Project 3
Parametric Equalizer

Coordidnator Teacher:
Dumitrescu Anamaria

Titular teacher:
Dumitrescu Anamaria

Students:
Marin Cătălin
Matei Daniel
Nechifor Silviu-Mihail

2023-2024

Contents

List of Figures

1 Abstract

1.1 Description of the project

In the world of audio processing today, there's a lot of excitement around an area that scientists are delving into deeply, and one of the tools grabbing a ton of attention is the equalizer. It's popular because it lets the user interact with how the different parts of sound (the frequencies) can affect what you hear.

An equalizer is basically a control panel for tweaking specific frequencies in audio. It's like having sliders that allow you to boost or cut certain parts of the sound spectrum, making it possible to adjust how bassy, trebly, or middy something sounds.

People who work with sound, like music producers, sound engineers, and even folks who just want to tweak their home audio setup, find the equalizer to be an essential tool. It's a bit like an artist's palette, giving you the ability to fine-tune and craft the audio in a way that suits your preferences or the requirements of a specific project.

Because of its widespread use and importance in shaping sound, the equalizer isn't just a basic tool anymore. It's become a hot topic for researchers and scientists in the audio field. They're constantly exploring new ways to improve its functionality, making it more precise, user-friendly, and adaptable to the ever-evolving demands of audio processing. This constant curiosity and innovation keep pushing the boundaries of what an equalizer can do, making it an exciting area for ongoing exploration and development.

1.2 Description of the project

The designed project involved the design and implementation of a sophisticated equalizer using MATLAB, aimed at replicating the functionalities of a real-world equalizer. To achieve this, principles of digital signal processing (DSP), specifically leveraging Fourier transform methods and various types of filters were employed. The equalizer developed is equipped with 5 adjustable frequency bands and employs 8 different types of filters, allowing users to precisely control and manipulate specific frequency ranges within the audio spectrum. This granularity provides users with a high degree of flexibility in shaping the sound according to their preferences or project requirements.

To enhance user experience and usability, a MATLAB Graphical User Interface (GUI) was implemented featuring intuitive knobs and sliders. These user-friendly controls mimic the functionality of physical equalizer sliders, enabling users to interactively adjust the settings for each frequency band and filter type in real-time.

To summarize, the combination of Fourier transform techniques, digital signal processing methods, a range of filters, and an intuitive GUI, the MATLAB-based equalizer strives to accurately simulate the capabilities and functionalities of a real-world equalizer.

2 Audio Processing

2.1 Reading the audio

One of the most important parts in designing an implementation that works with altering and modifying audio signals is the processing itself that has to be performed around the input audio in order to achieve an preferred result. To to that, the need of a tool required using the dsp.AudioFileReader function in Matlab's DSP (Digital Signal Processing) System Toolbox, which is a powerful tool designed for reading audio files and processing their contents. This function offers a robust platform to access audio data in various file formats, such as WAV, MP3, and others, which is exactly what the application is able to use, allowing for direct integration into MATLAB workflows.

It provides flexibility in configuring parameters like the number of samples to read per frame, output data type, and file format specifications, thus largely simplifying the process and Widely used in audio processing applications. Dsp.AudioFileReader facilitates tasks such as real-time audio analysis, filtering, feature extraction, and signal manipulation. Its seamless integration with MATLAB's signal processing capabilities makes it an indispensable tool for professionals, researchers, and enthusiasts working in audio-related fields seeking efficient and versatile audio file handling and processing within the MATLAB environment.

For the reading, the function was used in the following manner for the equalizer: it creates an audio file reader object named "reader" to read audio data from the file specified by "app.audiofile". The reader is configured to read data in frames, each containing 1024 samples. In a real-time processing loop, the audio data is continuously retrieved the file using "reader()" method and assigned to the variable "audioIn". This data is subsequently processed, potentially filtered by a five-band equalizer, and played back. The release(reader) statement ensures the proper release of system resources associated with the audio file reader, particularly when the processing loop is stopped, indicated by the "actuallyStopped" flag. Overall, the function facilitates the dynamic handling of audio input in the equalizer app.

```
          function updateGraph(app,flag)
        if ~exist('flag','var')
            flag  = 0;
        end

        frequencies = [app.freq1, app.freq2, app.freq3, app.freq4, app.freq5] * 10
        bandwidths = [app.bandw1, app.bandw2, app.bandw3, app.bandw4, app.bandw5];

            reader = dsp.AudioFileReader(app.audiofile, ...
         'SamplesPerFrame', 1024);

    player = audioDeviceWriter('SampleRate', reader.SampleRate,BitDepth='32-bit fl
```

In the previous code, the function responsible for updating the viewing graph is described, and some key components and parameters. The functions begins by checking if the flag parameter is provided, and if not, it is set to the default value of 0. The flag is used to control specific aspects of the update process, and more flags are used in the following processing functions for a logical and accurately showing of the results.

The functions then extracts frequency and bandwidths values from the properties of the app object and these values are obtained from the properties named freq1, freq2 and so on. representing user-defined settings for the five-band equalizer. The arrays are thus created by joining the values for future usage. After that, the process of reading the audio file is performed, as mentioned before, configured to read 1024 samples per frame. Also, the ”player” object of type ”audioDeviceWriter” is created. This objects is responsible for playing back audio data and it is configured with the same sample rate as the reader object and a bit depth of 32-bit float.

The next part of the audio playing is the following:

```
        if app.pushed
            [app.audiofile] = selectTrack(app.newTrack);
            app.pushed = 0;

            if(flag)
            flag = 0;

    while ~isDone(reader)
        if app.actuallyStopped
            app.actuallyStopped=0;
            release(player)
            release(reader)
        return;
        end
        pause(0);
        audioIn = reader();
        audioIn = audioIn(:,1);
        if app.needsNewCoefficients
            app.needsNewCoefficients = 0;
            [app.b,app.a] = ...
            CreateFiveFilter(audioIn, reader.SampleRate,...
            app.filtertype, ...
        end

    ...
    ...
    ...

    release(reader);
    release(player);
    app.PLAYButton.Text = 'PLAY';
            end
        elseif app.stopped
            while ~strcmp(app.PAUSEButton.Text,'PAUSE')
                pause(.1);
            end
            app.stopped = 0;
        end
```

The code begins with a conditional check (if app.pushed) to determine if the button associated with track selection has been pressed. If true, the selectTrack function is called with the argument app.newTrack, updating the audio file path in the app.audiofile property. Subsequently, the app.pushed flag is reset to 0. The nested conditional check (if flag) then resets the flag variable to 0.

Following this, a while loop is initiated, iterating as long as the audio file reader has not completed reading the entire audio file. Within this loop, there is a check for the app.actuallyStopped flag, and if true, it releases the audio player (player) and audio file reader (reader) objects and returns from the function. The pause(0) statement introduces a short delay to prevent excessive CPU usage in the loop. Audio data is continuously read from the file using the reader() method, and only the first channel is retained (audioIn = audioIn(:,1)).

In such manner, if the app.needsNewCoefficients flag is set, it is reset, and a function (CreateFiveFilter) is called to generate new filter coefficients based on the audio input, sample rate, filter type, and user-defined frequency and bandwidth parameters stored in the app object properties (app.freq1, app.freq2, ..., app.bandw5). The coefficients app.b and app.a are used for real-time audio filtering.

At last, the last part of the code segment handles the transition to a play state by updating the text on a button and waits for a user interaction, specifically the pressing of a 'PAUSE' button, when the application is in a stopped state. Once the pause button is pressed, it resets the stopped flag, allowing the application to proceed.

2.2 ADSR

In order to have control over the time-varying amplitude of sounds and enabling a wide range of expressive possibilities in creating music and designing soundscape, the need to use ADSR tool was taken into account to produce best output results.The ADSR envelope is a fundamental concept in audio synthesis and music production. It is commonly used to shape the amplitude of a musical note or sound over time. The ADSR envelope provides control over how the volume of a sound evolves from the moment it is triggered to the moment it ends.

The performing of the ADSR action is done in the following code snippet:

```
A = linspace(0, 0.9, (200)) ;
D = linspace(0.9, 0.7,(500));
S = linspace(0.7, 0.7,(100));
R = linspace(0.7, 0,(1024 -800 ));
ADSR = [A D S R];
audioOut = audioOut .* ADSR';
```

The abbreviation comes from Attack, Decay, Sustain and Release, a compound envelope on an audio signal, followed by the spectral analysis and playback of the sound. The modified audio signal is played back using the player object. The ADSR segments are concatenated into a single envelope (ADSR). The original audio signal (audioOut) is multiplied element-wise by the ADSR envelope.

This process modifies the amplitude of the audio signal based on the ADSR envelope, simulating the dynamic changes in loudness typically associated with musical notes.

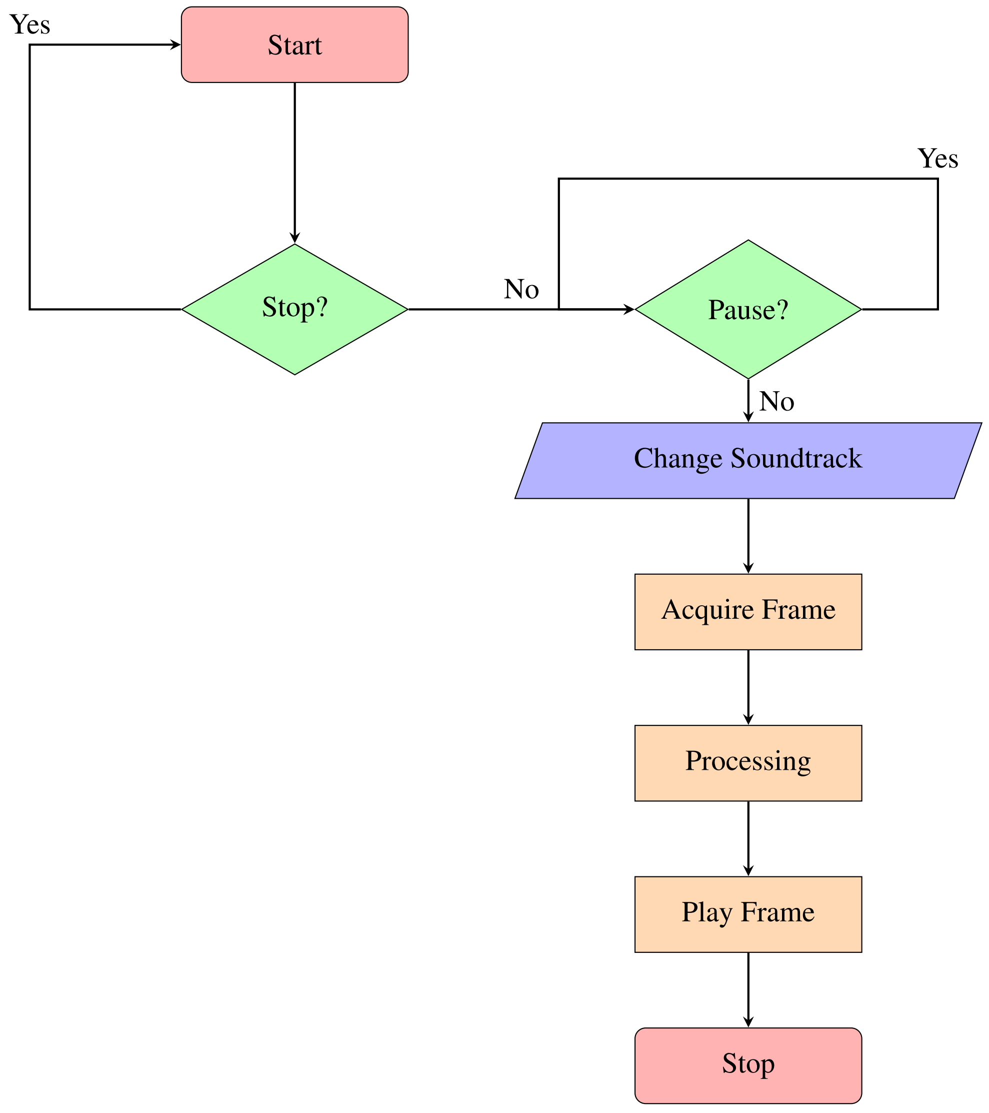
The following code performs a fast Fourier transform (FFT) on the modified audio signal to obtain its frequency spectrum, and the result (Y) represents the magnitudes of the frequencies in the signal. The ”loglog” function is then used to plot the spectrum on a logarithmic scale, specifically on the Axes of the Matlab app.

2.3 Playing the audiofile

Finally, the modified audio signal is then played back using the object ”player” created before, and updating the visualization of the spectrum with the help of ”drawnow limitrate” function to a real-time limiting rate of updates, in order to avoid excessive computation.

```
Y = abs(fft(audioOut, 2 * app.fs));
loglog(app.UIAxes,1 : length(Y)/2,Y(1:length(Y)/2));
drawnow limitrate;
player(audioOut);
```

Below, there is the flowchart for the equalizer, which contains the main blocks of decision process in the realization of the plugin:



3 Comparison of filter implementations

3.1 Theoretical background

Normally, an equalizer is an essential audio processing tool used to adjust the frequency response of audio signals. Digital equalizers, based on digital filters, provide a flexible and precise means of modifying the frequency content of a signal. Common types of digital filters employed in equalizer implementations include low-pass, high-pass, band-pass, or stop-band(notch) filters.

In digital signal processing, these filters are commonly realized through techniques like Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filter structures. For this implementation, an IIR filter has been chosen, as its main characteristic consist in have feedback loops, potentially resulting in more efficient implementations.

The coefficients of these filters are determined based on the desired filter characteristics, such as cutoff frequencies, bandwidths, and filter order. Implementing these filters digitally allows for precise control over the equalization process, enabling audio engineers to shape the frequency response of audio signals with high accuracy.

In the design of the parametric equalizer, the following filters have been chosen, both for standard requirements for an equalizer, and also for the practical uses of each filter, concluding to a number of 7 basic filters which have the most applications in the majority of audio engineering world.

The filters are classified as low-pass and high-pass, filters that select only the desired frequencies and attenuate the rest of the frequencies according to the central frequency chosen, band-pass which selects a specific interval, and its complementary filter, the band-reject or band-stop, also known as a notch filter, which cuts the exact interval of frequencies chosen.

An all-pass filter has also been considered to be implemented for accurate implementation purposes, which passes all the frequencies detected in the signal, alongside with low-shelf and high-shelf filters, which play a crucial role in equalizing. The difference between them and normal high-pass or low-pass filters it that high-pass and low-pass filters are designed to selectively pass or attenuate frequencies either above or below a cutoff point, whereas shelving filters are capable of adjusting the amplitude of frequencies over a broader range, either above or below the cutoff frequency, which is why the decision of implementing them is important when considering the need of each specific filter.

These filters are represented in figures below and detailed further for following understanding of the work.

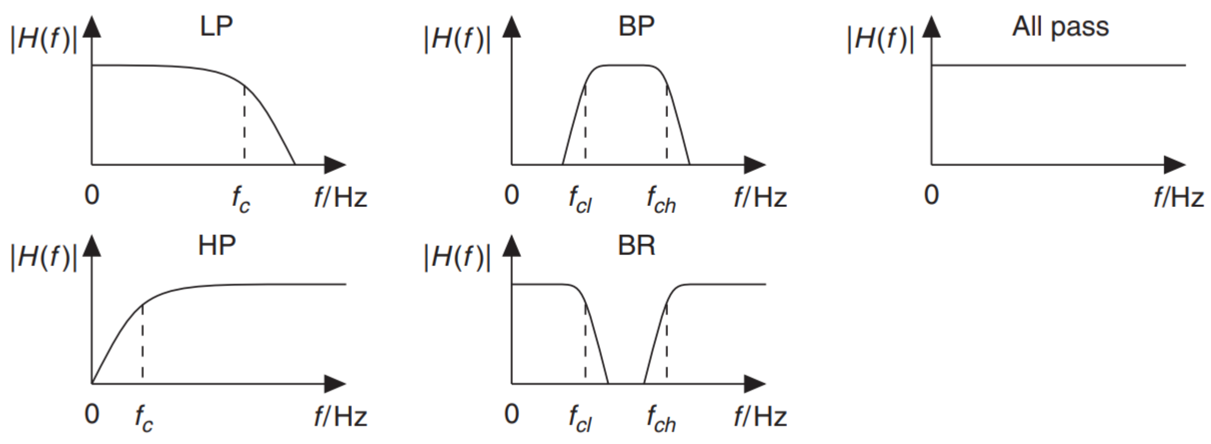


Figure 3.1: Filters used

3.2 DAFX implementation

For the first implementation, the DAFX approach has been chosen, presented in [x]. The decision to prioritize DAFX was underpinned by the model's comprehensive theoretical foundation, which includes established filter types such as Butterworth, Chebyshev, and elliptic filters. The DAFX implementation exhibits a nuanced understanding of filter characteristics, enabling the creation of digital filters with superior response characteristics. The accompanying documentation from DAFX further bolstered its appeal, providing in-depth explanations and mathematical formulations that facilitated a thorough grasp of the underlying principles.

When working with second-order filters, determining their coefficients involves more than just specifying the cutoff or center frequency — it also requires defining the Q factor. This factor serves different purposes depending on the type of filter:

For lowpass and highpass filters, the Q factor influences resonance height. THe Q value results in a maximally flat response up to the cutoff frequency. Lower Q values increase pass-band attenuation, while higher Q values introduce amplification around the cutoff frequency.

In bandpass and bandreject filters, the Q factor is related to the bandwidth (fb) through Q = fc/fb, representing the inverse of the relative bandwidth (fb/fc).

Despite the simplicity of canonical filters, deriving their coefficients from parameters like cutoff frequency and bandwidth can be intricate. To simplify this process, the documentation explores slightly more complex filter structures that offer easier parametrization while retaining the desired filter characteristics.

	b_0	b_1	b_2	a_1	a_2
Lowpass	$\frac{K^2Q}{K^2Q+K+Q}$	$\frac{2K^2Q}{K^2Q+K+Q}$	$\frac{K^2Q}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$
Highpass	$\frac{Q}{K^2Q+K+Q}$	$-\frac{2Q}{K^2Q+K+Q}$	$\frac{Q}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$
Bandpass	$\frac{K}{K^2Q+K+Q}$	0	$-\frac{K}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$
Bandreject	$\frac{Q\cdot(1+K^2)}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{Q\cdot(1+K^2)}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$
Allpass	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	1	$\frac{2Q\cdot(K^2-1)}{K^2Q+K+Q}$	$\frac{K^2Q-K+Q}{K^2Q+K+Q}$

Figure 3.2: Filters table

Example of implementation for the high-pass filter:

```
switch filter_type

    case 1 % APF FILTER
        f0 = 800;             % center frequency of the APF (Hz)
        bw = 1000;            % bandwidth of the APF (Hz)
        K = tan(pi*f0/fs);    % coefficient based on the tangent function
        Q = f0/bw;            % quality factor

    % coefficients taken in order for the numerator and the denominator (a & b)

        b0 = (K^2*Q-K+Q)/(K^2*Q+K+Q);
        b1 = 2*Q*(K^2-1)/(K^2*Q+K+Q);
        b2 = 1;
        a0 = 1;
        a1 = 2*Q*(K^2-1)/(K^2*Q+K+Q);
        a2 = (K^2*Q-K+Q)/(K^2*Q+K+Q);

     % store the coefficients in separate arrays

        b = [ b0 b1 b2 ];
        a = [ a0 a1 a2 ];

    % applying the filter to the input signal

        y = filter(b,a,x);
```

3.3 Cookbook implementation

The Bilinear Transform (BLT) is a crucial methodology in digital signal processing, providing a means to convert continuous-time analog systems into discrete-time digital counterparts. By facilitating the transformation from the analog frequency domain (s-plane) to the digital frequency domain (z-plane), the BLT ensures stability preservation in the digital realm, making it a fundamental tool in the design of digital filters.

In the development of the equalizer, the foundation lies in the transfer functions derived from analog prototypes, a process often employed to model the desired frequency response characteristics. The RBJ Cookbook [2], renowned for its wealth of mathematical formulations and coefficients, serves as an invaluable guide for designing various types of equalizers. The process entails selecting the appropriate filter type (such as parametric or shelving) and obtaining the requisite coefficients from the cookbook.

Once armed with these coefficients, the Bilinear Transform comes into play as the digital bridge. Each analog prototype's transfer function is transformed using the BLT, mapping analog frequencies to their corresponding digital counterparts. Notably, the implementation takes into account the nuances introduced by the BLT, necessitating prewarping to ensure accurate preservation of critical frequencies and compensating for bandwidth adjustments during the analog-to-digital mapping. The result is an effective and versatile tool for shaping audio responses in the digital domain, encapsulating the synergy between theoretical foundations and practical implementation.

Example of implementation for the high shelf filter

```
% Case: High-Shelf Filter (HSF)
    case 'HSF'

        % Parameters for the high-shelf filter
        S = 10000;                        % Shelf slope parameter
        A = 10^(-bandwidth(i)/40);        % Amplitude gain in decibels
        omega = 2*pi*frequency(i)/fs;     % Angular frequency in radians

        alfa = sin(omega)/2 * sqrt((A + 1/A)*(1/S - 1) + 2); % Calculate alpha
        % Calculate filter coefficients
        b0 = A * ((A+1) + (A-1) * cos(omega) + 2 * sqrt(A) * alfa);
        b1 = -2* A * ((A-1) + (A+1)*cos(omega));
        b2 = A * ((A+1) + (A-1)*cos(omega) - 2*sqrt(A) * alfa);
        a0 = (A+1) - (A-1) * cos(omega) + 2 * sqrt(A) * alfa;
        a1 = 2*((A-1) - (A+1) * cos(omega));
        a2 = (A+1) - (A-1) * cos(omega) - 2 * sqrt(A) * alfa;

        % Store coefficients in arrays for the filter
        b(i,:) = [b0, b1, b2];
        a(i,:) = [a0, a1, a2];

    end
```

For reference, this is an ideal representation of the same filter by the same filter, taken from [3].

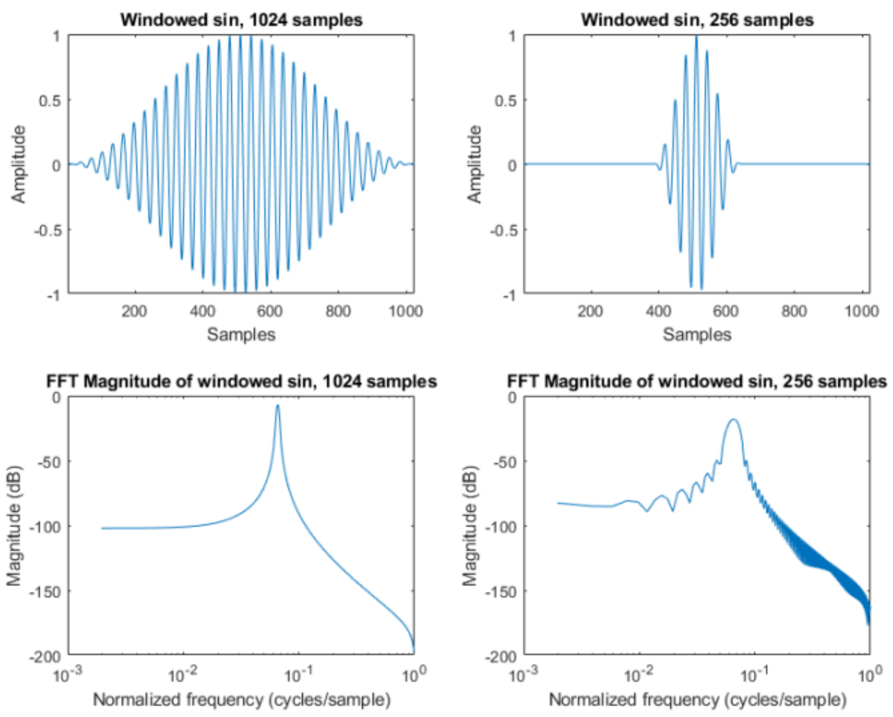
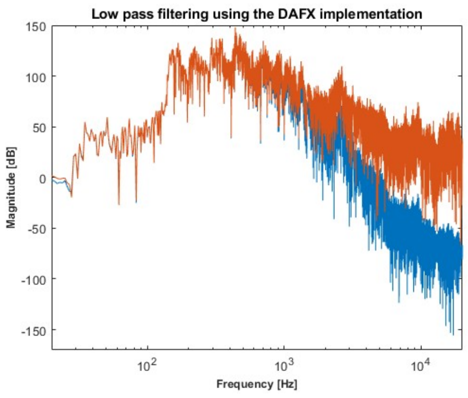


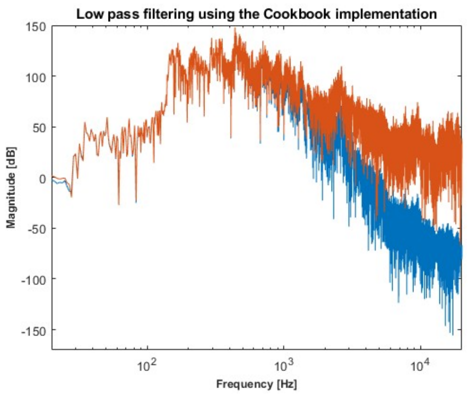
Figure 3.3: Filter representation

3.4 Comparison conclusion

Choosing the method of BLT over the DAFX implementation was taken into consideration as the final decision because of its improved response to the filtering process through its improved warping compensation stability. In simpler terms, it ensures that the crucial frequencies, the heart of our sound, stay true and recognizable in the digital world. BLT acts like a steady hand, keeping our equalizer strong and reliable, unfazed by different kinds of sounds it might encounter. Experts in the field trust BLT because it's like a well-known route in the world



(a) DAFX



(b) RBJ-Cookbook

Figure 3.4: Comparison

of digital signal processing. Using something trusted means that the equalizer is more likely to be reliable, just like keeping the musical harmony intact when when moving from analog to digital.

4 Equalizer GUI

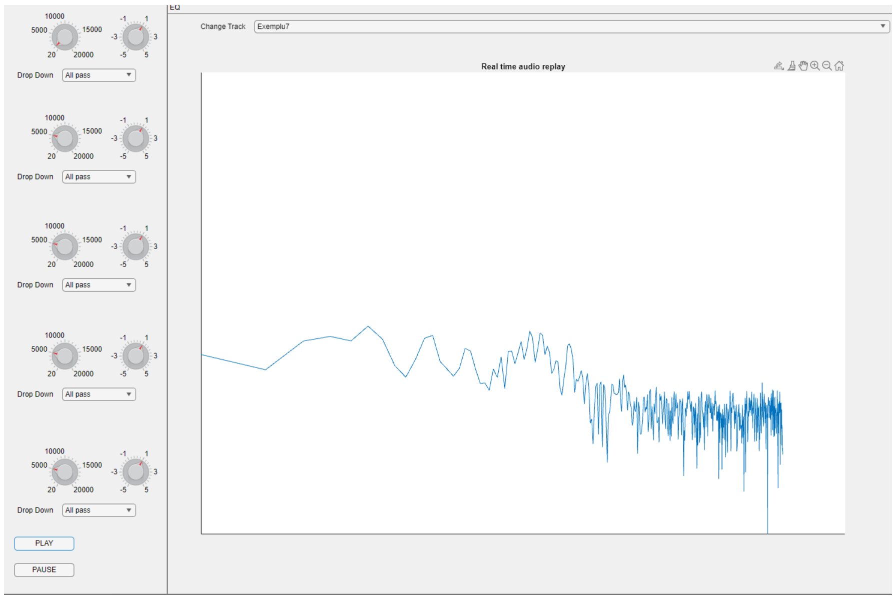


Figure 4.1: GUI

A Graphical User Interface (GUI) serves as a visual interface that allows users to interact with software applications using graphical elements such as buttons, sliders, and displays. In the context of an equalizer, a GUI provides an intuitive and user-friendly platform for controlling and adjusting audio parameters.

In MATLAB, the GUI that was implemented was created using the MATLAB App Designer (mlapp), a visual development environment. With App Designer, designing an equalizer GUI involves dragging and dropping components like sliders, buttons, and real-time plots to visualize the frequency response changes. MATLAB's App Designer simplifies the process of implementing callback functions to handle user interactions, ensuring a seamless connection between the graphical controls and the underlying equalizer functionality. This approach not only enhances the accessibility of the equalizer but also provides users with a straightforward and interactive means to tailor the audio output to their preferences.

In the designing of the interface for the equalizer, the application was designed according to some well-known interfaces of popular implementations, which contain some simple input designs, as presented generally before. Knobs were used in order to control and adjust the frequency chosen and respectively the bandwidth of each individual frequency, in an implementation which uses 5 separate ones, which represents enough coverage in order to achieve a equalization over the full audible range.

4.1 Choosing values in interface

This following code snippet defines two functions, f1ValueChanging and bw1ValueChanging, associated with user interface controls. The f1ValueChanging function updates the frequency parameter (freq1) of the filter in response to user interaction, triggering a graphical update through the updateGraph function. Similarly, the bw1ValueChanging function modifies the bandwidth parameter (amp1) and calls updateGraph to reflect the changes in the equalizer's graphical representation. These functions enable real-time adjustment of filter parameters via a user-friendly interface.

```
% Value changing function: f1
    function f1ValueChanging(app, event)
```

```
        app.freq1 = event.Value;
        updateGraph(app);
    end

    % Value changing function: bw1
    function bw1ValueChanging(app, event)
        app.bandw1 = event.Value;
        updateGraph(app);
    end
```

Another function which needed implementation is described in the following code:

```
        % Value changed function: filter1
    function filter1ValueChanged(app, event)
            app.filtertype(1,:) = killafonic(app.filter1.Value);
            updateGraph(app);
    end
```

This function represents the usage of a dictionary function, named both arbitrary and accordingly, related to the audio spectrum. Because of the issue with MATLAB not being able to change properly the text in the graphic interface because of different number of characters between 2 types of filters set by the user, for example changing from ”all pass” to ”band-pass filter” (they have different number of characters), the decision was using this workaround function, which connects a type of filter with the corresponding name containing only 3 letters, the minimum necessary in order to define all the filters that are used.

```
function short = killafonic(long)

    if strcmp(long(1, :), 'All pass')
        short = 'APF';
    elseif strcmp(long(1, :), 'Low pass')
        short = 'LPF';
    elseif strcmp(long(1, :), 'High pass')
        short = 'HPF';
    elseif strcmp(long(1, :), 'Band pass')
        short = 'BPF';
    elseif strcmp(long(1, :), 'Stop band')
    ...
    ...
    % the same for the rest of the filters

end
```

4.2 State machine for buttons

The provided code contains the two main functions associated with the essential buttons in an equalizer application, the PLAY/STOP and the PAUSE/RESUME buttons.

Code snippet for the PLAY/STOP button:

```
        % Button pushed function: PLAYButton
    function PLAYButtonPushed(app, event)
            %boolvalue = app.PLAYButton.Text ;
            %app.pushed = boolvalue;
            if strcmp(app.PLAYButton.Text,'PLAY')
            app.pushed = 1;
            app.PLAYButton.Text = "STOP";
            updateGraph(app,1);

            elseif strcmp(app.PLAYButton.Text,'STOP')
            app.actuallyStopped = 1;

            updateGraph(app,1);
            app.PLAYButton.Text = 'PLAY';
            end
    end
```

When triggered by a button press event, it first checks the current text displayed on the 'PLAY' button. If the text is 'PLAY', it sets the app.pushed flag to 1 to initiate playback, changes the button text to 'STOP', and then calls the updateGraph function with the argument 1 to update the graph display. On subsequent button presses when the text is 'STOP', it sets the app.actuallyStopped flag to 1, updates the graph display, and changes the button text back to 'PLAY'.

Code snippet for the PAUSE/RESUME button:

```
        % Button pushed function: PAUSEButton
    function PAUSEButtonPushed(app, event)

            app.stopped =1;
            if strcmp(app.PAUSEButton.Text,'RESUME')
                app.PAUSEButton.Text = 'PAUSE';
            else
                app.PAUSEButton.Text = 'RESUME';
            end

            updateGraph(app);
            startupFcn(app);
            cla(app.UIAxes);
    end
```

The PAUSEButtonPushed function controls the behavior of the 'PAUSE'/'RESUME' button. Upon pressing, it sets the app.stopped flag to 1 to indicate pausing. Depending on the current text displayed on the button, if it reads 'RESUME', the function changes it to 'PAUSE'. Conversely, if it displays 'PAUSE', the function changes it to 'RESUME'. It then calls updateGraph to update the graph display, startupFcn for potential startup procedures, and clears the content displayed on the UIAxes (the graph area).

These functions are integral to the playback control mechanisms of the equalizer application. The PLAYButtonPushed function toggles between playing and stopping audio playback, while PAUSEButtonPushed handles pausing and resuming playback. Both functions modify various flags and UI elements to control the playback state and update the display accordingly, ensuring a smooth user experience within the application.

5 Conclusion

Future implementations can be brought to the table, such as using adaptive filters, or Kalman filtering, that given the requirements of our applications, this filter would be the best fit, considering its low time complexity, and ease of implementation, and also to reduce the distortions produced. Another option could be implementing it in C++ or assembly, which would be more complex to work, but they should provide better results, due to being able to perform finer with memory heavy tasks.

The project represented a challenge itself, as multiple sources of information from different backgrounds such as math, DSP(Digital Signal Processing), and programming required to be involved when approaching the desired task. It is also worth noting that although the designed application has a smaller amount of controllable knobs, the advantage comes in the fact that the user has more control over the frequencies, by being able to reposition them as preferred.

The equalizer holds significant potential for integration into advanced applications, bringing a new wave of possibilities within audio engineering. Additionally, this equalizer's design might serve as a cornerstone for developing AI-powered adaptive audio systems, enhancing real-time audio processing for different multimedia experiences. Overall, its implementation opens doors to a multitude of innovative projects that could redefine the landscape of audio engineering in the future.

6 Bibliography

[1] Udo Zölzer, ”DAFX - Digital Audio Effects”, in ”John Wiley and Sons”, 2002

[2] Robert Bristow-Johnson, ”Audio EQ Cookbook”, in ”W3C Working Group Note”, 2021

[3] Royer, Théo, ”Pitch-Shifting Algorithm Design and Applications in Music” Stockholm, Sweden, 2019

[4] P. P. Rayan Kutty and A. Sreenivasa Murthy, ”Kalman filter using quantile based noise estimation for audio restoration,”2011 International Conference on Emerging Trends in Electrical and Computer Technology, Nagercoil, India, 2011, pp. 616-620, doi: 10.1109/ICETECT.2011.5760191.

7 Annex

Under this annex the code used for the Matlab app designer can be found

```
classdef eq_bun < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                  matlab.ui.Figure
        GridLayout                matlab.ui.container.GridLayout
        LeftPanel                 matlab.ui.container.Panel
        PAUSEButton               matlab.ui.control.Button
        PLAYButton                matlab.ui.control.Button
        filter5                   matlab.ui.control.DropDown
        DropDown_5Label           matlab.ui.control.Label
        bw5                       matlab.ui.control.Knob
        f5                        matlab.ui.control.Knob
        filter4                   matlab.ui.control.DropDown
        DropDown_4Label           matlab.ui.control.Label
        bw4                       matlab.ui.control.Knob
        f4                        matlab.ui.control.Knob
        filter3                   matlab.ui.control.DropDown
        DropDown_3Label           matlab.ui.control.Label
        bw3                       matlab.ui.control.Knob
        f3                        matlab.ui.control.Knob
        filter2                   matlab.ui.control.DropDown
        DropDown_2Label           matlab.ui.control.Label
        bw2                       matlab.ui.control.Knob
        f2                        matlab.ui.control.Knob
        filter1                   matlab.ui.control.DropDown
        DropDownLabel             matlab.ui.control.Label
        bw1                       matlab.ui.control.Knob
        f1                        matlab.ui.control.Knob
        RightPanel                matlab.ui.container.Panel
        ChangeTrackDropDown       matlab.ui.control.DropDown
        ChangeTrackDropDownLabel  matlab.ui.control.Label
        UIAxes                    matlab.ui.control.UIAxes
        ContextMenu               matlab.ui.container.ContextMenu
        Menu                      matlab.ui.container.Menu
        Menu2                     matlab.ui.container.Menu
        ContextMenu2              matlab.ui.container.ContextMenu
        Menu_2                    matlab.ui.container.Menu
        Menu2_2                   matlab.ui.container.Menu
    end

    % Properties that correspond to apps with auto-reflow
    properties (Access = private)
        onePanelWidth = 576;
    end


    properties (Access = private)

        actuallyStopped
        newTrack
        audiofile
        fs
        pushed
        freq1
        bandw1
        freq2
        bandw2
        freq3
        bandw3
        freq4
        bandw4
        freq5
        bandw5
        filtertype
        stopped
        needsNewCoefficients
        b
        a
    end

    methods (Access = public)
        % gask - dropdown, slect froma bunch of audio files
        %additioanally, add a flag  on the stop button, that pseudo-pauses
        %the audio file
        %p.s. copy the code from playbutton

        function updateGraph(app,flag)
            if ~exist('flag','var')
                flag  = 0;
            end
                reader = dsp.AudioFileReader(app.audiofile, ...
             'SamplesPerFrame', 1024);

        player = audioDeviceWriter('SampleRate', reader.SampleRate);

            if app.pushed
                [app.audiofile] = selectTrack(app.newTrack);
                app.pushed = 0;

                if(flag)
                flag = 0;

        while ~isDone(reader)
            if app.actuallyStopped
                app.actuallyStopped=0;
                release(player)
                release(reader)
            return;
            end
            pause(0);
            audioIn = reader();
            audioIn = audioIn(:,1);
            if app.needsNewCoefficients
                app.needsNewCoefficients = 0;
                [app.b,app.a] = ...
                CreateFiveFilter(audioIn, reader.SampleRate,...
                app.filtertype, ...
                [app.freq1, app.freq2, app.freq3, app.freq4, app.freq5 ],  ...
                [app.bandw1, app.bandw2, app.bandw3, app.bandw4, app.bandw5 ] );
            end
            app.b = real(app.b);
            app.a = real(app.a);
            y1 = filter(app.b(1,:),app.a(1,:),audioIn);
            y2 = filter(app.b(2,:),app.a(2,:),audioIn);
            y3 = filter(app.b(3,:),app.a(3,:),audioIn);
            y4 = filter(app.b(4,:),app.a(4,:),audioIn);
            y5 = filter(app.b(5,:),app.a(5,:),audioIn);

            g1 = .2;
            g2 = .2;
            g3 = .2;
            g4 = .2;
            g5 = .2;

            direct_path_gain = 0.05;

            audioOut = g1 .* y1 +g2 .* y2 +g3 .* y3 +g4 .* y4 +g5 .* y5 + direct_pa
            if (max(audioOut) > 1)
                audioOut = audioOut/max(audioOut);
            end

            A = linspace(0, 0.9, (200)) ;     %rise 35% of signal
            D = linspace(0.9, 0.7,(500));    %drop of 5% of signal
            S = linspace(0.7, 0.7,(100));    %delay of 40% of signal
            R = linspace(0.7, 0,(1024 -800 ));      %drop of 25% of signal
            ADSR = [A D S R];
            audioOut = audioOut .* ADSR';

            Y = abs(fft(audioOut, 2 * app.fs));
            loglog(app.UIAxes,1 : length(Y)/2,Y(1:length(Y)/2));
            ylim(app.UIAxes,[0,150]);
            drawnow limitrate
            pause(1e-100);

            player(audioOut);


        end
        release(reader);
        release(player);
        app.PLAYButton.Text = 'PLAY';
                end

                %pass - think how to clear the figure and how to reset
                %knobs, evntually will also have to reset dropdowns
            elseif app.stopped
                while ~strcmp(app.PAUSEButton.Text,'PAUSE')
                    pause(.1);
                end
                app.stopped = 0;
            end

        end
```

```
end


% Callbacks that handle component events
methods (Access = private)

    % Code that executes after component creation
    function startupFcn(app)

        [app.audiofile] = selectTrack(app.newTrack);
        app.freq1 = 20;
        app.freq2 = 20;
        app.freq3 = 20;
        app.freq4 = 20;
        app.freq5 = 20;
        app.bandw1 = 1;
        app.bandw2 = 1;
        app.bandw3 = 1;
        app.bandw4 = 1;
        app.bandw5 = 1;
        app.pushed = 0;
        app.stopped = 0;
        app.filtertype = ['APF';'APF';'APF';'APF';'APF'];
        app.needsNewCoefficients = 1;
    end

    % Button pushed function: PLAYButton
    function PLAYButtonPushed(app, event)
        %boolvalue = app.PLAYButton.Text ;
        %app.pushed = boolvalue;
        if strcmp(app.PLAYButton.Text,'PLAY')
        app.pushed = 1;
        app.PLAYButton.Text = "STOP";
        updateGraph(app,1);

        elseif strcmp(app.PLAYButton.Text,'STOP')
        app.actuallyStopped = 1;

        updateGraph(app,1);
        app.PLAYButton.Text = 'PLAY';
        end
    end
```

```
% Button pushed function: PAUSEButton
function PAUSEButtonPushed(app, event)

    app.stopped =1;
    if strcmp(app.PAUSEButton.Text,'RESUME')
        app.PAUSEButton.Text = 'PAUSE';
    else
        app.PAUSEButton.Text = 'RESUME';
    end


    updateGraph(app);
    startupFcn(app);
    cla(app.UIAxes);
end

% Value changing function: f1
function f1ValueChanging(app, event)
    app.freq1 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);
end

% Value changing function: bw1
function bw1ValueChanging(app, event)
    app.bandw1 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);
end

% Value changing function: f2
function f2ValueChanging(app, event)
    app.freq2 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);
end

% Value changing function: bw2
function bw2ValueChanging(app, event)
    app.bandw2 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);

end
```

```
% Value changing function: f3
function f3ValueChanging(app, event)
    app.freq3 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changing function: bw3
function bw3ValueChanging(app, event)
    app.bandw3 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changing function: f4
function f4ValueChanging(app, event)
    app.freq4 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changing function: bw4
function bw4ValueChanging(app, event)
    app.bandw4 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changing function: f5
function f5ValueChanging(app, event)
    app.freq5 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changing function: bw5
function bw5ValueChanging(app, event)
    app.bandw5 = event.Value;
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changed function: filter1
function filter1ValueChanged(app, event)
    app.filtertype(1,:) = killafonic(app.filter1.Value);
    app.needsNewCoefficients = 1;
    updateGraph(app);
end

% Value changed function: filter2
function filter2ValueChanged(app, event)
    app.filtertype(2,:) = killafonic(app.filter2.Value);
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changed function: filter3
function filter3ValueChanged(app, event)
    app.filtertype(3,:) = killafonic(app.filter3.Value);
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changed function: filter4
function filter4ValueChanged(app, event)
    app.filtertype(4,:) = killafonic(app.filter4.Value);
    app.needsNewCoefficients = 1;
    updateGraph(app);

end

% Value changed function: filter5
function filter5ValueChanged(app, event)
    app.filtertype(5,:) = killafonic(app.filter5.Value);
    app.needsNewCoefficients = 1;
    updateGraph(app);

end
```

```
    % Value changed function: ChangeTrackDropDown
    function ChangeTrackDropDownValueChanged(app, event)
        app.newTrack = app.ChangeTrackDropDown.Value;
        [app.audiofile] = selectTrack(app.newTrack);
        updateGraph(app);

    end

    % Changes arrangement of the app based on UIFigure width
    function updateAppLayout(app, event)
        currentFigureWidth = app.UIFigure.Position(3);
        if(currentFigureWidth <= app.onePanelWidth)
            % Change to a 2x1 grid
            app.GridLayout.RowHeight = {987, 987};
            app.GridLayout.ColumnWidth = {'1x'};
            app.RightPanel.Layout.Row = 2;
            app.RightPanel.Layout.Column = 1;
        else
            % Change to a 1x2 grid
            app.GridLayout.RowHeight = {'1x'};
            app.GridLayout.ColumnWidth = {283, '1x'};
            app.RightPanel.Layout.Row = 1;
            app.RightPanel.Layout.Column = 2;
        end
    end
end

% Component initialization
methods (Access = private)

    % Create UIFigure and components
    function createComponents(app)

        % Create UIFigure and hide until all components are created
        app.UIFigure = uifigure('Visible', 'off');
        app.UIFigure.AutoResizeChildren = 'off';
        app.UIFigure.Position = [100 100 1498 987];
        app.UIFigure.Name = 'MATLAB App';
        app.UIFigure.SizeChangedFcn = createCallbackFcn(app, @updateAppLayout,

        % Create GridLayout
        app.GridLayout = uigridlayout(app.UIFigure);
        app.GridLayout.ColumnWidth = {283, '1x'};
        app.GridLayout.RowHeight = {'1x'};
        app.GridLayout.ColumnSpacing = 0;
        app.GridLayout.RowSpacing = 0;
        app.GridLayout.Padding = [0 0 0 0];
        app.GridLayout.Scrollable = 'on';

        % Create LeftPanel
        app.LeftPanel = uipanel(app.GridLayout);
        app.LeftPanel.Layout.Row = 1;
        app.LeftPanel.Layout.Column = 1;

        % Create f1
        app.f1 = uiknob(app.LeftPanel, 'continuous');
        app.f1.Limits = [20 20000];
        app.f1.MajorTicks = [20 5000 10000 15000 20000];
        app.f1.MajorTickLabels = {'20', '5000', '10000', '15000', '20000'};
        app.f1.ValueChangingFcn = createCallbackFcn(app, @f1ValueChanging, true
        app.f1.Position = [86 834 39 39];
        app.f1.Value = 1000;

        % Create bw1
        app.bw1 = uiknob(app.LeftPanel, 'continuous');
        app.bw1.Limits = [-5 5];
        app.bw1.ValueChangingFcn = createCallbackFcn(app, @bw1ValueChanging, t
        app.bw1.Position = [192 834 39 39];
        app.bw1.Value = 1;

        % Create DropDownLabel
        app.DropDownLabel = uilabel(app.LeftPanel);
        app.DropDownLabel.HorizontalAlignment = 'right';
        app.DropDownLabel.Position = [28 785 65 22];
        app.DropDownLabel.Text = 'Drop Down';

        % Create filter1
        app.filter1 = uidropdown(app.LeftPanel);
        app.filter1.Items = {'All pass', 'Low pass', 'High pass', 'Band pass ',
        app.filter1.ValueChangedFcn = createCallbackFcn(app, @filter1ValueChan
        app.filter1.Position = [108 785 100 22];
        app.filter1.Value = 'All pass';

        % Create f2
        app.f2 = uiknob(app.LeftPanel, 'continuous');
        app.f2.Limits = [20 20000];
        app.f2.MajorTicks = [20 5000 10000 15000 20000];
        app.f2.MajorTickLabels = {'20', '5000', '10000', '15000', '20000'};
        app.f2.ValueChangingFcn = createCallbackFcn(app, @f2ValueChanging, true
        app.f2.Position = [86 682 39 39];
        app.f2.Value = 5000;

        % Create bw2
        app.bw2 = uiknob(app.LeftPanel, 'continuous');
        app.bw2.Limits = [-5 5];
        app.bw2.ValueChangingFcn = createCallbackFcn(app, @bw2ValueChanging, t
        app.bw2.Position = [192 682 39 39];
        app.bw2.Value = 1;

        % Create DropDown_2Label
        app.DropDown_2Label = uilabel(app.LeftPanel);
        app.DropDown_2Label.HorizontalAlignment = 'right';
        app.DropDown_2Label.Position = [28 633 65 22];
        app.DropDown_2Label.Text = 'Drop Down';

        % Create filter2
        app.filter2 = uidropdown(app.LeftPanel);
        app.filter2.Items = {'All pass', 'Low pass', 'High pass', 'Band pass ',
        app.filter2.ValueChangedFcn = createCallbackFcn(app, @filter2ValueChan
        app.filter2.Position = [108 633 100 22];
        app.filter2.Value = 'All pass';

        % Create f3
        app.f3 = uiknob(app.LeftPanel, 'continuous');
        app.f3.Limits = [20 20000];
        app.f3.MajorTicks = [20 5000 10000 15000 20000];
        app.f3.MajorTickLabels = {'20', '5000', '10000', '15000', '20000'};
        app.f3.ValueChangingFcn = createCallbackFcn(app, @f3ValueChanging, true
        app.f3.Position = [86 519 39 39];
        app.f3.Value = 5000;

        % Create bw3
        app.bw3 = uiknob(app.LeftPanel, 'continuous');
        app.bw3.Limits = [-5 5];
        app.bw3.ValueChangingFcn = createCallbackFcn(app, @bw3ValueChanging, t
        app.bw3.Position = [192 519 39 39];
        app.bw3.Value = 1;

        % Create DropDown_3Label
        app.DropDown_3Label = uilabel(app.LeftPanel);
        app.DropDown_3Label.HorizontalAlignment = 'right';
        app.DropDown_3Label.Position = [28 470 65 22];
        app.DropDown_3Label.Text = 'Drop Down';

        % Create filter3
        app.filter3 = uidropdown(app.LeftPanel);
        app.filter3.Items = {'All pass', 'Low pass', 'High pass', 'Band pass ',
        app.filter3.ValueChangedFcn = createCallbackFcn(app, @filter3ValueChan
        app.filter3.Position = [108 470 100 22];
        app.filter3.Value = 'All pass';

        % Create f4
        app.f4 = uiknob(app.LeftPanel, 'continuous');
        app.f4.Limits = [20 20000];
        app.f4.MajorTicks = [20 5000 10000 15000 20000];
        app.f4.MajorTickLabels = {'20', '5000', '10000', '15000', '20000'};
        app.f4.ValueChangingFcn = createCallbackFcn(app, @f4ValueChanging, tru
        app.f4.Position = [86 354 39 39];
        app.f4.Value = 5000;

        % Create bw4
        app.bw4 = uiknob(app.LeftPanel, 'continuous');
        app.bw4.Limits = [-5 5];
        app.bw4.ValueChangingFcn = createCallbackFcn(app, @bw4ValueChanging, t
        app.bw4.Position = [192 354 39 39];
        app.bw4.Value = 1;

        % Create DropDown_4Label
        app.DropDown_4Label = uilabel(app.LeftPanel);
        app.DropDown_4Label.HorizontalAlignment = 'right';
        app.DropDown_4Label.Position = [28 305 65 22];
        app.DropDown_4Label.Text = 'Drop Down';

        % Create filter4
        app.filter4 = uidropdown(app.LeftPanel);
        app.filter4.Items = {'All pass', 'Low pass', 'High pass', 'Band pass ',
        app.filter4.ValueChangedFcn = createCallbackFcn(app, @filter4ValueChan
        app.filter4.Position = [108 305 100 22];
        app.filter4.Value = 'All pass';

        % Create f5
        app.f5 = uiknob(app.LeftPanel, 'continuous');
        app.f5.Limits = [20 20000];
        app.f5.MajorTicks = [20 5000 10000 15000 20000];
        app.f5.MajorTickLabels = {'20', '5000', '10000', '15000', '20000'};
        app.f5.ValueChangingFcn = createCallbackFcn(app, @f5ValueChanging, tru
        app.f5.Position = [86 178 39 39];
        app.f5.Value = 5000;

        % Create bw5
        app.bw5 = uiknob(app.LeftPanel, 'continuous');
        app.bw5.Limits = [-5 5];
        app.bw5.ValueChangingFcn = createCallbackFcn(app, @bw5ValueChanging, tr
        app.bw5.Position = [192 178 39 39];
        app.bw5.Value = 1;

        % Create DropDown_5Label
        app.DropDown_5Label = uilabel(app.LeftPanel);
        app.DropDown_5Label.HorizontalAlignment = 'right';
        app.DropDown_5Label.Position = [28 129 65 22];
        app.DropDown_5Label.Text = 'Drop Down';

        % Create filter5
        app.filter5 = uidropdown(app.LeftPanel);
        app.filter5.Items = {'All pass', 'Low pass', 'High pass', 'Band pass ',
        app.filter5.ValueChangedFcn = createCallbackFcn(app, @filter5ValueChanc
        app.filter5.Position = [108 129 100 22];
        app.filter5.Value = 'All pass';

        % Create PLAYButton
        app.PLAYButton = uibutton(app.LeftPanel, 'push');
        app.PLAYButton.ButtonPushedFcn = createCallbackFcn(app, @PLAYButtonPusł
        app.PLAYButton.Position = [28 73 100 23];
        app.PLAYButton.Text = 'PLAY';

        % Create PAUSEButton
        app.PAUSEButton = uibutton(app.LeftPanel, 'push');
        app.PAUSEButton.ButtonPushedFcn = createCallbackFcn(app, @PAUSEButtonPu
        app.PAUSEButton.Position = [28 29 100 23];
        app.PAUSEButton.Text = 'PAUSE';

        % Create RightPanel
        app.RightPanel = uipanel(app.GridLayout);
        app.RightPanel.Title = 'EQ';
        app.RightPanel.Layout.Row = 1;
        app.RightPanel.Layout.Column = 2;

        % Create UIAxes
        app.UIAxes = uiaxes(app.RightPanel);
        title(app.UIAxes, 'Real time audio replay')
```

```
app.UIAxes.XTick = [];
app.UIAxes.YTick = [];
app.UIAxes.Position = [51 95 1083 793];

% Create ChangeTrackDropDownLabel
app.ChangeTrackDropDownLabel = uilabel(app.RightPanel);
app.ChangeTrackDropDownLabel.HorizontalAlignment = 'right';
app.ChangeTrackDropDownLabel.Position = [50 905 80 22];
app.ChangeTrackDropDownLabel.Text = 'Change Track';

% Create ChangeTrackDropDown
app.ChangeTrackDropDown = uidropdown(app.RightPanel);
app.ChangeTrackDropDown.Items = {'Exemplu7', 'interface', 'Baby again',
app.ChangeTrackDropDown.ValueChangedFcn = createCallbackFcn(app, @Chanc
app.ChangeTrackDropDown.Position = [145 905 653 22];
app.ChangeTrackDropDown.Value = 'Exemplu7';

% Create ContextMenu
app.ContextMenu = uicontextmenu(app.UIFigure);

% Create Menu
app.Menu = uimenu(app.ContextMenu);
app.Menu.Text = 'Menu';

% Create Menu2
app.Menu2 = uimenu(app.ContextMenu);
app.Menu2.Text = 'Menu2';

% Assign app.ContextMenu
app.f1.ContextMenu = app.ContextMenu;

% Create ContextMenu2
app.ContextMenu2 = uicontextmenu(app.UIFigure);

% Create Menu_2
app.Menu_2 = uimenu(app.ContextMenu2);
app.Menu_2.Text = 'Menu';

% Create Menu2_2
app.Menu2_2 = uimenu(app.ContextMenu2);
app.Menu2_2.Text = 'Menu2';

% Assign app.ContextMenu2
app.PLAYButton.ContextMenu = app.ContextMenu2;
```

```
            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = eq_bun

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```