# EE 321: Digital Signal Processing

# Python Assignment: Set - P01

## Group Members:

**Avinash Sharma - 210102111**

**Swapn Agarwal – 210102130**

**Ambesh Dixit – 210102122**

### Task 01: -

Load the noisy audio signal from the file “noisy audio.wav” into a numpy array. Design a low-pass FIR filter with a cutoff frequency of 2 kHz to remove high-frequency noise from the signal. Choose an appropriate filter order of your choice. Plot the magnitude and phase response of the designed filter. Apply the designed filter to the noisy audio signal. Plot the original noisy signal and the filtered signal in the same graph to compare their frequency content. (Do not use any DSP Python libraries for filter design. Build your own filter function.)

### Characteristics of the Chosen Low-Pass FIR Filter:

1. **Filter Order and Cutoff Frequency:**

- **Filter Order:** 511
    - The filter order determines the number of taps in the FIR filter. A higher order allows for sharper roll-off but increases computational complexity.
- **Cutoff Frequency:** 2 kHz
    - The cutoff frequency defines the point at which the filter starts attenuating high-frequency components.

2. **Filter Design:**

- **Hamming Window:**
    - A Hamming window is applied to the ideal low-pass filter kernel.
    - The Hamming window provides a good compromise between narrowing the main lobe for improved frequency resolution and suppressing side lobes to reduce spectral leakage.
    - The mathematical formulation of the Hamming window is straightforward, making it easy to implement in various digital signal processing applications
- **Normalization:**
    - The filter is normalized to have unity gain at DC (0 Hz), preventing overall magnitude changes in the signal.

## Filter Equations and Steps:

1. **Ideal Low-Pass Filter Equation:**

- The ideal low-pass filter function is given by the sinc function:

$$hd[n] = (fs/fc) \cdot sinc(2fc(n - \frac{(N-1)}{2})/fs)$$

where:

- *fc* is the cutoff frequency,
- *fs* is the sampling frequency,
- *N* is the filter order

2. **Hamming Window Equation:**

- The Hamming window function is given by:

$$w(n) = \alpha - \beta cos(2\pi n/(N-1))$$

where:

- $\alpha$=0.54 and $\beta$=0.46 are coefficients chosen for the Hamming window,
- *N* is the number of taps (filter order),
- *n* is the index vector.

3. **Filtering Process:**

- The ideal filter kernel is multiplied by the Hamming window to form the final filter kernel.

$$h(n) = hd[n] \cdot w(n)$$

- The filter is then normalized to have unity gain at DC (0 Hz).

4. **Filtering the Signal:**

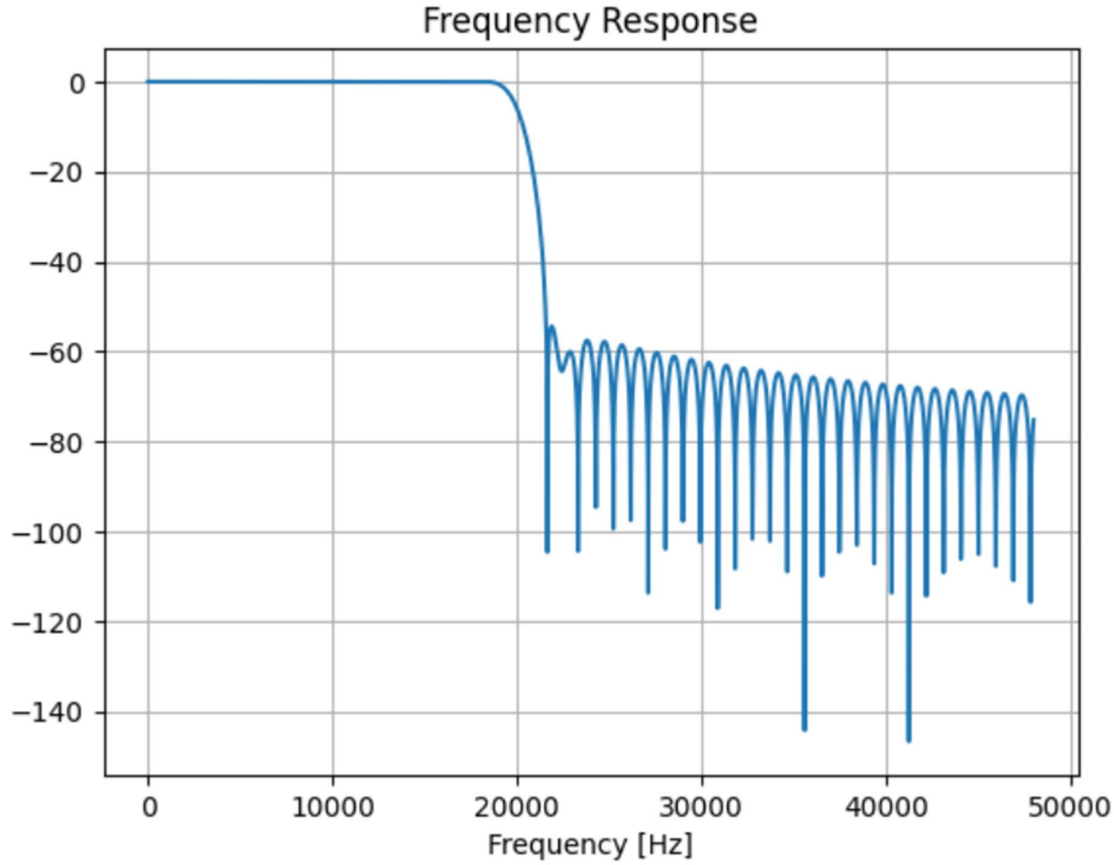
- The filter is applied to the input signal using convolution:

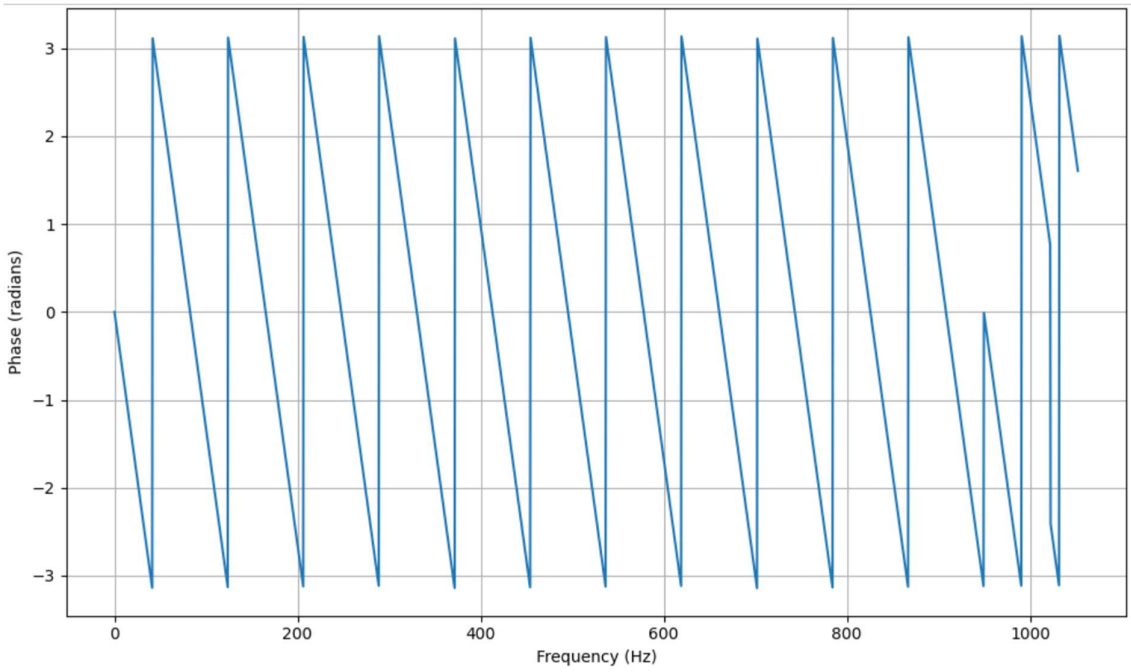
$$y(n) = x(n) * h(n)$$

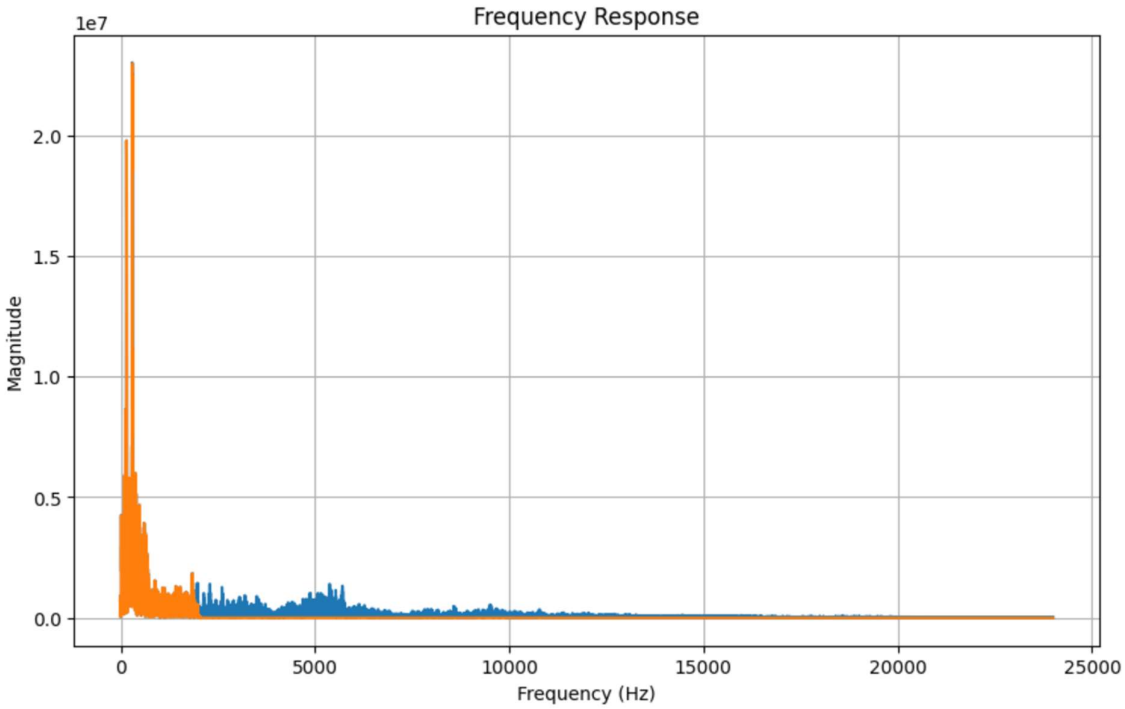
- The convolution is performed in both the forward and backward directions to ensure a linear phase response.

5. **Frequency Response Plots:**

- The frequency response of the filter is visualized by plotting:
  - Magnitude response in dB.
  - Phase response in radians.
  - Magnitude response of the original and filtered signals.


Frequency Response
0
−20
−40
−60
−80
−100
−120
−140
0
10000
20000
30000
40000
50000
Frequency [Hz]


Phase (radians)
3
2
1
0
−1
−2
−3
0
200
400
600
800
1000
Frequency (Hz)




## TASK 2:

Compute and plot the spectrogram of the filtered signal obtained in Part 1. Identify and mark the frequency bands containing the remaining noise in the spectrogram. Suggest a suitable filter type (e.g., bandstop or notch filter) to remove the remaining noise components based on your analysis.

The objective of this subtask is to analyze the spectrograms of the original signal and the signal after applying a low-pass FIR filter.

- The original signal was subjected to spectrogram analysis. The resulting spectrogram illustrates the time-frequency representation of the original signal.
- An FIR low-pass filter was designed with 2000 KHz cutoff frequency. The filtered signal was then utilized to compute a spectrogram to observe any frequency changes induced by the filtering process.
- Two subplots were created in a single figure for side-by-side comparison. The first subplot displays the spectrogram of the original signal, providing insight into its frequency content over time. The second subplot illustrates the spectrogram of the filtered signal, highlighting the impact of the low-pass filter on the signal's frequency characteristics.

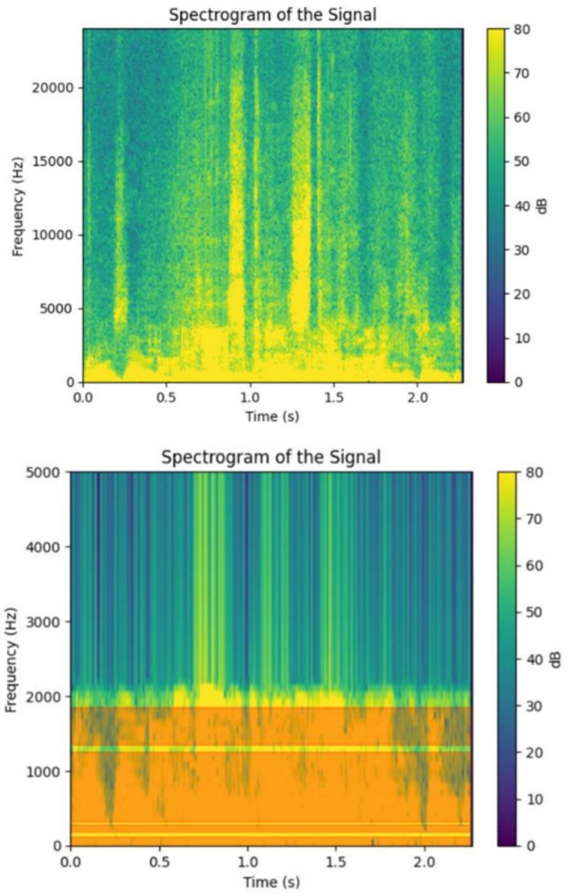


FIG: Required Spectrograms

## Iterative Approach to choose the appropriate filter:

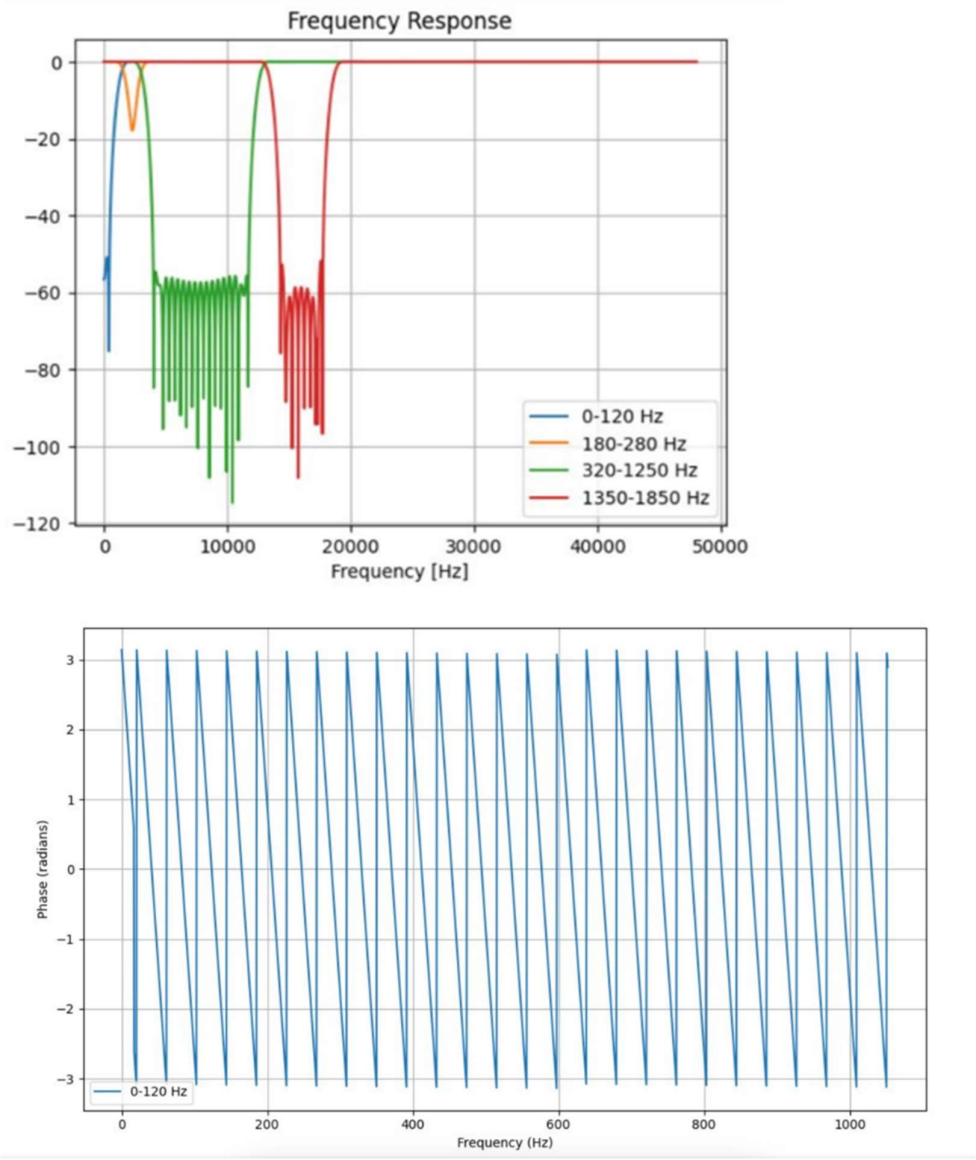
- We iteratively applied the band reject filter with different cutoff frequencies and observe the spectrogram after each application.
- Fine-tune the cutoff frequencies to effectively eliminate the remaining noise components.
- Consider analyzing the resulting signals after applying each band-reject filter to ensure that the filters effectively attenuate the undesired frequency components while preserving the essential signal.

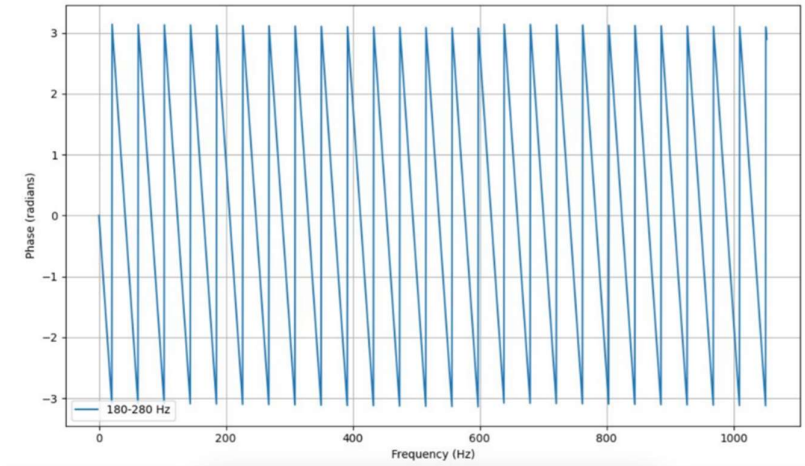
Here's a breakdown of the filters:

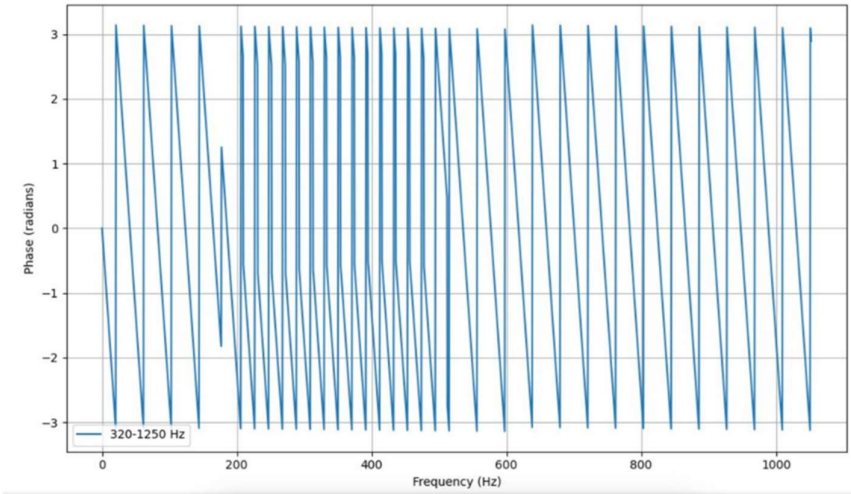
1. **First Band-Reject Filter (h1):**
    - Cutoff frequencies: 0 to 120 Hz
    - Targets the lower frequency noise components.
2. **Second Band-Reject Filter (h2):**
    - Cutoff frequencies: 180 to 280 Hz
    - Targets another band of noise components.
3. **Third Band-Reject Filter (h3):**
    - Cutoff frequencies: 320 to 1250 Hz
    - Addresses a broader range of mid-frequency noise.

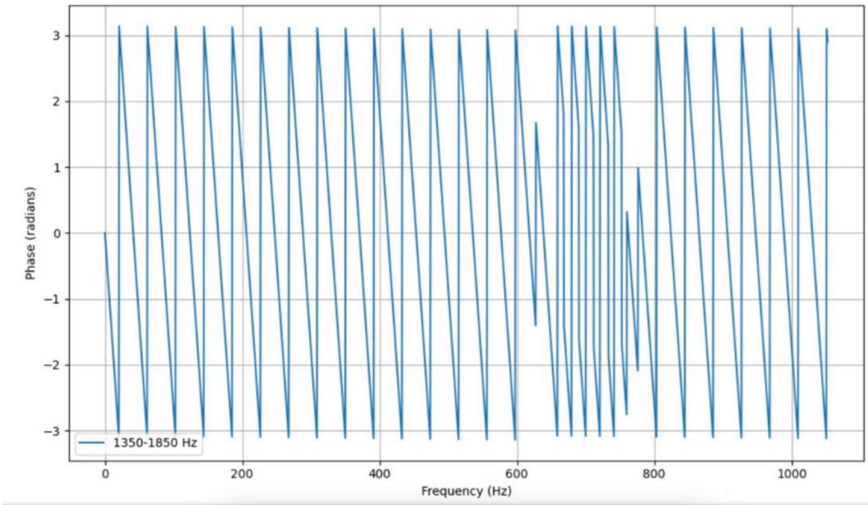
4. **Fourth Band-Reject Filter (h4):**
    - Cutoff frequencies: 1350 to 1850 Hz
    - Targets higher frequency noise components.

Below are the Frequency Response Plots:






## Filter Equations:

1. **Hamming Window:**

$$w(n) = \alpha - \beta cos(2\pi n/(N-1))$$

where:

- $\alpha$=0.54 and $\beta$=0.46 are coefficients chosen for the Hamming window,
- $N$ is the number of taps (filter order),
- $n$ is the index vector.

2. **Filter Coefficients:**
   - The filter coefficients $h(n)$ are calculated using the formula:

$$h(n) = w(n)(sin(\pi(n - Tuo)) - (sin(wc2(n - Tuo)) - sin(wc1(n - Tuo))))/(\pi(n - Tuo))$$

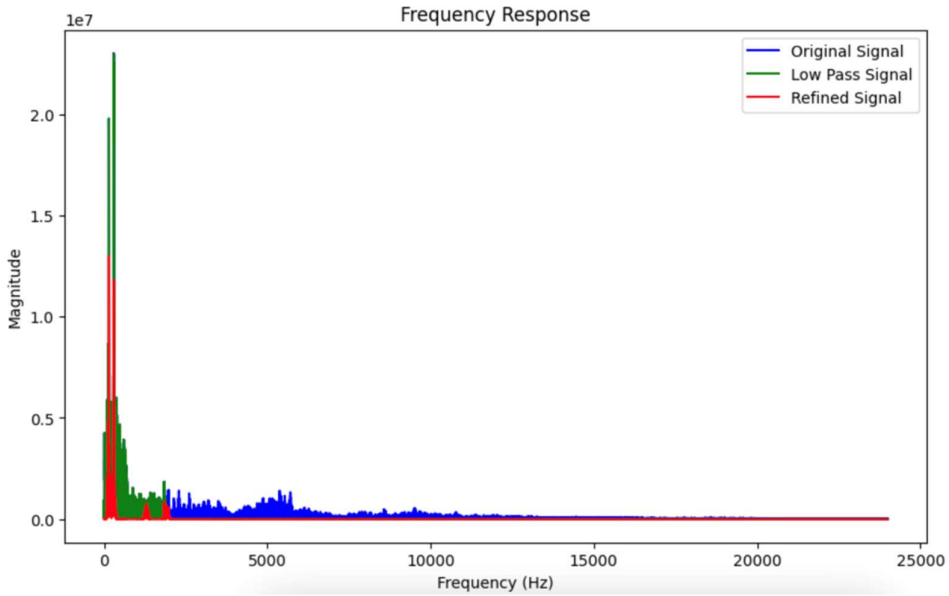
   where $Tuo$ is $(N-1)/2$, $wc1$ and $wc2$ are the angular frequencies corresponding to the band edges.
3. **Normalization:**
   - The filter is normalized by dividing the coefficients by the sum of the coefficients to achieve unity gain.

## TASK 3:

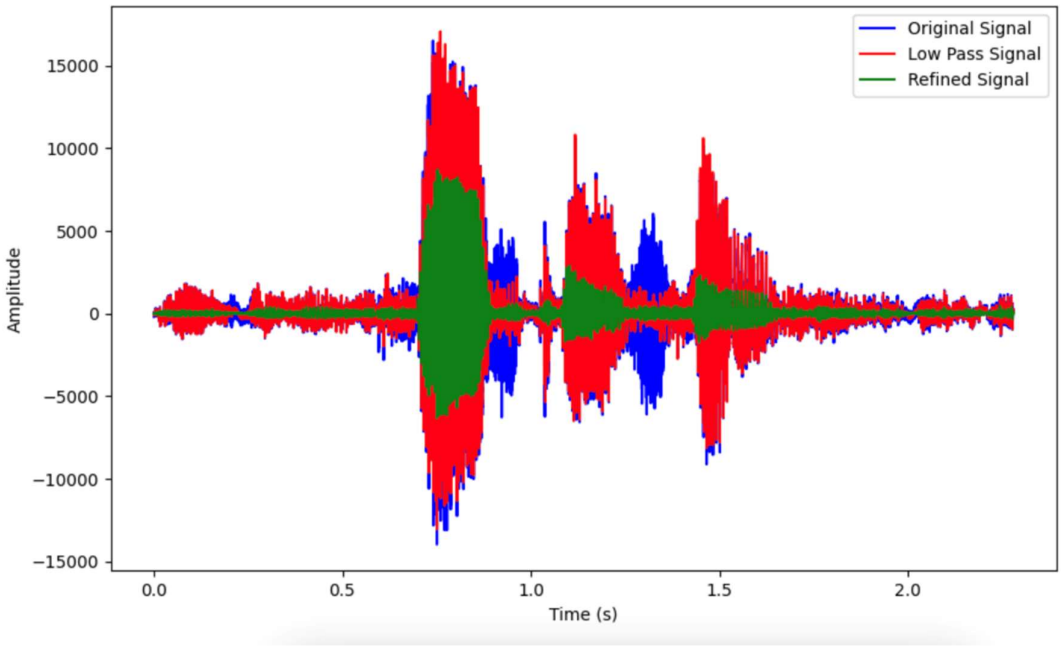
Implement the filter you suggested in Part 2 to remove the remaining noise components from the filtered signal. Plot the magnitude and phase response of the designed filter. Also, plot the original noisy signal, the filtered signal from Part 1, and the final cleaned signal with the remaining noise removed. Calculate and display the signal-to-noise ratio (SNR) of the cleaned signal compared to the original noisy signal. (Do not use any DSP Python libraries for filter design. Build your own filter function.)

## Frequency Analysis:

The objective is to gain insights into how the frequency content of the signals has been altered through the processing steps.
The frequency axis is computed using the Fast Fourier Transform (FFT) to convert the time-domain signals into the frequency domain. The freq_axis variable represents the frequencies corresponding to the DFT components.
The DFT is computed for the original signal, low pass signal and filtered signal.



The original audio signal (signal_data) is represented in blue in the plots. This signal serves as the baseline for comparison.
The low-pass filtered signal (filtered_signal), depicted in red, is obtained by applying a low-pass filter to the original signal. This filtering process aims to attenuate high-frequency components and preserve the lower-frequency content of the signal.
The refined signal (signal), illustrated in green, represents the final processed signal after additional steps such as noise reduction or enhancement. This refined signal is expected to exhibit improvements over the original signal.

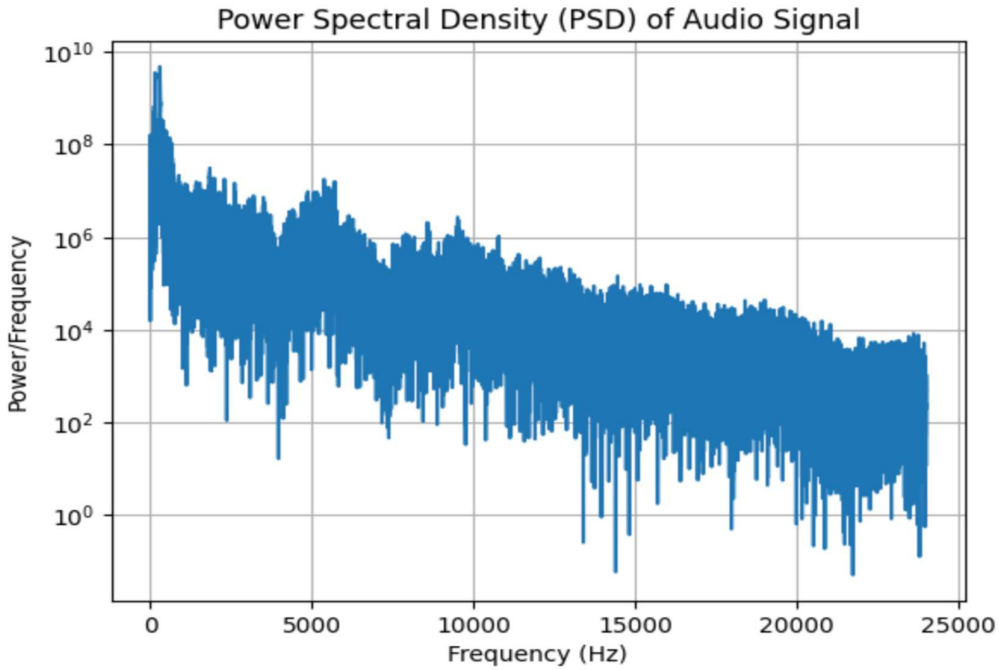


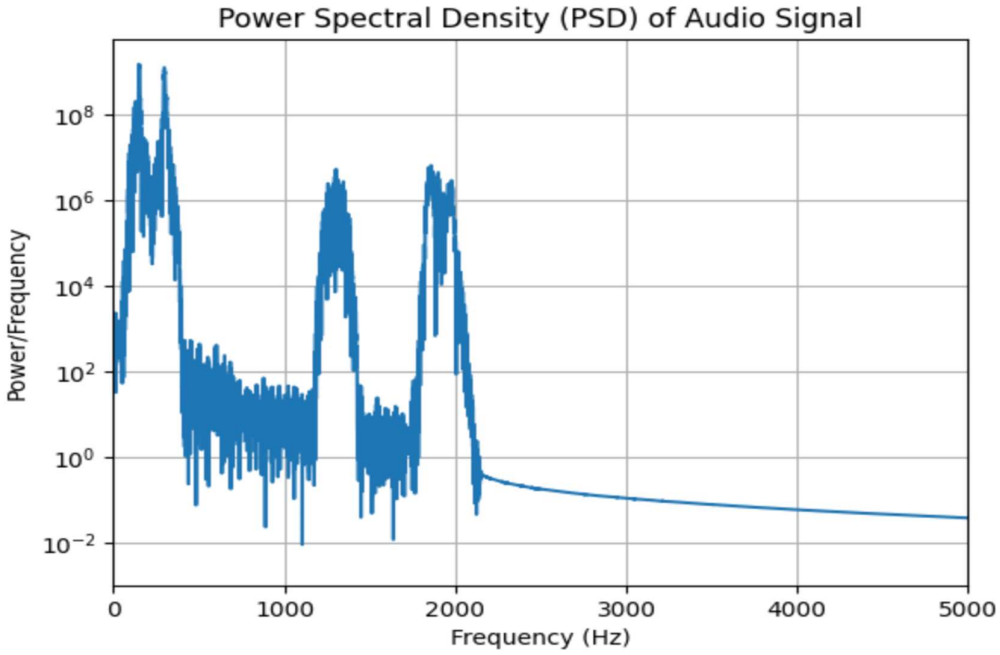

## SNR

The Signal-to-Noise Ratio (SNR) is a measure of the quality of a signal and is often used to quantify how much a signal stands out from background noise. In the context of comparing a clean signal and a noisy signal, the SNR is calculated using Summation of squares in the difference.
Result:- 3.9649968216285285

## Power Spectral Density

One-Sided FFT: The Fast Fourier Transform (FFT) was applied to the audio signal to transform it from the time domain to the frequency domain. The frequencies were computed using numpy.fft.fftfreq, and the FFT values were obtained.
**Magnitude Spectrum:** The magnitude spectrum was calculated by taking the absolute values of the FFT output. This spectrum represents the magnitude of each frequency component in the signal.
**Power Spectral Density (PSD):** The PSD was computed by squaring the magnitude spectrum values and normalizing them by the length of the signal.


Power Spectral Density (PSD) of Audio Signal
Power/Frequency
Frequency (Hz)
$10^{10}$
$10^{8}$
$10^{6}$
$10^{4}$
$10^{2}$
$10^{0}$
0
5000
10000
15000
20000
25000


Power Spectral Density (PSD) of Audio Signal
Power/Frequency
Frequency (Hz)
$10^{8}$
$10^{6}$
$10^{4}$
$10^{2}$
$10^{0}$
$10^{-2}$
0
1000
2000
3000
4000
5000