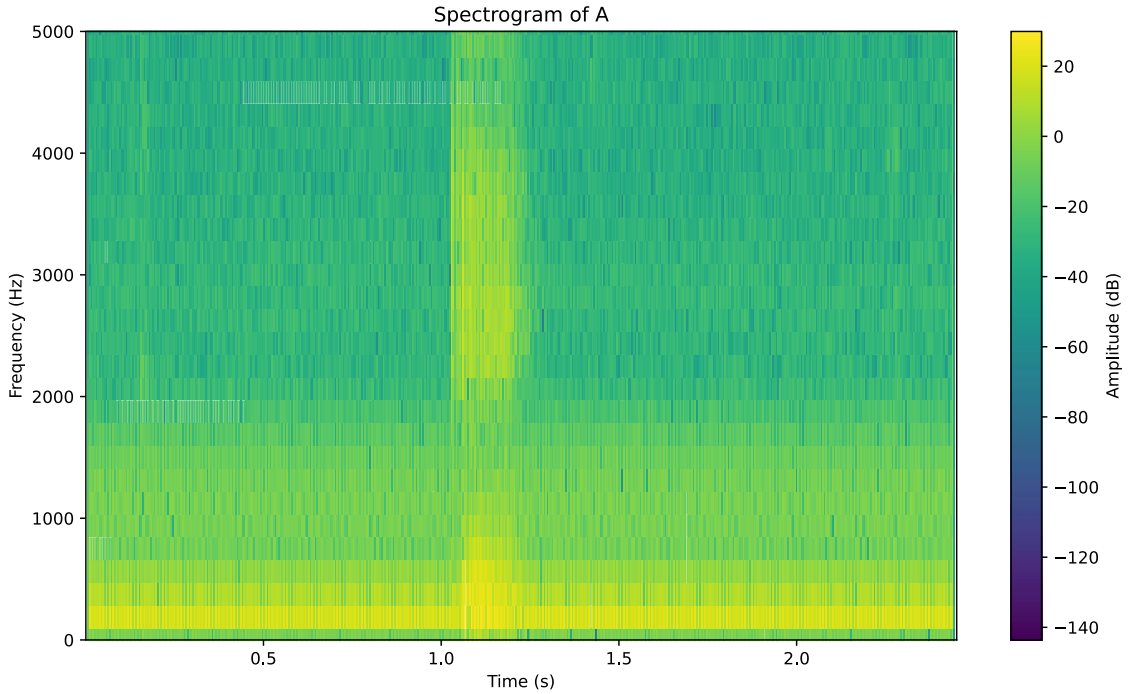

Figure 5 - Spectrogram of A spoken at 1.2th sec

- **Consonants:** In contrast, the spectrograms of consonants, such as **"G"** (figure 5) and **"M"** (figure 6) show a **wider frequency range** compared to vowels. This broader range is typical of consonants, especially voiced sounds like **"G"** and **"M,"** where energy extends across both low and high frequencies. The **"G"** sound exhibits a strong concentration of energy in the lower frequencies, below **1 kHz**, and then spreads upward with visible energy components reaching up to around **17.5 kHz**. This reflects the voiced and somewhat plosive nature of the consonant **"G,"** which involves both vocal cord vibration and a burst of air. The consonant **"M"**, being a nasal sound, shows more concentrated energy in the lower frequencies but with less prominence in the higher frequencies compared to **"G"**. The broad range seen in the **"G"** spectrogram is consistent with its mixed plosive and voiced nature, where both low and mid-to-high frequencies are activated during articulation.

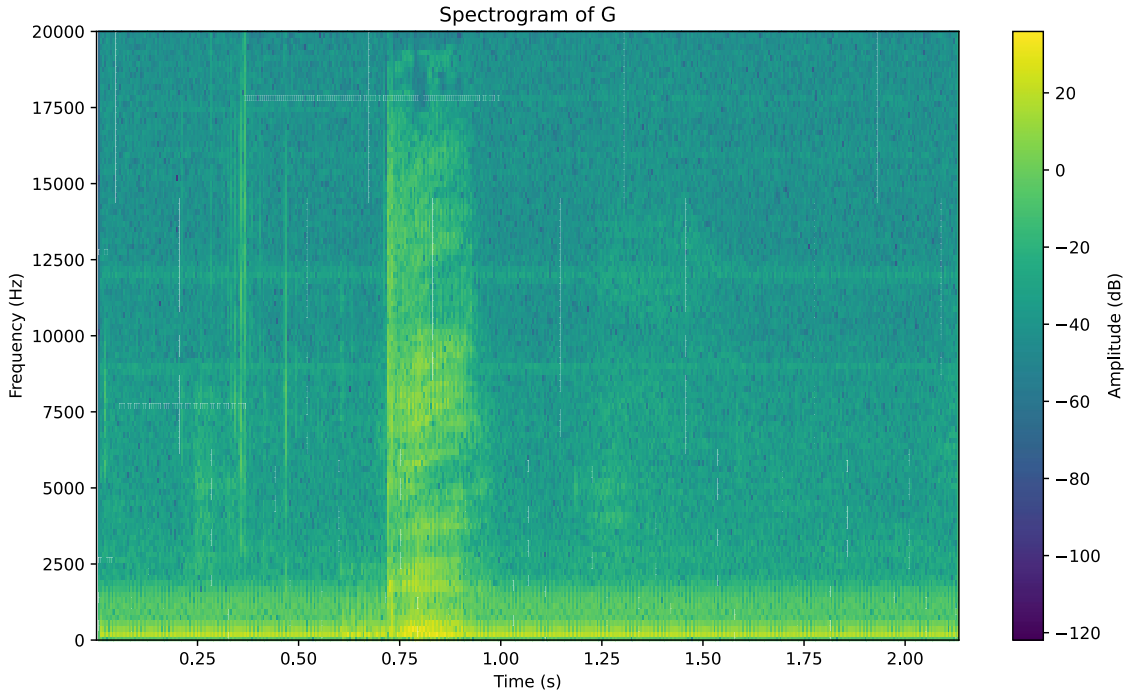

Figure 6 - Spectrogram of G spoken at 0.75th sec

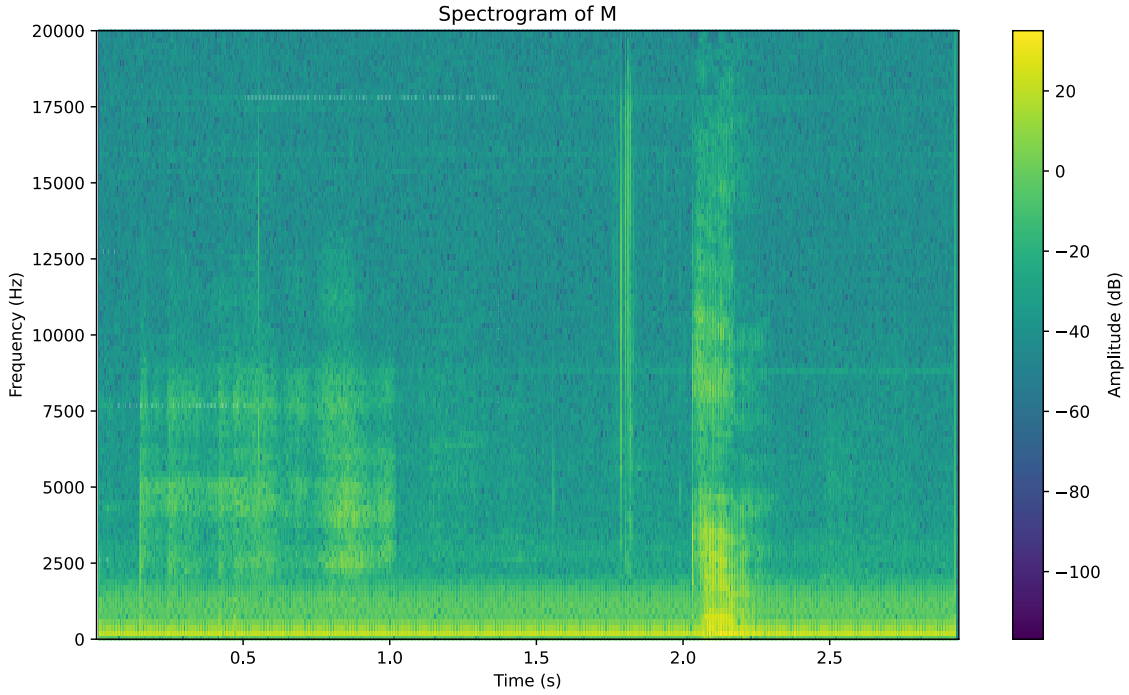

Figure 7 -Spectrogram of M spoken at 2.2th sec

Task 3: Improving Voice Quality

In this task, the objective was to improve the quality of the voice by manipulating the frequency bands above 3 kHz using the Fourier Transform. This enhancement aimed not at noise reduction but at making the voice sound perceptually more pleasant and engaging.

- **Noise Gating**: Noise was detected at the beginning of the audio, and its duration was identified to be 0.5 seconds. During this period, the absence of speech and the presence of noise, likely due to environmental factors or recording artefacts, were observed. To address this, the first 0.5 seconds were extracted from the audio and converted to its frequency domain form and all the frequency components in this segment were set to zero. This approach was chosen because no valuable speech information was present in this portion, and retaining the noise would have compromised the overall clarity of the audio. Later, it was converted back to the time domain and added to the original audio.

- **Frequency Manipulation**: The frequency range above 3 kHz is crucial for the intelligibility and brightness of human speech, especially for the clarity of consonant sounds. By focusing on this range, it was intended to enhance the details that make the speech sound clearer and more distinct. To achieve this, the Fourier Transform (FFT) was used to convert the cleaned audio signal from the time domain to the frequency domain, enabling us to isolate and manipulate the specific frequency components of interest.

- **Custom Smoother Filter**: A custom smoother filter was applied to frequencies between 3 kHz and 10 kHz. This filter was designed to create a smooth transition in this range, avoiding abrupt changes that could introduce harshness or unnatural artifacts into the audio. The filter mask applied a gradual attenuation, which enhanced the targeted frequency components in a subtle and controlled manner.

```
# Custom smoother filter for frequencies between 3kHz and 10kHz with smooth transition
def smooth_transition_filter(fft_freq, cutoff_low, cutoff_high):
    mask = np.ones_like(fft_freq)  # Start by keeping all frequencies
    transition_band = (np.abs(fft_freq) > cutoff_low) & (np.abs(fft_freq) <= cutoff_high)
    mask[transition_band] = (cutoff_high - np.abs(fft_freq[transition_band])) /
(cutoff_high - cutoff_low)
    return mask
```

- **Inverse FFT and Normalization**: After applying the filter, an inverse FFT was used to convert the enhanced frequency-domain signal back into the time domain. The audio was then normalized to ensure that the output signal remained within an appropriate amplitude range, preventing clipping or distortion. This step was crucial for maintaining consistent volume levels and ensuring that the enhanced audio did not introduce any artifacts.

- **Challenges and Considerations**: During this process, several approaches were considered, such as enhancing a broader range of frequencies. Initially, bandpass and low-pass filters were experimented to compare their effects on the voice quality. The bandpass filter was the most effective in enhancing clarity and brightness; it is not allowed to use standard bandpass filters for this assignment. Therefore, a custom smoother filter was created to achieve similar results while adhering to the assignment requirements. However, such approaches often introduced excessive brightness or unnatural artifacts. By specifically targeting the 3 kHz to 10 kHz range and using a smooth filter, an optimal balance between naturalness and enhancement was obtained. The tuning of the filter required careful adjustment to avoid over-enhancement, which could lead to an artificial sound. After several iterations, the parameters were tuned to achieve the desired perceptual quality without compromising the natural tone of the voice.

Task 4: Enhancing the Voice with Aural Exciter

The objective of Task 4 was to further enhance the vocal quality by applying an aural exciter effect, which aims to add brightness and vibrancy to the voice. This process was intended to make the voice sound perceptually louder and more engaging, leveraging the concept of harmonic distortion.

- **Aural Exciter Concept**: The aural exciter is a technique that works by adding a small amount of harmonic distortion to specific frequency ranges, enhancing the perceived loudness and clarity of the audio without increasing its actual volume significantly. The idea is to pass the audio signal through a non-linearity, in this case, the hyperbolic tangent (tanh), to emphasize harmonic content that contributes to a more engaging sound.
- **Frequency Range Selection**: In our implementation, the frequency range between 3 kHz and 10 kHz was chosen for the exciter. This range is particularly important for speech intelligibility and brightness, capturing key consonant sounds and enhancing the overall clarity of the voice. The FFT was used to convert the audio signal into the frequency domain, which allowed us to isolate and manipulate this specific range.
- **Application of Non-Linearity**: The selected frequency components between 3 kHz and 10 kHz were passed through a non-linearity using the tanh function. The tanh function was chosen because it provides a smooth form of distortion that can enhance harmonic content without introducing excessive harshness. By applying this non-linearity, the harmonic content in this frequency band was amplified, adding a sense of brightness and richness to the voice.
- **Scaling and Combining**: After applying the non-linearity, the excited frequency components were scaled by a factor of 0.3 before being added back to the original signal. The scaling factor was crucial to ensure that the effect was noticeable but not overwhelming. Adding too much of the excited signal could result in an unnatural or overly processed sound, whereas too little would not provide the desired enhancement. The chosen scaling factor provided a good balance, enhancing the vocal characteristics without compromising naturalness.
- **Frequency Limitation**: To ensure that the enhanced audio did not introduce any unwanted high-frequency noise, a frequency limitation step was applied. This involved zeroing out any frequency components above 10 kHz, ensuring that only the desired range was preserved. This step helped maintain a clean and controlled enhancement without introducing additional artifacts.
- **Inverse FFT and Normalization**: Once the enhancement was complete, an inverse FFT was performed to convert the signal back to the time domain. The audio was then normalized to keep the output within a suitable amplitude range, ensuring that the enhancements did not cause clipping or distortion. Normalization also helped maintain a consistent volume level, making the enhanced audio more pleasant to listen to.
- **Challenges and Considerations**: One challenge during this process was determining the optimal amount of harmonic distortion to add. Initially, different non-linear functions and scaling factors were experimented with. Functions such as simple clipping or other non-linearities often introduced harshness or artifacts that were not desirable. The tanh function provided a smoother form of harmonic enhancement, and after several iterations, we settled on a scaling factor that provided a subtle yet impactful enhancement. Additionally, it was found that applying the exciter only to the targeted frequency range was crucial for achieving a balanced and natural-sounding result, as a broader application often led to an over-processed sound.

Results

- The **time domain plot** of the original audio showed the natural amplitude variations of the recorded voice.
- The **frequency domain plot** highlighted the fundamental frequency, harmonics, and noise bands, providing insights into the signal composition.
- After **manipulating the frequency bands above 3 kHz**, the audio showed enhanced clarity, making it sound more pleasant and distinct.
- The application of the **aural exciter** resulted in enhanced vocal brightness and richness, as evidenced by the final frequency domain plot, which showed a more pronounced harmonic structure in the key frequency ranges.

Conclusion

This assignment analyzed and enhanced an audio signal by visualizing and modifying key components in both the time and frequency domains. The use of a smoother filter and aural exciter significantly improved the perceptual quality of the audio. The final enhanced audio exhibits better clarity, enriched harmonic content, and an overall more pleasant listening experience.

GitHub Repository

The full code and additional lab assignments for this Digital Signal Processing (DSP) course can be found on the GitHub repository: DSP Lab Assignments. The repository contains all tasks and scripts, for FFT which will also be used for future assignments FIR, and IIR filter implementations.

References

- **Python Libraries**: numpy, matplotlib, wave, scipy.io.wavfile
- **Aural Exciter Concept**: Aphex Aural Exciter

Appendix

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile

from scipy.io.wavfile import write


# TASK - 1 - LOADING THE WAV FILE AND PLOTTING GRAPHS


# Load the original audio file

file_path = '/home/basav/DSP/Digital-Signal-Processing/Assignment_1_FFT/audio files/orig.wav'

sampling_rate, audio_data = wavfile.read(file_path)


# Normalize the audio data to be between -1 and +1
```

```
audio_data = audio_data / np.max(np.abs(audio_data), axis=0)

# Create a time axis
time = np.linspace(0, len(audio_data) / sampling_rate, num=len(audio_data))

# Plot original audio signal in time domain
plt.figure(figsize=(10, 6))
plt.plot(time, audio_data)
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.title('Original Audio Signal - Time Domain')
plt.grid()
plt.savefig('original_time_domain_plot.svg', format='svg')
plt.show()

# Plot the audio signal in the frequency domain
# Perform Fourier Transform
audio_fft = np.fft.fft(audio_data)
frequencies = np.fft.fftfreq(len(audio_fft), d=1/sampling_rate)

# Use only the positive frequencies
positive_freqs = frequencies[:len(frequencies)//2]
positive_fft = audio_fft[:len(audio_fft)//2]

# Convert amplitude to dB
amplitude_db = 20 * np.log10(np.abs(positive_fft))

# Plot the frequency domain with log axis for both freq. and ampliitude
plt.figure(figsize=(10, 6))
plt.plot(positive_freqs, amplitude_db, label='Frequency Spectrum')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (dB)')
plt.title('Audio Signal in Frequency Domain')
plt.grid()
plt.savefig('original_frequency_domain_plot.svg', format='svg')
plt.show()
```

```
# TASK - 2 - APPLY FOURIER TRANSFORM (FFT) AND FIND FUNDAMENTAL FREQ., HARMONICS AND NOISE


# Plot the frequency domain representation and mark the fundamental frequency and harmonics range

plt.figure(figsize=(10, 6))

plt.plot(positive_freqs, amplitude_db, label='Frequency Spectrum')

plt.xscale('log')

plt.yscale('log')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude (dB)')

plt.title('Audio Signal in Frequency Domain')

plt.grid()


# Mark the fundamental frequency

fundamental_freq = 168.174

fundamental_amplitude = 20 * np.log10(np.abs(positive_fft[np.argmin(np.abs(positive_freqs - fundamental_freq))]))

plt.plot(fundamental_freq, fundamental_amplitude, 'ro', label='Fundamental Frequency (168.174 Hz)')


# Mark the harmonics range (assumed to be from 2x to 5x of the fundamental frequency)

harmonics_range_start = 2 * fundamental_freq

harmonics_range_end = 5 * fundamental_freq

plt.axvspan(harmonics_range_start, harmonics_range_end, color='yellow', alpha=0.3, label='Harmonics Range')


# Mark the harmonics in the harmonics range

harmonics_freqs = [fundamental_freq * i for i in range(2, 6)]

for harmonic_freq in harmonics_freqs:

    if harmonics_range_start <= harmonic_freq <= harmonics_range_end:

        harmonic_amplitude = 20 * np.log10(np.abs(positive_fft[np.argmin(np.abs(positive_freqs - harmonic_freq))]))

        plt.plot(harmonic_freq, harmonic_amplitude, 'rx', label=f'Harmonic Frequency ({harmonic_freq:.2f} Hz)')


# Mark the noise bands (assumed to be high-frequency bands and low-frequency bands that do not contribute to the perceived sound)

# Noise typically resides in frequencies below 85 Hz and above 8 kHz

low_noise_band_end = 85  # 85 Hz

high_noise_band_start = 8000  # 8 kHz

plt.axvspan(positive_freqs[0], low_noise_band_end, color='red', alpha=0.2, label='Low-Frequency Noise Band < 85Hz')
```

```
plt.axvspan(high_noise_band_start, max(positive_freqs), color='red', alpha=0.2, label='High-Frequency Noise Band > 8KHz')

plt.legend()
plt.savefig('frequency_domain_plot.svg', format='svg')
plt.show()

# TASK - 3 - Improving the quality of the voice

# Isolate the first half second (where the noise is present)
half_second_samples = int(0.5 * sampling_rate)
first_half_second = audio_data[:half_second_samples]

# Perform FFT on the first half second
fft_first_half = np.fft.fft(first_half_second)

# Set the frequency components of the noise to zero
fft_first_half_cleaned = np.zeros_like(fft_first_half)  # Zero out all frequency components

# Perform Inverse FFT to get back the time-domain signal (cleaned-up first half second)
cleaned_first_half_second = np.real(np.fft.ifft(fft_first_half_cleaned))

# Reconstruct the full audio by combining cleaned first half with the rest of the audio
cleaned_audio_data = np.concatenate((cleaned_first_half_second, audio_data[half_second_samples:]))

# Perform FFT on the cleaned full audio
fft_data_cleaned = np.fft.fft(cleaned_audio_data)
fft_freq_cleaned = np.fft.fftfreq(len(fft_data_cleaned), d=1/sampling_rate)

# Custom smoother filter for frequencies between 3kHz and 10kHz with smooth transition
def smooth_transition_filter(fft_freq, cutoff_low, cutoff_high):
    mask = np.ones_like(fft_freq)  # Start by keeping all frequencies
    transition_band = (np.abs(fft_freq) > cutoff_low) & (np.abs(fft_freq) <= cutoff_high)
    mask[transition_band] = (cutoff_high - np.abs(fft_freq[transition_band])) / (cutoff_high - cutoff_low)
    return mask

# Apply the smooth filter (for frequencies between 3kHz and 10kHz)
cutoff_low = 3000  # 3kHz
```

```
cutoff_high = 10000  # 10kHz
filter_mask_cleaned = smooth_transition_filter(fft_freq_cleaned, cutoff_low, cutoff_high)
filtered_fft_data_cleaned = fft_data_cleaned * filter_mask_cleaned

# TASK - 4 - Voice Enhancement Using Aural Exciter

# Apply Aural Exciter using a non-linearity (tanh) in the 3kHz to 10kHz range
def apply_aural_exciter(fft_data, fft_freq, exciter_range=(3000, 10000)):
    exciter_band = (np.abs(fft_freq) >= exciter_range[0]) & (np.abs(fft_freq) <= exciter_range[1])
    excited_fft_data = np.zeros_like(fft_data)  # Start with zeros
    excited_fft_data[exciter_band] = np.tanh(fft_data[exciter_band])  # Applying non-linearity only to this band
    return excited_fft_data

# Apply the exciter to the smooth-filtered FFT data (only between 3kHz and 10kHz)
exciter_range = (3000, 10000)  # Frequency range for the exciter
excited_fft_data = apply_aural_exciter(filtered_fft_data_cleaned, fft_freq_cleaned, exciter_range)

# Add the excited frequencies back to the original signal
scaling_factor = 0.3  # Scale the excited data to add a smaller amount to the original
enhanced_fft_data = fft_data_cleaned + scaling_factor * excited_fft_data  # Add back the enhanced frequencies

# limit the frequency band post-excitation (but ensure frequencies up to 10kHz are preserved)
def limit_frequency_band(fft_data, fft_freq, limit_range=(85, 10000)):
    limited_fft_data = np.copy(fft_data)
    limit_band = (np.abs(fft_freq) > limit_range[1])  # Only limit frequencies above 10kHz
    limited_fft_data[limit_band] = 0  # Zero out frequencies above 10kHz
    return limited_fft_data

# Limiting frequencies above 10kHz
limited_enhanced_fft_data = limit_frequency_band(enhanced_fft_data, fft_freq_cleaned, limit_range=(85,
10000))

# Inverse FFT to return to time domain after excitation and enhancement
final_enhanced_audio_data = np.fft.ifft(limited_enhanced_fft_data)
final_enhanced_audio_data_real = np.real(final_enhanced_audio_data)

# Normalize the final output
def normalize_signal(signal):
```

```
    max_amplitude = np.max(np.abs(signal))
    return signal / max_amplitude if max_amplitude != 0 else signal

final_enhanced_audio_data_normalized = normalize_signal(final_enhanced_audio_data_real)

# Save the final cleaned and enhanced audio
output_file_enhanced = "final_enhanced_audio.wav"
write(output_file_enhanced, sampling_rate, np.int16(final_enhanced_audio_data_normalized * 32767))

# Plot final enhanced audio in time domain
plt.figure(figsize=(10, 6))
plt.plot(time, final_enhanced_audio_data_normalized)
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.title('Final Enhanced Audio - Time Domain')
plt.grid()
plt.savefig('final_enhanced_time_domain.svg', format='svg')
plt.show()

# Plot final enhanced audio in frequency domain
final_enhanced_fft = np.fft.fft(final_enhanced_audio_data_normalized)
plt.figure(figsize=(10, 6))
plt.plot(fft_freq_cleaned[:len(fft_freq_cleaned)//2], 20 *
np.log10(np.abs(final_enhanced_fft[:len(final_enhanced_fft)//2])))
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (dB)')
plt.title('Final Enhanced Audio - Frequency Domain')
plt.grid()
plt.savefig('final_enhanced_frequency_domain.svg', format='svg')
plt.show()

print("Final enhanced audio saved as 'final_enhanced_audio.wav'")
```