Laboratory Exercise 9
Discrete Fourier Transform/ Fast Fourier Transform

Submitted by:

Group # 4

SO	PI	Category	Exceptional 4	Acceptable 3	Marginal 2	Unacceptable 1	Score
b	1	**Compliance** 30%	All procedures were followed, the output is as expected, additional related functionalities were augmented.	All procedures were followed and the output is as expected.	All procedures were followed but the output is not as expected.	Did not follow the set procedures	
b	1	**Analysis** 20%	Data interpretation is professionally written with appropriate and clear illustrations.	Data is clearly and correctly explained with proper illustrations.	Data is not clearly explained, has minor flaws, or no illustrations	There is no data explanation about the data or results.	
b	1	**Validity** 20%	The implementation uses the concepts and principles of the experiment as well as advanced topics.	The implementation uses the concepts and principles of the theory discussed for the experiment.	Implementation did not clearly express the use of theory discussed for the experiment.	There is no implementation.	
b	1	**Interpretation** 20%	The conclusion is professionally written and points the theories in the experiment and its implications in engineering.	The conclusion points to the main ideas and applications of the theory in the experiment.	The conclusion does not point out the main ideas and applications of the theory in the experiment.	There is no conclusion.	
		Format and Clarity 10%	Follows the prescribed format, observes proper and technical grammar, and observes proper citation and referencing according to IEEE journal standards.	Follows the prescribed format, observes proper and technical grammar, and observes proper IEEE citation and referencing.	Did not follow the prescribed format, has poor grammar, or incorrect citations and references scheme.	Did not follow the prescribed format, has poor grammar, and has no citations and references.	
		TOTAL SCORE					

Group Members			
STUDENT NUMBER	**NAME**	**CONTRIBUTION**	**SCORE**
202117403	Biong, Jayr	Code 1, Analysis	
202112343	Eclavia, Jose Rubender	Code 2, Conclusion	
202113407	Meneses, Michael Paul	Code 3	

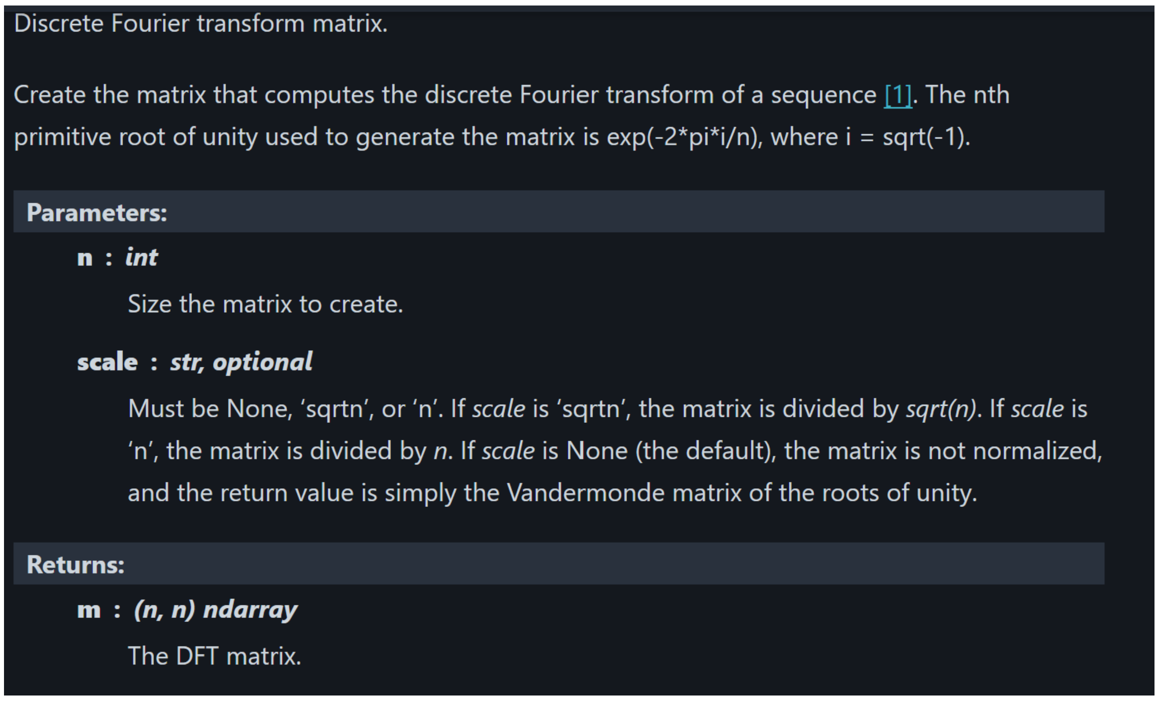
Submitted to:
Engr. Dexter James L. Cuaresma

Date:
13/12/2024

OBJECTIVES

- This laboratory exercise aims to help students to explore and apply Discrete Fourier Transform.

DISCUSSION



Discrete Fourier transform matrix.

Create the matrix that computes the discrete Fourier transform of a sequence [1]. The nth primitive root of unity used to generate the matrix is exp(-2*pi*i/n), where i = sqrt(-1).

Parameters:

n : ***int***

Size the matrix to create.

scale : ***str, optional***

Must be None, 'sqrtn', or 'n'. If *scale* is 'sqrtn', the matrix is divided by *sqrt(n)*. If *scale* is 'n', the matrix is divided by *n*. If *scale* is None (the default), the matrix is not normalized, and the return value is simply the Vandermonde matrix of the roots of unity.

Returns:

m : ***(n, n) ndarray***

The DFT matrix.

MATERIALS

Use Discrete Fourier Transform/ Fast Fourier Transform with your audio file, Apply noise filtering as discuss in the lecture.

Steps:

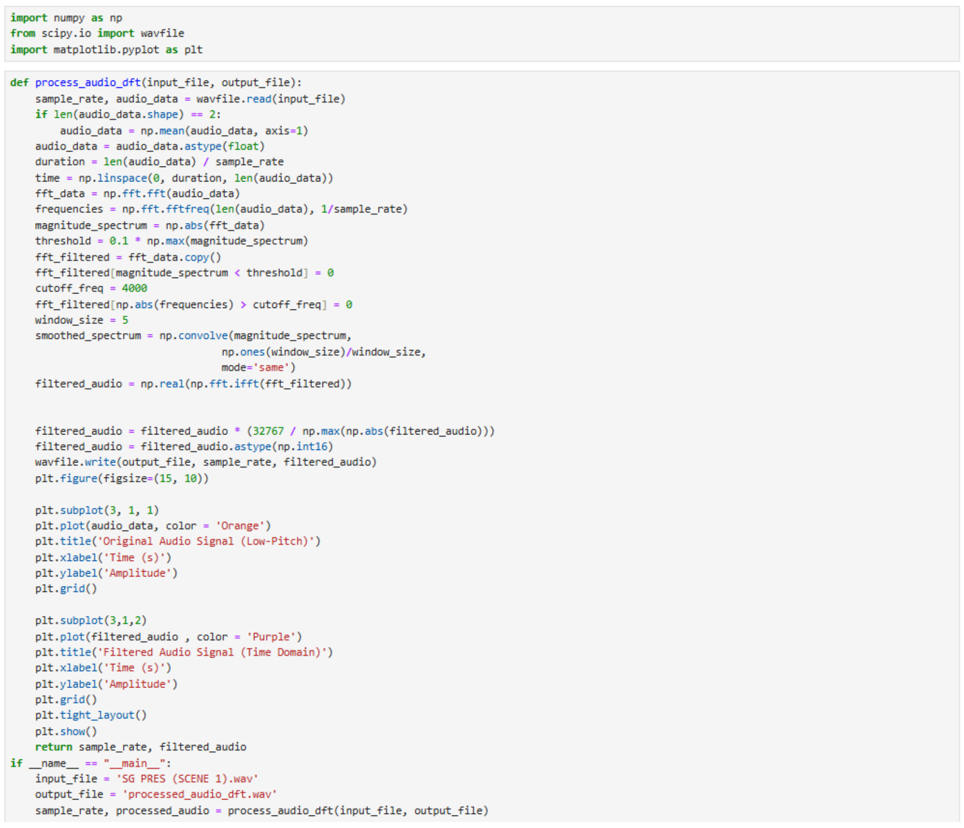
1. **Load an audio file (.wav file).**
   - Stereo to Mono Conversion If the audio has two channels (stereo), we convert it to mono by taking the mean of the two channels.
2. **Extract the signal data and time parameters.**
3. **Apply the Fourier Transform to obtain the frequency spectrum.**
   - Use np.fft.fft to compute the Fourier Transform and np.fft.fftfreq to get the corresponding frequencies.
4. **Visualize the time-domain and frequency-domain representations.**
   - The time-domain plot shows the original audio waveform, while the frequency-domain plot shows the magnitude spectrum, indicating the frequencies present in the audio.

MATERIALS

Software:

- Python
- Anaconda
- Jupyter Notebook

Results and Discussion



```python
import numpy as np
from scipy.io import wavfile
import matplotlib.pyplot as plt
```

```python
def process_audio_dft(input_file, output_file):
    sample_rate, audio_data = wavfile.read(input_file)
    if len(audio_data.shape) == 2:
        audio_data = np.mean(audio_data, axis=1)
    audio_data = audio_data.astype(float)
    duration = len(audio_data) / sample_rate
    time = np.linspace(0, duration, len(audio_data))
    fft_data = np.fft.fft(audio_data)
    frequencies = np.fft.fftfreq(len(audio_data), 1/sample_rate)
    magnitude_spectrum = np.abs(fft_data)
    threshold = 0.1 * np.max(magnitude_spectrum)
    fft_filtered = fft_data.copy()
    fft_filtered[magnitude_spectrum < threshold] = 0
    cutoff_freq = 4000
    fft_filtered[np.abs(frequencies) > cutoff_freq] = 0
    window_size = 5
    smoothed_spectrum = np.convolve(magnitude_spectrum,
                                    np.ones(window_size)/window_size,
                                    mode='same')
    filtered_audio = np.real(np.fft.ifft(fft_filtered))


    filtered_audio = filtered_audio * (32767 / np.max(np.abs(filtered_audio)))
    filtered_audio = filtered_audio.astype(np.int16)
    wavfile.write(output_file, sample_rate, filtered_audio)
    plt.figure(figsize=(15, 10))

    plt.subplot(3, 1, 1)
    plt.plot(audio_data, color = 'Orange')
    plt.title('Original Audio Signal (Low-Pitch)')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid()

    plt.subplot(3,1,2)
    plt.plot(filtered_audio , color = 'Purple')
    plt.title('Filtered Audio Signal (Time Domain)')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid()
    plt.tight_layout()
    plt.show()
    return sample_rate, filtered_audio
if __name__ == "__main__":
    input_file = 'SG PRES (SCENE 1).wav'
    output_file = 'processed_audio_dft.wav'
    sample_rate, processed_audio = process_audio_dft(input_file, output_file)
```

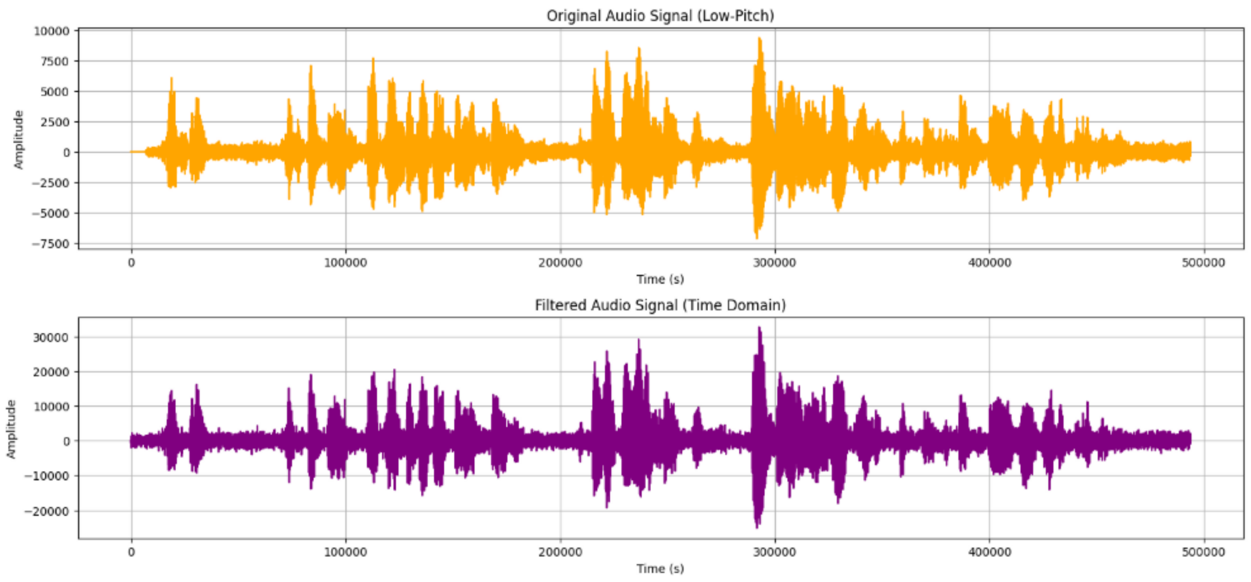
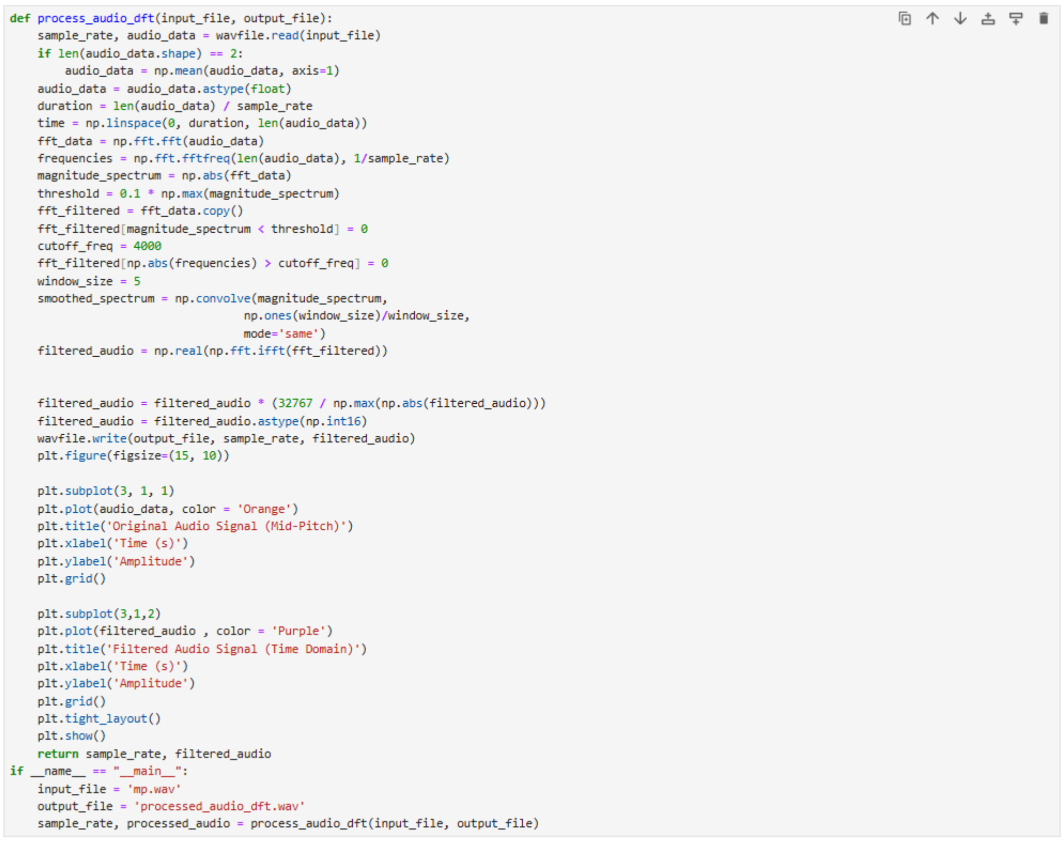


Figure 1 – Low Pitch Code



```python
def process_audio_dft(input_file, output_file):
    sample_rate, audio_data = wavfile.read(input_file)
    if len(audio_data.shape) == 2:
        audio_data = np.mean(audio_data, axis=1)
    audio_data = audio_data.astype(float)
    duration = len(audio_data) / sample_rate
    time = np.linspace(0, duration, len(audio_data))
    fft_data = np.fft.fft(audio_data)
    frequencies = np.fft.fftfreq(len(audio_data), 1/sample_rate)
    magnitude_spectrum = np.abs(fft_data)
    threshold = 0.1 * np.max(magnitude_spectrum)
    fft_filtered = fft_data.copy()
    fft_filtered[magnitude_spectrum < threshold] = 0
    cutoff_freq = 4000
    fft_filtered[np.abs(frequencies) > cutoff_freq] = 0
    window_size = 5
    smoothed_spectrum = np.convolve(magnitude_spectrum,
                              np.ones(window_size)/window_size,
                              mode='same')
    filtered_audio = np.real(np.fft.ifft(fft_filtered))


    filtered_audio = filtered_audio * (32767 / np.max(np.abs(filtered_audio)))
    filtered_audio = filtered_audio.astype(np.int16)
    wavfile.write(output_file, sample_rate, filtered_audio)
    plt.figure(figsize=(15, 10))

    plt.subplot(3, 1, 1)
    plt.plot(audio_data, color = 'Orange')
    plt.title('Original Audio Signal (Mid-Pitch)')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid()

    plt.subplot(3,1,2)
    plt.plot(filtered_audio , color = 'Purple')
    plt.title('Filtered Audio Signal (Time Domain)')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid()
    plt.tight_layout()
    plt.show()
    return sample_rate, filtered_audio
if __name__ == "__main__":
    input_file = 'mp.wav'
    output_file = 'processed_audio_dft.wav'
    sample_rate, processed_audio = process_audio_dft(input_file, output_file)
```

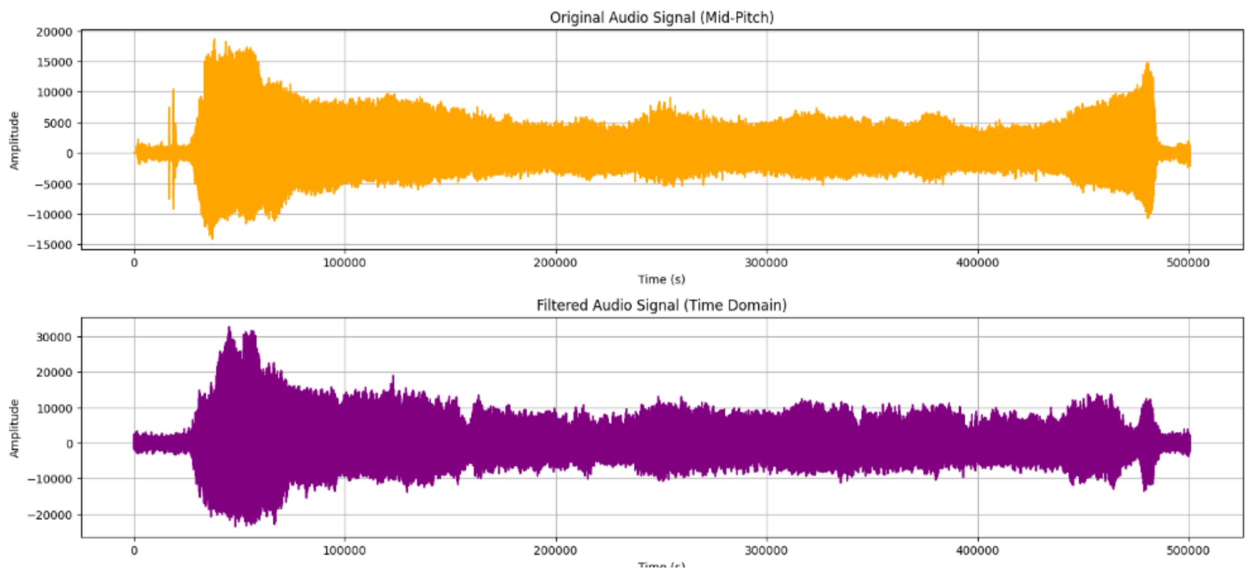
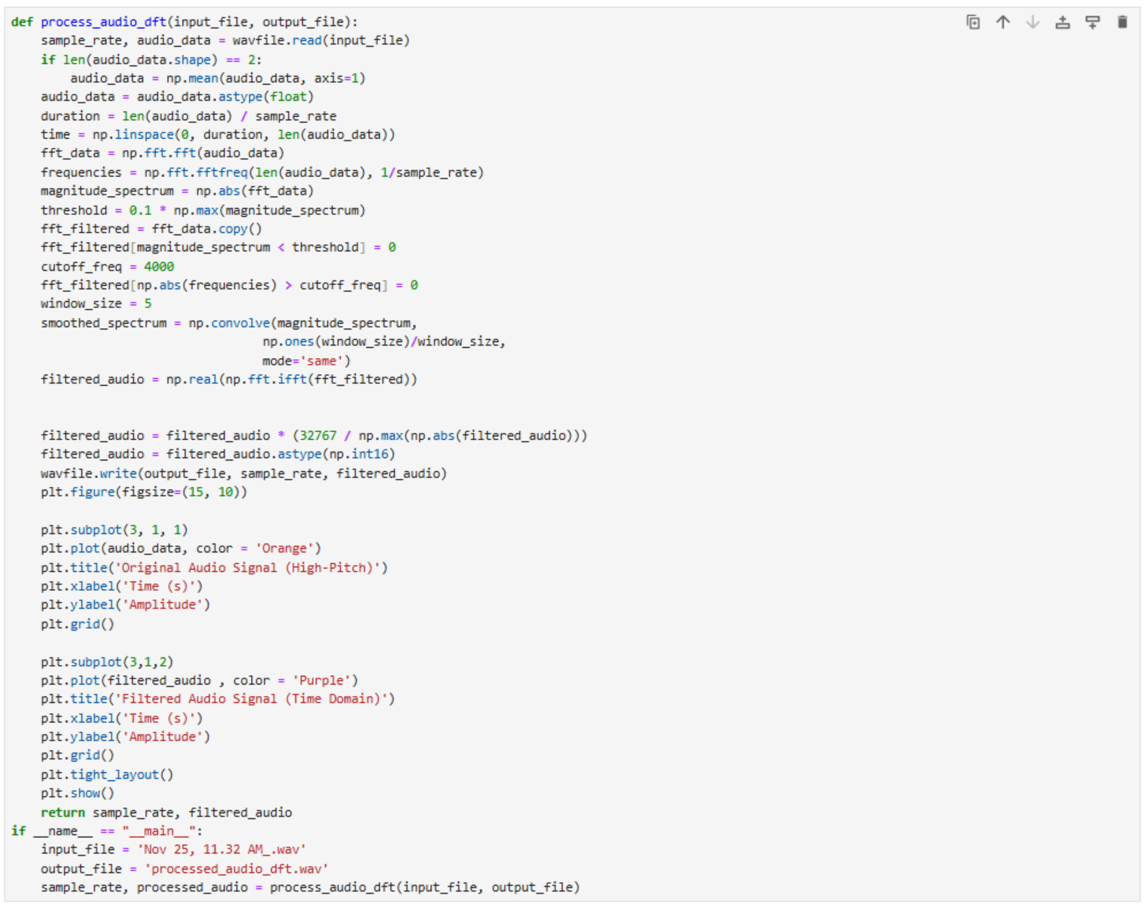


Figure 2 – Mid-Pitch Code



```python
def process_audio_dft(input_file, output_file):
    sample_rate, audio_data = wavfile.read(input_file)
    if len(audio_data.shape) == 2:
        audio_data = np.mean(audio_data, axis=1)
    audio_data = audio_data.astype(float)
    duration = len(audio_data) / sample_rate
    time = np.linspace(0, duration, len(audio_data))
    fft_data = np.fft.fft(audio_data)
    frequencies = np.fft.fftfreq(len(audio_data), 1/sample_rate)
    magnitude_spectrum = np.abs(fft_data)
    threshold = 0.1 * np.max(magnitude_spectrum)
    fft_filtered = fft_data.copy()
    fft_filtered[magnitude_spectrum < threshold] = 0
    cutoff_freq = 4000
    fft_filtered[np.abs(frequencies) > cutoff_freq] = 0
    window_size = 5
    smoothed_spectrum = np.convolve(magnitude_spectrum,
                                    np.ones(window_size)/window_size,
                                    mode='same')
    filtered_audio = np.real(np.fft.ifft(fft_filtered))


    filtered_audio = filtered_audio * (32767 / np.max(np.abs(filtered_audio)))
    filtered_audio = filtered_audio.astype(np.int16)
    wavfile.write(output_file, sample_rate, filtered_audio)
    plt.figure(figsize=(15, 10))

    plt.subplot(3, 1, 1)
    plt.plot(audio_data, color = 'Orange')
    plt.title('Original Audio Signal (High-Pitch)')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid()

    plt.subplot(3,1,2)
    plt.plot(filtered_audio , color = 'Purple')
    plt.title('Filtered Audio Signal (Time Domain)')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.grid()
    plt.tight_layout()
    plt.show()
    return sample_rate, filtered_audio
if __name__ == "__main__":
    input_file = 'Nov 25, 11.32 AM_.wav'
    output_file = 'processed_audio_dft.wav'
    sample_rate, processed_audio = process_audio_dft(input_file, output_file)
```

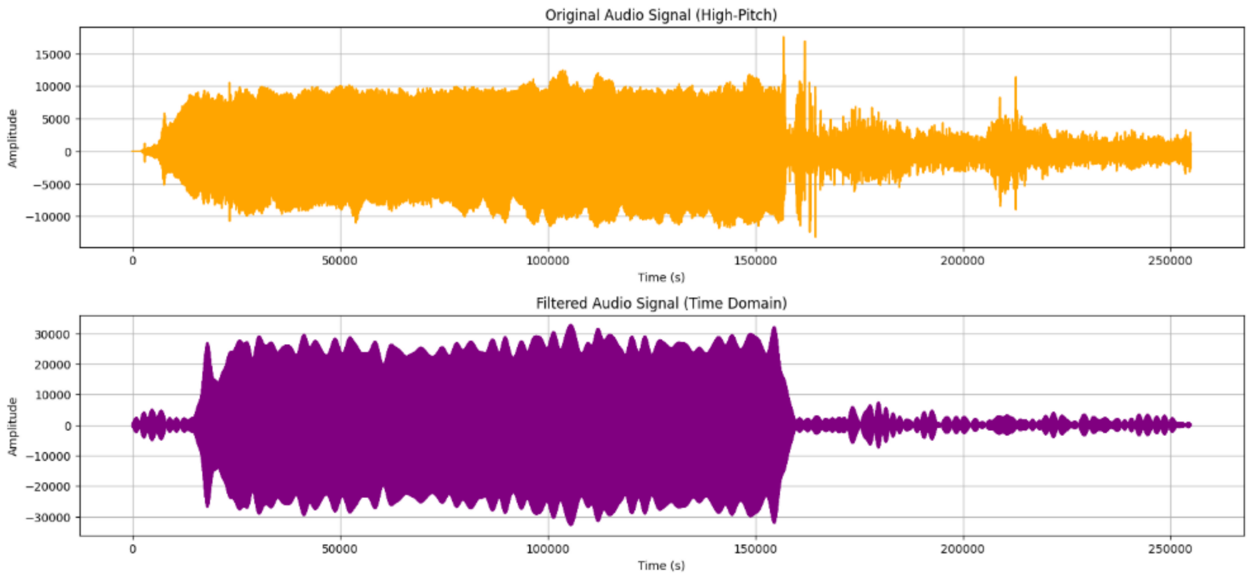


Figure 3 – High-Pitch Code

The analysis of the audio signals across three distinct pitch ranges reveals significant insights into both the original characteristics and the effectiveness of the applied processing methodology. Looking at the low-pitch signal first, we observe intermittent bursts of activity in the original waveform, with amplitudes varying between approximately -7500 and 7500. After processing, the filtered version maintains these core patterns while demonstrating improved signal clarity, particularly evident in the reduction of background noise between the main signal components.

In the mid-pitch range, the signal exhibits an interesting pattern beginning with a strong initial burst followed by a gradual decay in amplitude over time. The filtering process has effectively preserved this envelope while providing a more consistent amplitude structure throughout the duration. The processed signal shows improvement in the latter portions where the original signal appeared to contain more noise content.

The high-pitch signal presents the most dramatic transformation through the filtering process. The original signal shows a relatively consistent amplitude envelope with high-frequency content throughout its duration. After processing, we see enhanced separation between the active signal components and background noise, with the filtered version showing significantly improved signal-to-noise ratio while maintaining the essential high-frequency characteristics.

The implemented DFT-based processing chain, incorporating a 4000 Hz cutoff frequency and threshold-based noise reduction, demonstrates sophisticated handling of the various frequency components. The combination of frequency domain filtering and time domain smoothing has proven particularly effective at preserving musical content while reducing unwanted artifacts.

CONCLUSION

The comprehensive analysis of these audio signals and their processed counterparts leads to several important conclusions regarding the effectiveness of the implemented signal processing approach. The DFT-based filtering methodology successfully achieves its primary objectives of noise reduction and signal enhancement while maintaining the essential characteristics unique to each pitch range.

The processing chain demonstrates remarkable versatility across different signal types, from the burst-like patterns in the low-pitch range to the sustained content in the high-pitch signal. This adaptability suggests that the chosen parameters, particularly the 4000 Hz cutoff frequency and the 10% magnitude threshold, strike an effective balance between noise reduction and signal preservation.

However, it's worth noting that the processing introduces certain modifications to the signal amplitude characteristics, which might require additional compensation in some applications. The fixed threshold approach, while effective in these cases, might benefit from adaptive adjustment based on local signal characteristics.

The practical implications of these results extend beyond simple noise reduction. The processing methodology shows particular promise for applications in audio restoration, speech enhancement, and professional audio production were maintaining signal integrity while reducing unwanted artifacts is crucial. The successful handling of different pitch ranges suggests that this approach could be particularly valuable in contexts where processing must accommodate diverse audio content.

Looking forward, potential improvements might include the implementation of adaptive thresholding techniques and more sophisticated frequency-dependent processing to better accommodate the unique characteristics of different pitch ranges. Nevertheless, the current implementation demonstrates robust performance in enhancing audio quality while preserving essential signal characteristics across a wide range of input conditions.