Voice Spectrum Analyzer: Real-Time Voice Signal Processing and Spectral Analysis

Kareem R. Hussein
Ahmed Haitham
Abd-Allah Emam
Ayman El-Shazly
Omar El-Beltagy
Department of Electronics and Communication Engineering
Nile University
5/1/2025

January 5, 2025

Abstract

This report outlines the design and implementation of a Voice Spectrum Analyzer developed using MATLAB, aimed at real-time acquisition and spectral analysis of voice signals. The system captures audio input for a user-defined duration and visualizes the resulting signal in both the time and frequency domains. Using the fast Fourier transform (FFT), the tool computes the spectral content of the recorded signal, showing key characteristics such as amplitude, phase, and energy across various frequencies. The accompanying graphical user interface (GUI) provides users with intuitive control over parameters including sampling frequency, maximum recording time, and spectrum type (Amplitude, Phase, or Energy). This report provides a detailed explanation of the methodologies used for signal acquisition, processing, and visualization. In addition, it evaluates the performance of the system and highlights its potential applications in fields such as speech analysis, audio signal processing, and communication systems. The developed tool exemplifies the practical application of MATLAB's computational and visualization capabilities in a user-friendly environment for real-time signal analysis.

Contents

0.1 Introduction

Voice signal analysis is essential for applications in telecommunications, speech recognition, voice authentication, and entertainment industries. The human voice is a complex signal with a wide frequency and amplitude range, varying over time. Analyzing its behavior in both time and frequency domains helps reveal characteristics such as pitch, tone, and energy distribution, crucial for fields like communication systems and acoustics.

Mathematically, voice signals can be analyzed using tools such as the Fast Fourier Transform (FFT). The FFT converts the time-domain signal into the frequency domain, allowing us to analyze its spectral content. The amplitude, phase, and energy spectra are derived from the FFT result. The amplitude spectrum reveals the strength of different frequency components, the phase spectrum describes the phase shift at each frequency, and the energy spectrum provides insight into the distribution of energy across frequencies. These mathematical techniques are fundamental for understanding the underlying characteristics of the voice signal and for various signal processing applications.

This project provides an interactive platform for analyzing voice signals by capturing real-time audio, processing it digitally, and visualizing its spectral content. It serves as both an educational tool for signal processing concepts and a practical application to explore dominant frequencies, spectral comparisons, and phase shifts. By integrating signal acquisition, processing, and graphical display, the project bridges theoretical knowledge with real-world analysis.

MATLAB's functionality makes it ideal for this project, combining signal processing and visualization with built-in tools. Key components of the implementation include the following.

- **Audio Recording**: Using `audiorecorder` to capture voice signals, with `recordblocking` for synchronous recording and `getaudiodata` to retrieve the audio.
- **Fourier Transform**: The function `fft` converts the signal from the time domain to the frequency domain, and the amplitude, phase, and energy spectra are derived from the FFT output.
- **Graphical User Interface (GUI)**: MATLAB components like `uifigure`, `uiaxes`, and `uidropdown` create an intuitive interface for user control.
- **Signal Visualization**: The `plot` function displays both the signal in the time domain and the frequency domain spectrum, with dynamic updates for user-selected analysis types.
- **Interactive Features**: Functions like `zoom` and `pan` allow users to explore the plots in more detail.

This project leverages MATLAB's powerful features to create an efficient, user-friendly voice spectrum analyzer.

0.2 Detailed Mathematical Background

The project involves capturing and analyzing a voice signal in both the time and frequency domains. Below is a detailed explanation of the mathematics used:

0.2.1 Signal in Time Domain

In the time domain, a signal $x(t)$ represents variations in sound pressure as a function of time. For digital processing, the continuous-time signal is sampled to create a discrete-time signal:

$$x[n] = x(nT_s), \quad n = 0, 1, 2, \ldots, N-1$$

Where:

- $T_s = \frac{1}{f_s}$ is the sampling period.
- f_s is the sampling frequency.
- N is the total number of samples.

The time-domain signal directly reflects the amplitude of the sound wave at each sampled point in time. This domain is essential for observing signal patterns, such as waveform shape and duration.

0.2.2 Signal in Frequency Domain

The frequency domain provides a representation of how the energy of a signal is distributed across different frequency components. To transform a signal from the time domain to the frequency domain, the Fourier Transform is used.

0.2.3 Fast Fourier Transform (FFT)

The Fourier Transform for a discrete signal is represented as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}}, \quad k = 0, 1, 2, \ldots, N-1$$

Where:

- $X[k]$ is the frequency-domain representation at frequency index k.
- $e^{-j\frac{2\pi kn}{N}}$ represents the complex exponential basis functions.

The FFT is an efficient algorithm for computing the Discrete Fourier Transform (DFT), significantly reducing the computational complexity from $O(N^2)$ to $O(N \log N)$.

The FFT output is as follows:

- Symmetric about the center for real-valued signals, so only the first half of the spectrum ($N/2$) is typically used.
- Split into:
  - Magnitude (Amplitude): $|X[k]|$
  - Phase: $\angle X[k]$

0.2.4 Amplitude Spectrum

The amplitude spectrum shows the magnitude of each frequency component:

$$A[k] = |X[k]| = \sqrt{(\text{Re}(X[k]))^2 + (\text{Im}(X[k]))^2}$$

Where:

- $\text{Re}(X[k])$ is the real part of $X[k]$.
- $\text{Im}(X[k])$ is the imaginary part of $X[k]$.

The amplitude spectrum provides information on the frequencies that dominate the signal.

0.2.5 Phase Spectrum

The phase spectrum indicates the phase shift of each frequency component:

$$\phi[k] = \arg(X[k]) = \tan^{-1}\left(\frac{\text{Im}(X[k])}{\text{Re}(X[k])}\right)$$

Phase information is crucial for reconstructing signals accurately and understanding their timing relationships.

0.2.6 Energy Spectrum

The energy spectrum shows the distribution of signal energy across frequencies. It is derived as the square of the amplitude:

$$E[k] = |X[k]|^2$$

This spectrum is particularly useful in applications like audio analysis and compression, where energy concentration is critical.

0.3 Code Implementations

0.3.1 Audio Capture

The Audio Capture section in this code is responsible for recording audio data from the user's microphone, which is then processed and visualized. This section uses the `audiorecorder` object, a built-in MATLAB function, to facilitate audio capture. The following provides a detailed explanation of how the audio capture functionality is implemented B.1

Explanation of the Audio Capture Process

Starting Audio Capture

When the user presses the `START` button, the `STARTButtonPushed` function is triggered. The maximum recording time (`Tmax`) and sampling frequency (`Fs`) are retrieved from the user interface (`TmaxField` and `FsField`, respectively). These values define the duration and the quality of the audio recording. The sampling frequency (`Fs`) determines how many samples per second are recorded, and the duration (`Tmax`) specifies how long the recording will last.

```
Tmax = app.TmaxField.Value;  % Max recording time
Fs = app.FsField.Value;      % Sampling frequency
```

Creating the Audio Recorder Object

The `audiorecorder` function is used to create an audio recorder object. This object is responsible for capturing the audio data from the microphone. The parameters passed to `audiorecorder` are:

- `Fs`: The sampling frequency in Hertz (samples per second).
- `16`: The bit depth, indicating that each audio sample will be 16 bits (standard quality for audio recording).
- `1`: The number of channels (mono audio in this case, meaning one audio stream is recorded).

```
app.Recorder = audiorecorder(Fs, 16, 1);  % 16-bit, mono, Fs sampling
    frequency
```

Recording the Audio

The `recordblocking` function is used to start the recording process. It records the audio for a duration of `Tmax` seconds and blocks further execution until the recording is complete. This means that the MATLAB script will not continue until the user finishes speaking or the time limit expires.

```
recordblocking(app.Recorder, Tmax);  % Start and stop automatically after
    Tmax seconds
```

The audio data is now stored within the `app.Recorder` object, and the recording process is complete.

Processing the Recorded Audio

Once the recording finishes, the `processRecording` function is called to process and visualize the captured audio. This function retrieves the audio data using `getaudiodata` and then processes it to generate both time-domain and frequency-domain plots.

```
    processRecording(app);
```

Stopping the Audio Capture

The `STOP` button allows the user to manually stop the recording before the maximum time (`Tmax`) is reached. When the `STOP` button is pressed, the `STOPButtonPushed` function is triggered. It checks if the recording is still ongoing using `isrecording(app.Recorder)`. If it is, it stops the recording immediately using the `stop` function.

```
if isrecording(app.Recorder)
    stop(app.Recorder);  % Stop recording
end
```

Post-Recording Audio Processing

After the recording is stopped or finished, the `processRecording` function is called again to process and display the audio data in both the time and frequency domains.

Summary of the Audio Capture Flow

1. **User Input**: The user specifies the recording duration (`Tmax`) and sampling frequency (`Fs`).
2. **Audio Recorder Setup**: The `audiorecorder` function is used to initialize an audio recorder with the user-defined parameters (sampling rate, bit depth, and channels).
3. **Start Recording**: The `recordblocking` function captures audio for the specified duration and stores the audio data in the `app.Recorder` object.
4. **Post-Processing**: Once the recording is finished or stopped, the `processRecording` function is called to analyze and display the audio data.

0.3.2 Signal Visualizations

The **Signal Visualization** section in the **Voice Spectrum Analyzer** app is responsible for displaying the recorded audio signal in both the **time domain** and the **frequency domain**. This section of the code leverages MATLAB's powerful plotting functions to give users visual insights into the captured signal's behavior and its spectral content.

Key Elements in Signal Visualization:

Time Domain Visualization

The **Time Domain Plot** displays the audio signal as it evolves over time, showing how the signal amplitude changes during the recording. This is a crucial visualization for understanding the raw waveform of the recorded voice or sound. The time domain plot is a fundamental signal representation, especially for examining the signal's periodicity, amplitude variations, and overall shape.

Code Implementation for Time Domain Plot The **plot** function is used to visualize the audio signal in the time domain. In the `processRecording` function, the following steps are implemented for time domain visualization B.2

- `t = (0:length(audioSignal)-1) / Fs;`: This line generates a time vector `t`, which corresponds to the time points at which the signal values (`audioSignal`) are sampled. It divides the sample index `(0:length(audioSignal)-1)` by the sampling frequency `Fs` to convert sample indices into time in seconds.
- `plot(app.TimeDomainPlot, t, audioSignal);`: This line generates the plot of the audio signal against time. The `audioSignal` is plotted on the y-axis, and the time vector `t` is plotted on the x-axis.
- `xlabel` and `ylabel` functions label the axes, while `title` adds a title to the plot.
- The `grid` function is enabled to improve readability, making it easier to discern the signal's variations.

Frequency Domain Visualization

The **Frequency Domain Plot** displays the spectral content of the recorded audio signal. This visualization is derived from the **Fast Fourier Transform (FFT)** of the audio signal, which converts the time-domain signal into its frequency components. It provides a clearer understanding of the frequencies present in the voice or sound recording, showing the amplitude, phase, or energy distribution across different frequencies.

Code Implementation for Frequency Domain Plot The FFT analysis is performed using the `fft` function, and the corresponding frequency spectrum is plotted as shown B.3 B.4

- `fft(audioSignal)` computes the **Fast Fourier Transform (FFT)** of the recorded audio signal. This transforms the signal from the time domain to the frequency domain, representing the signal's frequency content.
- The frequency vector `frequencies` is calculated to correspond to the bins of the FFT output. It defines the range of frequencies, from 0 Hz up to the Nyquist frequency (half the sampling rate). `frequencies = (0:N-1)*(Fs/N);` calculates the frequency range, where `Fs` is the sampling frequency, and `N` is the length of the audio signal.

- Since the FFT output is symmetric, the code only selects the first half of the FFT results (`fftSignal(1:N/2)`) to focus on positive frequencies. The `frequencies = frequencies(1:N/2);` statement ensures that only the corresponding positive frequencies are used.

Once the FFT is performed, the **spectrum type** selected by the user (Amplitude, Phase, or Energy) is displayed in the frequency domain plot.

User-Selected Spectrum Type

Based on the user's selection from the drop-down menu, the program displays different representations of the frequency domain spectrum. The spectrum can be displayed as **Amplitude**, **Phase**, or **Energy**. This flexibility allows the user to choose how they want to interpret the frequency content of the audio signal. B.5

- The `spectrumType` variable holds the value selected by the user (Amplitude, Phase, or Energy).
- A `switch` statement is used to compute the spectrum type:
  - **Amplitude**: `abs(fftSignal)` computes the magnitude of the FFT.
  - **Phase**: `angle(fftSignal)` computes the phase angle of the FFT.
  - **Energy**: $abs(fftSignal)^2$ computes the energy.
- The **ylabelText** and **xlabelText** variables are updated to reflect the selected spectrum type, ensuring the plot labels correspond to the displayed data.

Plotting the Spectrum

The `plot` function is used to visualize the spectrum in the frequency domain:

```
plot(app.FrequencyDomainPlot, frequencies, spectrumData);  % Plot the
    frequency spectrum
xlabel(app.FrequencyDomainPlot, 'Frequency (Hz)');  % Label x-axis
    (frequency in Hz)
ylabel(app.FrequencyDomainPlot, ylabelText);  % Label y-axis
    (Amplitude/Phase/Energy)
title(app.FrequencyDomainPlot, [spectrumType, ' Spectrum']);  % Set plot
    title
grid(app.FrequencyDomainPlot, 'on');  % Enable grid for clarity
```

- The `plot` function plots the `spectrumData` against the corresponding `frequencies`. The frequency axis (x-axis) is in Hertz (Hz), while the y-axis depends on the selected spectrum type (Amplitude, Phase, or Energy).
- The `xlabel`, `ylabel`, and `title` functions label the axes and set the plot title based on the spectrum type.

0.3.3 FFT for spectral analysis

In the **FFT for Spectral Analysis** section, the code utilizes a **switch-case** structure to provide the user with different options for viewing the frequency-domain representation of the recorded voice signal. The `switch-case` structure allows the user to select from three different spectrum types: **Amplitude**, **Phase**, and **Energy**. Based on the selected spectrum type, the code processes the FFT data accordingly and visualizes the desired spectral representation. Here's a detailed breakdown of how the **switch-case ??**

Use of the `switch-case` Structure

- **User-Selected Spectrum Type (`spectrumType`)**: The value selected by the user from the `SpectrumTypeDropDown` menu is stored in the variable `spectrumType`. This value determines which case to execute in the `switch` block, allowing the user to dynamically choose the type of spectrum they want to view.

```
spectrumType = app.SpectrumTypeDropDown.Value;
```

- **`switch` Statement**: The `switch` statement evaluates the value of `spectrumType` and executes the corresponding `case` block. The possible values are:
  - `'Amplitude'`: This case computes the **amplitude spectrum**, which represents the magnitude of the signal's frequency components.
  - `'Phase'`: This case computes the **phase spectrum**, which shows the phase (or angle) of each frequency component in the signal.
  - `'Energy'`: This case computes the **energy spectrum**, which is the square of the magnitude of each frequency component, representing the energy content at each frequency.

- **Case 1: `'Amplitude'`**: When the user selects "Amplitude", the `abs(fftSignal)` function is applied to the FFT result to compute the magnitude of each frequency component. The result is stored in the `spectrumData` variable. The x-axis and y-axis labels are set to "Amplitude", as the spectrum displays the magnitude of the signal.

```
spectrumData = abs(fftSignal);  % Magnitude (Amplitude Spectrum)
ylabelText = 'Amplitude';
xlabelText = 'Amplitude';
```

- **Case 2: `'Phase'`**: When the user selects "Phase", the `angle(fftSignal)` function is applied to the FFT result to extract the phase (angle) of each frequency component. The result is stored in the `spectrumData` variable. The x-axis and y-axis labels are set to "Phase (radians)", as the spectrum displays the phase in radians.

```
spectrumData = angle(fftSignal);  % Phase Spectrum
ylabelText = 'Phase (radians)';
xlabelText = 'Phase (radians)';
```

- **Case 3:** `'Energy'`: When the user selects "Energy", the code calculates the energy spectrum by squaring the magnitude of the FFT result ($abs(fftSignal)^2$) This gives the energy distribution across frequencies. The x-axis and y-axis labels are set to "Energy", reflecting the energy representation of the signal.

```
    spectrumData = abs(fftSignal).^2;  % Energy Spectrum
    ylabelText = 'Energy';
    xlabelText = 'Energy';
```

- **otherwise**: The `otherwise` block serves as a default case, which will execute if the user selects an invalid spectrum type (although this is unlikely since the drop-down menu limits the options to `'Amplitude'`, `'Phase'`, and `'Energy'`). By default, if an invalid selection occurs, the amplitude spectrum is computed.

```
     spectrumData = abs(fftSignal);  % Default to Amplitude Spectrum
    ylabelText = 'Amplitude';
    xlabelText = 'Amplitude';
```

Why Use a `switch-case` Structure?

The `switch-case` structure is particularly useful here because it provides a clean, organized way to handle multiple possible user inputs (spectrum types) and ensures that the correct spectral analysis is performed based on the user's choice. It allows the program to:

- **Handle multiple options easily**: The structure cleanly separates the different spectrum types into individual blocks, making the code more readable and maintainable.
- **Optimize for performance**: The `switch` statement is efficient when checking for multiple possible values of a single variable, especially when compared to using multiple `if-else` conditions.
- **Adapt to user input**: The user can dynamically change the spectrum type during interaction, and the program responds immediately by updating the frequency-domain plot accordingly.

In summary, the **switch-case** structure allows the user to choose between different spectral analyses (Amplitude, Phase, Energy) and ensures that the correct transformation is applied to the FFT data. This approach enhances the flexibility and interactivity of the app, enabling users to explore various characteristics of the voice signal in the frequency domain.

0.3.4 Interactive Control

The **Interactive Control** section of the code allows the user to interact with and manipulate the displayed signal in both the time and frequency domains. This section enhances the user experience by providing controls such as zooming, panning, and selecting different spectrum types for analysis. The interactive features are implemented using MATLAB's GUI components and plotting functions, allowing users to visually explore the recorded audio data in various ways.

Key Interactive Features and Code Implementation

Zooming and Panning in Plots

The **zooming** and **panning** functionality allows users to interactively explore the plots, making it easier to inspect specific portions of the audio signal or the frequency spectrum.

Zooming:

- MATLAB provides a built-in `zoom` function that allows users to zoom in and out of the plots, either along the x-axis (time/frequency) or y-axis (amplitude/energy/phase).
- In this application, zooming is enabled on the frequency domain plot (which displays the spectrum of the audio signal) using the command:

```
zoom(app.FrequencyDomainPlot, 'on'); % Enable zooming in frequency
    plot
```

- This command enables the zoom tool, which allows users to click and drag to zoom in or out on the plot for a more detailed view.

Panning:

- The `pan` function is used to allow users to move the plot along the x-axis (time or frequency axis). This is useful when the user wants to shift the viewable area of the plot without changing the zoom level.
- The panning functionality is similarly activated for the frequency domain plot:

```
    pan(app.FrequencyDomainPlot,'on');% Enable panning in frequency
        plot
```

- Once activated, the user can click and drag to shift the plot along the horizontal axis.

Spectrum Type Selection via Drop-Down Menu

One of the key interactive features in this application is the ability to select the type of spectrum that is displayed in the frequency domain plot. The user can choose between **Amplitude**, **Phase**, or **Energy** spectra. This is implemented using a drop-down menu (`uidropdown`), which lets the user select a specific analysis type. **Label for Spectrum Type**:

- The **Spectrum Type Label** provides a description for the spectrum type drop-down, guiding the user to choose the type of spectral analysis they wish to view.

```
    app.SpectrumTypeLabel = uilabel(app.Panel);
    app.SpectrumTypeLabel.HorizontalAlignment = 'right';
    app.SpectrumTypeLabel.Position = [577 16 85 22];
    app.SpectrumTypeLabel.Text = 'Spectrum Type';
```

The code for the drop-down menu is as follows:

```
app.SpectrumTypeDropDown = uidropdown(app.Panel);
app.SpectrumTypeDropDown.Items = {'Amplitude', 'Phase', 'Energy'};
app.SpectrumTypeDropDown.Position = [677 16 130 22];
app.SpectrumTypeDropDown.Value = 'Amplitude';  % Default selection
```

- The drop-down menu is initialized with three possible options: 'Amplitude', 'Phase', and 'Energy'.
- When the user selects a new spectrum type, the `processRecording` function is called again to update the frequency domain plot according to the selected spectrum.

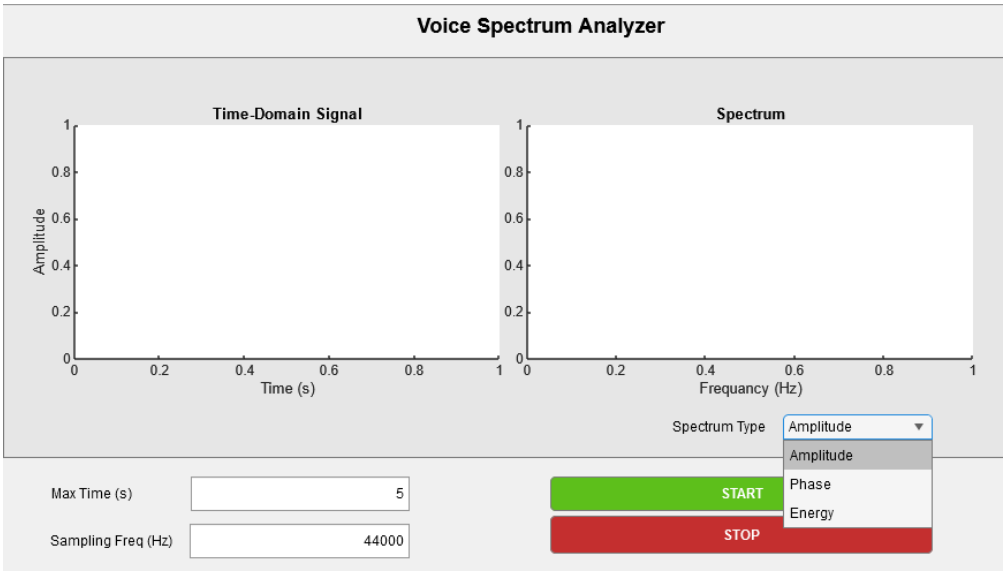

Figure 1: Drop-down menu for spectrum type selection.

0.3.5 GUI

Various other UI components are used to allow interaction with the app:

Numeric Fields for Input:

- The user can specify the **maximum time** (`TmaxField`) and **sampling frequency** (`FsField`) for the audio capture. These fields are numeric input boxes where users can directly enter values.

```
app.TmaxField = uieditfield(app.UIFigure, 'numeric');
app.TmaxField.ValueDisplayFormat = '%.0f';
app.TmaxField.Position = [164 56 190 30];
app.TmaxField.Value = 5;  % Default value
```

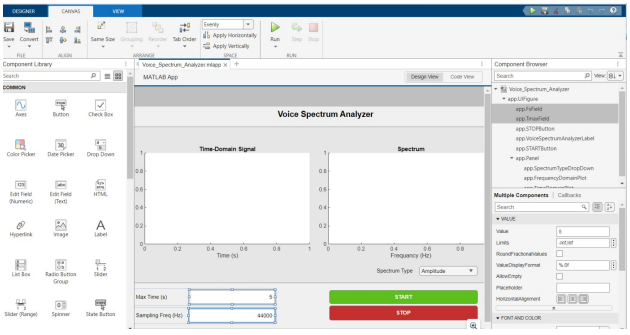

Figure 2: Input fields for maximum time and sampling frequency.

These input fields allow users to define key parameters for audio recording, such as how long to record and at what sampling frequency.

START and STOP Buttons:

- The **START** and **STOP** buttons are used to begin and end the audio recording. These buttons are implemented as `uibutton` components and are linked to callback functions (`STARTButtonPushed` and `STOPButtonPushed`), which control the audio recording process.

```
app.STARTButton = uibutton(app.UIFigure, 'push');
app.STARTButton.ButtonPushedFcn = createCallbackFcn(app,
    @STARTButtonPushed, true);
```

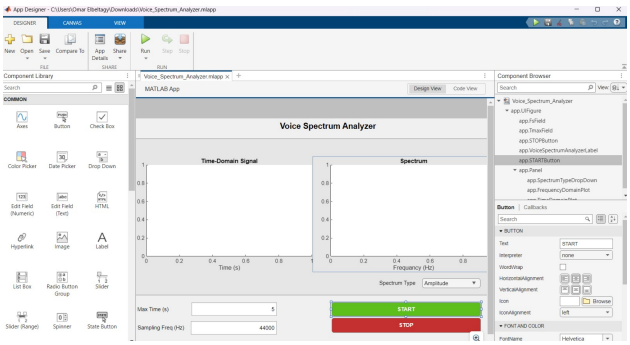

Figure 3: Start and stop buttons for audio recording.

0.4 Results and Discussion

The application was tested by recording voice signals under different conditions. The time-domain and frequency-domain plots were analyzed to verify the accuracy of the system. The FFT successfully displayed the spectral components of the voice signal characteristics.

0.4.1 Time-Domain Analysis

In the time-domain analysis, users were able to set the **maximum time** (`Tmax`) and **sampling frequency** (`Fs`) before recording. This flexibility allowed for customization based on the specific needs of the analysis.

During testing, users could initiate audio recording by clicking the `START` button. The system continued recording until either the maximum time was reached or the user manually stopped the recording by clicking the `STOP` button. This feature ensured that recordings could be tailored in real time to match the length of the speech or audio of interest.

The recorded voice signal was immediately displayed in the *Time-Domain Plot*, providing a clear visualization of the waveform amplitude over time.

0.4.2 Frequency-Domain Analysis

After recording, the voice signal was transformed into the frequency domain using the Fast Fourier Transform (FFT). This transformation provided insights into the spectral components of the recorded audio.

The application allowed users to analyze the signal in three different spectrum types:

- **Amplitude Spectrum:** Displayed the magnitude of each frequency component, highlighting the strength of the signal at various frequencies.

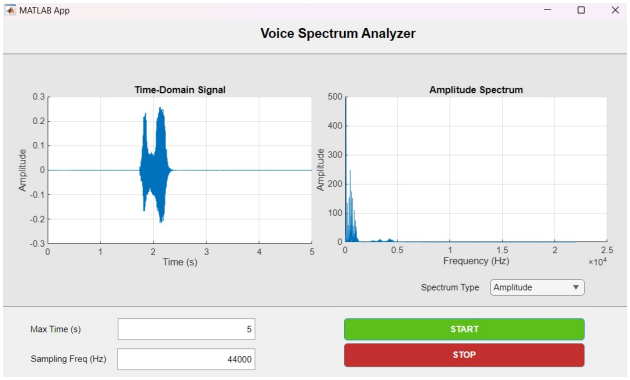

Figure 4: Amplitude spectrum of the recorded voice signal. The x-axis represents frequency (Hz), while the y-axis shows the amplitude.

- **Phase Spectrum:** Showed the phase (angle) of each frequency component, providing information about the relative timing of the signal components.

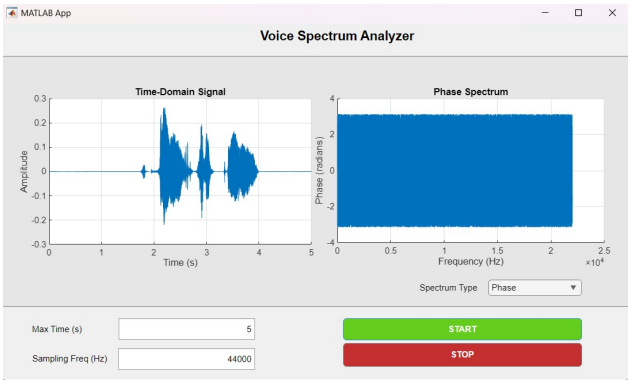

Figure 5: Phase spectrum of the recorded voice signal. The x-axis represents frequency (Hz), and the y-axis shows the phase in radians.

- **Energy Spectrum:** Represented the energy content at each frequency by squaring the magnitude of the frequency components.

Figure 6: Energy spectrum of the recorded voice signal. The x-axis represents frequency (Hz), and the y-axis shows the energy.

Interactive controls enabled users to select the desired spectrum type via a drop-down menu, and the frequency-domain plot updated dynamically. Figures 4, 5, and 6 demonstrate the output for each spectrum type. The results confirm that the application accurately captures and processes the voice signal in both time and frequency domains. Users can interactively explore the spectral properties of the signal, enabling a deeper understanding of its characteristics.

0.5 Conclusion

The **Voice Spectrum Analyzer** project successfully demonstrates the application of signal processing techniques in an interactive and user-friendly MATLAB-based interface. By utilizing MATLAB's App Designer framework, the project integrates real-time audio capture, signal analysis, and visualization in both the time and frequency domains. The ability to analyze the amplitude, phase, and energy of the signal through FFT showcases the flexibility and power of spectral analysis in understanding the frequency components of voice or audio signals.

The project features key functionalities, including audio recording, FFT-based spectral analysis, and interactive controls that allow users to modify the view of the time and frequency domain plots. These functionalities are implemented using MATLAB's built-in functions and GUI components, such as **audiorecorder**, **fft**, `plot`, and **uidropdown**, ensuring ease of use and intuitive interaction for the end user.

The system's versatility is enhanced by the ability to display different spectral representations—Amplitude, Phase, and Energy—empowering users to gain insights into the signal's characteristics from various perspectives. Additionally, the interactive zoom and pan features allow for detailed exploration of the visualized signals, adding to the overall user experience.

In conclusion, this project successfully bridges the gap between theoretical signal processing concepts and practical application, providing a robust platform for exploring and analyzing voice signals. The combination of real-time audio processing, spectral analysis, and graphical visualization makes this tool a valuable resource for both educational and practical purposes in the field of signals and systems.

Appendix A

Figures and Descriptions

- **Figure 1: Drop-down Menu for Spectrum Type Selection** This figure illustrates the drop-down menu used for selecting the spectrum type (Amplitude, Phase, Energy). It shows the UI element within the application interface, demonstrating its default appearance and layout.
- **Figure 2: Input Fields for Maximum Time and Sampling Frequency** This figure showcases the numeric input fields (`TmaxField` and `FsField`) where users can specify key parameters such as recording duration and sampling frequency. These fields are central to customizing the recording process.
- **Figure 3: Start and Stop Buttons for Audio Recording** This figure displays the start and stop buttons, critical UI components that allow users to control the recording process. These buttons are positioned prominently in the application for easy access.
- **Figure 4: Amplitude Spectrum of the Recorded Voice Signal** This figure represents the amplitude spectrum of the recorded voice signal. It highlights the strength of the audio signal at different frequencies, with frequency (Hz) on the x-axis and amplitude on the y-axis.
- **Figure 5: Phase Spectrum of the Recorded Voice Signal** This figure displays the phase spectrum, showing the phase (angle) of each frequency component. The x-axis represents frequency (Hz), and the y-axis represents phase in radians.
- **Figure 6: Energy Spectrum of the Recorded Voice Signal** This figure depicts the energy spectrum, which represents the energy content at each frequency by squaring the magnitude of the frequency components. The x-axis represents frequency (Hz), and the y-axis shows the energy values.

Appendix B

Code Implementations

B.1 Audio Recording Functions

This section contains code for handling audio recording, including start and stop functionality.

```
% Button pushed function: STARTButton
function STARTButtonPushed(app, event)
    Tmax = app.TmaxField.Value;  % Retrieve maximum recording time
    Fs = app.FsField.Value;      % Retrieve sampling frequency

    % Create audio recorder object
    app.Recorder = audiorecorder(Fs, 16, 1);  % 16-bit, mono, Fs sampling
       frequency
    recordblocking(app.Recorder, Tmax);  % Record for Tmax seconds

    % Retrieve and process audio data
    processRecording(app);
end

% Button pushed function: STOPButton
function STOPButtonPushed(app, event)
    if isrecording(app.Recorder)
        stop(app.Recorder);  % Stop the audio recording if it's still
           ongoing
    end
    % Process the recorded audio
    processRecording(app);
end
```

B.2 Data Visualization: Time Domain

This code plots the recorded audio signal in the time domain. Jump to Data Visualization: Time Domain

```
% Plot signal in time domain
function plotTimeDomain(signal, Fs)
    t = (0:length(signal)-1)/Fs;  % Time vector
    plot(t, signal);
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Time-Domain Signal');
end
```

B.3 Data Visualization: Frequency Domain

This code plots the frequency spectrum of the recorded signal. Jump to Data Visualization: Frequency Domain

```
% Plot signal in frequency domain
function plotFrequencyDomain(fftSignal, Fs)
    N = length(fftSignal);
    f = (0:N-1)*(Fs/N);  % Frequency vector
    plot(f, abs(fftSignal));
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('Frequency-Domain Signal');
end
```

B.4 Signal Processing: FFT

This code computes the Fast Fourier Transform (FFT) to analyze the frequency domain of audio signals. Jump to Signal Processing: FFT

```
% Perform FFT on the recorded signal
function spectrum = computeFFT(signal, Fs)
    N = length(signal);            % Number of samples
    f = (0:N-1)*(Fs/N);           % Frequency vector
    spectrum = abs(fft(signal));  % Compute FFT and magnitude
end
```

B.5 Spectrum Switch Case

```
% User-selected spectrum type
spectrumType = app.SpectrumTypeDropDown.Value;

% Compute Spectrum
switch spectrumType
    case 'Amplitude'
        spectrumData = abs(fftSignal);
        ylabelText = 'Amplitude';
        xlabelText = 'Amplitude';
    case 'Phase'
        spectrumData = angle(fftSignal);
        ylabelText = 'Phase (radians)';
        xlabelText = 'Phase (radians)';
    case 'Energy'
        spectrumData = abs(fftSignal).^2;
        ylabelText = 'Energy';
        xlabelText = 'Energy';
    otherwise
        spectrumData = abs(fftSignal);
        ylabelText = 'Amplitude';
        xlabelText = 'Amplitude';
end
```

Bibliography

[1] LaTeX - A document preparation system. Accessed: Apr. 7, 2024. [Online]. Available: https://www.latex-project.org/.

[2] LyX — LyX - The Document Processor. Accessed: Apr. 7, 2024. [Online]. Available: https://www.lyx.org/.

[3] MATLAB GUI, en. Accessed: Dec. 2, 2023. [Online]. Available: https://www.mathworks.com/discovery/matlab-gui.html.

[4] Oppenheim, A. V., *Signals and Systems*, 2nd ed., Prentice Hall, 1997.