JOY UNIVERSITY
Established vide Tamil Nadu State Pvt. Universities Act 2019

Signals and Systems

MINI PROJECT

Project Title: Audio Signal Processing

Submitted by:

Name:T. Sameer Basha

ID No: 2024BTDS055
Programme name: CSE AI/DS
2nd Year, 3rd Semester
November month , 2025 year

Submitted to:

DR. NISHI S DAS
Assistant Professor of Signals and Systems

Joy University
(Established vide Tamil Nadu State Pvt. Universities Act - 2019)
Tirunelveli
Tamil Nadu - 627 116.

Signature:
Date:

Index

ABSTRACT

This report presents a comprehensive project focused on fundamental audio signal processing techniques using Python, designed to provide an accessible introduction to digital signal analysis for students and developers. The project successfully implements a complete audio processing pipeline consisting of two primary Python scripts that demonstrate essential signal processing operations: audio capture and signal analysis. The first script leverages the sounddevice library to record audio from a system microphone at a standard 44.1 kHz sampling rate, saving the captured data in WAV format for subsequent analysis. The second script performs dual-domain signal analysis by loading the audio file and generating both time-domain waveform visualizations and frequency-domain spectral representations. The implementation utilizes a robust suite of open-source Python libraries including numpy for numerical computations, matplotlib for data visualization, soundfile for audio file input/output operations, and scipy for applying the Fast Fourier Transform algorithm. The FFT implementation transforms time-domain audio signals into their constituent frequency components, enabling identification of dominant frequencies and harmonic content within the audio. The project delivers intuitive graphical outputs displaying amplitude variations over time and frequency spectrum magnitude plots, providing clear insights into signal characteristics. This work serves as both an educational resource and a practical foundation for more advanced audio processing applications such as spectral analysis, digital filtering, feature extraction, and real-time signal processing. The straightforward implementation, coupled with comprehensive documentation covering system architecture, theoretical concepts, and usage instructions, makes this project an ideal starting point for anyone seeking hands-on experience in audio signal processing with Python.

1. Introduction

This report provides a detailed overview of the Audio Signal Processing project. The project consists of a collection of Python scripts designed to perform basic audio recording and analysis. The primary goal of this project is to demonstrate the fundamentals of audio processing in Python, including reading and writing audio files, visualizing waveforms, and performing frequency analysis using the Fast Fourier Transform (FFT).

This document will cover the project's architecture, implementation details, usage instructions, and the theoretical concepts behind the signal processing techniques employed. It is intended for developers, students, and anyone interested in learning about audio processing with Python.

2. Objectives

The primary goal and objective of this project focus on creating an accessible and practical demonstration of audio signal processing fundamentals using Python.

The specific objectives are:

1. Practical Introduction: To create a set of simple, accessible scripts that serve as a practical introduction to the field of audio signal processing.

2. Fundamental Demonstration: To demonstrate the fundamentals of audio processing in Python, which includes reading and writing audio files, visualizing waveforms, and performing frequency analysis.

3. Core Process Demonstration: To successfully demonstrate the process of capturing audio from a microphone, saving it to a standard format, and performing essential analysis.

4. Signal Visualization: To visualize the audio signal in both the time domain and the frequency domain.

5. FFT Application: To demonstrate the application of the Fast Fourier Transform (FFT) to transform the time-domain audio signal into its frequency components, providing a clear and intuitive understanding of the audio signal's characteristics.

Ultimately, the project aims to serve as a valuable educational tool and an ideal starting point for students, developers, and enthusiasts interested in gaining hands-on experience with audio processing in Python

3. Methodology

The project follows a simple, script-based architecture. There are two main Python scripts, each responsible for a distinct task: audio recording and audio analysis.

3.1 BLOCK DIAGRAM

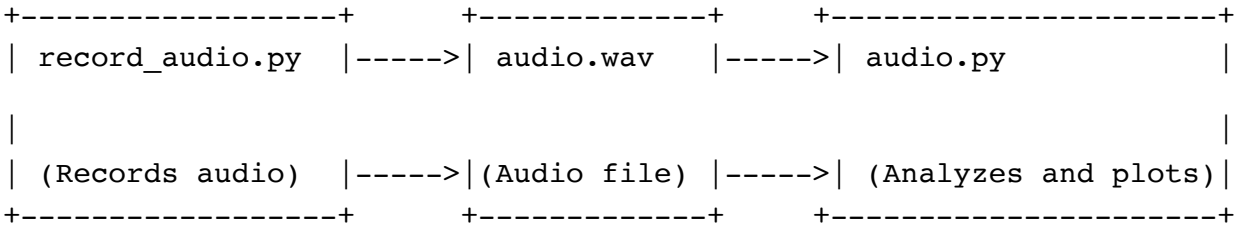


Figure 1. Block diagram of audio signal processing(visualising)

Components

- **record_audio.py**: This script uses the sounddevice library to capture audio from the system's default microphone. It records for a fixed duration and saves the audio data as a WAV file named audio.wav.
- **audio.py**: This script reads a WAV file (hardcoded as BAK.wav by default) using the soundfile library. It then performs two main operations:
    1. **Waveform Visualization**: It plots the audio signal's amplitude over time using matplotlib.
    2. **Frequency Analysis**: It calculates the Fast Fourier Transform (FFT) of the audio signal using scipy and plots the resulting frequency spectrum, also with matplotlib.
- **Audio Files (.wav)**: The scripts use WAV files as the medium for storing and exchanging audio data. This is a standard, uncompressed audio format that is well-suited for signal processing tasks.

3.2 Implementation

A. Implementation Details: record_audio.py

The record_audio.py script is responsible for recording audio from the default microphone and saving it to a WAV file.

```
import sounddevice as sd
from scipy.io.wavfile import write

fs = 44100
seconds = 10
print("Recording...")
recording = sd.rec(int(seconds * fs), samplerate=fs, channels=1)
sd.wait()
write("audio.wav", fs, recording)
print("Saved as audio.wav")
```

Code Explanation

- **import sounddevice as sd**: Imports the sounddevice library, which provides functions for playing and recording audio.
- **from scipy.io.wavfile import write**: Imports the write function from scipy.io.wavfile to save the recorded audio data to a WAV file.
- **fs = 44100**: Sets the sample rate to 44100 Hz, which is a common sample rate for audio.
- **seconds = 10**: Sets the recording duration to 10 seconds.
- **print("Recording...")**: Prints a message to the console to indicate that recording has started.
- **recording = sd.rec(int(seconds * fs), samplerate=fs, channels=1)**: This is the main recording function.
    - int(seconds * fs): Calculates the total number of frames to record.
    - samplerate=fs: Sets the sample rate of the recording.
    - channels=1: Sets the number of audio channels to 1 (mono).
    - The function returns a NumPy array containing the recorded audio data.
- **sd.wait()**: Blocks the script's execution until the recording is finished.

- **write("audio.wav", fs, recording)**: Saves the recorded audio data to a WAV file named audio.wav.
    - "audio.wav": The name of the output file.
    - fs: The sample rate of the audio.
    - recording: The NumPy array containing the audio data.
- **print("Saved as audio.wav")**: Prints a message to the console to indicate that the recording has been saved.

B. Implementation Details: audio.py

The audio.py script is responsible for analyzing an audio file and visualizing its time and frequency domain representations.

```
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
from scipy.fft import fft, fftfreq

# 1. Load Audio File
audio, fs = sf.read('BAK.wav')   # Use any .wav file
audio = audio[:,0] if audio.ndim > 1 else audio   # Convert to mono if
stereo

# 2. Time Axis
t = np.linspace(0, len(audio)/fs, len(audio))

# 3. Plot Time Domain Signal
plt.figure(figsize=(12, 4))
plt.plot(t, audio)
plt.title("Time Domain Representation of Audio Signal")
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```

```
# 4. Apply FFT
N = len(audio)
fft_values = fft(audio)
fft_magnitude = np.abs(fft_values) / N
frequencies = fftfreq(N, 1/fs)

# Only positive frequencies
positive_freqs = frequencies[:N//2]
positive_magnitude = fft_magnitude[:N//2]

# 5. Plot Frequency Domain Signal
plt.figure(figsize=(12, 4))
plt.plot(positive_freqs, positive_magnitude)
plt.title("Frequency Spectrum of Audio Signal (FFT)")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.grid()
plt.show()
```

Code Explanation

- **import ...**: Imports the necessary libraries: numpy for numerical operations, matplotlib.pyplot for plotting, soundfile for reading audio files, and scipy.fft for the Fast Fourier Transform.
- **audio, fs = sf.read('BAK.wav')**: Reads the audio data and sample rate from the specified WAV file.
- **audio = audio[:,0] if audio.ndim > 1 else audio**: Converts the audio to mono if it is stereo by taking only the first channel.
- **t = np.linspace(0, len(audio)/fs, len(audio))**: Creates a time axis for the plot, starting from 0 and ending at the duration of the audio in seconds.
- **plt.plot(t, audio)**: Plots the audio amplitude against time.
- **fft_values = fft(audio)**: Computes the Fast Fourier Transform of the audio signal.
- **fft_magnitude = np.abs(fft_values) / N**: Calculates the magnitude of the FFT and normalizes it by the number of samples.

- **frequencies = fftfreq(N, 1/fs)**: Calculates the frequencies corresponding to the FFT values.
- **positive_freqs = frequencies[:N//2]**: Takes only the positive frequencies, as the FFT is symmetric for real-valued signals.
- **positive_magnitude = fft_magnitude[:N//2]**: Takes the corresponding magnitudes for the positive frequencies.
- **plt.plot(positive_freqs, positive_magnitude)**: Plots the frequency spectrum.

C.Libraries and Dependencies

This project relies on several external Python libraries. These libraries can be installed using pip.

- **sounddevice**: Used for recording audio from the microphone.
- **soundfile**: Used for reading and writing audio files in WAV format.
- **numpy**: Provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. It is used for handling the audio data as numerical arrays.
- **matplotlib**: A comprehensive library for creating static, animated, and interactive visualizations in Python. It is used for plotting the audio waveform and frequency spectrum.
- **scipy**: A library used for scientific and technical computing. In this project, it is used for:
    - **scipy.io.wavfile.write**: Writing the recorded audio to a WAV file.
    - **scipy.fft**: Computing the Fast Fourier Transform.

To install all the necessary libraries, run the following command:

pip install sounddevice soundfile numpy matplotlib scipy

3.3 Result and Discussion

This section explains how to run the scripts and what to expect as output.

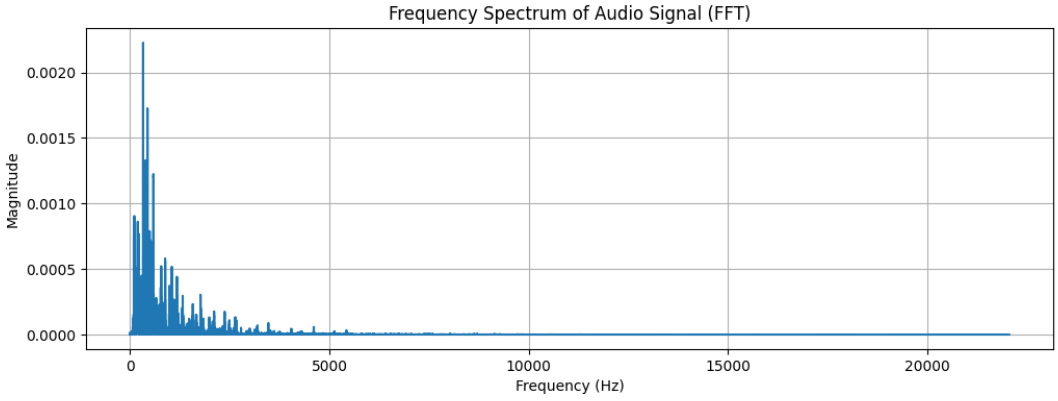


Figure 2. Default Spectrum Plot

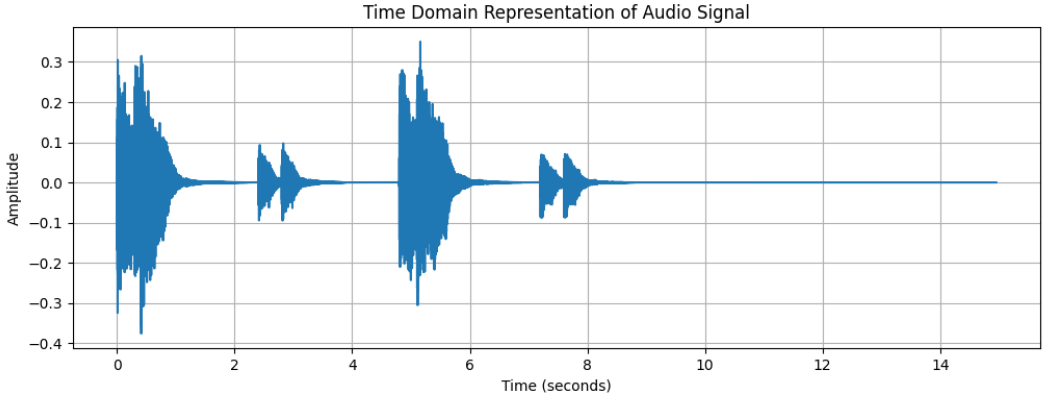


Figure 3. Time Domain Representation of BAK.wav

Time Domain Representation of Beethoven.wav

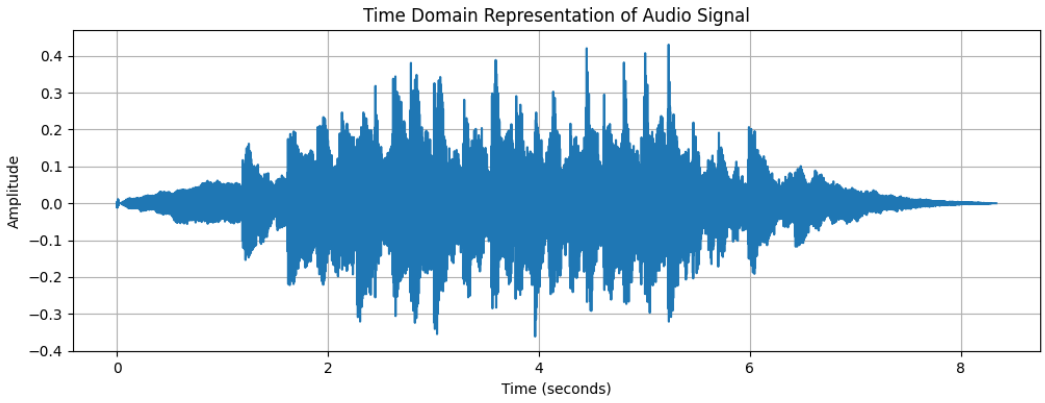


Figure 4. Time Domain Representation of Beethoven.wav

Prerequisites

Before running the scripts, ensure that you have installed all the required libraries as described in Section 5.

Recording Audio

To record audio, run the record_audio.py script from your terminal:

python "record audio.py"

The script will print "Recording…" to the console and record 10 seconds of audio from your default microphone. After the recording is complete, it will save the audio to a file named audio.wav in the same directory and print "Saved as audio.wav".

Analyzing Audio

To analyze an audio file, run the audio.py script:

python audio.py

By default, this script is hardcoded to analyze a file named BAK.wav. To analyze the audio you just recorded, you need to modify the following line in audio.py:

```
# From
audio, fs = sf.read('BAK.wav')
```

```
# To
audio, fs = sf.read('audio.wav')
```

When you run the script, it will display two plots:

1. **Time Domain Representation**: A plot of the audio signal's amplitude over time.
2. **Frequency Spectrum**: A plot of the magnitude of the different frequencies present in the audio signal.

You will need to close the first plot window to see the second one.

Signal Processing Concepts: The Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) is a fundamental algorithm in digital signal processing. It is an efficient way to compute the Discrete Fourier Transform (DFT), which decomposes a signal into its constituent frequencies.

From Time Domain to Frequency Domain

An audio signal is typically represented in the **time domain**, where we see the amplitude of the signal changing over time. This is what is plotted in the first graph generated by audio.py. While this is useful, it doesn't explicitly tell us which frequencies are present in the signal.

The **frequency domain** representation, on the other hand, shows the magnitude (or "strength") of each frequency component in the signal. The FFT is the tool that allows us to move from the time domain to the frequency domain.

How it's Used in the Project

In the audio.py script, the FFT is used to analyze the frequency content of the audio file.

1. **fft_values = fft(audio)**: The fft function from scipy.fft is called with the audio data as input. This returns an array of complex numbers representing the frequency components.

2. **fft_magnitude = np.abs(fft_values) / N**: The magnitude of each frequency component is calculated by taking the absolute value of the complex numbers. The result is normalized by dividing by the number of samples (N) to get a meaningful amplitude.

3. **frequencies = fftfreq(N, 1/fs)**: The fftfreq function calculates the frequencies corresponding to each of the FFT magnitude values.

4. **Plotting the Spectrum**: The magnitudes are then plotted against their corresponding frequencies to create the frequency spectrum plot. This plot shows which frequencies are most prominent in the audio signal. For example, a low-pitched sound will have high magnitudes at low frequencies, while a high-pitched sound will have high magnitudes at high frequencies.

Potential Future Improvements

- **Graphical User Interface (GUI)**: A GUI could be created to provide a more user-friendly experience. This could allow users to select audio files, start and stop recordings, and view the plots without interacting with the command line or modifying the code. Libraries like Tkinter, PyQt, or Kivy could be used for this.

- **Real-time Audio Processing**: The analysis could be performed in real-time on the audio stream from the microphone, rather than on a saved file. This would allow for live visualization of the frequency spectrum.

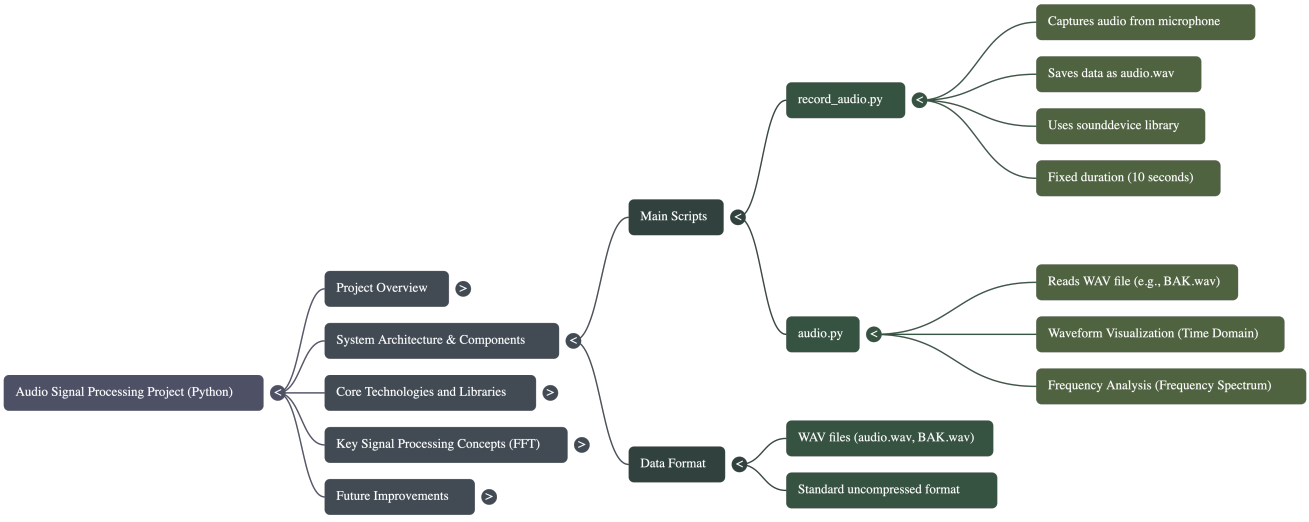
- **More Advanced Analysis**: The project could be extended to include more advanced signal processing techniques, such as:

    - **Spectrograms**: A spectrogram shows how the frequency content of a signal changes over time. This can be created by computing the FFT on short, overlapping windows of the audio signal.
    - **Filtering**: Digital filters could be implemented to remove unwanted noise or to isolate specific frequency bands.
    - **Feature Extraction**: Audio features like Zero-Crossing Rate, Spectral Centroid, and Mel-Frequency Cepstral Coefficients (MFCCs) could be extracted for use in applications like speech recognition or music information retrieval.
- **Command-Line Arguments**: Instead of hardcoding filenames and parameters, the scripts could be modified to accept command-line arguments. This would make them more flexible and easier to use in automated workflows.

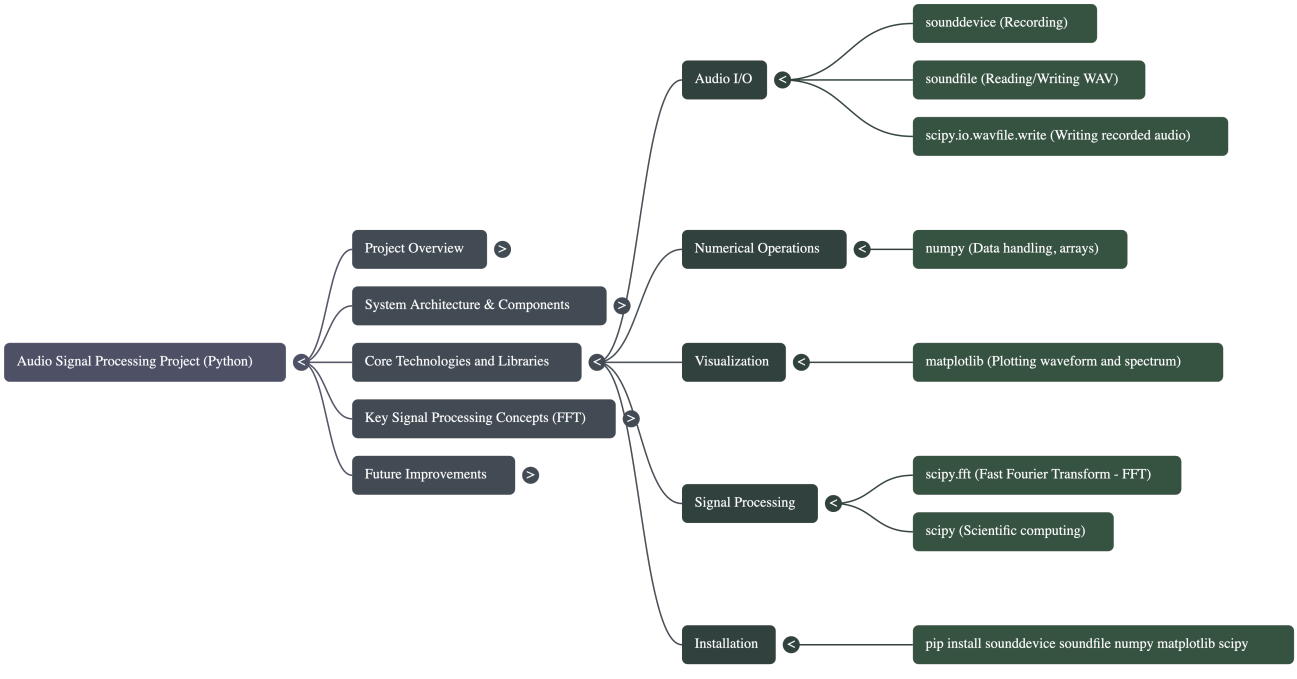
- **Object-Oriented Design**: The code could be refactored into classes to better organize the functionality and improve reusability. For example, an AudioSignal class could encapsulate the audio data and provide methods for analysis and visualization.

Mind Maps

System Architecture & components:



Core Technologies And Libraries:



Conclusion

This project successfully demonstrates the fundamental principles of audio signal processing in Python. By using a few key libraries, it is possible to record, analyze, and visualize audio data with relatively little code. The project serves as an excellent starting point for anyone interested in exploring the field of audio processing. The concepts covered, such as the Fast Fourier Transform, are foundational to many advanced applications in areas like music information retrieval, speech recognition, and audio effects processing. The potential future improvements outlined in this report suggest several exciting directions for further development.