Scilab case study project on

Design and Analysis of an Audio Equalizer

Vishesh Vinod Munghate

Visvesvaraya National Institute of Technology Nagpur

Digital Signal Processing

August 28, 2024

Abstract

The objective of this case study is to develop a comprehensive understanding of digital signal processing within a computational environment. This study involves the detailed analysis of digital audio signals using a logic-driven approach grounded in mathematical principles. The primary focus is on analyzing audio signals in both the time and frequency domains, ultimately leading to the creation of a Scilab-based graphical user interface (GUI) for an audio equalizer. This equalizer is designed to perform key operations such as reading and playing audio files, visualizing signals in time and frequency domains, and applying various filters to manipulate the audio signal.

1. Introduction

In the field of digital signal processing, it's important to understand how audio signals can be analyzed and adjusted. This case study explores the process of converting an analog audio signal into a digital one, allowing for detailed examination in both time and frequency domains. Using the concepts of Fourier transform and discrete Fourier transform, the project focuses on creating a Scilab-based graphical user interface (GUI) that functions as an audio equalizer. This equalizer can read and play audio files, as well as apply different filters to change the sound. By combining theory with practical application, this study aims to provide a clear understanding of how digital signal processing techniques can be used in everyday situations.

2. Problem Statement

The problem at hand involves the analysis of a given audio file through digital signal processing techniques. The goal is to examine the audio file in both the time and frequency domains and develop a Scilab-based graphical user interface (GUI) that functions as an audio equalizer. The equalizer should be capable of performing the following tasks:

1. Reading the audio file.
2. Playing the audio file.
3. Displaying the input signal in both the time and frequency domains.
4. Designing five filters: one low-pass filter, one high-pass filter, and three band-pass filters.
5. Applying these filters to the input signal.
6. Displaying the filtered signal in the frequency domain.

This problem requires a comprehensive understanding of digital signal processing principles and the ability to implement them effectively within a computational environment.

3. Basic concepts related to the topic

The concepts of Fourier transform and discrete Fourier transform are fundamental to understanding digital signal processing. A signal can be defined as the variability of any physical value that can be represented as a function of one or more variables. In this context, we focus on one-dimensional time functions. In the real world, time functions typically exist in the continuous domain. However, with advancements in computer science, analog signal processing has become less common. It is now more cost-effective to develop, implement, and test signal processing algorithms in the digital realm than to design and create analog (electronic) devices.

From Continuous to Discrete Domain: To digitally represent an analog signal, it must be converted into the discrete-time domain and quantized. The Nyquist-Shannon sampling theorem serves as the connection between continuous-time signals and discrete-time signals. This theorem addresses how to sample a continuous-time signal to obtain a discrete-time signal, from which the original continuous-time signal can be accurately reconstructed.

According to the theorem, to achieve a properly sampled discrete-time signal, the sampling frequency must be at least twice the highest frequency present in the original signal.

If a function f(x) is periodic with a period T and integrable (i.e., its integral is finite) over the interval [x0, x0 + T] it can be expressed as a series.

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cdot \cos\left(\frac{2n\pi}{T} x\right) + b_n \cdot \sin\left(\frac{2n\pi}{T} x\right) \right)$$

where

$$\mathrm{a}_n = \frac{2}{T} \int_{x_0}^{x_0+T} f(x) \cdot \cos\left(\frac{2n\pi}{T} x\right) dx$$

$$\mathrm{b}_n = \frac{2}{T} \int_{x_0}^{x_0+T} f(x) \cdot \sin\left(\frac{2n\pi}{T} x\right) dx$$

Discrete Fourier Transform:

The Fourier series is considered the precursor to the Fourier Transform. For digital signals, the Discrete Fourier Transform (DFT) is described by the following formula:

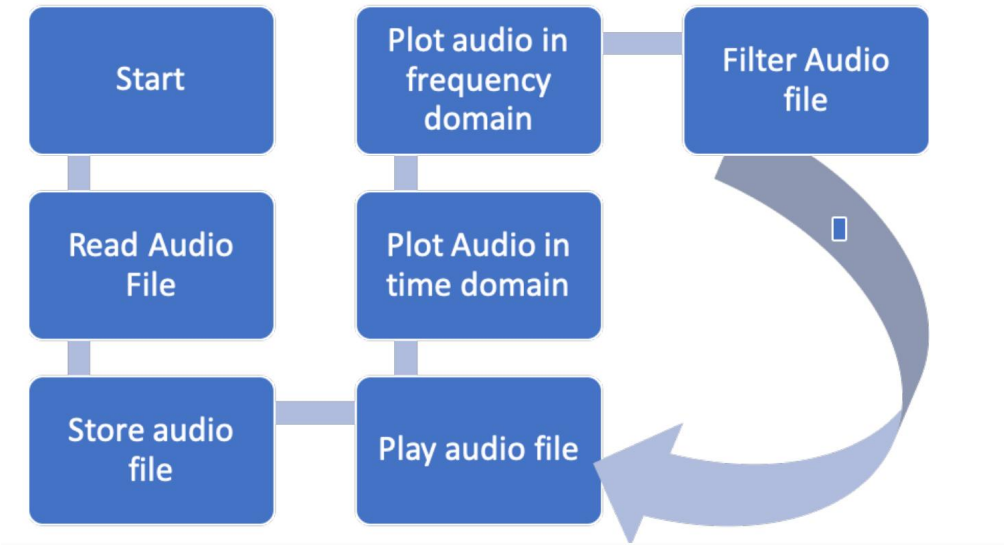
$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) \cdot e^{-j \frac{2\pi}{N} kn}$$

Fourier transformation is reversible, allowing us to return to the time domain using the formula:

$$x(n) = \sum_{k=0}^{N-1} X(k) \cdot e^{j \frac{2\pi}{N} kn}$$

The DFT converts a finite sequence of equally spaced samples of a function into a sequence of equally spaced samples of the Discrete-Time Fourier Transform (DTFT), which is a complex-valued function of frequency. The sampling interval of the DTFT is the reciprocal of the duration of the input sequence.

4. Flowchart



5. Software/Hardware used

Operating System: Windows 10 Home

Scilab Version: 2024.1.0

Toolbox Used: None

Hardware: MacBook Air 2019 with Windows 10 Home installed via Boot Camp

6. Procedure of execution

No dependency files have been used. All three `.sce` files are independent of each other. They are as follows:

1. main.sce: GUI-based equalizer.
2. reverse.sce: Reverses the chosen audio file and plays the reversed audio.
3. fft.sce: Plots the frequency domain of the chosen audio file using the FFT function and the `plot2d` function.

Steps to Execute `main.sce` Code:

1. Click on the “Read File” button and select the audio file of your choice.
2. Click on the “Play” button to play the selected audio file.
3. Click on the “input (time domain)” button to view the amplitude vs. time graph of the input signal.
4. Click on the “input (frequency domain)” button to view the amplitude vs. frequency graph of the input signal.

5. Click on the “Low pass” button to hear the audio file after it has been processed through a low-pass filter that cuts off frequencies above 2000 Hz.
6. Click on the “bandpass1” button to hear the audio file after it has been processed through a band-pass filter allowing frequencies in the range 2000 Hz to 3000 Hz.
7. Click on the “bandpass2” button to hear the audio file after it has been processed through a band-pass filter allowing frequencies in the range 3000 Hz to 4000 Hz.
8. Click on the “bandpass3” button to hear the audio file after it has been processed through a band-pass filter allowing frequencies in the range 4000 Hz to 5000 Hz.
9. Click on the “highpass” button to hear the audio file after it has been processed through a high-pass filter that cuts off frequencies below 5000 Hz.
10. After applying any of the above filters, click on the “output (frequency domain)” button to see the amplitude vs. frequency plot of the filtered signal.

Steps to Execute `reverse.sce` Code:

1. Open the file `reverse.sce` from the Scilab window.
2. After the code has opened in Scilab Notes, click on the Run or Play button to execute the code.
3. A window will prompt you to select an audio file.
4. After selecting the audio file, you will hear the original audio first, followed by the reversed audio after a short delay. Additionally, two graphs will appear: the top graph displays the amplitude vs. time of the input signal, and the bottom graph shows the amplitude vs. time of the reversed input signal.

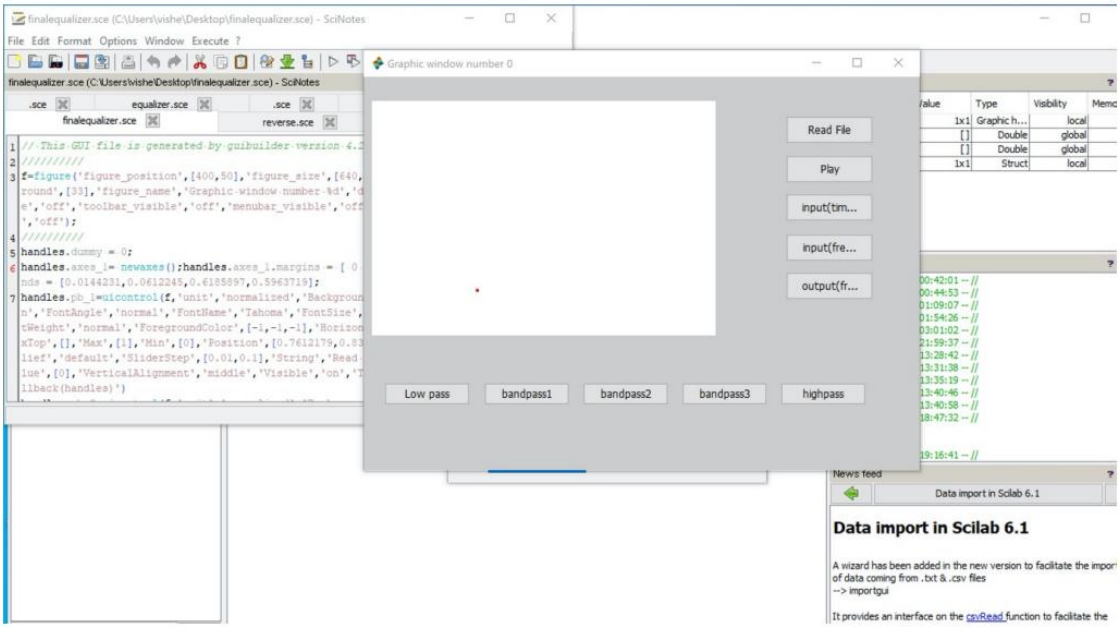
Steps to Execute `fft.sce` Code:

1. Open the file `fft.sce` from Scilab.
2. After the code has opened in Scilab Notes, click on the Run or Play button to execute the code.
3. A window will prompt you to select an audio file.
4. After selecting the audio file, you can view the frequency plot of the input signal. The plot is generated by plotting the absolute values of the fast Fourier transform (FFT) of the input signal.

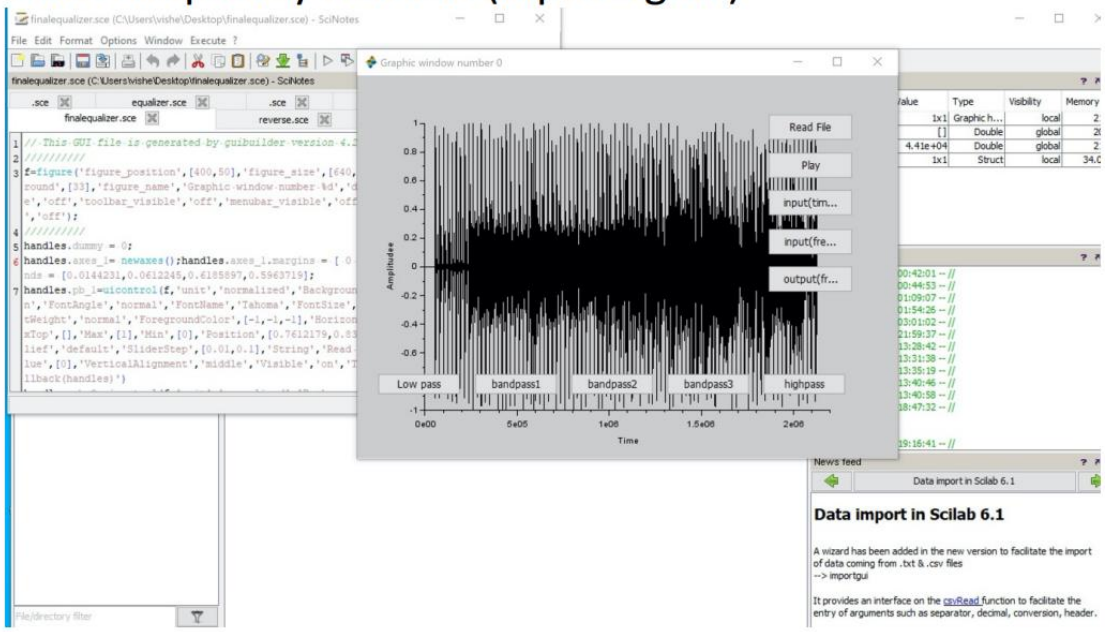
7. Result

A detailed description of the results from the simulation. Use contours and/or plots wherever necessary. Inference(s) drawn from the results also need to be mentioned in detail.

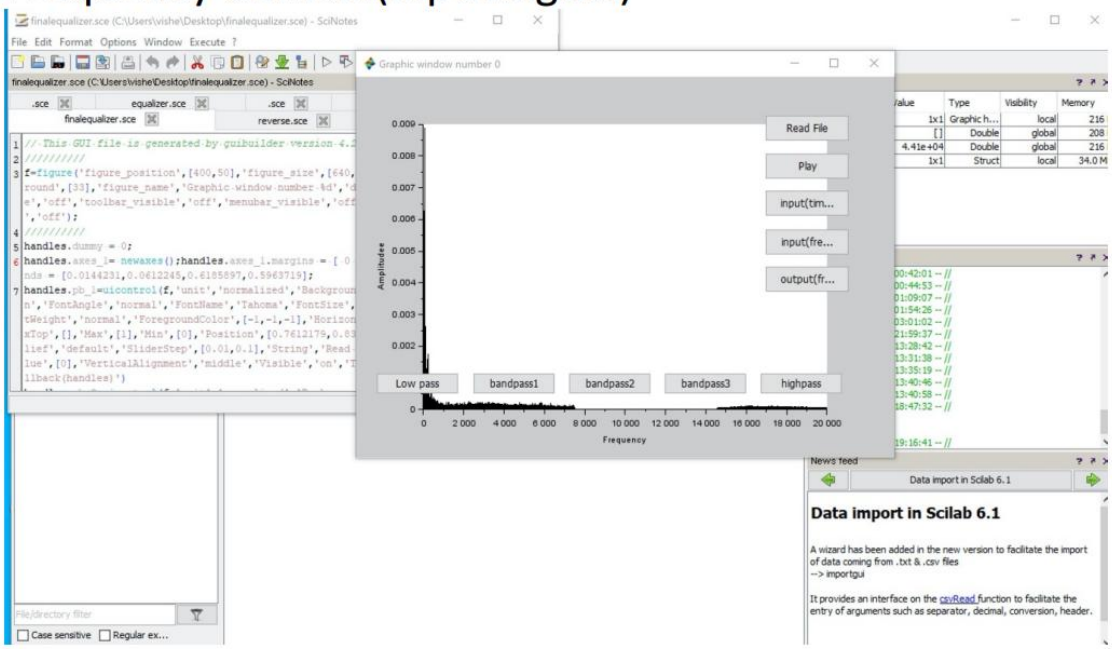
GUI



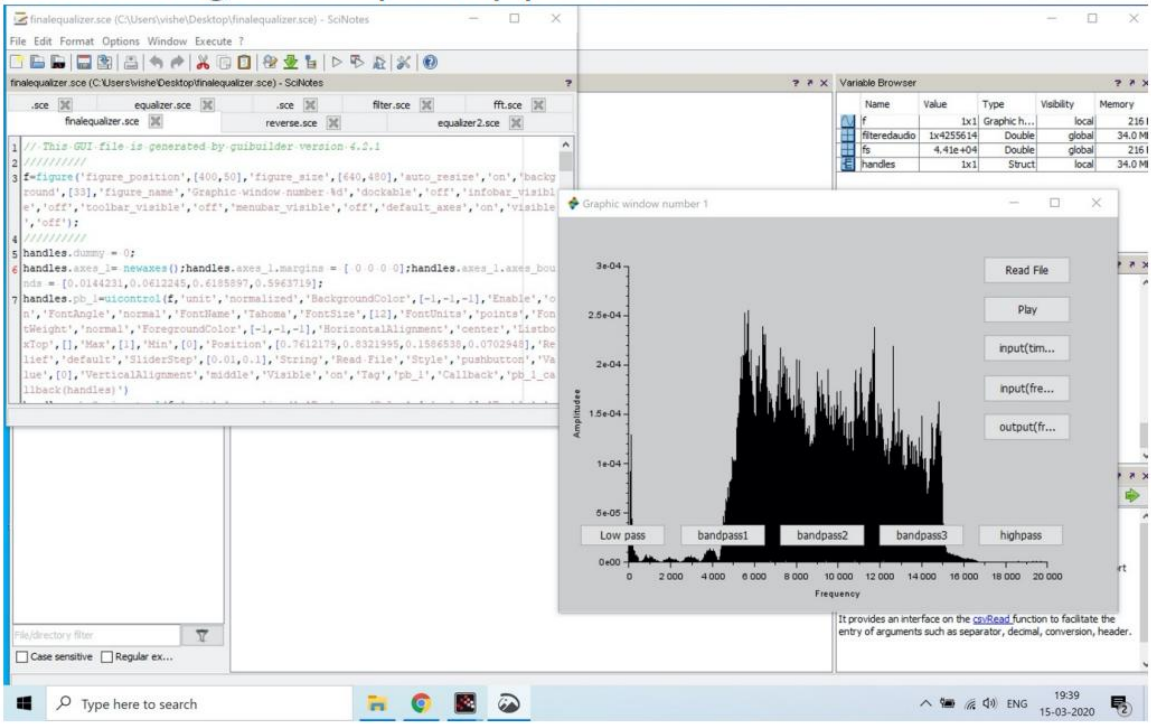
Time frequency domain (input signal)



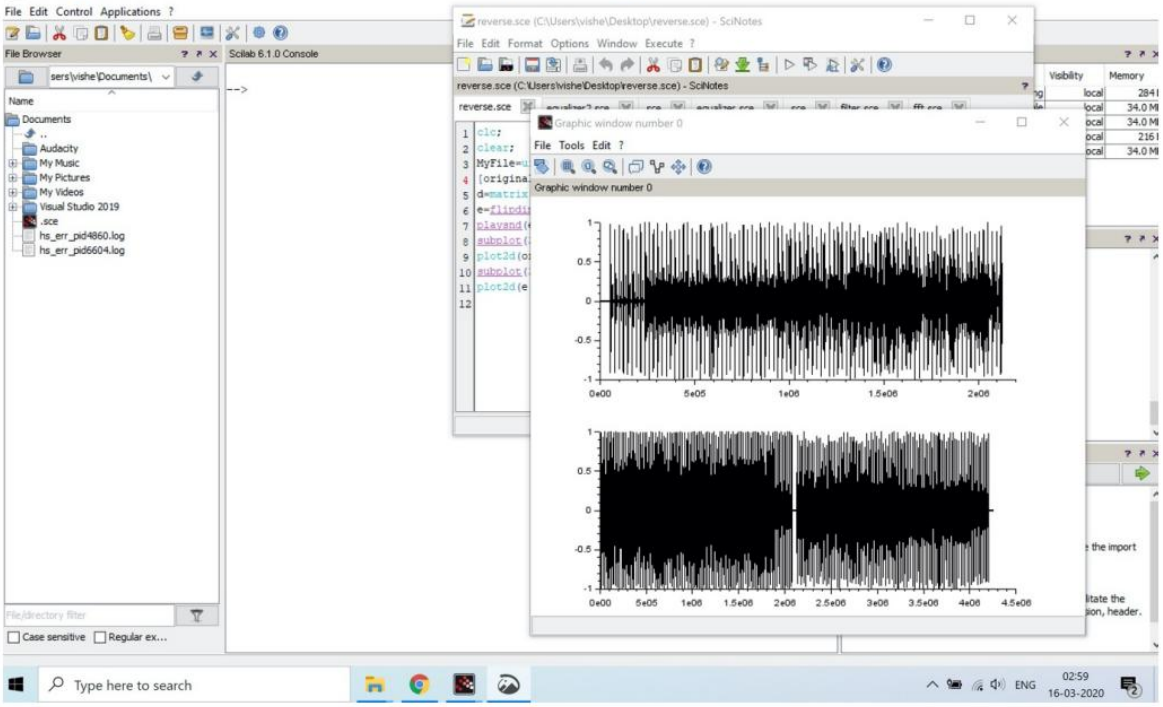
Frequency domain(input signal)



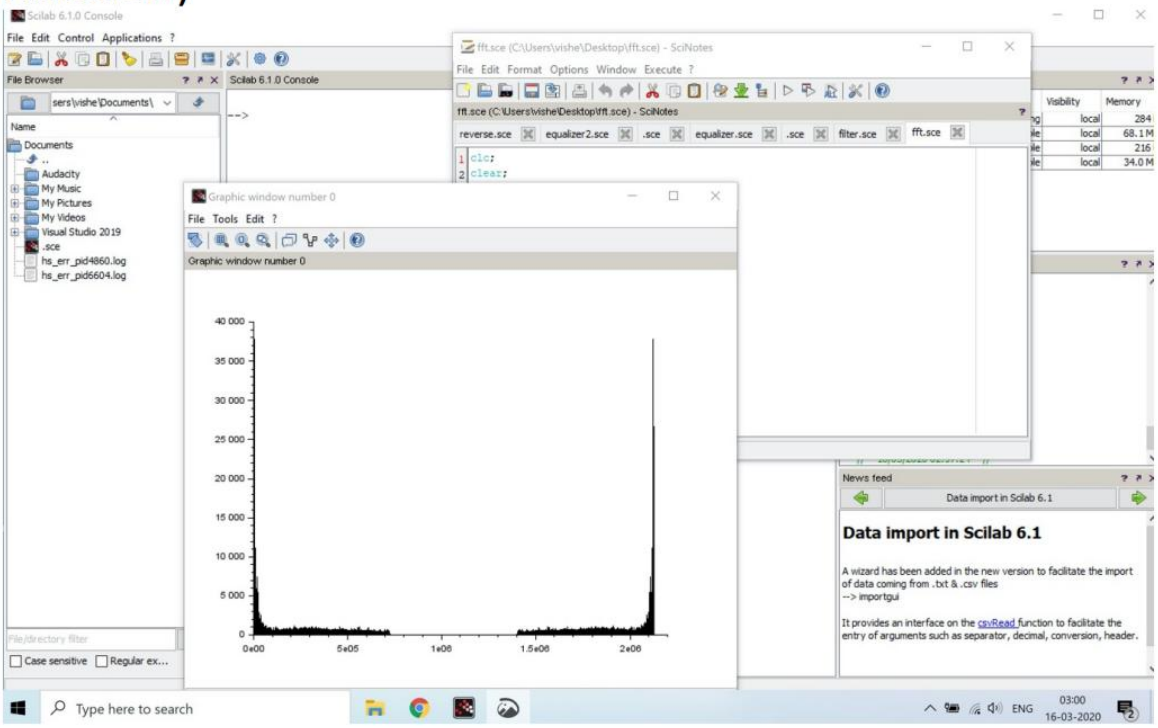
Filtered signal frequency plot



Original and reversed input signal in time domain



Plot of input signal in frequency domain(using fft function)



8. References

1. Oppenheim, A. V., & Schafer, R. W. (Year). Discrete-Time Signal Processing. Publisher.

2. Roy, N. (2020). Audio Signal Processing. Retrieved from [GitHub](https://www.github.com/neelabhro/Audio-Signal-Processing). Accessed May 4, 2020.

3. Chaguaro Aldaz, E. D. (n.d.). Digital Signal Processing Filtering Algorithm - Audio Equalization Using MATLAB. Retrieved from [Semantic Scholar](https://pdfs.semanticscholar.org/b367/c0375672e83a57ec9ee7c2e3cffa569bc6e3.pdf).