**EC3093D**

**Digital Signal Processing Lab**

**Mini Project Report**

**Real Time Audio Equalizer**

**Group Members:**

|  |  |
| --- | --- |
| NAME | ROLL NUMBER |
| Aditya Pulyapudi | B201015EC |
| Pranith Kumar Boge | B201032EC |
| Mohinder Chawan | B201036EC |
| Rishi Dharmeshkumar Shah | B200929EC |

**AIM:**

To design a Real Time Audio Equalizer that provides high-quality sound adjustment with low latency and user-friendly controls, aimed at enhancing the overall listening experience for music enthusiasts and professionals alike.

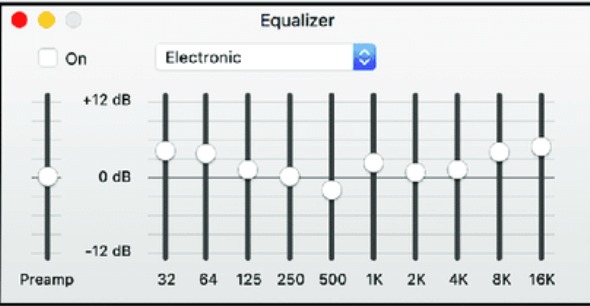


**THEORY:**

An audio equalizer is a device or software that allows the user to adjust the frequency response of an audio signal. The equalizer works by **selectively boosting or cutting specific frequency ranges of the audio spectrum to shape the overall sound to the user's liking.**

In the case of a Real-Time Audio Equalizer implemented in code, the audio signal is processed in the digital domain using digital signal processing (DSP) techniques.

* The input audio signal is first sampled at a specific sampling rate, and then divided into blocks of samples for processing. In the code provided, a block size of 1024 samples is used for processing.
* Next, the frequency spectrum of the audio signal is computed using the Fast Fourier Transform (FFT) algorithm, which converts the time-domain signal into the frequency-domain representation. The frequency spectrum is then divided into frequency bands, each with a specific range of frequencies. The number of bands and their frequency ranges are specified by the user in the code.
* The user can also specify the gain (in dB) to be applied to each frequency band to boost or cut its amplitude. The gains are typically set using a graphic equalizer interface that displays the frequency spectrum with sliders that can be adjusted to boost or cut specific frequency ranges.
* In the code provided, the gains are defined as an array of dB values for each frequency band. The frequency spectrum within each band is then multiplied by a scaling factor derived from the corresponding gain value. The scaling factor is computed as a power of 10^(gain/20), where gain is the gain value in dB.
* After applying the gains to the frequency spectrum, an inverse FFT is used to convert the modified frequency spectrum back to the time-domain representation. The resulting time-domain signal is then played back through the speakers in real-time.



**DESIGN:**

1. Sampling Rate and Block Size:

The first step is to define the sampling rate and block size. The sampling rate determines the number of samples taken per second, and the block size determines the number of samples processed at once.

1. Audio Input and Output:

Next, System objects are created to read audio input from the microphone and play audio output through the speakers. The input and output are connected to the computer's default audio input and output devices.

1. Frequency Bands:

The frequency bands for the equalizer are defined as non-overlapping intervals between the specified frequency values. The number of frequency bands is determined by the number of intervals, which is equal to the length of the freqBands vector minus one.

1. Equalization Gains:

The equalization gains are specified in decibels (dB) and initialized to default values. The gains determine the amount of boost or cut applied to each frequency band.

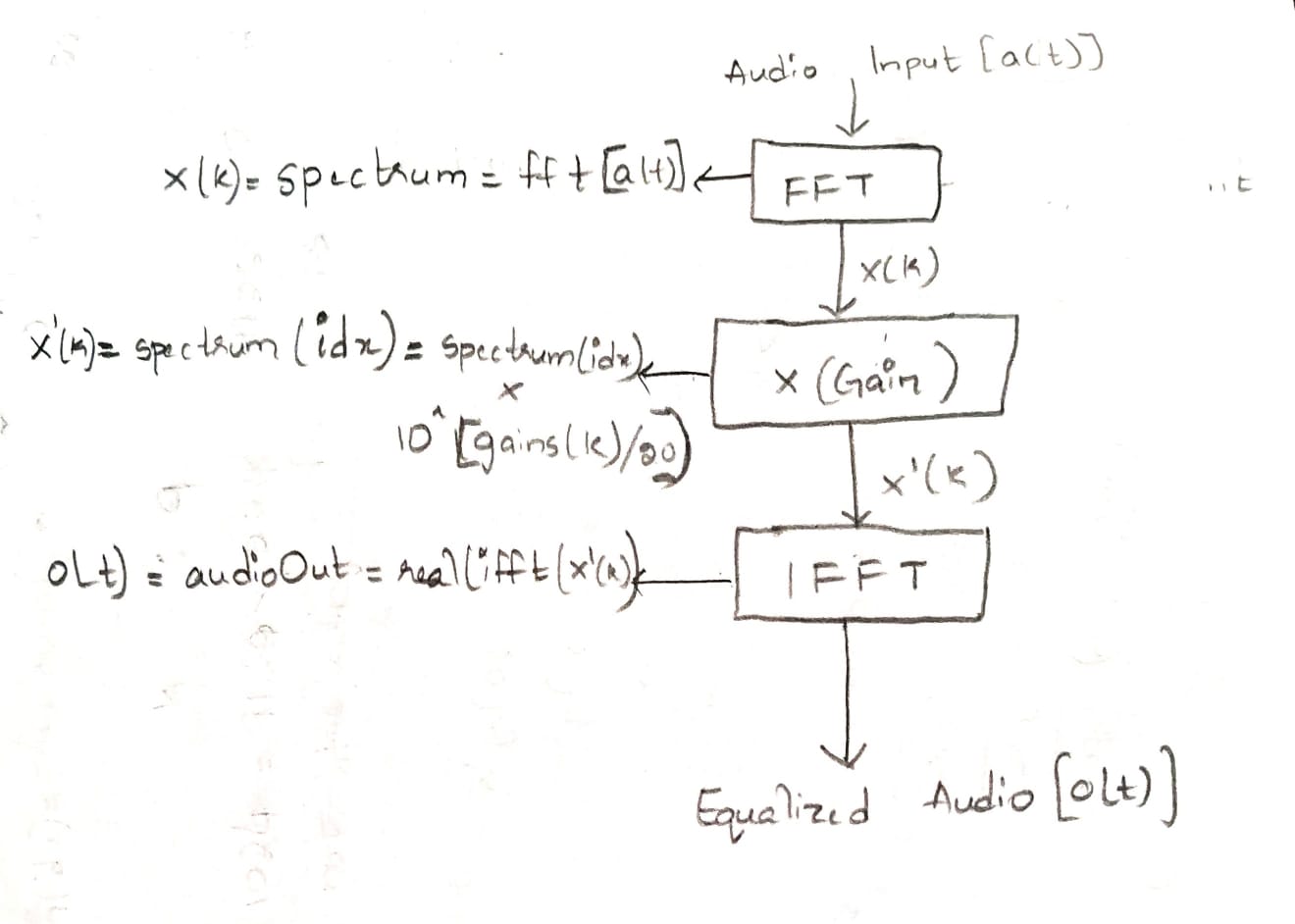
1. Real-Time Processing Loop:

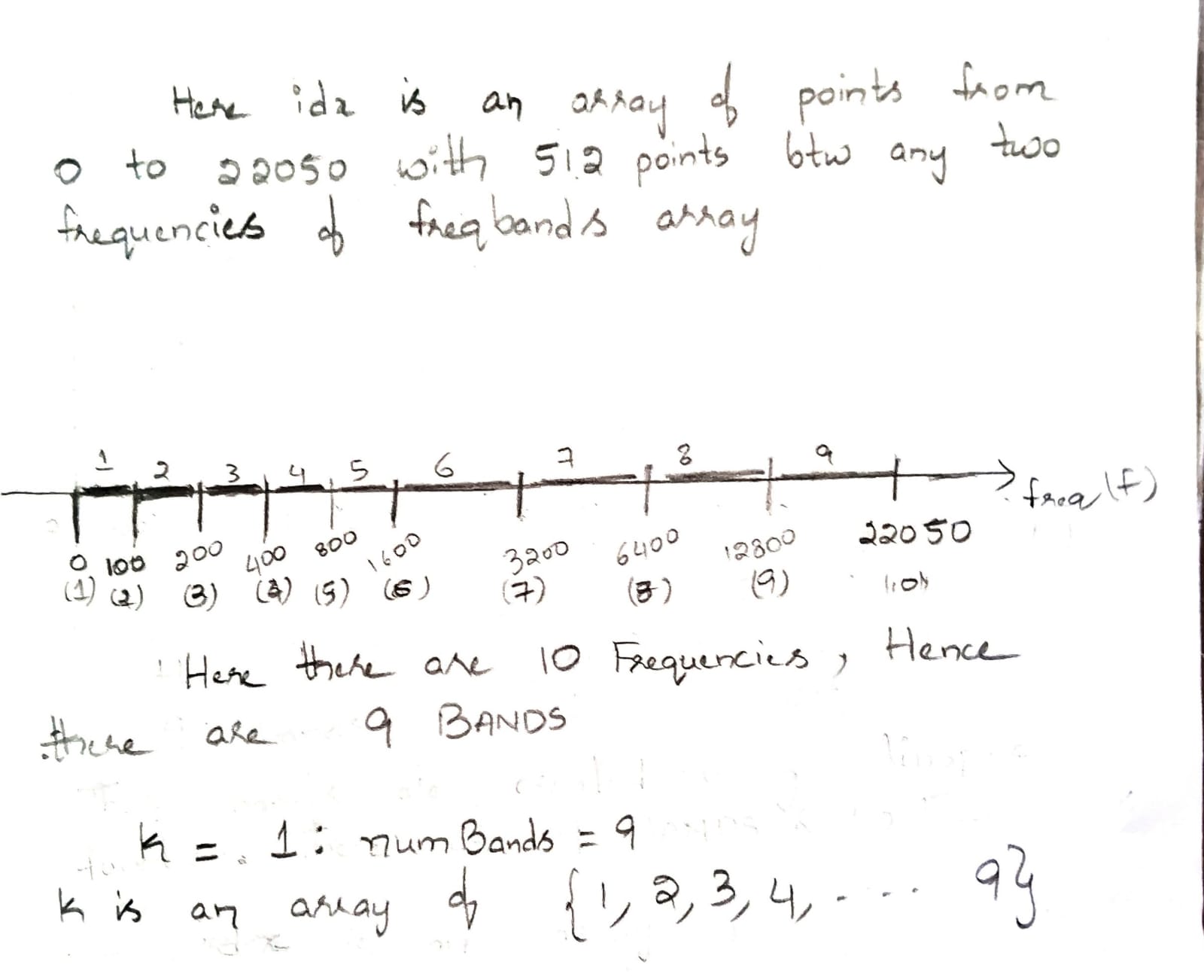
The main loop continuously reads a block of audio input from the microphone, computes the frequency spectrum of the audio signal, applies the equalization gains to the frequency spectrum, and then applies an inverse FFT to obtain the time-domain signal. The equalized audio signal is then written to the speakers.

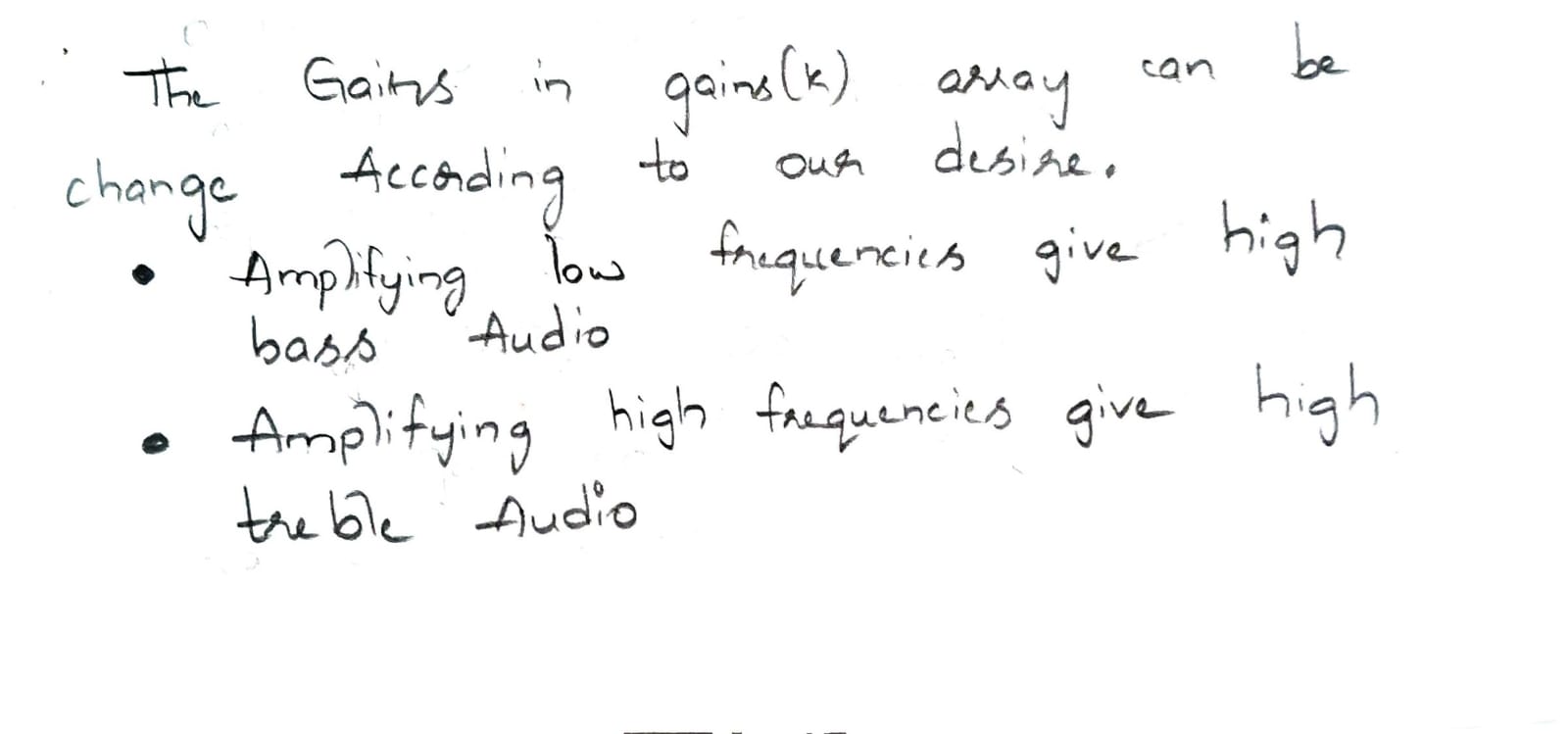
1. Cleanup:

The System objects for audio input and output are released to clean up the resources used.

**BLOCK DIAGRAM:**







**CODE:**

clc

clear all

close all

% Real-time Audio Equalizer using Digital Signal Processing

% This code assumes that you have a microphone and speakers connected to your computer.

% Define the sampling rate and block size

fs = 44100; % Sampling rate (Hz)

blockSize = 1024; % Block size for processing (samples)

% Create a System object to read audio input from the microphone

audioInput = audioDeviceReader('SamplesPerFrame', blockSize, 'NumChannels', 1, 'SampleRate', fs);

% Create a System object to play audio output through the speakers

audioOutput = audioDeviceWriter('SampleRate', fs, 'SupportVariableSizeInput', true);

% Define the frequency bands for the equalizer

freqBands = [0 100 200 400 800 1600 3200 6400 12800 22050]; % Frequency bands (Hz)

numBands = length(freqBands)-1; % Number of frequency bands

% Initialize the equalization gains in decibels (dB)

% gains = [0 0 0 0 0 0 0 0 0];

% gains = [-20 -20 -20 -20 -20 0 3 6 9];

% gains = [10 6 3 3 -3 -6 -9 -10 -20];

%gains = [-20 -10 -6 -3 0 3 6 9 12];

% gains = [200 200 200 200 200 200 200 200 200];

% gains = [-200 -200 -200 -200 -200 -200 -200 -200 -200];

gains = [-10 -20 -20 0 50 50 50 50 50];

% Create a System object to write audio output to a file

filename = 'RealTimeEqu\_File\_Output.wav';

audioFileWriter = dsp.AudioFileWriter(filename, 'FileFormat', 'WAV', 'SampleRate', fs );

% Main loop for real-time processing

while true

% Read a block of audio input from the microphone

audioIn = audioInput();

% Compute the frequency spectrum of the audio signal

spectrum = fft(audioIn);

% Apply the equalization gains to the frequency spectrum

for k = 1:numBands

idx = (freqBands(k) < linspace(0, fs/2, blockSize/2)) & (linspace(0, fs/2, blockSize/2) < freqBands(k+1));

spectrum(idx) = spectrum(idx) \* 10^(gains(k)/20);

end

% Apply an inverse FFT to obtain the time-domain signal

audioOut = (ifft(spectrum));

% Write the equalized audio signal to the speakers

audioOutput(audioOut);

% Write the equalized audio signal to a file

audioFileWriter(audioOut);

end

% Cleanup when the loop is exited

release(audioInput);

release(audioOutput);

**Code Explanation:**

**“freqBands”** is an array that specifies the frequency bands in Hz. Each element in the array represents the upper limit of a frequency band. The first element is 0, which represents DC or the zero-frequency component. The last element is the Nyquist frequency, which is half of the sampling rate (22050 Hz in this case) and represents the highest frequency that can be represented in the signal.

**“numBands”** is a variable that represents the number of frequency bands, which is determined by the length of “**freqBands”** minus one.

**“gains”** is an array that specifies the gain for each frequency band in decibels (dB). In this case, there are 9 frequency bands, so there are 9 gain values. A negative gain reduces the level of the corresponding frequency band, while a positive gain increases it. The values in gains are arbitrary and can be adjusted according to the desired equalization curve.

The frequency spectrum of the audio signal is obtained using the fast Fourier transform (FFT), and then the gains are applied to specific frequency bands. The loop iterates over each frequency band defined by freqBands, and for each band, it computes the index of the frequency components within that band using the “**linspace”** function. This index is used to select the corresponding components of the frequency spectrum. The gains in dB for that frequency band are converted to linear scale and used to multiply the selected components of the frequency spectrum.

In other words, this block of code amplifies or attenuates specific frequency bands in the audio signal according to the values in “**gains”**. Then the resulting frequency spectrum is converted back to a time domain signal using IFFT (Inverse Fast Fourier Transform) function.

**RESULT:**

Because Real Time signals are difficult to plot. We are taking an audio file (.wav) and applying equalization on it.

**Audio Equalizer on a Sample file (.wav):**

clc

clear all

close all

% Define the sampling rate and block size

fs = 44100; % Sampling rate (Hz)

blockSize = 1024; % Block size for processing (samples)

% Load the input audio file

[input, fs\_input] = audioread('C:\Users\91944\OneDrive\Documents\MATLAB\DSP Mini\Equ\_File\_Input.wav');

if size(input, 2) > 1

input = mean(input, 2); % convert stereo to mono

end

% Create a System object to play audio output through the speakers

audioOutput = audioDeviceWriter('SampleRate', fs, 'SupportVariableSizeInput', true);

% Define the frequency bands for the equalizer

freqBands = [0 100 200 400 800 1600 3200 6400 12800 22050]; % Frequency bands (Hz)

numBands = length(freqBands)-1; % Number of frequency bands

% Initialize the equalization gains in decibels (dB)

% gains = [0 0 0 0 0 0 0 0 0];

% gains = [-200 -200 -200 -200 -200 -200 -200 -200 -200];

gains = [200 200 200 200 200 200 200 200 200];

% gains = [9 6 3 0 -3 -6 -9 -12 -20];

% Create a System object to write audio output to a file

filename = 'Equ\_File\_Output.wav';

audioFileWriter = dsp.AudioFileWriter(filename, 'FileFormat', 'WAV', 'SampleRate', fs );

% Compute the frequency spectrum of the input signal

spectrum\_input = fft(input);

spectrum\_input2 = fft(input);

% Apply the equalization gains to the frequency spectrum

for k = 1:numBands

idx = (freqBands(k) < linspace(0, fs/2, blockSize/2)) & (linspace(0, fs/2, blockSize/2) < freqBands(k+1));

spectrum\_input(idx) = spectrum\_input(idx) \* 10^(gains(k)/20);

end

% Apply an inverse FFT to obtain the time-domain signal

output = (ifft(spectrum\_input));

% Write the equalized audio signal to a file

audioFileWriter(output);

% Compute the frequency spectrum of the output signal

spectrum\_output = fft(output);

% Plot the frequency spectra of the input and output signals

figure;

subplot(2,1,1);

plot(linspace(0,fs/2,blockSize/2),20\*log10(abs(spectrum\_input2(1:blockSize/2))));

title('Input Signal Frequency Spectrum');

xlabel('Frequency (Hz)');

ylabel('Magnitude (dB)');

xlim([0 23000])

subplot(2,1,2);

plot(linspace(0,fs/2,blockSize/2),20\*log10(abs(spectrum\_output(1:blockSize/2))));

title('Output Signal Frequency Spectrum');

xlabel('Frequency (Hz)');

ylabel('Magnitude (dB)');

xlim([0 23000])

% Cleanup

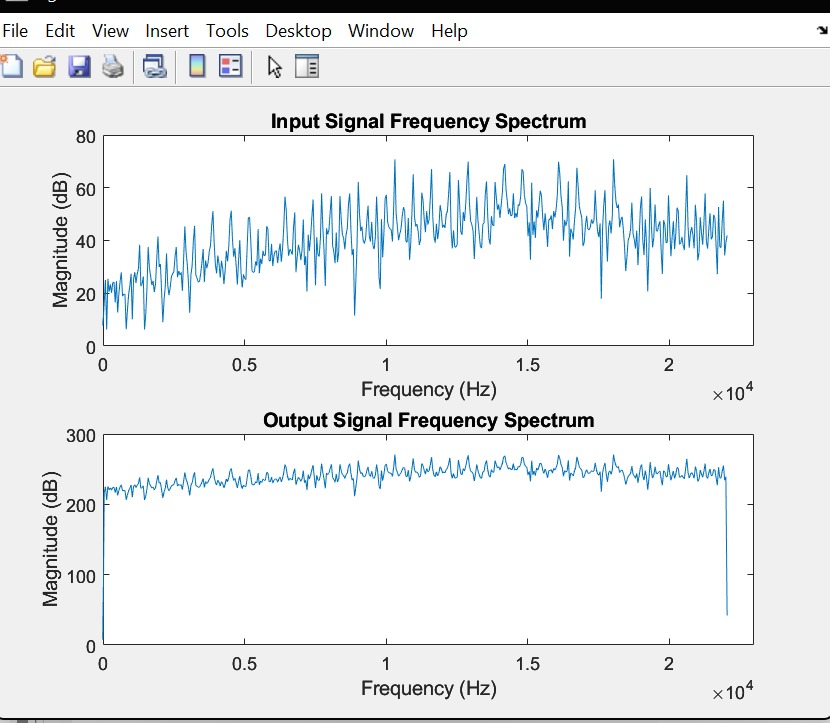
release(audioOutput);

release(audioFileWriter);

Output Magnitude Respones for different Gain Values

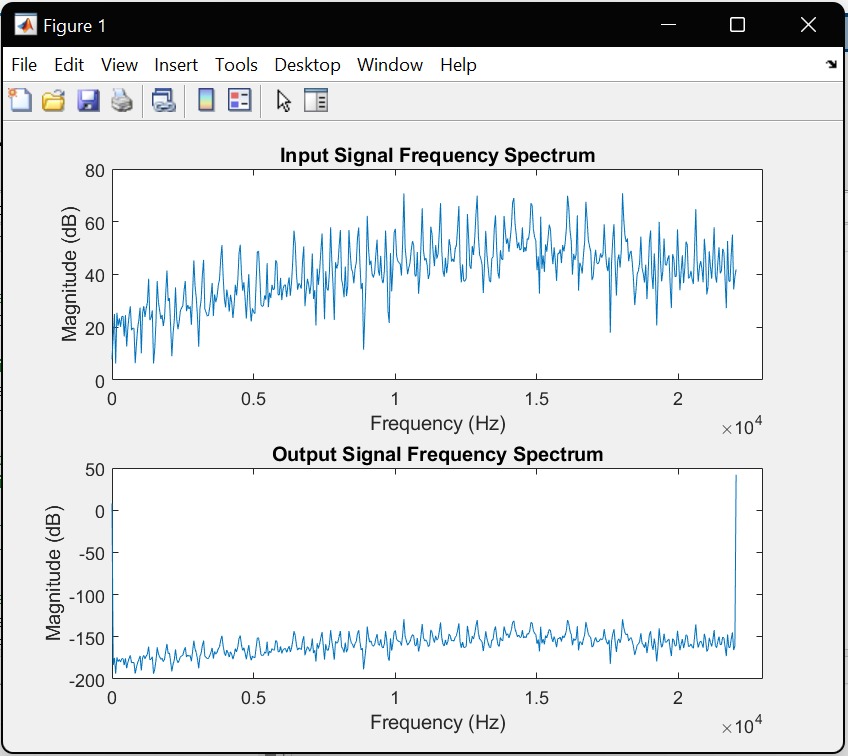
for gains = [200 200 200 200 200 200 200 200 200];

**All Frequencies Amplified**



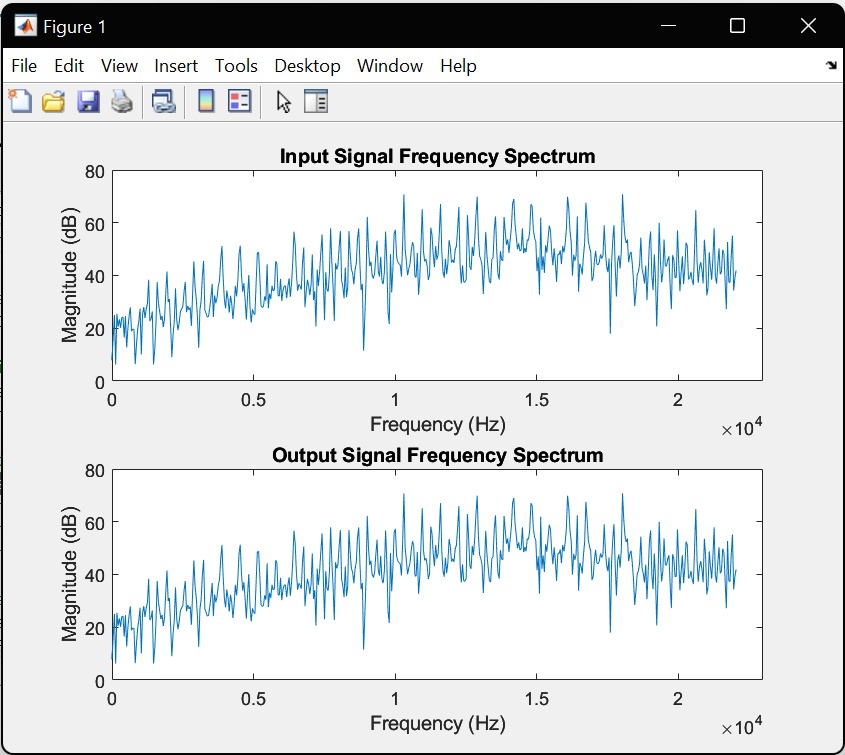
for gains = [-200 -200 -200 -200 -200 -200 -200 -200 -200];

**All Frequencies Supressed**



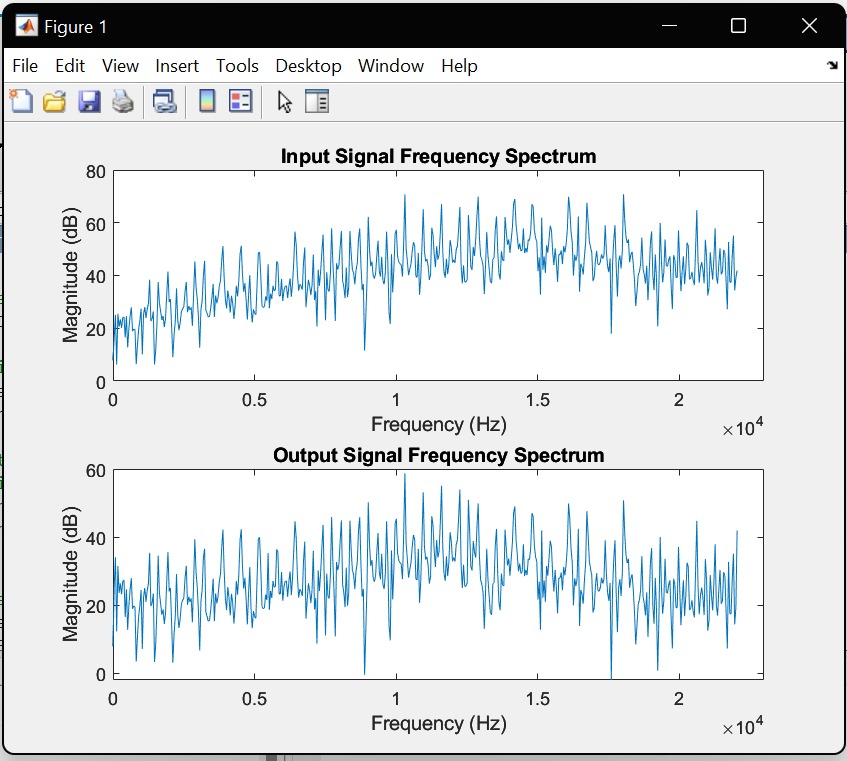
for gains = [0 0 0 0 0 0 0 0 0];

**All Frequencies are INTACT (Unchanged)**



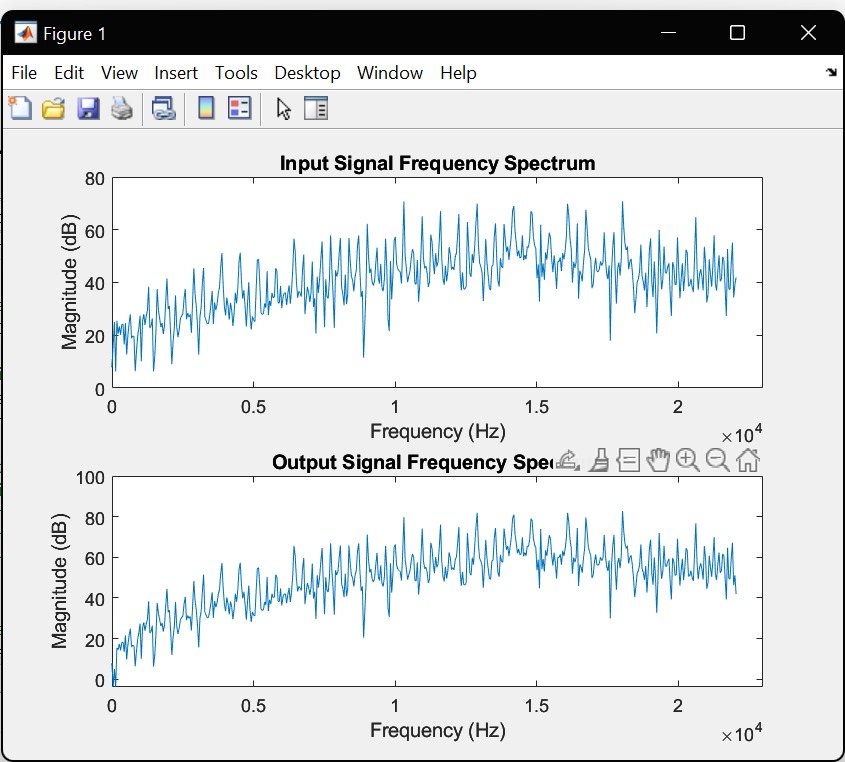
for gains = [9 6 3 0 -3 -6 -9 -12 -20]:

**Lower Frequencies Amplified Upper Frequencies Suppressed**



for gains = [-20 -10 -6 -3 0 3 6 9 12 ];

**Lower Frequencies Suppressed Upper Frequencies Amplified**



**CONCLUSION :**

The project makes use of several MATLAB functions and Signal Processing Toolbox objects to process the audio signal. We have used the “**audioDeviceReader”** and “**audioDeviceWriter”** objects to read and write audio data to and from the computer's audio input and output devices.

We have also used the “**fft”** function to compute the frequency spectrum of the audio signal and applied the gain levels to the frequency spectrum using a For Loops. Finally, we applied an inverse FFT to obtain the time-domain signal and played the equalized audio signal through the speakers using the “**audioOutput”** object.

This project can be further extended by adding more advanced features such as additional frequency bands, dynamic range compression, or user-defined filters. It also serves as a good introduction to real-time audio processing using MATLAB and the Signal Processing Toolbox.