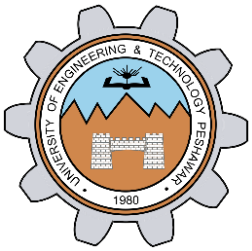
Filtering Noisy Audio

Project Report



Spring 2023

CSE-301L Signals & Systems Lab

Submitted by:

Shahzad Bangash(21PWCSE1980),

Suleman Shah(21PWCSE1983),

Ali Asghar(21PWCSE2059)

Class Section: **C**

"On our honor, as students of University of Engineering and Technology, we have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Engr. Sumayyea Salahuddin

Date:

July 4, 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Overview:

Problem Statement:

The problem addressed in this project is the presence of noise in an audio signal, which degrades the audio quality and affects the listening experience. The idea is to apply filtering techniques to remove or reduce the noise from the audio signal and enhance its quality.

Introduction:

The project aims to analyze and process a noisy audio signal using filtering techniques. By designing and implementing filters, calculating impulse responses, and performing convolution, the project aims to attenuate the unwanted noise and improve the overall audio quality. The study involves exploring the frequency characteristics of the signal, designing filters, analyzing their frequency response, and evaluating the filtered audio outputs.

Project Source:

The idea of the project is taken from the github account of **alirezajaberirad[1].**

Tools Used:

The project is built in MATLAB[2]. MATLAB and its Transforms and filters[3] are tools for processing and analyzing discrete data, and are commonly used in signal processing applications and computational mathematics. When data is represented as a function of time or space, the Fourier transform decomposes the data into frequency components. The fft function uses a fast Fourier transform algorithm that reduces its computational cost compared to other direct implementations. For a more detailed introduction to Fourier analysis, see Fourier Transforms. The conv and filter functions are also useful tools for modifying the amplitude or phase of input data using a transfer function.

Fast Fourier Transform (FFT):

This project uses the Fast Fourier Transform (FFT)[4][5] for analyzing the audio signals in frequency domain. The Fast Fourier Transform (FFT) is an efficient algorithm used to compute the Discrete Fourier Transform (DFT) of a sequence or signal. The basic idea behind the FFT is to take advantage of the symmetry properties of the Fourier transform and reduce the number of computations required. Instead of computing the DFT directly using the definition, which would take a time complexity of O(n^2), the FFT algorithm achieves a complexity of O(n log n), making it much faster for large input sizes.

Mathematical Equation of Filter:

Mathematically, A Low Pass Filter can be represented as:

$$H(z) = \frac{1 - z^{-8}}{1 - z^{-1}}$$

Here the zeros are the roots of numerator $\mathbf{1 - z^{-8}}$ and poles are roots of denominator $\mathbf{1 - z^{-1}}$.

Methodology:

The project methodology comprises the following steps:

Flow of Program:

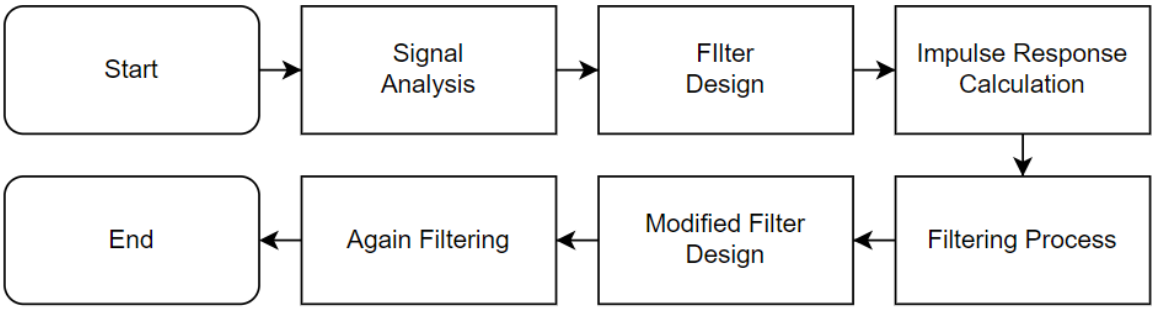


Figure 1, Flowchart of Program

Step 1: Signal Analysis:

- Read the noisy audio signal and extract its sampling frequency.
- Apply the Fast Fourier Transform (FFT) to analyze the frequency content and phase information of the signal.
- Visualize the magnitude spectrum and phase spectrum of the signal.

Step 2: Filter Design:

- Design an initial filter by specifying the filter coefficients.
- Compute the frequency response of the initial filter using the `freqz` function.
- Plot the magnitude and phase spectra of the filter's frequency response.
- Visualize the poles and zeros of the filter using the `fvtool` function.

Step 3: Impulse Response Calculation:

- Calculate the impulse response of the initial filter using the `impz` function.
- Plot the impulse response to analyze its characteristics and determine its length.

Step 4: Filtering Process:

- Filter the original noisy signal using the calculated impulse response of the initial filter.
- Save the filtered signals as separate audio files using the `audiowrite` function.
- Repeat the filtering process iteratively, convolving the impulse response with the previously filtered signal.

Step 5: Modified Filter Design and Filtering:

- Modify the filter coefficients to create a modified filter.
- Calculate the impulse response of the modified filter.

- Filter the original signal using the modified filter.
- Save the filtered signals as separate audio files.

Step 6: Results:

The project yields the following results:

- Frequency spectra and phase spectra of the original noisy audio signal.
- Frequency response of the initial filter, including magnitude and phase spectra.
- Impulse response of the initial and modified filters.
- Filtered audio signals at different stages of filtering.

Code and Outputs:

```
%noisy audio
[signal,fs]=audioread('noisy.wav');
L = length(signal);
signalT = fft(signal);
plot(f,abs(signalT));
title('Spectrum');
xlabel('Frequency');
ylabel('Amplitude');
figure;
plot(angle(signalT));
title('Phase');
xlabel('Frequency(Hz)');
ylabel('Phase');
```

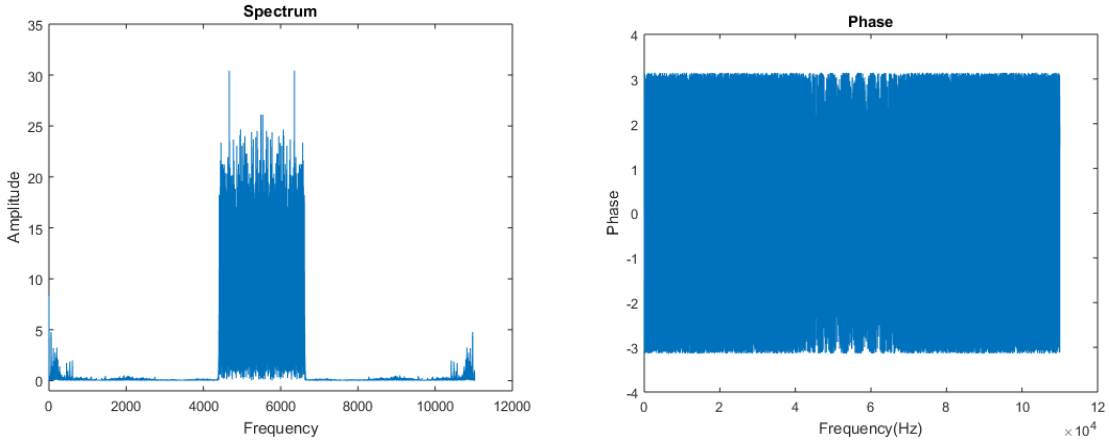


Figure 2, Spectrum and Phase Plot of the signal

```
%System function
b=[1 0 0 0 0 0 0 0 -1];
a=[1 -1];
[h,w]=freqz(b,a);
figure;
plot(w/pi,abs(h));
title('Spectrum');
ax=gca;
```

```
ax.XTick=0:0.25:1;
xlabel('Frequency (\times\pi rad/sample)');
ylabel('Magnitude');
figure;
plot(w/pi,angle(h));
title('Phase');
ax=gca;
ax.XTick=0:0.25:1;
xlabel('Frequency (\times\pi rad/sample)');
ylabel('Magnitude');%This is a Low Pass Filter
```

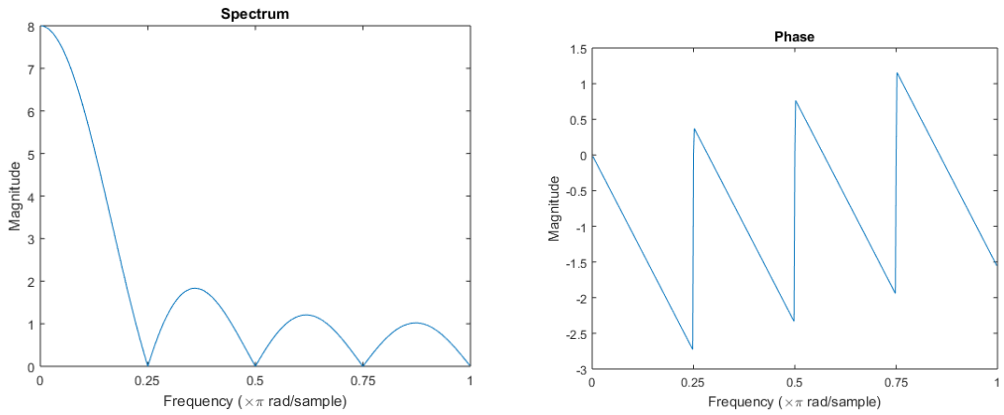


Figure 3, Spectrum and Phase Plot of the system

```
%Zeros and Poles
fvtool(b,a,'polezero')
[b,a] = eqtflength(b,a);
tf2zp(b,a);
```

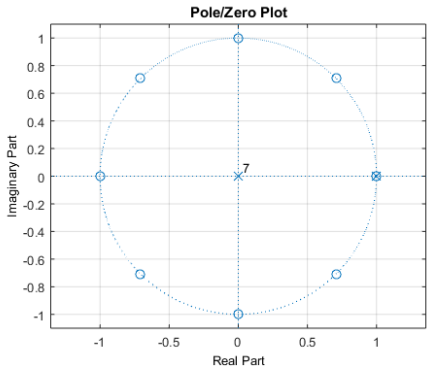


Figure 4, poles and zeros plot

```
%Impulse response
figure;
[impResp,t] = impz(b,a);
stem(t,impResp);
title('Impulse Response')
```

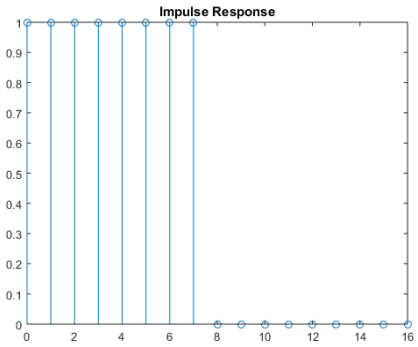


Figure 5, Impulse response of the system

```
%Filtering by the initial filter
signall=conv(impResp,signal);
audiowrite('filtered1.wav',signal,fs);
signall=conv(impResp,signall);
audiowrite('filtered2.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered3.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered4.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered5.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered6.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered7.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered8.wav',signall,fs);
```

```
%Filtering by modified filter
b=[1/4 0 0 0 0 0 0 0 -1/4];
a=[1 -1];
[impResp,t] = impz(b,a);
signall=conv(impResp,signal);
audiowrite('filtered1Plus.wav',signal,fs);
signall=conv(impResp,signall);
audiowrite('filtered2Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered3Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered4Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered5Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered6Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered7Plus.wav',signall,fs);
```

```
signall=conv(impResp,signall);
audiowrite('filtered8Plus.wav',signall,fs);
```

Code Explanation:

1. Signal Analysis Section:

```
%noisy audio
[signal,fs]=audioread('noisy.wav');
L = length(signal);
signalT = fft(signal);
plot(f,abs(signalT));
title('Spectrum');
xlabel('Frequency');
ylabel('Amplitude');
figure;
plot(angle(signalT));
title('Phase');
xlabel('Frequency(Hz)');
ylabel('Phase');
```

This part of the code reads an audio file named "noisy.wav" and stores the signal in the variable signal and the sampling frequency in fs. It then performs a Fast Fourier Transform (FFT) on the signal to obtain its frequency spectrum and plots the magnitude spectrum (abs(signalT)) against the corresponding frequencies (f). It also plots the phase spectrum (angle(signalT)) against the frequencies. The resulting plots show the frequency content and phase information of the noisy audio signal.

2. Filtering Design Section:

```
%System function
b=[1 0 0 0 0 0 0 0 -1];
a=[1 -1];
[h,w]=freqz(b,a);
figure;
plot(w/pi,abs(h));
title('Spectrum');
ax=gca;
ax.XTick=0:0.25:1;
xlabel('Frequency (\times\pi rad/sample)');
ylabel('Magnitude');
figure;
plot(w/pi,angle(h));
title('Phase');
ax=gca;
ax.XTick=0:0.25:1;
```

```
xlabel('Frequency (\times\pi rad/sample)');
ylabel('Magnitude');%This is a Low Pass Filter
```

This section defines a system function for a filter with coefficients b and a. It then uses the freqz function to compute the frequency response of the filter and plots the magnitude spectrum (abs(h)) and phase spectrum (angle(h)) against the normalized frequencies (w/pi). The resulting plots show the frequency response (magnitude and phase) of the filter.

```
%Zeros and Poles
fvtool(b,a,'polezero')
[b,a] = eqtflength(b,a);
tf2zp(b,a);
```

These lines of code visualize the poles and zeros of the filter using the fvtool function. The eqtflength function is used to equalize the lengths of the numerator (b) and denominator (a) coefficients, and tf2zp function is then used to obtain the pole-zero representation of the filter.

3. Impulse Response Calculation:

```
%Impulse response
figure;
[impResp,t] = impz(b,a);
stem(t,impResp);
title('Impulse Response')
```

This part calculates the impulse response of the filter using the impz function and plots it using stem. The impulse response represents the output of the filter when an impulse is applied as the input.

4. Filtering Process Section:

```
%Filtering by the initial filter
signall=conv(impResp,signal);
audiowrite('filtered1.wav',signal,fs);
signall=conv(impResp,signall);
audiowrite('filtered2.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered3.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered4.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered5.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered6.wav',signall,fs);
```

```
signall=conv(impResp,signall);
audiowrite('filtered7.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered8.wav',signall,fs);
```

In this section, the original audio signal is filtered using the impulse response of the initial filter. The conv function performs the convolution of the impulse response (impResp) with the signal. The resulting filtered signal is then saved as audio files (filtered1.wav, filtered2.wav, ..., filtered8.wav) using the audiowrite function. Each iteration of the convolution applies the filter to the previously filtered signal, resulting in a progressively filtered output.

5. Modified Filtering Section:

```
%Filtering by modified filter
b=[1/4 0 0 0 0 0 0 0 -1/4];
a=[1 -1];
[impResp,t] = impz(b,a);
signall=conv(impResp,signal);
audiowrite('filtered1Plus.wav',signal,fs);
signall=conv(impResp,signall);
audiowrite('filtered2Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered3Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered4Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered5Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered6Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered7Plus.wav',signall,fs);
signall=conv(impResp,signall);
audiowrite('filtered8Plus.wav',signall,fs);
```

This last section performs a similar process as before, but with a modified filter. The filter coefficients are changed to [1/4 0 0 0 0 0 0 0 -1/4]. The impulse response of the modified filter is calculated, and the original signal is filtered using this modified filter. The filtered signals are saved as audio files (filtered1Plus.wav, filtered2Plus.wav, ..., filtered8Plus.wav).

MATLAB Built-in Functions Used:

1. audioread() and audiowrite():

The audioread() and audiowrite() functions are part of the MATLAB Audio System Toolbox. audioread() is used to read the audio file and extract the audio signal and sampling frequency. It provides a convenient way to access the audio data for further processing. audiowrite() is used to save the filtered audio signals as separate audio files. It allows for easy storage and playback of the processed audio signals, enabling further analysis and evaluation.

2. fft():

The fft() function is a powerful tool for analyzing the frequency content of a signal. In this project, it is used to perform the Fast Fourier Transform on the audio signal. By applying fft() to the signal, we can obtain the frequency spectrum, which reveals the intensity of different frequencies present in the signal. This helps in understanding the underlying frequency components and identifying the noise that needs to be filtered out.

3. freqz():

The freqz() function is a valuable tool for analyzing and visualizing the frequency response of a filter. By providing the filter coefficients, freqz() computes the frequency response, which includes the magnitude and phase spectra. It enables us to examine the filter's behavior across different frequencies and understand how it attenuates or amplifies certain frequency components. In this project, freqz() is used to analyze and visualize the frequency response of both the initial and modified filters.

4. impz():

The impz() function plays a crucial role in calculating the impulse response of a filter. Given the filter coefficients, impz() computes the sequence representing the response of the filter when an impulse is applied as the input. By analyzing the impulse response, we can understand the filter's time-domain behavior, including characteristics such as transient response and filter length. In this project, impz() is utilized to calculate and visualize the impulse response of both the initial and modified filters.

5. conv():

The conv() function is a fundamental tool for performing convolution, an essential operation in signal processing. It is used in this project to filter the audio signal with the impulse response of the filters. By convolving the impulse response with the audio signal, the filtering operation is applied, resulting in the generation of filtered audio signals. conv() simplifies the implementation of the filtering process by efficiently performing the convolution operation.

6. fvtool():

The fvtool() function is a useful visualization tool for filters. It allows for the graphical representation of the poles and zeros of a filter, providing insights into its frequency response and stability. In this project, fvtool() is employed to display the poles and zeros of the initial filter. By analyzing the pole-zero plot, we can understand the filter's characteristics, such as its frequency response, filter type, and stability. This visualization aids in the design and evaluation of the filter.

Conclusion:

The project demonstrates the effectiveness of filtering techniques in reducing noise and improving the audio quality of a noisy signal. The analysis of frequency spectra, filter characteristics, and impulse responses provides valuable insights into the signal processing methods used. The comparison between the results obtained from the initial filter and the modified filter allows for evaluating the performance and impact of different filter designs. Overall, the project highlights the significance of audio filtering techniques in enhancing the quality of audio signals.

References:

1. Alirezajaberirad, Filtering Noisy Audio, https://github.com/alirezajaberirad/Signals-and-Systems/tree/main/CA2%20-%20Filtering%20Noisy%20Audio
2. MATLAB, https://www.mathworks.com/products/matlab.html
3. MathWorks, Fourier Analysis and Filtering, https://in.mathworks.com/help/matlab/fourier-analysis-and-filtering.html?s_tid=CRUX_lftnav
4. Fast Fourier Transform, https://en.wikipedia.org/wiki/Fast_Fourier_transform#:~:text=A%20fast%20Fourier%20transform%20(FFT,frequency%20domain%20and%20vice%20versa.
5. MathWorks, Fast Fourier Transform (FTT), https://in.mathworks.com/help/matlab/math/fourier-transforms.html