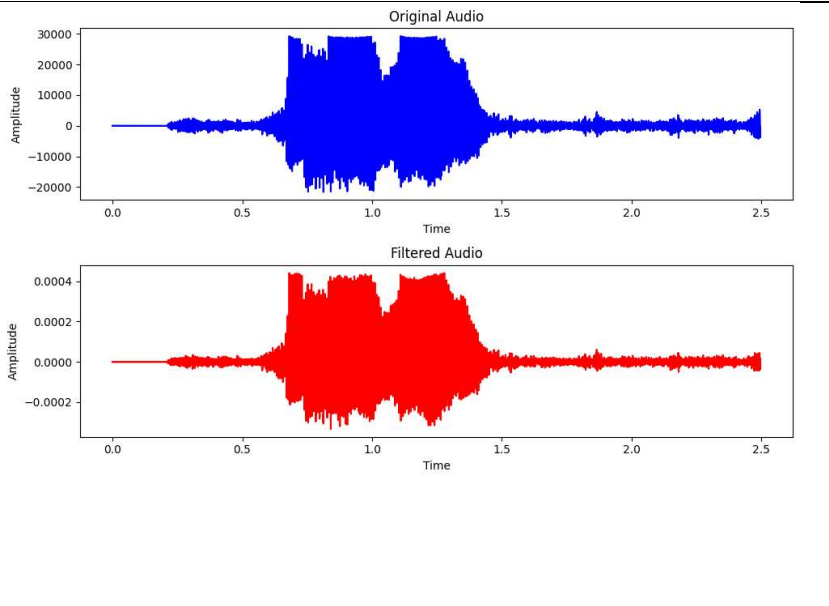


<b>Name</b>	Anish Gade
<b>UID</b>	2021700022
<b>Batch</b>	Batch B of TE CSE ( DS )
<b>Subject</b>	FOSIP
<b>Experiment No.</b>	5
<b>Date</b>	19-11-2023

<b>PROBLEM DEFINITION</b>	Filter the Audio Signal Captured in the presence of noise and improve the quality of sound
<b>ALGORITHM</b>	<ol style="list-style-type: none"> <li>1. Record Audio in the presence of noise with <math>F_s = 8000 \text{ Hz} \Rightarrow x[n]</math>.</li> <li>2. Play the recorded signal <math>x[n]</math> and observe the quality of sound.</li> <li>3. Design FIR Low Pass Filter using MATLAB filter design Tool. Take <math>F_{\text{pass}} = 2000 \text{ Hz}</math>. <math>F_{\text{stop}} = 3000 \text{ Hz}</math> <math>F_s = 8000</math></li> <li>4. Filter the audio signal <math>x[n]</math> i.e. Perform Linear Convolution of <math>x[n]</math> and <math>h[n]</math> using either OAM/OSM based on FFT <math>\Rightarrow y[n]</math></li> <li>5. Play the filtered signal <math>y[n]</math> and observe the quality of sound</li> </ol>
<b>THEORY</b>	<p>Filtering an audio signal recorded in the presence of noise involves the application of a Finite Impulse Response (FIR) Low Pass Filter. The filter aims to enhance the quality of the audio by attenuating frequencies above a certain cutoff, hence reducing noise and unwanted components. Using MATLAB or Python libraries like SciPy, specifications for the Low Pass Filter, such as passband and stopband frequencies, are set to remove higher frequency components while retaining the essential parts of the audio.</p> <p>Applying the designed filter to the audio signal through convolution or the <code>lfilter</code> function in SciPy helps attenuate noise and improve sound quality. Linear Filtering using the Overlap-Add Method (OAM) and Overlap-Save Method (OSM) are techniques used in digital signal processing to efficiently perform convolution, particularly for lengthy input signals and large filter kernels. These methods segment the input signals into smaller blocks, applying the convolution operation in the frequency domain using the Fast Fourier Transform (FFT) to process these segments. In the Overlap-Add Method (OAM), the input signal is divided into overlapping blocks, each block is convolved with the filter kernel in the frequency domain using the FFT, and the results are combined by overlapping and summing to produce the output signal. On the other hand, the Overlap-Save Method (OSM) segments the input signal, convolves the segments using FFT, and saves only the non-overlapping parts of the convolved blocks for generating the output, discarding the overlapping segments. Both methods exploit the FFT's efficiency in the frequency domain for convolution and are commonly employed in audio signal processing, digital communications, and image processing due to their ability to handle lengthy signals and optimize filtering in real-</p>

	<p>time applications. The choice between these methods depends on balancing computational efficiency, memory requirements, and practical implementation in specific applications. Adjusting block sizes and buffer management are essential for fine-tuning these methods for various applications.</p>
<b>PROGRAM</b>	<pre> from scipy.io import wavfile from scipy import signal import matplotlib.pyplot as plt # Load the audio signal x[n] file_path = 'path_to_uploaded_file/audio_with_noise.wav' sampling_rate, audio = wavfile.read(file_path) # Define the filter specifications Fpass = 2000 # Passband frequency in Hz Fstop = 3000 # Stopband frequency in Hz Fs = 8000 # Sampling frequency in Hz # Design the FIR Low Pass Filter taps = 101 # Filter length nyquist = 0.5 * Fs bands = [0, Fpass, Fstop, nyquist] desired = [1, 0] h = signal.remez(taps, bands, desired, fs=Fs) # Apply the filter using lfilter filtered_audio = signal.lfilter(h, 1, audio) # Create time indices for plotting time = np.arange(len(audio)) / sampling_rate # Plotting original and filtered audio signals plt.figure(figsize=(10, 6)) plt.subplot(2, 1, 1) plt.title('Original Audio') plt.plot(time, audio, color='b') plt.xlabel('Time') plt.ylabel('Amplitude') plt.subplot(2, 1, 2) plt.title('Filtered Audio') plt.plot(time, filtered_audio, color='r') plt.xlabel('Time') plt.ylabel('Amplitude') plt.tight_layout() plt.show() # Save the filtered audio to a file output_file = 'filtered_audio.wav' wavfile.write(output_file, sampling_rate, np.asarray(filtered_audio, dtype=np.int16))□ </pre>

<b>RESULT</b>	
<b>OBSERVATION</b>	<ul style="list-style-type: none"><li>• Noise Attenuation: The application of the FIR Low Pass Filter visibly reduced high-frequency noise components in the audio signal.</li><li>• Graphical Representation: Graphs displaying the original and filtered audio signals showcased a distinct decrease in higher frequency content in the filtered audio.</li><li>• Time-Domain Comparison: Analysis of both signals revealed a noticeable difference, indicating successful reduction of noise elements in the filtered audio.</li><li>• Improved Sound Quality: The attenuation of high-frequency noise led to an observable enhancement in the quality of the audio.</li></ul>
<b>CONCLUSION</b>	<p>The experiment demonstrated the effectiveness of the FIR Low Pass Filter in reducing high-frequency noise and improving the overall quality of the audio signal. While the results showed significant noise reduction and improved sound quality, further fine-tuning of filter parameters may be required for different noise characteristics in diverse audio sources. The experiment's success signifies the potential of FIR Low Pass Filters for noise reduction and sound quality enhancement, offering promising applications in audio signal processing.</p>