UNIVERSITY OF
WEST LONDON

ASSESSMENT 2 REPORT: BUILDING A 3 BAND EQUALIZER USING MATLAB FRAMEWORK

Implementing Techniques from Digital Signal Processing

Al Ameer Asyraf Bin Mohamed Hassan (ID: 21500396)

21500396@student.uwl.ac.uwk

Table of Contents

List of figures and tables

Abstract

Equalizers are essential when it comes to sound recording and reproduction. Equalizing or Equalization (EQ) is a process where the volume of different frequency bands is altered (Wikipedia, 2022). It is built by a combination of Low Pass, High Pass, Shelving and Peaking filters. Nowadays, commonly used by musicians and producers, better and futuristic EQ's are being built all the time for a better frequency-magnitude altering tool. This paper serves as the design overview of the 3 Band EQ built for the Signal Processing module and results show that the program designed is functioning as expected. Analysis section show that the user desired audio sample is altered based on his or her parameters. The paper concludes by proving the comprehension of signal processing skills as well as suggestions of improving the program in future works.

Introduction

This paper overviews the assessment assigned in Digital Signal Processing (DSP) module for Assessment 2 and instructions were given to design a three-band control equalizer with a switchable loudness control. The user should have control over the Low Cut Off Frequency and High Cut Off Frequency, as well as the Boost/Cut control of gain in the Low, Mid and High Frequency region. The Loudness control is a binary '1' or '0' choice and this will make the filtered signal have a boost in gain. All these controls should be nicely packaged in a GUI designed in MATLAB. The program will use the techniques learnt in the DSP module and that will meet the criteria of the requirement. The sections in this paper will cover the program's method of development, the results by GUI presentation and unit testing using native MATLAB, as well as a simple analysis section discussing the impact of the program on a sample audio signal. The paper strives to show the comprehension of DSP techniques, GUI production and design specifically for MATLAB.

Method of Development

MATLAB's built-in App Designer was the base framework for this program. It was a closed system where everything was built within that one .MLAPP file without needed a dependency from any other file. The assessment requested a MATLAB script as a mandatory while the GUI as additional credit work. However, during the design phase of this program, the GUI as built first as a basis and due to time constraints, the program was only able to be completed in App format, instead of native MATLAB script (.m). Future works can implement sample code design in normal MATLAB format.

Graphical User Interface (GUI)

For the GUI, based on the requirements, a minimum of 3 knobs for the Gain Control, 2 knobs for Low and High Cut Off Frequencies, and a switch for Loudness Control. The knobs are built by using the components from the Component Library.

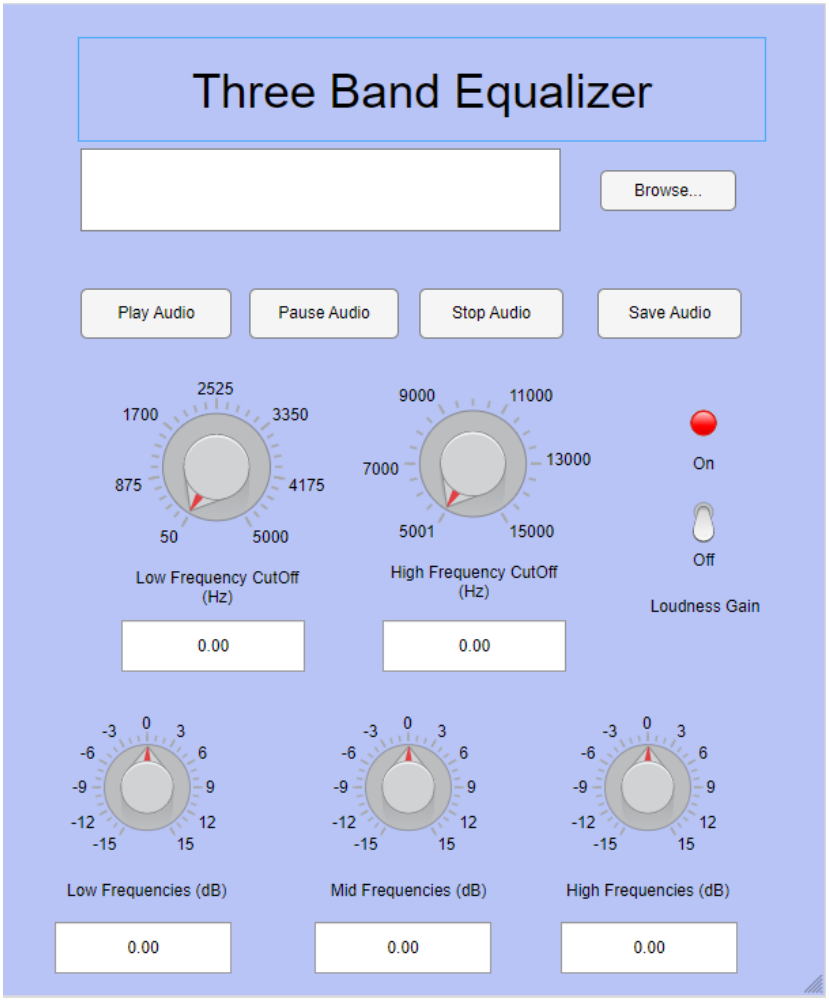

Figure 1.1: Three Band Equalizer GUI with Low and High Cut off Frequency, Gain Control for Low, Mid, High Frequencies.

Figure 1.1 in the previous page shows the GUI of the program with the required knobs. Each textbox corresponds to its knob by linking them in their respective callback functions.

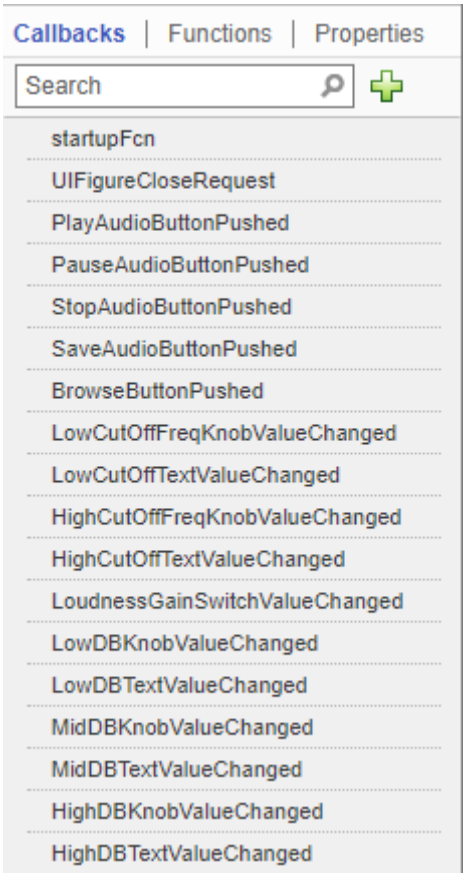

Figure 1.2: Three Band Equalizer GUI with Low and High Cut off Frequency, Gain Control for Low, Mid, High Frequencies.

Figure 1.2 above shows the respective callback functions for each component which holds the code that links the knob to the textboxes as well as get the value to perform the signal processing. The user is also able to choose the desired file by using the *uigetfile()* and specify the file formats to WAV, OGG, FLAC, and MP3. Figure 1.3 below shows the code that allows the user to load the file into the program, i.e., when the Browse button is pressed.

```
% Button pushed function: BrowseButton
function BrowseButtonPushed(app, event)
    [app.audio_file, app.audio_filepath] = uigetfile({'*.wav;*.ogg;*.oga;*.flac;*.mp3;',...
                                                      'All Audio Files (*.wav,*.ogg,*.oga,*.flac,*.mp3)';...
                                                      '*.wav','WAVE File (*.wav)';...
                                                      '*.ogg;*.oga','OGG File (*.ogg,*.oga)';...
                                                      '*.flac','FLAC File (*.flac)';...
                                                      '*.mp3','MP3 File (*.mp3)'},'Browse');
    figure(app.UIFigure);

    if ~isequal(app.audio_file, 0)
        [app.input_audio_samples, app.sample_rate] = audioread([app.audio_filepath, app.audio_file]);

        L = length(app.input_audio_samples);

        % Low, Mid and High Frequencies Boost/Cut Reset
        resetParameters(app);

        % Loaded Audio File Name
        [~, app.audio_filename, app.audio_file_ext] = fileparts([app.audio_filepath, app.audio_file]);
        app.FileNameTextBar.Value = [app.audio_filename, app.audio_file_ext];
    end
```

Figure 1.3: Code to the callback function of the Browse button when pushed.

In Figure 1.3 in the previous page, the lines in the highlighted box show the file information grabbed by the *uigetfile()* based on the formats. Using that information, it will be displayed in the textbox next to the button as well as read the file to get the sample and sample rate information. Using the sample information, the signal processing can be done using math and built-in functions.

Comparing to the JUCE framework in Assessment 1, the component building in MATLAB much simpler and very high level. MATLAB takes care of the back-end code of the component placement and design. The following section will discuss the signal processing method for this program.

Signal Manipulation and Processing (DSP)

For this program, the equalizer was built using 2 shelving filters (Low Shelf and High Shelf) as well as a peaking filter. The coefficients were derived from a table from *DAFX: Digital Audio Effects*, where the coefficients for a second order shelving and peaking filter were shown how to obtain and manipulate to almost control three parameters independently: center frequency, bandwidth, and gain (Zolzer, 2011).

```
function [a, b] = Shelving(app, filterType, gainValue)

    q_val = 1 / 0.8;

    if abs(gainValue) <= 0
        a = 1;
        b = 1;
    else
        if (strcmp(filterType, 'l'))
            K = tan( (pi * app.lowcutoff_val) / app.sample_rate);
        else
            K = tan( (pi * app.highcutoff_val) / app.sample_rate);
        end

        V0 = 10^(gainValue / 20);

        % Low Boost
        if (gainValue >= 0 && strcmp(filterType, 'l'))

            a1 = 2 * (K^2 - 1) / (1 + q_val * K + K^2);
            a2 = (1 - q_val * K + K^2) / (1 + q_val * K + K^2);
            b0 = (1 + (sqrt(V0) * q_val) * K + V0 * K^2) / (1 + q_val * K + K^2);
            b1 = 2 * (V0 * K^2 - 1) / (1 + q_val * K + K^2);
            b2 = (1 - (sqrt(V0) * q_val) * K + V0 * K^2) / (1 + q_val * K + K^2);

        % Low Cut
        elseif (gainValue < 0 && strcmp(filterType, 'l'))

            a1 = 2 * (K^2 - V0) / (V0 + (sqrt(V0) * q_val) * K + K^2);
            a2 = (V0 - (sqrt(V0) * q_val) * K + K^2) / (V0 + (sqrt(V0) * q_val) * K + K^2);
            b0 = V0 * (1 + q_val * K + K^2) / (V0 + (sqrt(V0) * q_val) * K + K^2);
            b1 = 2 * V0 * (K^2 - 1) / (V0 + (sqrt(V0) * q_val) * K + K^2);
            b2 = V0 * (1 - q_val * K + K^2) / (V0 + (sqrt(V0) * q_val) * K + K^2);

        % High Boost
        elseif (gainValue >= 0 && strcmp(filterType, 'h'))

            a1 = 2 * (K^2 - 1) / (1 + q_val * K + K^2);
            a2 = (1 - q_val * K + K^2)/(1 + q_val * K + K^2);
            b0 = (V0 + (sqrt(V0) * q_val) * K + K^2) / (1 + q_val * K + K^2);
            b1 = 2 * (K^2 - V0) / (1 + q_val * K + K^2);
            b2 = (V0 - (sqrt(V0) * q_val) * K + K^2) / (1 + q_val * K + K^2);
```

Figure 1.4: Code for the Shelving filter and equations for Low Boost, Low Cut, High Boost.

```
%High Cut
else

    a1 = 2 * (V0 * K^2 - 1) / (1 + (sqrt(V0) * q_val) * K + V0 * K^2);
    a2 = (1 - (sqrt(V0) * q_val) * K + V0 * K^2) / (1 + (sqrt(V0) * q_val) * K + V0 * K^2);
    b0 = V0 * (1 + q_val * K + K^2) / (1 + (sqrt(V0) * q_val) * K + V0 * K^2);
    b1 = 2 * V0 * (K^2 - 1) / (1 + sqrt(2*V0) * K + V0 * K^2);
    b2 = V0 * (1 - q_val * K + K^2) / (1 + (sqrt(V0) * q_val) * K + V0 * K^2);

end

a = [1, a1, a2];
b = [b0, b1, b2];
```

Figure 1.5: Code for the Shelving filter and equation for High Cut and final coefficients.

Figure 1.4 in the previous page and Figure 1.5 above shows how the coefficients are obtained. As previously mentioned, these equations are obtained from Zolzer's book, and the required values are calculated using his equations as well. The K value and V_0 are calculated using the respective formulas (Zolzer, 2011):

$$K = \tan(\pi fc/fs) \text{ and } V_0 = 10^{(G/20)}$$

```
function [a, b] = Peaking(app, gainValue)

    if abs(gainValue) <= 0.01
        a = 1;
        b = 1;
    else
        app.bandwidth_val = app.highcutoff_val - app.lowcutoff_val;
        K = tan(pi * app.bandwidth_val / app.sample_rate);
        V0 = 10^(gainValue / 20);
        Q = 0.8;

        if (gainValue >= 0)
        % Boost

            a1 = 2 * (K^2 - 1) / (1 + K / Q + K^2);
            a2 = (1 - K / Q + K^2) / (1 + K / Q + K^2);
            b0 = (1 + V0 * K / Q + K^2) / (1 + K / Q + K^2);
            b1 = 2 * (K^2 - 1) / (1 + K / Q + K^2);
            b2 = (1 - V0 * K / Q + K^2) / (1 + K / Q + K^2);

        else
        % Cut

            a1 = 2 * (K^2 - 1) / (1 + K / (V0 * Q) + K^2);
            a2 = (1 - K / (V0 * Q) + K^2) / (1 + K / (V0 * Q) + K^2);
            b0 = (1 + K / Q + K^2) / (1 + K / (V0 * Q) + K^2);
            b1 = 2 * (K^2 - 1) / (1 + K / (V0 * Q) + K^2);
            b2 = (1 - K / Q + K^2) / (1 + K / (V0 * Q) + K^2);

        end
        a = [1, a1, a2];
        b = [b0, b1, b2];
    end
end
```

Figure 1.6: Code for the Peaking filter and equation for Boost and Cut.

Figure 1.6 in the previous page shows the peaking filter and similar method of coefficient derivation is used. Figure 1.7 below shows an example code of how these functions are used to manipulate the signal. The user defined parameters are grabbed using the component objects and sent to the function as passing parameters.

```
% Value changed function: LowCutOffFreqKnob
function LowCutOffFreqKnobValueChanged(app, event)
    lowCutOffVal = app.LowCutOffFreqKnob.Value;
    app.LowCutOffText.Value = lowCutOffVal;
    app.lowcutoff_val = lowCutOffVal;

    [a, b] = Shelving(app, 'l', app.LowDBText.Value);
    [app.low_response, app.angular_response] = freqz(b, a, 2048);
end
```

Figure 1.7: Code to use the user defined parameters to alter the signal.

The function returns the coefficients a and b, and those values are used by the function *freqz()* to obtain the region frequency response and angular response. Similar steps are repeated for the Middle Frequencies and High Frequencies. The function in Figure 1.7 is specifically to obtain the Low Frequency Value while still using the value set in the Low DB Gain Value for the Shelving function.

The Peaking filter is used for the Middle Frequencies, and it possesses similar steps as the function in Figure 1.7.

```
function [output_audio] = Loudness(app, input_audio)
    if app.loudness_gain_flag == 0
        output_audio = input_audio;
    else
        weightFilt = weightingFilter('K-weighting', app.sample_rate);
        output_audio = weightFilt(input_audio);
    end
end
```

Figure 1.8: Code to affect Loudness Gain control.

Figure 1.8 in the previous page shows the Loudness control function and it is shown that based on the loudness gain flag, the audio output will be boosted in gain using the *weightingFilter()* and the parameter 'K-weighting' will normalize the audio using two stages of filtering: shelving filter and a high pass filter.

With that, a clear description of how the program was developed has been discussed. The following section will discuss the results of the design of the program and how the audio is impacted by the program.

Results

As shown in the previous section, the GUI is built as per the requirement. Figure 2.1 below shows the GUI after a sample is loaded into the program. A default value of 2525 Hz and 10000 Hz is set for Low and High Cut of Frequency respectively.

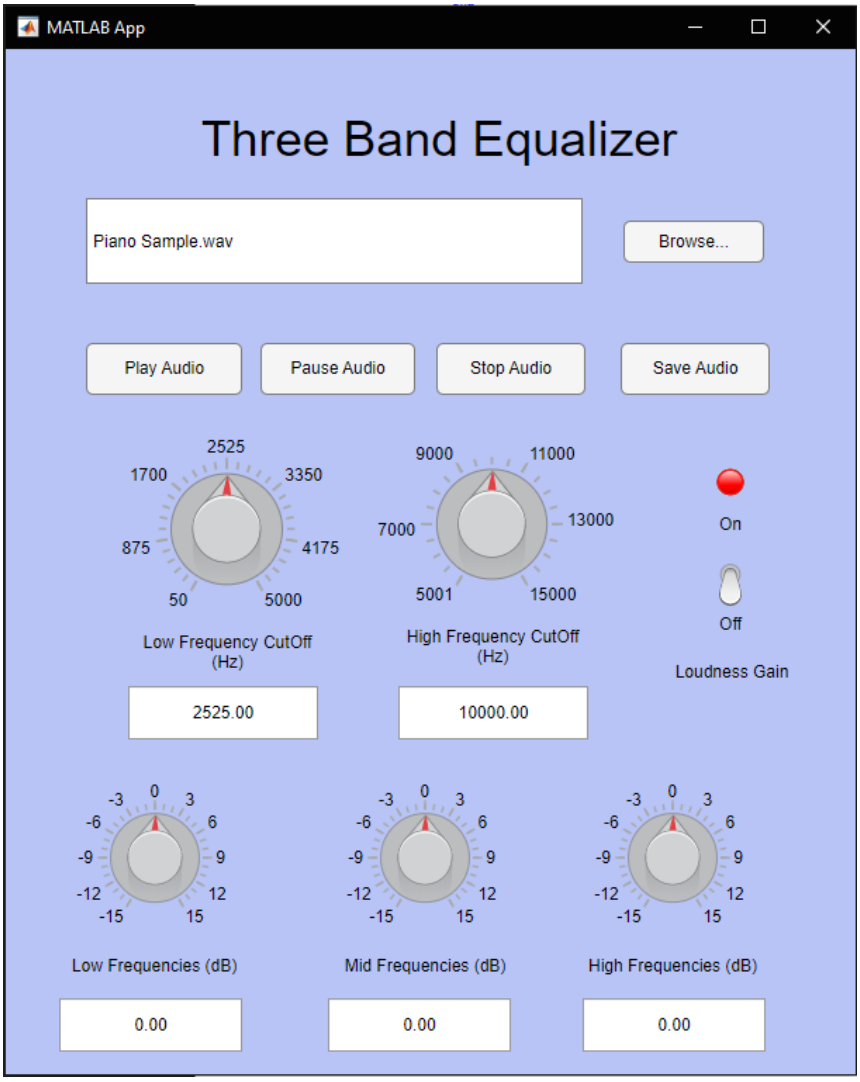

Figure 2.1: GUI after loading a sample by user.

When ‘Play Audio’ button is pressed, the audio loaded into the program is played with the values set as default. Figure 2.2 below shows the GUI after the user tempers with the values. The textbox corresponding to each knob shows the correct value that the knob is assigned to. The Loudness Gain control also comes with a lamp to indicate to the user if it is on or not.

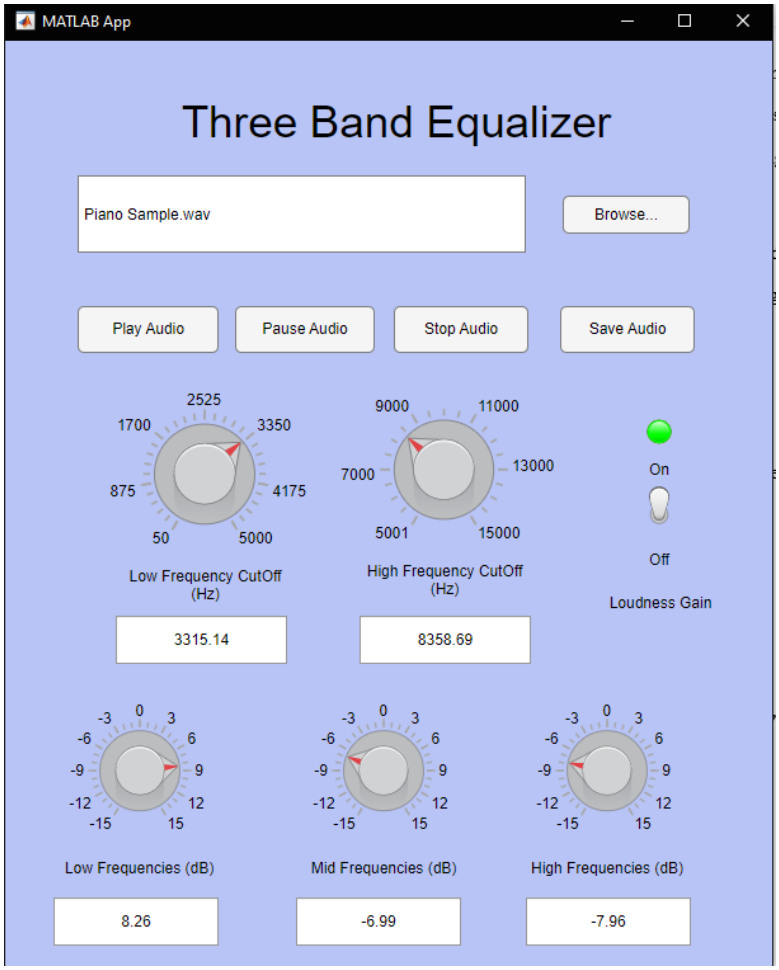

Figure 2.2: GUI after changing the values to alter sample by user.

Besides that, once the ‘Play Audio’ button is pressed, a Magnitude vs Frequency plot is shown to show how the filter is operating. The following section will cover the plot and discuss its impact and how it is impacted.

Currently, the ‘Pause Audio’ button is able to pause the audio at the correct sample position, however, to resume playing at the position is currently not successful due to time constraints. Future works can be done to improve the algorithm to be able to resume the audio at the paused place.

When the ‘Save Audio’ button is pressed, a pop-up window appears to allow the user to define the path, the name, and the format of the audio the user would like to save it in. If the user does not define the name, the default name is to add the string “_Filtered” to the end of the original sample name to ensure the user knows which is the new file. This concludes the results section of the program, and the next section will analyze how the program affects the audio.

Analysis

Figure 3.1 below shows the how the filter impacts the incoming audio signal. The frequency range has been normalized and its in rad/sample unit while the magnitude is in dB unit.

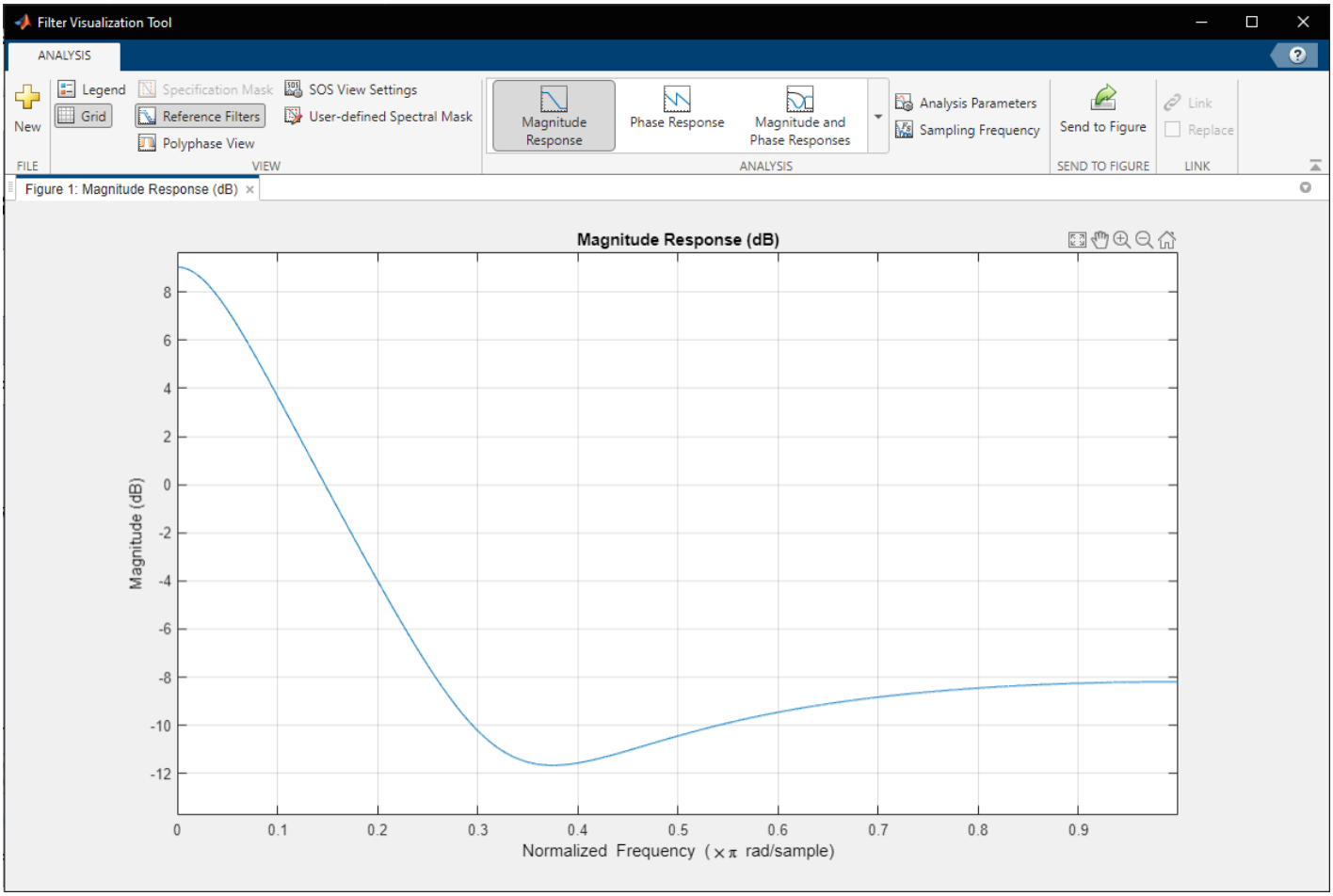

Figure 3.1: Magnitude vs Frequency plot after filter is applied.

It is clear from this plot that based on the parameters set (Low: 8.26 DB, Mid: -6.99, High: -7.96), the audio has been impacted accordingly in the respective regions. The Low Frequency region has been boosted above the 8 dB mark, while the High Frequency has been capped at close to -8 dB.

The Mid Frequency region has a much larger dip than ~-7 dB and that is due to the Q value set at the design stage (0.8) as well as the huge gain boost in the Low Frequency region. The audio sample can be heard to be altered in the affected region as expected.

Besides that, the program also runs into performance issues after a few complex steps. The program seems to not respond when playing audio after saving the audio file. Future works can look into how to solve this issue and build a more robust and efficient program.

Conclusion

In summary, the program has been built according to the requirements set by the assessment. Although a few requirements were not met due to time constraints, the concept needed to be displayed has been successfully applied in the program to obtain the desired effect. A GUI was built using the MATLAB framework, comprehension of filter design was shown by referencing the equations used, as well as plot analysis after applying filter to show desired impact of the combined filters. The techniques applied to alter CutOff Frequencies and Gain within a sample has been proved to be comprehensive by this technical report. There is significant room for improvement in this program from filter design perspective such as a better range of frequencies, a more robust algorithm as well as Q factor manipulation option.

References

1. Huang, Q., 2022. *Filter Design Using MatLab,* London: Microsoft PowerPower.
2. MathWorks, 2022. *Frequency-weighted filter - MATLAB.* [Online] Available at: https://uk.mathworks.com/help/audio/ref/weightingfilter-system-object.html#mw_13845833-724c-4a1a-8ba0-a53e43b88474 [Accessed 07 06 2022].
3. Reiss, J. D. & McPherson, A. P., 2014. *Audio Effects: Theory, Implementation and Application.* 1st ed. London: CRC Press.
4. Wikipedia, 2022. *Equalization (audio).* [Online] Available at: https://en.wikipedia.org/wiki/Equalization_(audio) [Accessed 07 06 2022].
5. Zolzer, U., 2011. *DAFX: Digital Audio Effects.* 2nd ed. West Sussex: John Wiley and Sons, Ltd..

Appendices

Due to the large nature of the code, this program has been uploaded to a GitHub repository at the link below:

https://github.com/alameerasyraf/3BandEqualizer

Please contact the author of this paper if any issues is caused by this method of submission.