EE321
DIGITAL SIGNAL PROCESSING

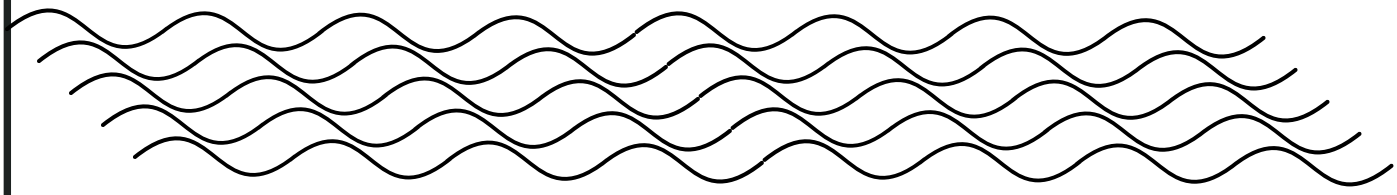
ASSIGNMENT (PS 03) REPORT

BY

ANSHU APOORVA
210102020

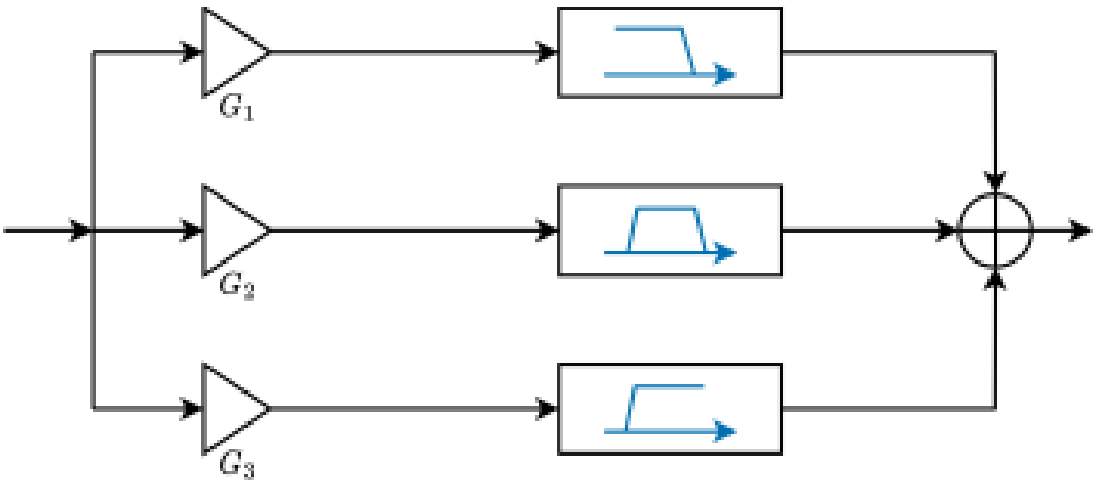
KARAN MARKANDEY
210102042

ASSMIT SHUKLA
210102129



PS 03

A music equalizer can adjust the amplitude of the audio signal at different frequency bands. A music file music.wav is provided. You will implement a 3-band music equalizer. The audio spectrum will be divided into 3 bands: 20 Hz – 200 Hz, 200 Hz – 2 KHz, and 2 KHz – 20 KHz, each with adjustable gains. The block diagram of the 3-band equalizer is shown as follows, where Gn ∈ [0, 1] is the gain of the n th band.



FIR Filter

Finite Impulse Response (FIR) filters are digital filters characterized by a finite-duration impulse response. They offer design flexibility, allowing us to shape frequency responses to meet specific requirements. FIR filters achieves linear phase while preserving the relative timing of input signal. They are stable and suitable for a wide range of applications.
Designing an FIR filter involves selecting the filter type and cut-off frequency, choosing appropriate window and calculating coefficients.
For this assignment we are using the Kaiser window.

Kaiser Window

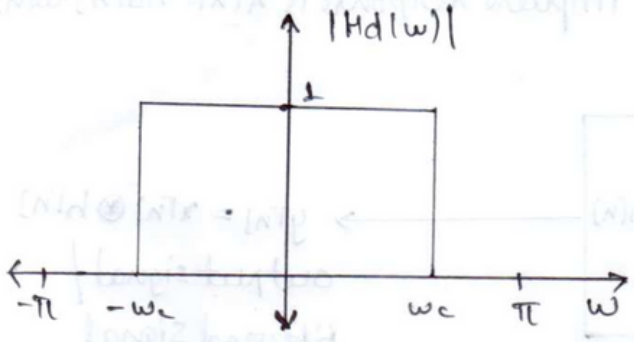
The Kaiser window is a window function commonly used in digital signal processing, particularly for designing Finite Impulse Response (FIR) filters. It is defined by a shape parameter, denoted as β, which governs the trade-off between the main lobe width and the level of side lobes.

The Kaiser window is defined as

$$w_K[n] = \begin{cases} \dfrac{I_0\left\{\beta\left[1-\left(\frac{n-\alpha}{\alpha}\right)^2\right]^{1/2}\right\}}{I_0(\beta)} & n = 0, \ldots, L-1 \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha = \frac{L-1}{2}$ and I_0 is the zeroth-order modified Bessel function of the first kind.

Kaiser window - Low Pass FIR Filter :-



using inverse fourier transform

$$h[t] = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega) \cdot e^{j\omega t} \, d\omega$$

we get,

$$h_d[n] = \begin{cases} \frac{\sin \omega_c (n-\alpha)}{\pi(n-\alpha)} & n \neq \alpha \\ \frac{\omega_c}{\pi} & n = \alpha \end{cases}$$

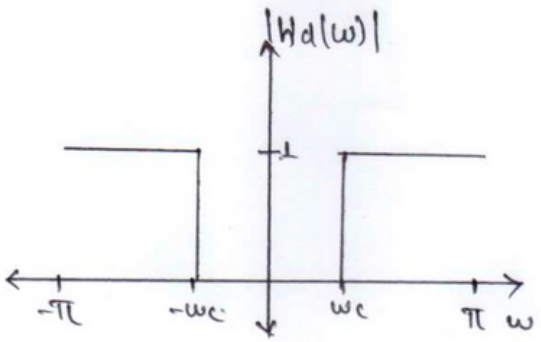
where $\alpha = \frac{N}{2}$, $\omega_c = \frac{2\pi f_c}{f_s}$

therefore,

$$h_d[n] = \begin{cases} \frac{2 \cdot f_c}{f_s} \operatorname{sinc}\left(\frac{2f_c}{f_s}(n-\alpha)\right) & n \neq \alpha \\ \frac{2f_c}{f_s} & n = \alpha \end{cases}$$

we have $h[n] = h_d[n] \cdot w[n]$; $w[n]$ = Kaiser window function

Kaiser window - High Pass FIR Filter :-



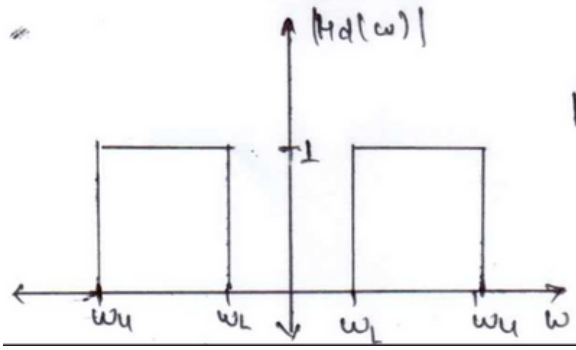
similarly using inverse fourier transform, we get

$$h_d[n] = \begin{cases} -\frac{2f_c}{f_s} \operatorname{sinc} \frac{2f_c}{f_s}(n-\alpha) & n \neq \alpha \\ -\frac{2f_c}{f_s} & n = \alpha \end{cases}$$

$$\alpha = N/2$$

$h[n] = h_d[n] \cdot w[n]$; $w[n]$ = kaiser window function

Kaiser-window Bandpass FIR Filter :-

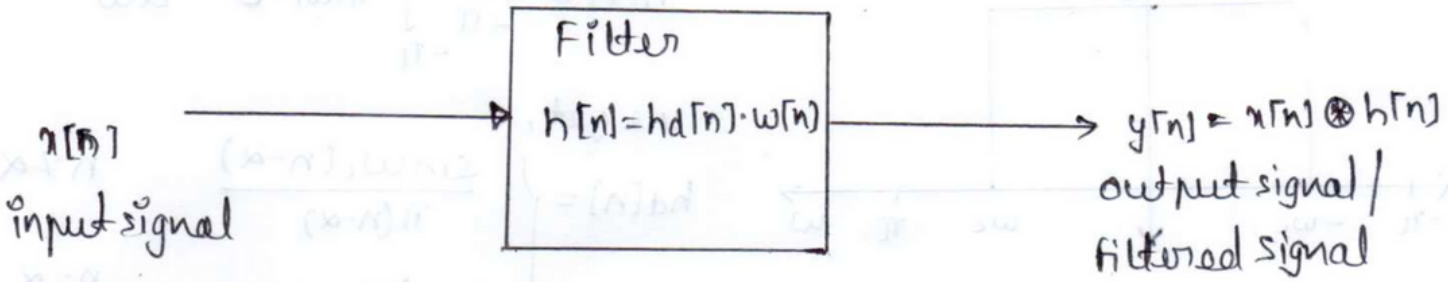


using inverse fourier transform, we get

$$h_d[n] = \begin{cases} \frac{2f_H}{f_s} \operatorname{sinc}\left(\frac{2f_H}{f_s}(n-\alpha)\right) - \frac{2f_L}{f_s} \operatorname{sinc}\left(\frac{2f_L}{f_s}(n-\alpha)\right) & n \neq \alpha \\ \frac{2(f_H - f_L)}{f_s} & n = \alpha \end{cases}$$

$$\alpha = N/2$$

The filter is an LTI system whose impulse response is $h[n] = h_d[n] \cdot w[n]$



Impulse response of an ideal filter $h_d[n]$ is not practically realisable. So we use window method to obtain $h[n] = h_d[n] \cdot w[n]$ which is practically realiseable.

Part 1

First you have to design three filters (low-pass filter, band-pass filter, and high-pass filter) in Python. Use FIR filter with Kaiser window, β = 0.5, order = 80, Fs = 8KHz. Plot the magnitude and phase response of all the designed filters. (Do not use any DSP Python libraries for filter design. Build your own filter function.)

Solution:

Link to Code : Google Colab

Approach:

Kaiser Window Function:

- The function kaiser_window(beta, M) generates Kaiser window coefficients. It takes a shape parameter beta and a window length M as input

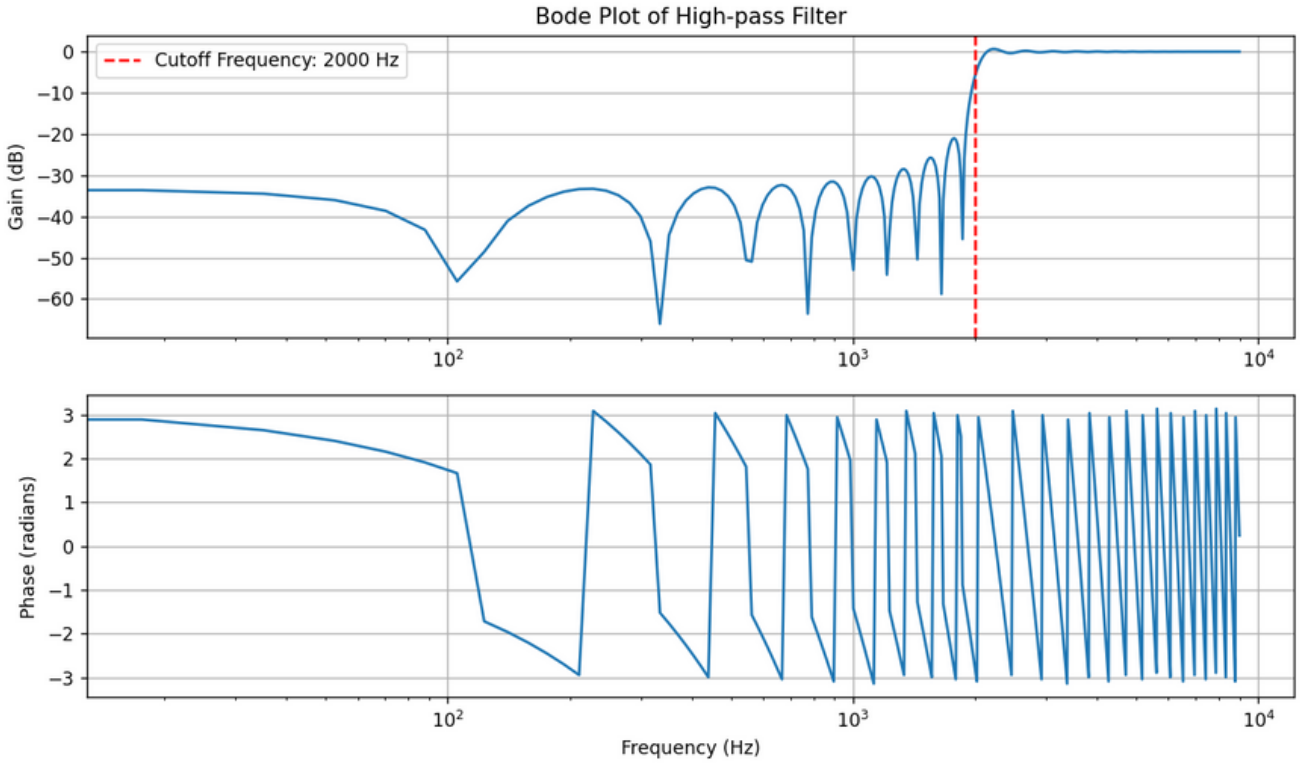
FIR Filter Design Function:

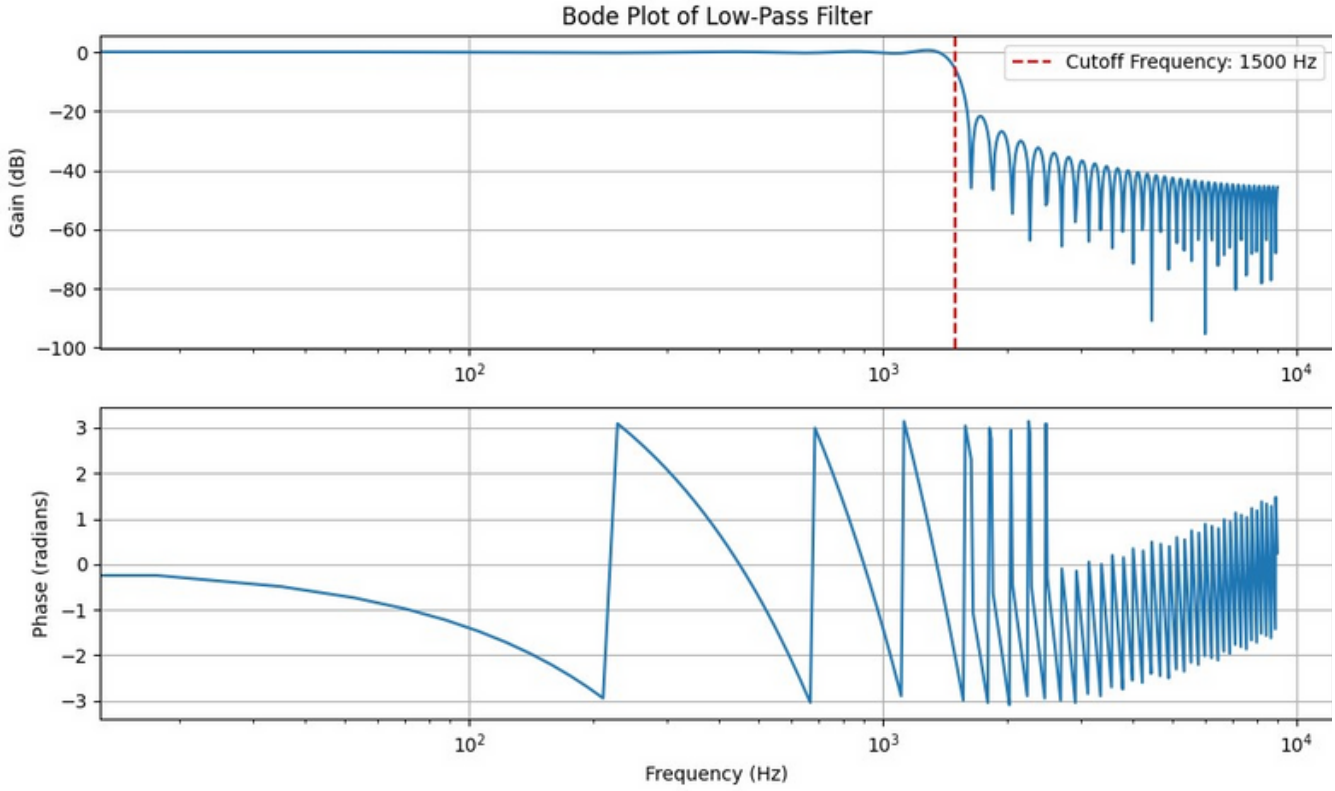
- The function fir_filter_design designs FIR filters using the Kaiser window.
- It takes parameters like cutoff_freq (either a scalar for low/high-pass or a list for band-pass), filter_type ('lowpass', 'highpass', or 'bandpass'), Kaiser window shape parameter beta, filter order order, and sampling frequency Fs.
- The filter coefficients are designed based on the specified filter type and cutoff frequencies, and then the Kaiser window is applied.

Bode Plotting Function:

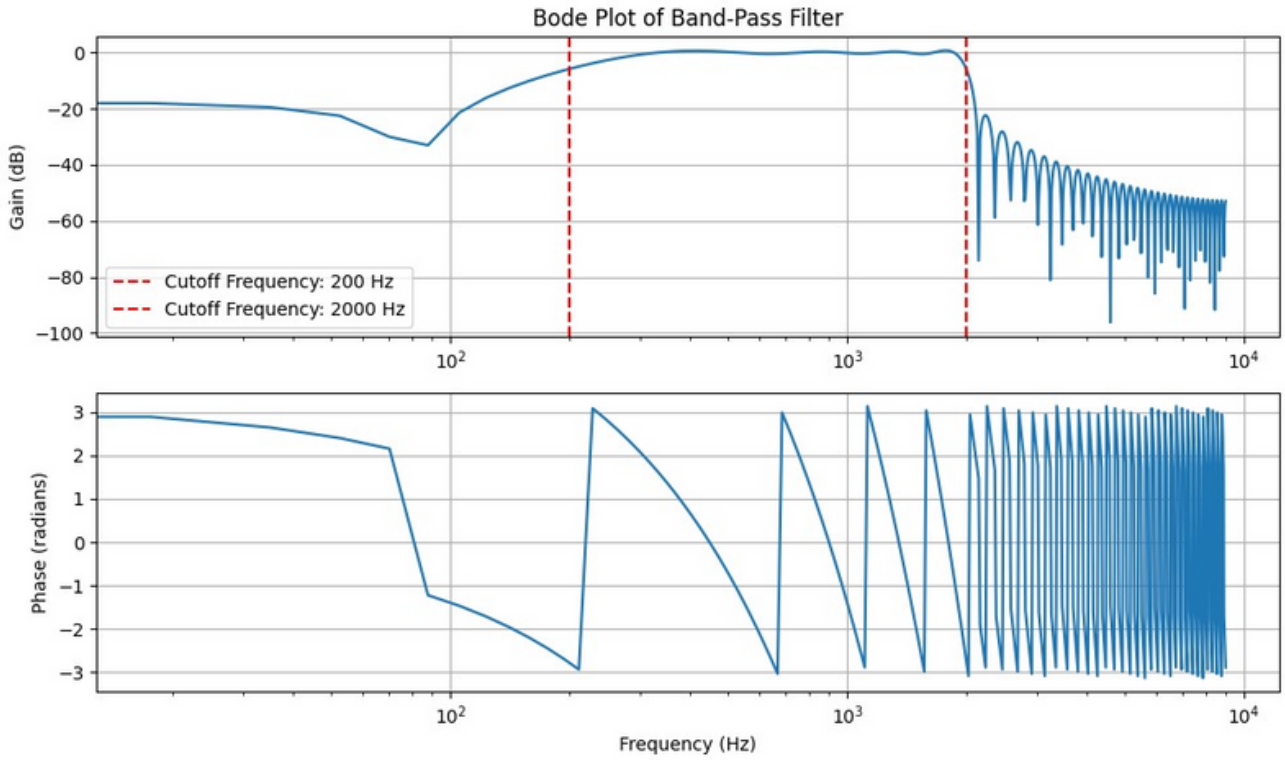
- The function bode_plot generates a Bode plot for a given filter system.
- It calculates the frequency response using freqz from scipy.signal and plots both the magnitude and phase responses.

Output Response





Fs = 8Khz ,β = 0.5 , N = 80



Fs = 8Khz ,β = 0.5 , N = 80

Part 2

First you have to design three filters (low-pass filter, band-pass filter, and high-pass filter) in Python. Use FIR filter with Kaiser window, β = 0.5, order = 80, Fs = 8KHz. Plot the magnitude and phase response of all the designed filters. (Do not use any DSP Python libraries for filter design. Build your own filter function.)

Solution:

Link to Code : Google Colab

Approach:

Audio File Reading and Parameters:

- The code reads an audio file (music.wav) and extracts the signal.
- Filter parameters like the Kaiser window parameter (beta), filter order (order), and sampling frequency (Fs) are set.

Band-Pass Filter Design for Equalizer Bands:

- Three high-pass filters are designed to create three bands for the equalizer. Each band has a different cutoff frequency and contributes to a specific frequency range.

.

Equalization and Output:

- The input signal is equalized by applying the three band-pass filters to different portions of the original signal.
- The filtered signals are then combined with different gains (Gain1, Gain2, Gain3).
- To avoid clipping, the input signal is divided by 3 before applying the filters.

Output Writing:

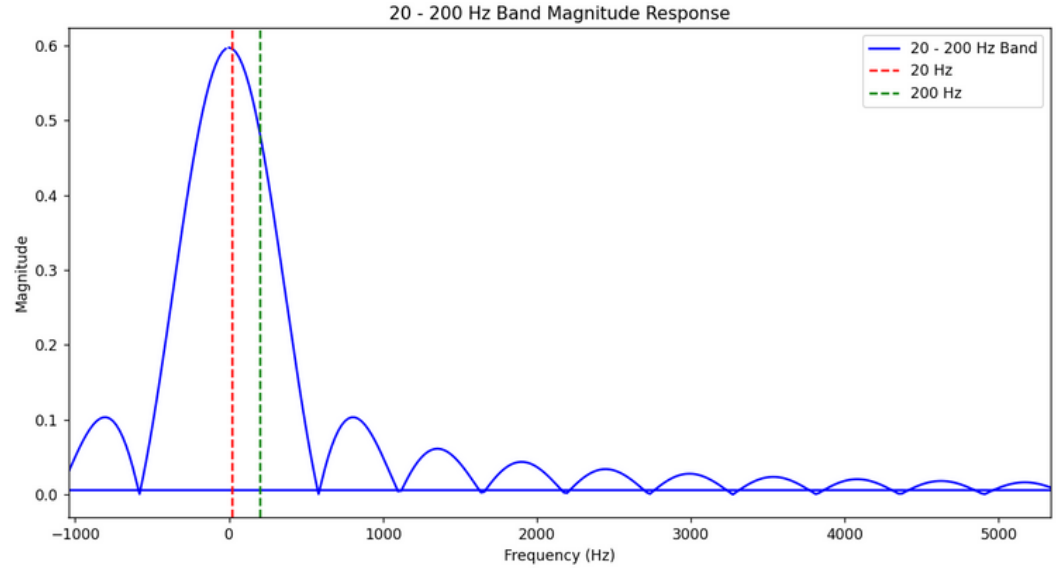
- The equalized signal is written to a new WAV file (equalized_music.wav).

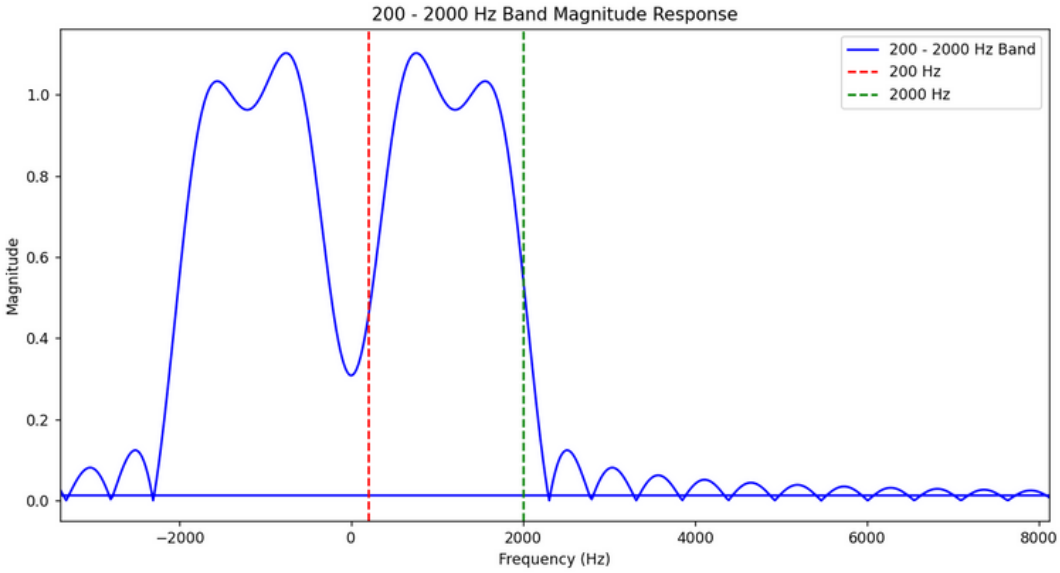
Plotting Frequency Responses:

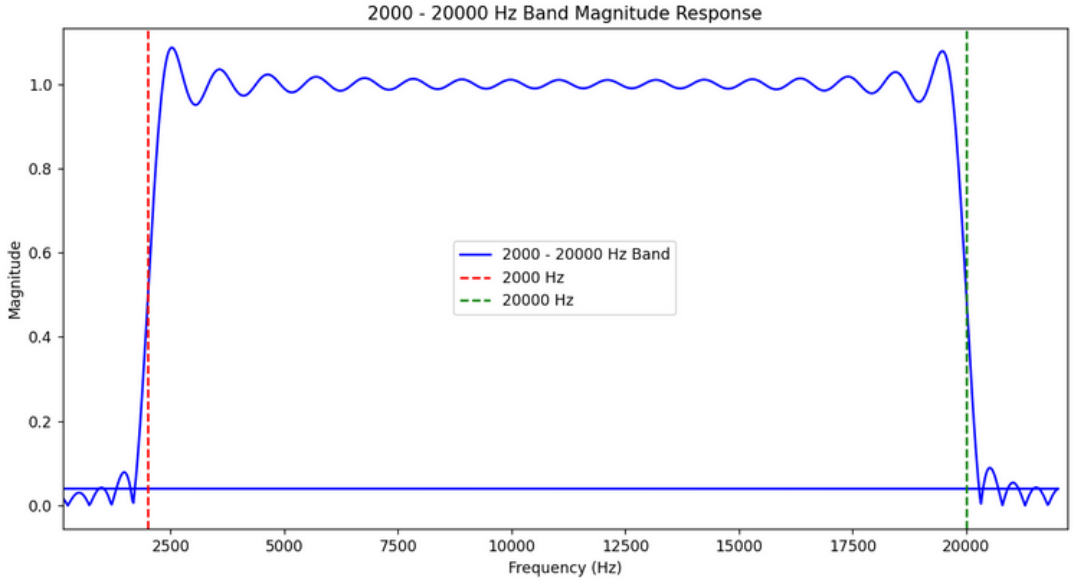
- The code uses matplotlib and scipy to plot the frequency responses of the input signal, the individual equalizer bands, and the filtered output.
- Plot 1: Input Signal Spectrum
  - Displays the magnitude spectrum of the original input signal.
- Plot 2: Equalizer Bands Magnitude Response
  - Shows the magnitude response of each band of the equalizer.
- Plot 3: Filtered Signal Spectrum
  - Displays the magnitude spectrum of the final filtered signal.
- The three plots are arranged vertically for easy comparison.

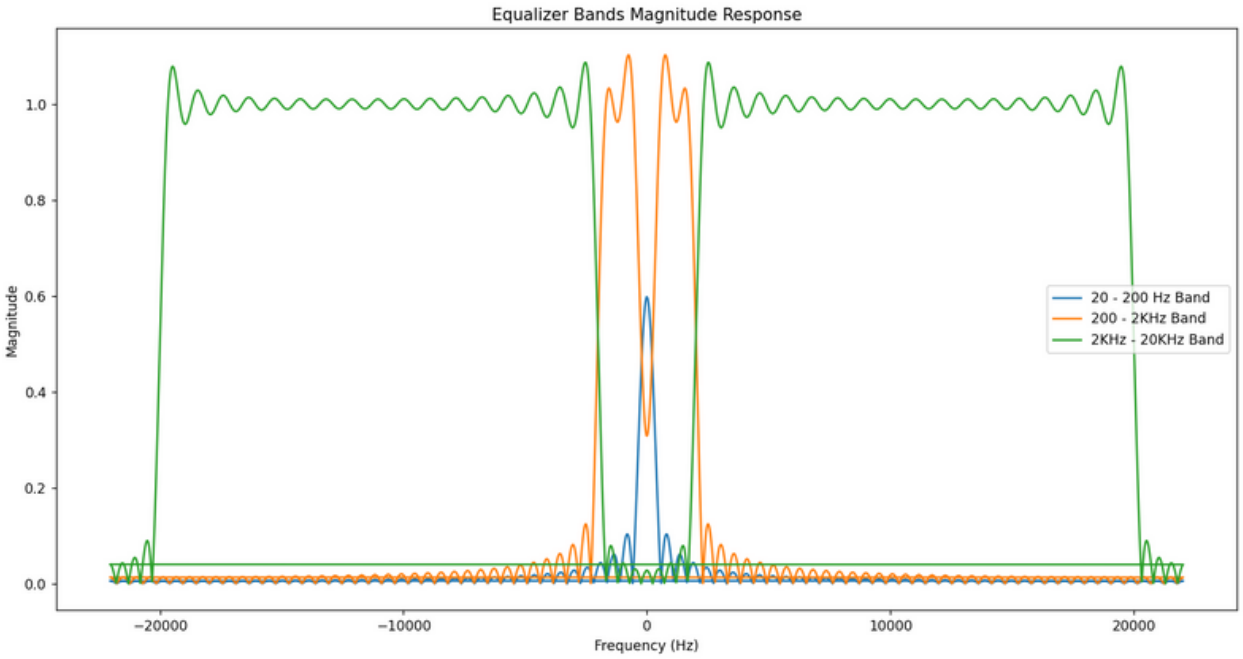
Output Response

Magnitude Response of Equalizer

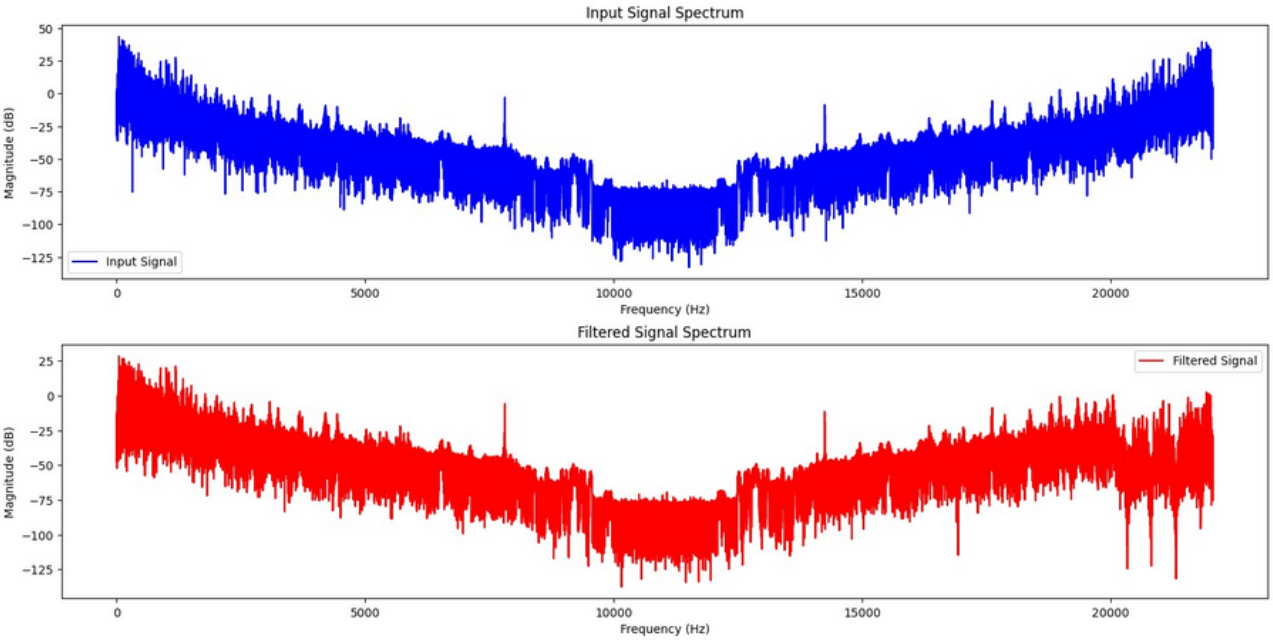








Signal Spectrum



Gain1 = 0.1 , Gain2 = 0.5 , Gain3 = 0.7

Part 3

Apply the designed equalizer to the loaded audio signal. Plot the spectrogram of the equalized audio. Adjust the gains at different bands and observe the effects. (to avoid clipping, please divide the gain of the original audio signal by 3 before passing it through the equalizer).

Solution:

Link to Code : Google Colab

Approach:

1.Loading Audio Files:

- The code uses `librosa.load` to load two audio files, "music.wav" and "equalized_music.wav," storing the audio signal (`y1` and `y2`) and their respective sampling rates (`sr1` and `sr2`).

2.Calculate Spectrograms:

- Spectrograms are calculated using the Short-Time Fourier Transform (STFT) through `librosa.stft`. The magnitudes of the complex STFT are converted to decibels using `librosa.amplitude_to_db`.

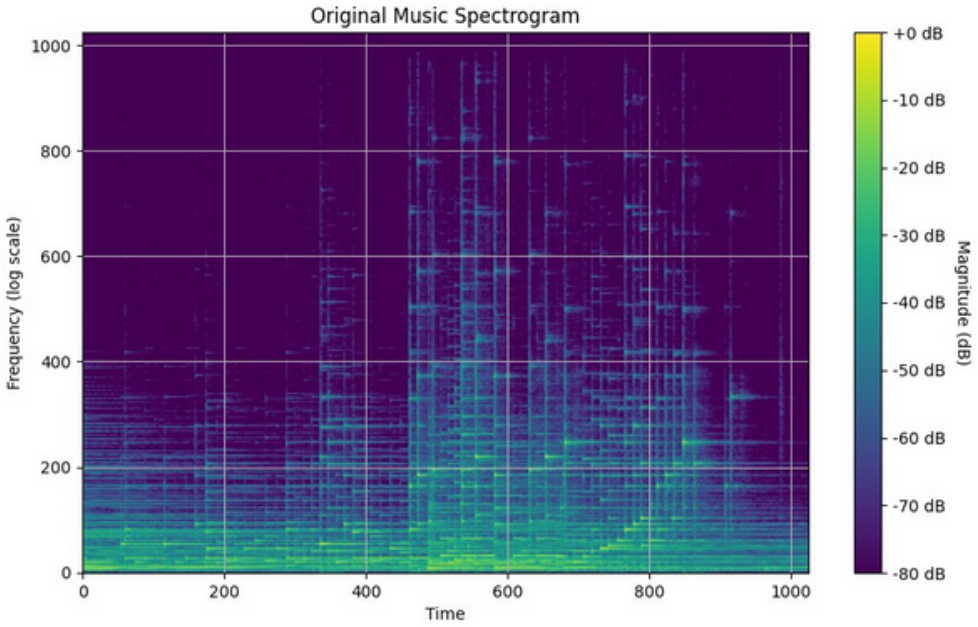
3.Plotting Spectrograms:

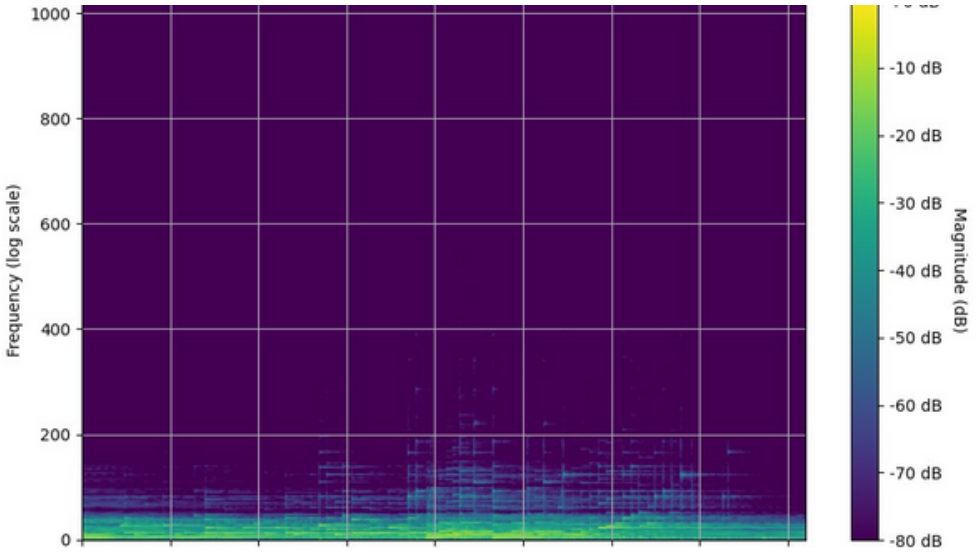
- The code creates a 2x1 subplot figure (`fig`) to display both spectrograms.
- For the first spectrogram (original music), it plots the image using `imshow` and sets titles, labels, grid, and colorbar properties.
- The same process is repeated for the second spectrogram (equalized music) in the second subplot.
- The `format='%+2.0f dB'` in the `colorbar` adds dB labels to the colorbar.

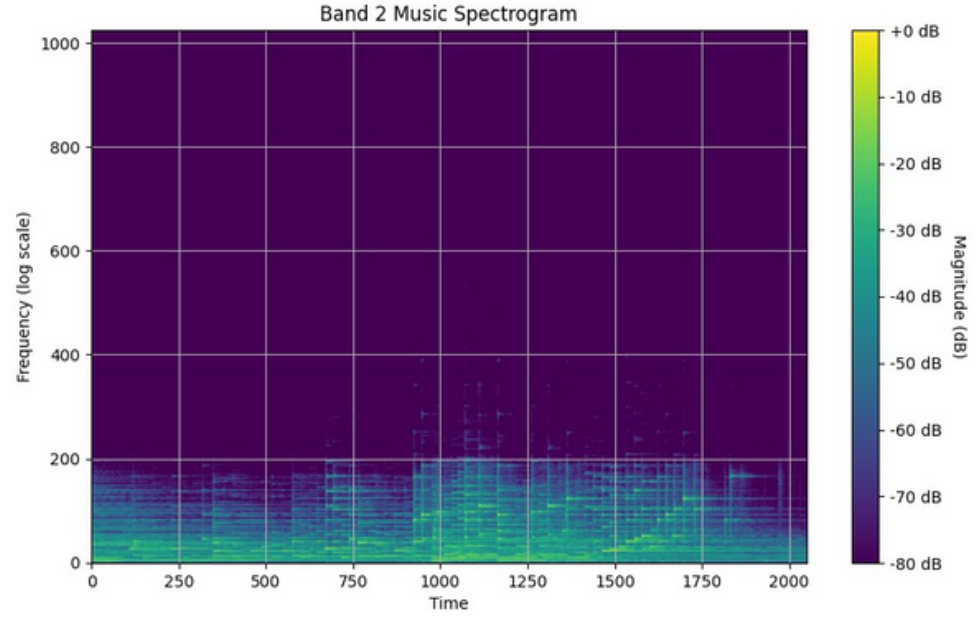
4.Display the Plot:

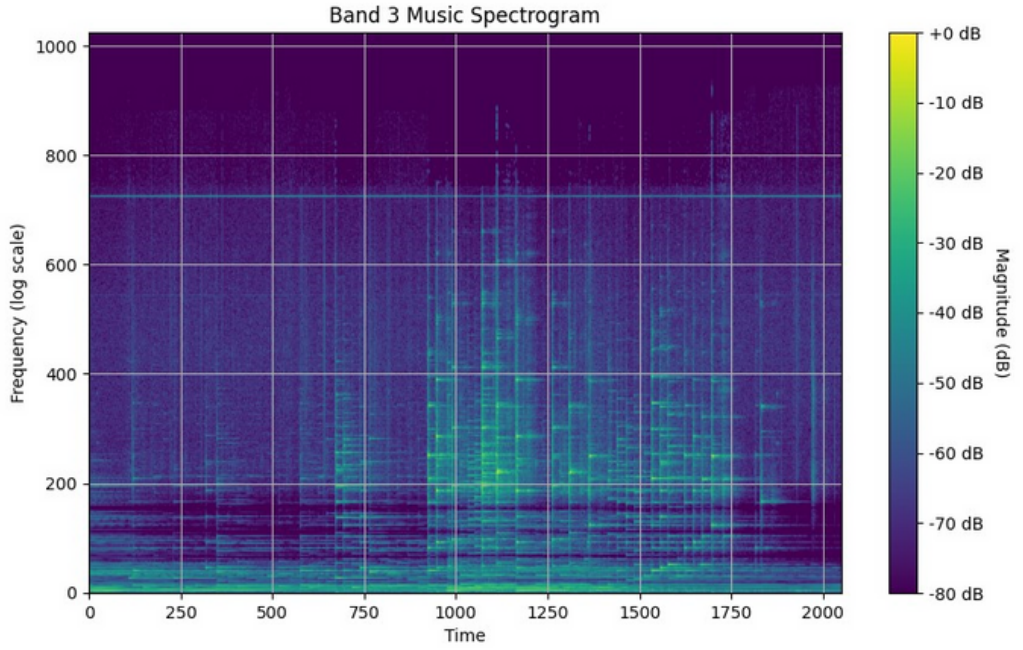
- Finally, the code uses `plt.tight_layout()` to ensure proper spacing between subplots and `plt.show()` to display the figure.

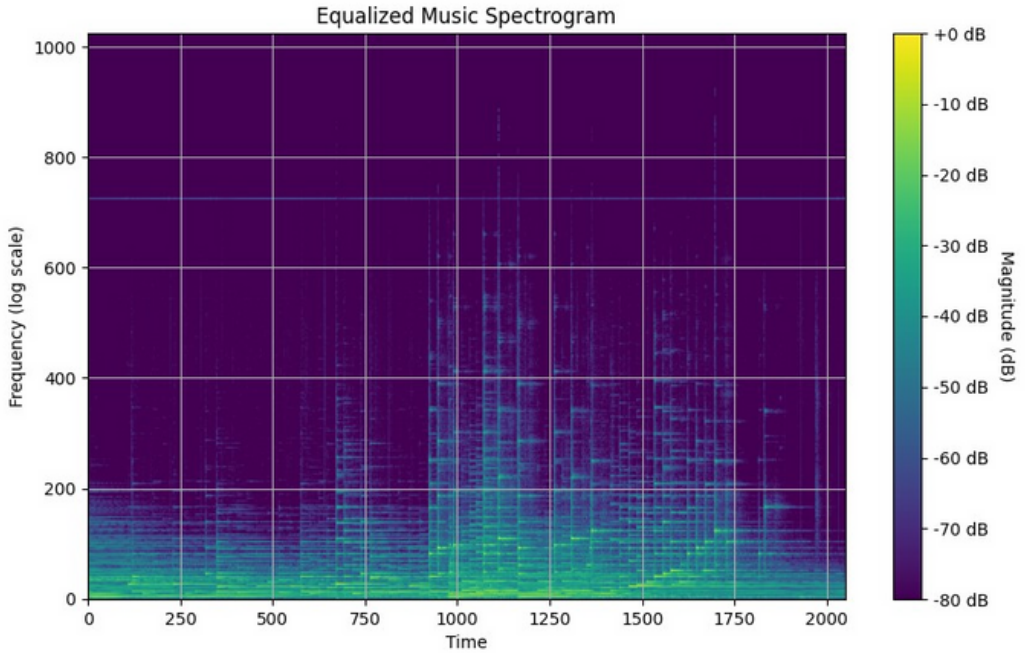
Output Response



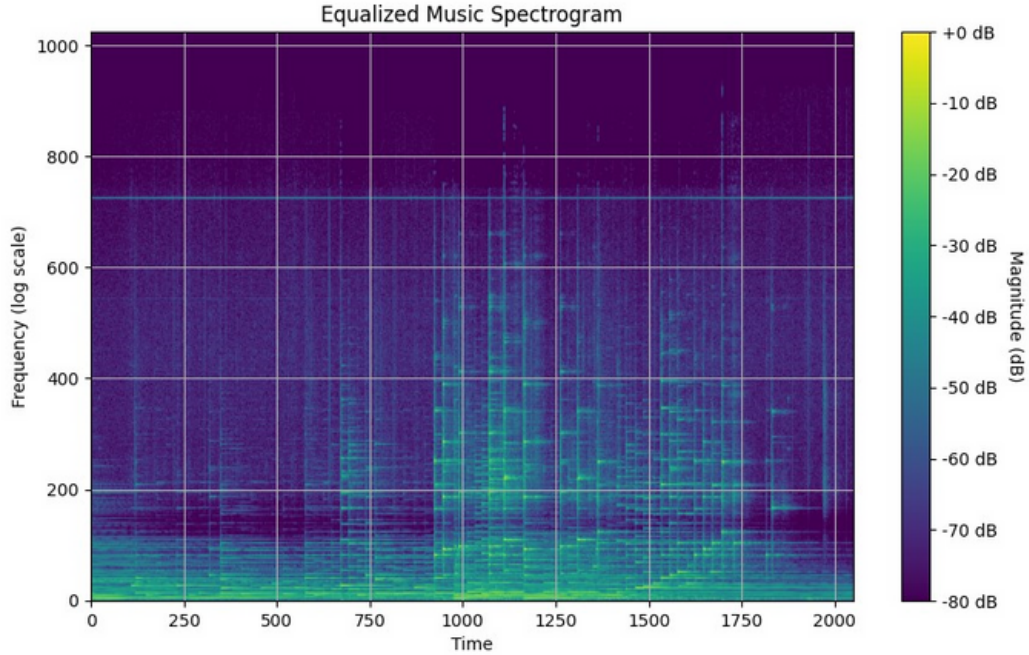




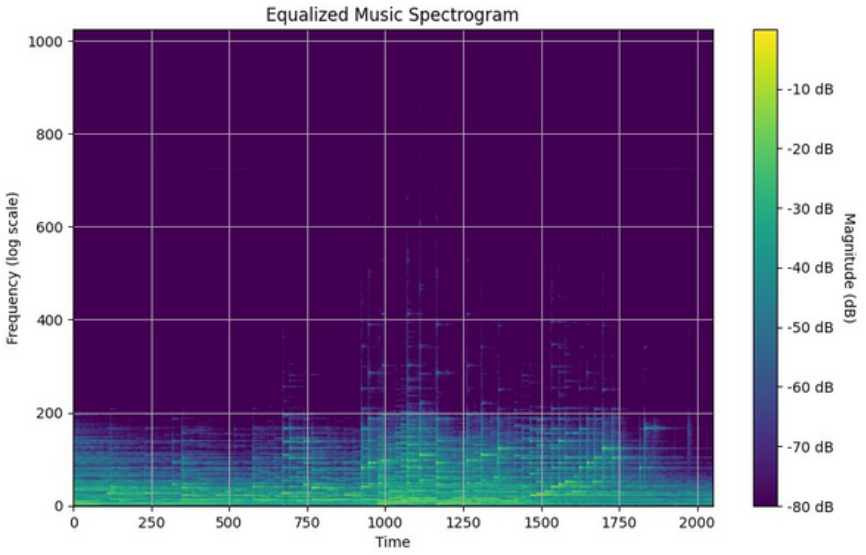




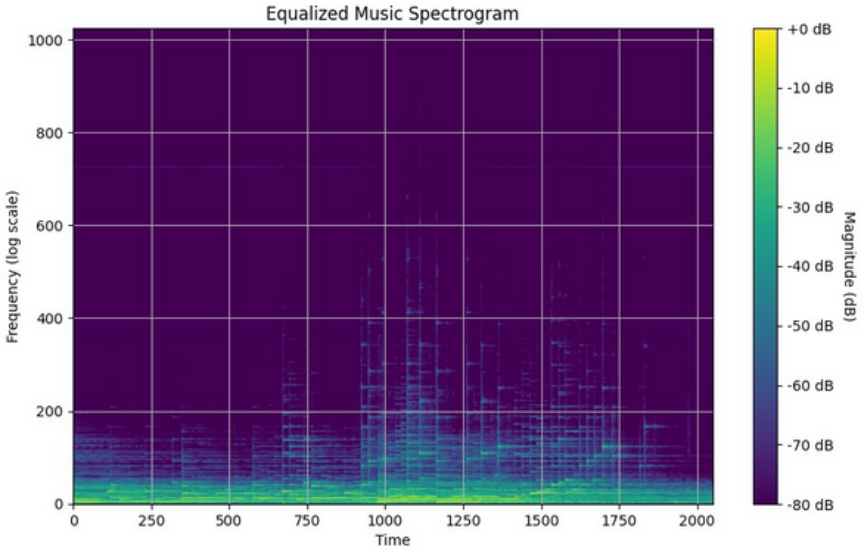
Gain1 = 0.1 , Gain2 = 0.5 , Gain3 = 0.7



Gain1 = 0.1 , Gain2 = 0.1 , Gain3 = 0.99



Gain1 = 0.1 , Gain2 = 0.99 , Gain3 = 0.1



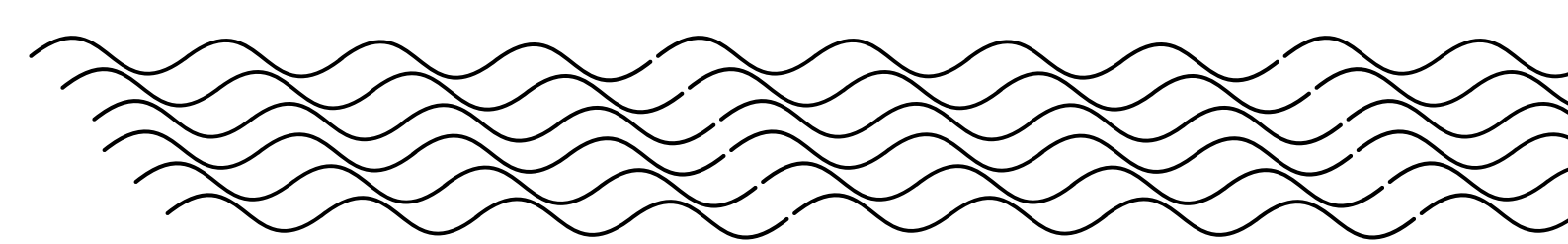
Gain1 = 0.99 , Gain2 = 0.1 , Gain3 = 0.1

Conclusion

We showcases a practical application of signal processing techniques to enhance audio signals. It employs Finite Impulse Response (FIR) filters, designed with Kaiser window functions, to shape the frequency response of the audio. The filters, categorized as lowpass, highpass, and bandpass, enable control over specific frequency ranges. The design parameters, such as cutoff frequencies and window shape, are adjustable for customization.

The second part focuses on audio equalization using a multi-band approach. By applying highpass FIR filters with varying cutoff frequencies and gains, the code achieves balanced amplification across different frequency bands. Careful gain adjustment ensures that the equalized signal avoids distortion.

Librosa's spectrogram visualization helps assess the impact of these processes on the audio's frequency content over time. In summary by comparing the spectrograms of the original and equalized signals, engineers gain insights into how the applied filters influence the distribution of energy across different frequency components over time.



Thank You