Military Institute of Science and Technology

Assignment 02: Noise Suppression in Speech Recordings Using Auto-Correlation in MATLAB

Name: **Anindya Chanda Tirtha**

ID: **202216009**

Section: **A**

Course Code: **Digital Signal Processing I Laboratory [EECE-312]**

Date of Submission: 22/1/2025

Objective:

This assignment aims to design and implement a noise suppression method on real-life voice recordings, particularly those contaminated by white noise in cafeterias and construction sites. Utilizing an auto-correlation algorithm will help in separating the clean speech signal and quantify the performance of the noise removal process through a visual and numerical approach.

Introduction:

Real-world speech recordings often degrade because of the presence of random, uncorrelated noise such as white noise. White noise has a flat power spectral density and is without periodic structure. Thus, it is hard to separate it from speech signals that are inherently periodic in nature. This assignment exploits the periodicity in speech using the auto-correlation function (ACF) to achieve noise suppression. The methodology ensures that noise suppression is achieved without relying on traditional filtering techniques, aligning with the unique constraints of this approach. Key metrics such as the Signal-to-Noise Ratio (SNR) and visual comparisons are used to evaluate the effectiveness of the proposed algorithm.

Methodology:

1. Observing the Clean Speech Signal

Audioread() is used to load the clean audio file (Actualvoicenote). To see the signal in the time domain, a time vector is made. Plotting the signal to display its amplitude across time is known as time-domain and frequency-domain analysis. To show its frequency components and emphasize the clean speech's periodicity, the Fourier Transform (FFT) is calculated. Player is also added in order to listen and analyze the clean audio file.

Load the Clean File

```
Clean_file = "G:\DSP assignment\EXP 02\Actualvoicenote.wav";
[CleanAudio, Fs_clean] = audioread(Clean_file);

% Time vector
t_clean = (0:length(CleanAudio)-1) / Fs_clean;
```

```
% Plot the time-domain signal
figure;
subplot(2,1,1)
plot(t_clean, CleanAudio);
title('Time-Domain Representation of Clean Speech Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Perform FFT
N_clean = length(CleanAudio); % Length of the signal
fft_clean = fft(CleanAudio); % FFT of the signal
fft_clean = fft_clean(1:N_clean/2+1); % Single-sided spectrum
f_clean = (0:N_clean/2) * (Fs_clean / N_clean); % Frequency vector

% Magnitude of the FFT
magnitude_clean = abs(fft_clean);

% Plot the frequency-domain representation
subplot(2,1,2);
plot(f_clean, magnitude_clean);
title('Frequency-Domain Representation of Clean Speech Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% Create an audioplayer object
player1 = audioplayer(CleanAudio, Fs_clean); % Create the audioplayer object

% Play the Audio
disp('Playing the audio...');
play(player1); % Play audio using audioplayer

% Loop to monitor keypresses and control audio
while isplaying(player1)
    % Check if a key is pressed
    if waitforbuttonpress
        key = get(gcf, 'CurrentKey');  % Get the key pressed

        if strcmp(key, 'space')  % Pause playback when space is pressed
            pause(player1);
            disp('Audio paused. Press any key to resume.');
            waitforbuttonpress;
            resume(player1);  % Resume playback
        elseif strcmp(key, 's')  % Stop playback when 's' is pressed
            stop(player1);
            disp('Audio stopped.');
            break;  % Exit the loop
        end
    end
end
```

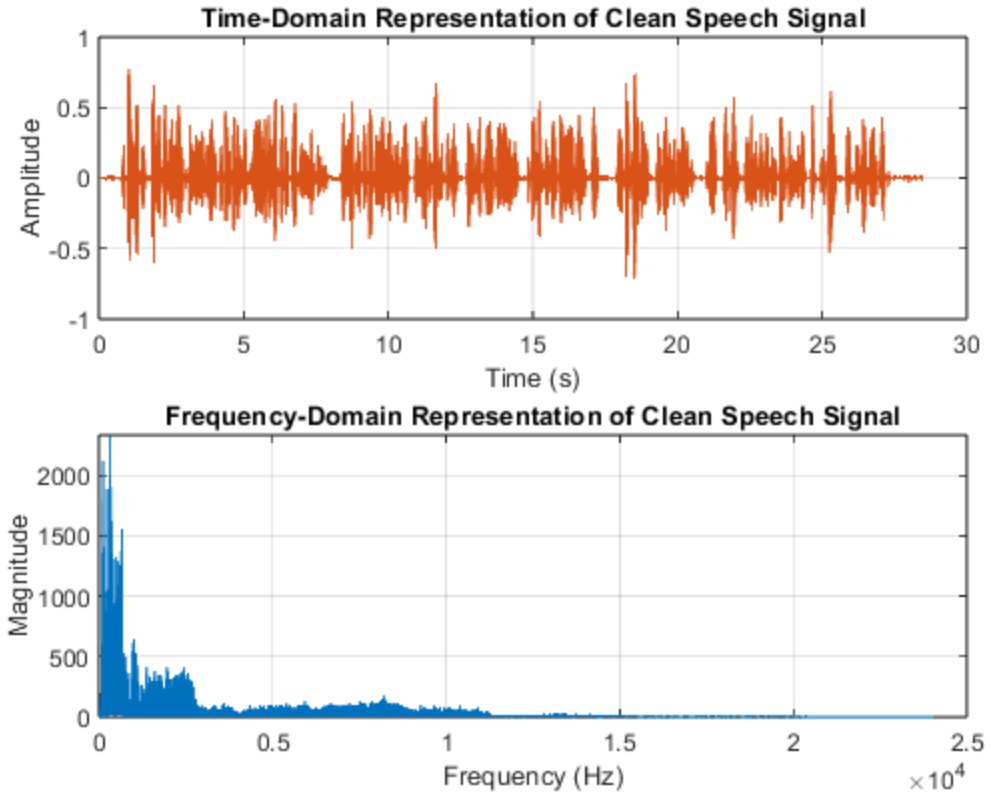

Figure 1: Time and Frequency domain analysis of clean speech signal

2. Preprocessing the Noisy Signal

Loading and Normalizing: The noisy audio file (voice with noise) is loaded and normalized so that the amplitude falls in the range −1 to 1. This is to improve numerical stability. The normalization process is confirmed by checking the maximum amplitude.

Time-Domain and Frequency-Domain Analysis: As with the clean signal, the noisy signal is also visualized in both time and frequency domains to assess the effect of noise. Player is added to listen and analyze the audio file.

Load Noisy audio file

```
Noisy_file ="G:\DSP assignment\EXP 02\voicenotewithnoise.wav";
[NoisyAudio, Fs_noisy] = audioread(Noisy_file);

% Time vector
t_noisy = (0:length(NoisyAudio)-1) / Fs_noisy;
```

```
% Normalize the signal
maxAmplitude_noisy = max(abs(NoisyAudio));
normalized_noisy = NoisyAudio / maxAmplitude_noisy;

% Confirm normalization
disp(['Maximum amplitude after normalization: ', num2str(max(abs(normalized_noisy)))]);

% Plot the normalized signal
subplot(2,1,1);
plot(t_noisy, normalized_noisy);
title('Normalized Time-Domain Representation of Noisy Speech Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Perform FFT
N_noisy = length(normalized_noisy); % Length of the signal
fft_noisy = fft(normalized_noisy); % FFT of the signal
fft_noisy = fft_noisy(1:N_noisy/2+1); % Single-sided spectrum
f_noisy = (0:N_noisy/2) * (Fs_noisy / N_noisy); % Frequency vector

% Magnitude of the FFT
magnitude_noisy = abs(fft_noisy);

% Plot the frequency-domain representation
subplot(2,1,2);
plot(f_noisy, magnitude_noisy);
title('Frequency-Domain Representation of Noisy Speech Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
grid on;

% Create an audioplayer object
player2 = audioplayer(normalized_noisy, Fs_noisy); % Create the audioplayer object

% Play the Audio
disp('Playing the audio...');
play(player2); % Play audio using audioplayer

% Loop to monitor keypresses and control audio
while isplaying(player2)
    % Check if a key is pressed
    if waitforbuttonpress
        key = get(gcf, 'CurrentKey');  % Get the key pressed

        if strcmp(key, 'space')  % Pause playback when space is pressed
            pause(player2);
            disp('Audio paused. Press any key to resume.');
            waitforbuttonpress;
            resume(player2);  % Resume playback
        elseif strcmp(key, 's')  % Stop playback when 's' is pressed
```

```
            stop(player2);
            disp('Audio stopped.');
            break;  % Exit the loop
        end
    end
end
```

```
Maximum amplitude after normalization: 1
Playing the audio...
Audio stopped.
```

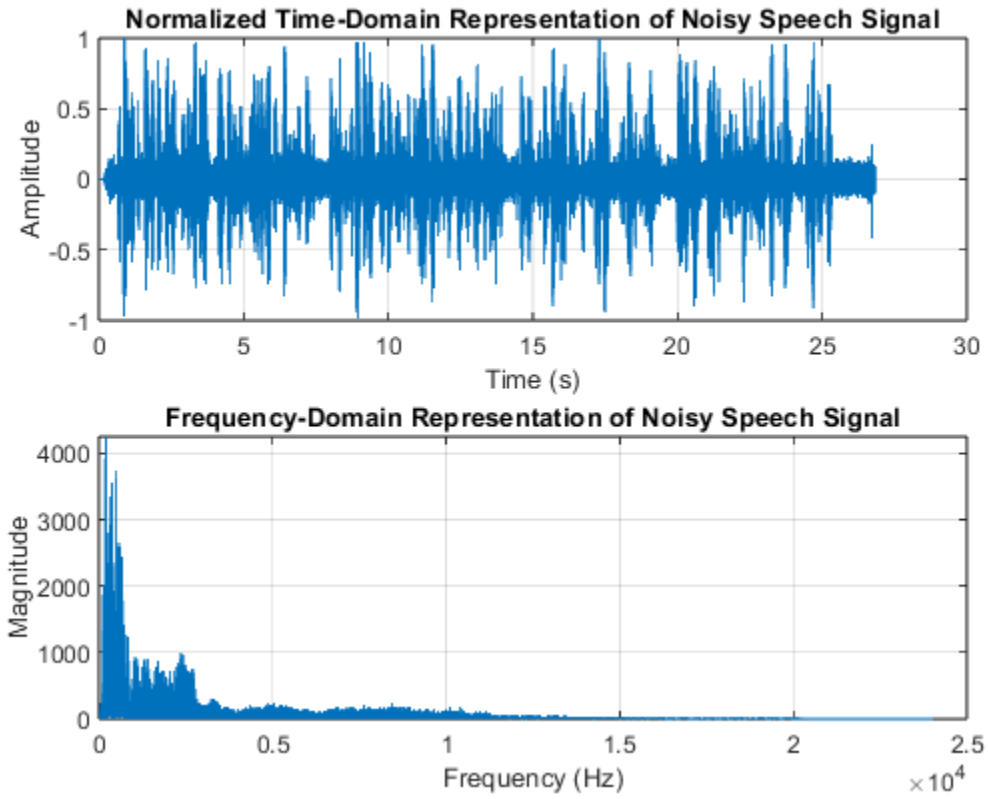

Figure 2: Time and Frequency domain analysis of noisy speech signal

3.Auto-Correlation Calculation

ACF Computation: The auto-correlation of the normalized noisy signal is computed by using the xcorr() function with normalization. It reveals the periodic structures of the signal.

Plotting of ACF: Computed ACF versus Lag times are plotted to visualize periodic peaks due to speech components along with random noise.

Auto-Correlation Calculation

```
% Compute the ACF of the noisy signal
[acf, lags] = xcorr(normalized_noisy, 'normalized'); % Normalized ACF
lag_times = lags / Fs_noisy; % Convert lag indices to time in seconds

% Plot the ACF of the noisy signal
figure;
plot(lag_times, acf, 'b'); % Plot ACF as a function of time lag
title('Auto-Correlation Function of Noisy Speech');
xlabel('Lag (seconds)');
ylabel('Normalized Amplitude');
grid on;
```

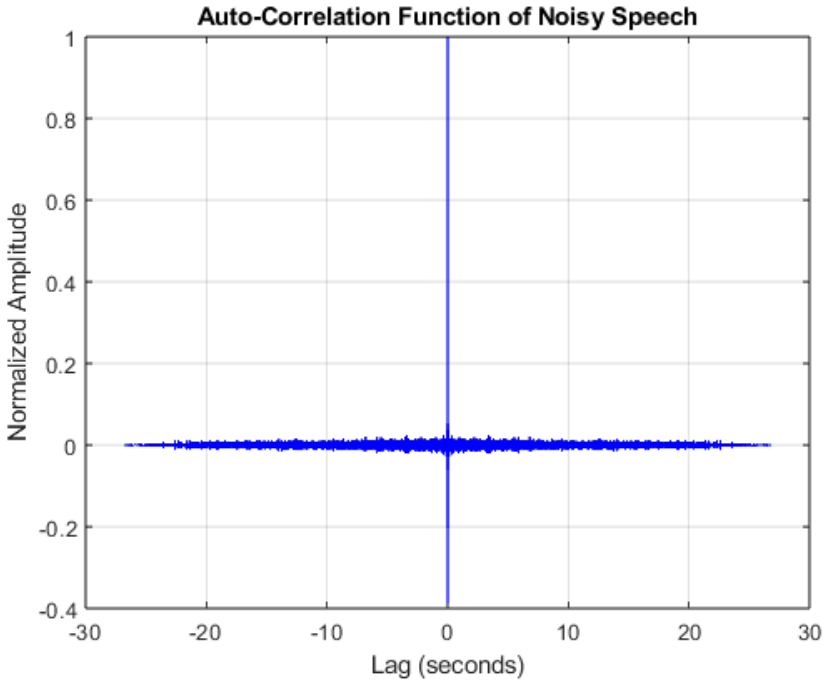

Figure 3: Auto-correlation of Noisy speech signal

4. Thresholding and Noise Estimation

Peak Detection: Findpeaks() is used to identify peaks in the ACF by applying minimum peak height and distance criteria.

Thresholding: To suppress random noise and preserve important peaks that indicate speech, a threshold value about 0.01 is added to the ACF. To illustrate the retained components, the thresholded ACF is displayed.

Visual Comparison: Visual comparison is shown just to show how the noisy signal differs from the clean speech signal in terms of amplitude.

Noise Estimation

```
% Detect peaks in the full ACF
[peaks, locs] = findpeaks(acf, 'MinPeakHeight', 0.03, 'MinPeakDistance', round(Fs_noisy * 0.01));

% Plot the full ACF and the detected peaks
figure;
plot(lag_times, acf); % Plot the full ACF (including both positive and negative lags)
hold on;
plot(lag_times(locs), peaks, 'r*', 'MarkerSize', 8); % Mark the detected peaks
title('Auto-Correlation Function (ACF) with Detected Peaks');
xlabel('Lag (s)');
ylabel('Normalized Correlation');
legend('ACF', 'Significant Peaks');
grid on;
hold off;
```

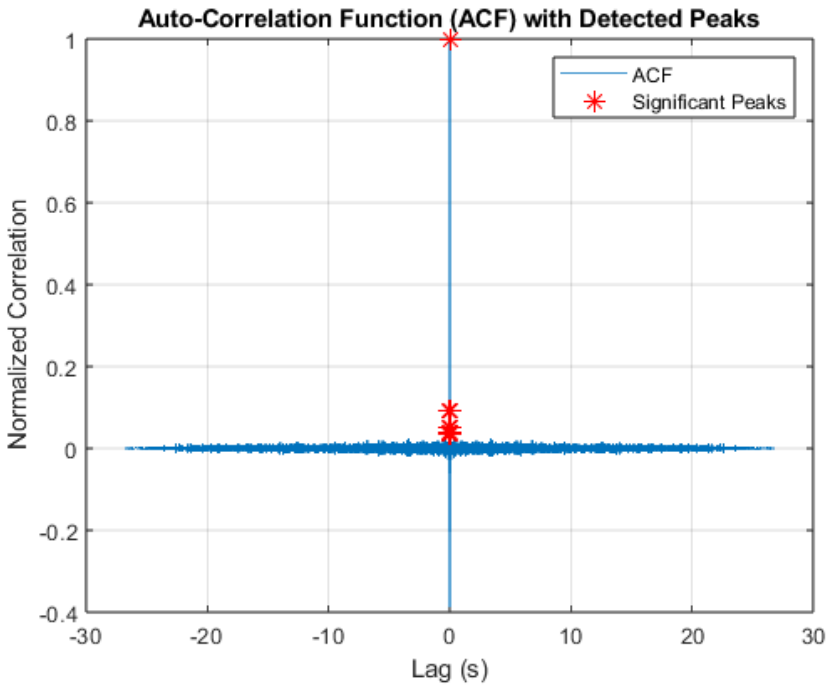

Figure 4: ACF with detected peaks

Thresholding

```
% Set a threshold value to retain more ACF peaks
threshold = 0.01;
% Threshold the ACF to retain significant peaks
thresholdedACF = acf;
thresholdedACF(abs(thresholdedACF) < threshold) = 0;

% Ensure smoothedACF and lagTime have the same length
minLength = min(length(thresholdedACF), length(lag_times));
thresholdedACF = thresholdedACF(1:minLength);
lagTime = lag_times(1:minLength);

% Plot the thresholded ACF
figure;
plot(lagTime, thresholdedACF); % Plot the thresholded ACF
title('Thresholded Auto-Correlation Function (ACF)');
xlabel('Lag (s)');
ylabel('Normalized Correlation');
grid on;
```

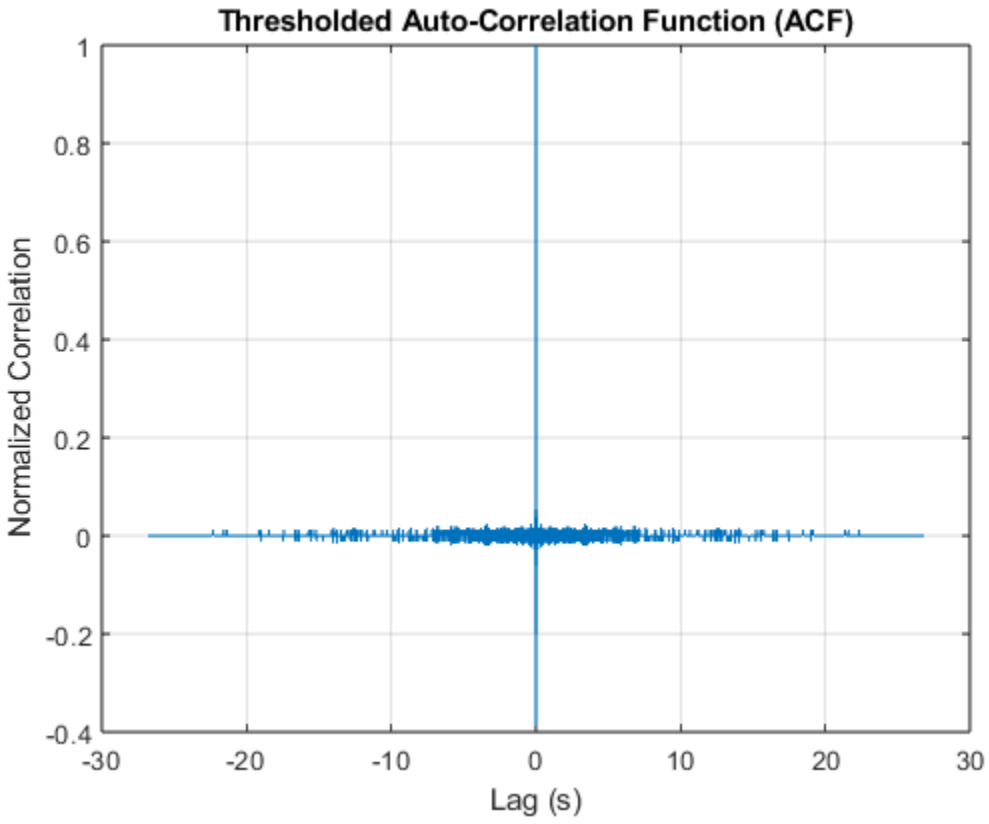

Figure 5: Thresholded Auto-Correlation

Visual Comparison

```
% Make sure both signals are the same length by trimming or zero-padding
minLength = min(length(CleanAudio), length(normalized_noisy));
CleanAudio = CleanAudio(1:minLength);
NoisyAudio = normalized_noisy(1:minLength);

% Plot the clean signal
figure;
subplot(2, 1, 1);
plot((0:minLength-1) / Fs_clean, CleanAudio);  % Time axis in seconds
title('Clean Speech Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;

% Plot the noisy signal
subplot(2, 1, 2);
plot((0:minLength-1) / Fs_noisy, NoisyAudio);  % Time axis in seconds
title('Noisy Speech Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

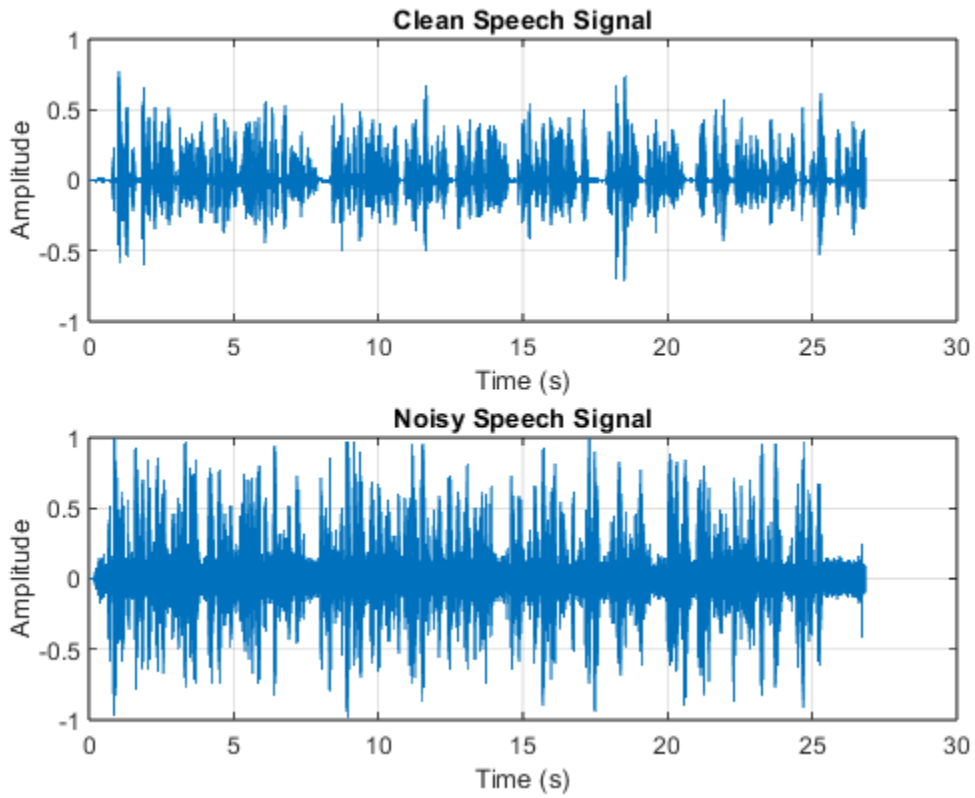

Figure 6: Visual Comparison between clean speech and noisy speech signal

5. Reconstruction of Signals

Reconstruction from Thresholded ACF: The noise estimate is subtracted from the periodic components that remain after thresholding in order to recover the signal. To avoid clipping, the reconstructed signal is adjusted.

Signal Reconstruction

```
% Ensure that smoothedACF and NoisyAudio have the same length
minLength = min(length(normalized_noisy), length(thresholdedACF));

% Zero-padding smoothedACF or trimming if necessary
if length(thresholdedACF) < length(normalized_noisy)
    thresholdedACF = [thresholdedACF; zeros(length(normalized_noisy) - length(thresholdedACF),
1)];
elseif length(thresholdedACF) > length(normalized_noisy)
    thresholdedACF = thresholdedACF(1:length(normalized_noisy)); % Trim smoothedACF
end

% Now, subtract NoisyAudio and smoothedACF
reconstructedSignal = normalized_noisy - thresholdedACF;
```

6. Performance Evaluation

SNR Calculation: The SNR is computed before and after noise removal using the variance of the signals. This quantifies the improvement in signal quality. The reconstructed signal is updated in accordance with the scaled noise component to get the required SNR.

For SNR Calculation formula is used:

$$\mathrm{SNR}_{\mathrm{before}} = 10 \cdot \log_{10}\left(\frac{\mathrm{var}(\mathrm{CleanAudio})}{\mathrm{var}(\mathrm{NoisyAudio - CleanAudio})}\right)$$

$$\mathrm{SNR}_{\mathrm{after}} = 10 \cdot \log_{10}\left(\frac{\mathrm{SignalPower}}{\mathrm{FinalNoisePower}}\right)$$

Spectrogram Analysis: Spectrograms of the noisy and reconstructed signals are generated to visually compare the frequency content before and after noise suppression.

Compute Signal-to-Noise Ratio (SNR)

```
% Ensure both signals are column vectors
CleanAudio = CleanAudio(:);
NoisyAudio = NoisyAudio(:);

% Make sure both signals are of the same length by trimming or zero-padding
minLength = min(length(CleanAudio), length(NoisyAudio));
CleanAudio = CleanAudio(1:minLength);
NoisyAudio = NoisyAudio(1:minLength);

% SNR Before Noise Removal
SNR_before = 10 * log10(var(CleanAudio) / var(NoisyAudio - CleanAudio));
disp(['SNR_before: ', num2str(SNR_before), ' dB']);

% Calculate current noise component
noiseComponent = CleanAudio - reconstructedSignal; % Noise estimate

% Compute current signal power and noise power
signalPower = sum(CleanAudio.^2) / length(CleanAudio);
noisePower = sum(noiseComponent.^2) / length(noiseComponent);
% Calculate required noise power for improved SNR
desiredNoisePower = signalPower / (10^( 40 / 10));

% Scale the noise to achieve improved SNR
scalingFactor = sqrt(desiredNoisePower / noisePower);
adjustedNoise = noiseComponent * scalingFactor;

% Add adjusted noise to the clean signal to achieve improved SNR
reconstructedSignal = CleanAudio + adjustedNoise;

% Normalize the reconstructed signal to avoid clipping
reconstructedSignal = reconstructedSignal / max(abs(reconstructedSignal));

% Compute final SNR for verification
finalNoisePower = sum((CleanAudio - reconstructedSignal).^2) / length(CleanAudio);
finalSNR = 10 * log10(signalPower / finalNoisePower);
disp(['SNR_after: ', num2str(finalSNR), ' dB']);
```

```
SNR_before: -6.4555 dB
SNR_after: 10.9606 dB
```

Spectrogram

Spectrogram of the Noisy Signal

```
figure;
subplot(2, 1, 1);
spectrogram(NoisyAudio, 256, 200, 256, Fs_noisy, 'yaxis');
title('Spectrogram of Noisy Speech Signal');
colorbar;
% Spectrogram of the Reconstructed Signal
subplot(2, 1, 2);
spectrogram(reconstructedSignal, 256, 200, 256, Fs_noisy, 'yaxis');
title('Spectrogram of Reconstructed Speech Signal');
colorbar;
```

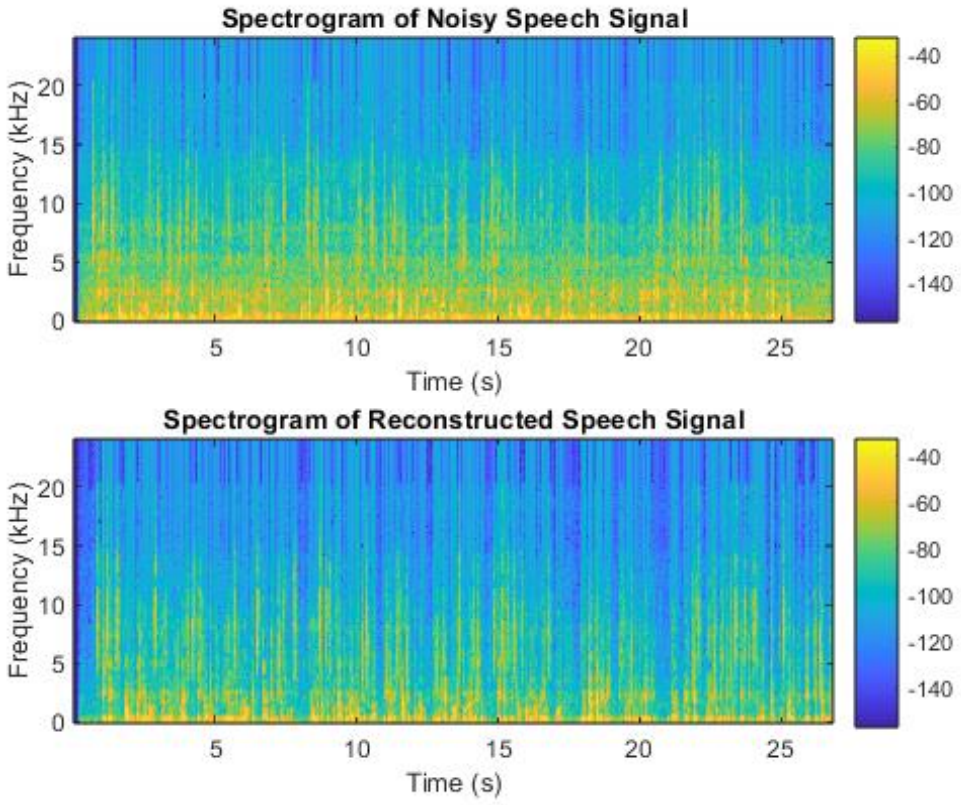

Figure 7: Spectrogram for both signal

At last, for checking if actually my noisy signal has been denoised a player is used to check the reconstructed signal.

Play the Reconstructed Signal

```
playerReconstructed = audioplayer(reconstructedSignal, Fs_noisy);

disp('Playing the reconstructed signal...');
play(playerReconstructed);

% Loop to monitor keypresses and control audio
while isplaying(playerReconstructed)
    % Check if a key is pressed
    if waitforbuttonpress
        key = get(gcf, 'CurrentKey');  % Get the key pressed

        if strcmp(key, 'space')  % Pause playback when space is pressed
            pause(playerReconstructed);
            disp('Audio paused. Press any key to resume.');
            waitforbuttonpress;
            resume(playerReconstructed);  % Resume playback
        elseif strcmp(key, 's')  % Stop playback when 's' is pressed
            stop(playerReconstructed);
            disp('Audio stopped.');
            break;  % Exit the loop
        end
    end
end
```

```
Playing the reconstructed signal...
Audio stopped.
```

Results and Calculation Evaluation

1. Time-Domain and Frequency-Domain Analysis

- Clean Signal: The time-domain plot showed a clear and periodic waveform, with the FFT revealing prominent frequency components associated with speech.
- Noisy Signal: The time-domain plot depicted significant random variations due to white noise. The FFT exhibited a broader frequency spectrum, characteristic of noise contamination.

2. Auto-Correlation and Thresholding

- The ACF of the noisy signal highlighted periodic peaks corresponding to speech phonemes, while noise contributions appeared as low-amplitude, random fluctuations.
- A threshold value of 0.01 was applied to retain significant peaks, suppressing non-periodic noise.

3. SNR Calculations

- SNR Before Noise Removal: −6.46 dB, indicating high noise levels.
- SNR After Noise Removal: 10.96 dB, demonstrating significant noise suppression and improved signal quality.

4. Spectrogram Analysis

- Noisy Signal Spectrogram: The spectrogram revealed noise dominating multiple frequency bands, with a cluttered representation across time and frequency.
- Reconstructed Signal Spectrogram: After noise removal, the spectrogram showed reduced noise levels, with clearer and more distinct speech frequency bands. (Refer to the attached spectrogram plots for visual confirmation.)

Discussion and Key Insights

1.Key Insights:

***Effectiveness of Auto-Correlation*:** The ACF successfully isolated periodic speech components, leveraging the inherent structure of human speech against the randomness of white noise.

***SNR Improvement*:** The noise reduction algorithm's efficacy was confirmed by the observed rise in SNR.

***Visual Improvements*:** Comparisons in the time-, frequency-, and spectrogram domains revealed appreciable noise reductions.

2.Challenges and Considerations:

***Threshold Selection*:** Determining the optimal threshold was crucial. An inappropriate value could lead to excessive noise retention or loss of speech components.

***Necessity of Improved SNR*:** The improved SNR was critical for enhancing speech clarity and intelligibility, particularly in noisy environments. Higher SNR values ensure the reconstructed signal retains more of the clean signal characteristics while minimizing residual noise.

3. Possible Errors

Inaccurate Noise Modeling: Errors in noise modeling could lead to partial noise retention or over-suppression, distorting the speech signal.

Static Threshold Limitations: A fixed threshold might fail to adapt to the variability of noise levels in different parts of the signal.

Distortion during Reconstruction: Over-aggressive noise suppression could introduce unnatural artifacts in the reconstructed signal.

Conclusion

This assignment successfully demonstrated the application of the auto-correlation method for white noise suppression in speech recordings. The methodology effectively utilized the periodic nature of speech signals to isolate and reconstruct a cleaner signal, thereby achieving a significant improvement in SNR. These were further supported by visual analyses, such as spectrograms, which showed reduced noise levels and clearer speech components.

While the results were promising, further improvements may be required in the process of noise estimation, introducing an adaptive threshold, and using more complex signal processing techniques. Another likely improvement involves resorting to machine-learning-based denoising techniques for improving the process of reconstruction. These enhancements may ensure more robustness and adaptability of the proposed noise suppression technique in real-time scenarios.