Line Echo Cancellation Project

Lana, Hadeel, Ahmad

Abstract—The primary objective of this project is to develop an adaptive line echo canceller (LEC) to reduce the negative impacts of echoes in phone communications, consequently improving voice quality for both near-end and far-end speakers. The project includes an analysis of the echo path's impulse and frequency responses, estimation of input and output powers, evaluation of the attenuation introduced by the echo path, training of an adaptive filter using the normalized least mean squares (NLMS) algorithm, and comparison of various adaptive algorithms for echo cancellation. The research aims to successfully minimize echoes and improve the overall intelligibility of telephone conversations by implementing these strategies.

Table of Contents:

Table of Figures:

I. INTRODUCTION

THIS This project centers around developing and accessing a Line Echo Cancellation (LEC) system intended to enhance the quality of voice transmission in phone line communications. The LEC system addresses the issue of echoes arising from disparities in circuitry, particularly at the near end, where the reflected signal from the far end disrupts the desired signal. Employing adaptive filtering techniques, the LEC system endeavors to estimate and eliminate the echo signal from the received audio, thereby improving overall clarity for both parties involved.

The project unfolds through distinct phases. Initially, the characteristics of the echo path are scrutinized by examining its impulse and frequency responses. Subsequently, a synthetic composite source signal, closely resembling speech, is employed to evaluate its properties and spectrum. The resulting echo signal, generated by transmitting concatenated source blocks through the echo path, is then analyzed. Furthermore, the project involves estimating the input and output powers and calculating the echo-return-loss (ERL) attenuation, which denotes the level of signal degradation introduced by the echo path.

In the subsequent stage, an adaptive line echo canceller is implemented, utilizing the far-end signal as the input and the echo signal obtained from the echo path as the reference. The canceller is trained using the normalized least mean squares (NLMS) algorithm with specific parameters. The performance of the canceller is assessed by analyzing the far-end signal, echo signal, and the resulting error signal. Additionally, the estimated response of the finite impulse response (FIR) channel is compared with the predefined FIR system.

Moreover, alternative adaptive algorithms are proposed and placed against the NLMS algorithm, aiming to identify a more suitable approach that can enhance the cancellation performance and surmount any limitations of the NLMS method. The project evaluates various criteria, including signal quality improvements, attenuation levels, and algorithmic comparisons, to quantitatively measure the efficacy of the LEC system.

In conclusion, this project serves to demonstrate the practical implementation of adaptive line echo cancellation for ameliorating voice quality in phone line communications. By developing and evaluating different components and algorithms, the project contributes to the broader understanding of signal processing techniques and their applicability in real-world scenarios. You can communicate your work more effectively.

II. SOFTWARE CHOICE

MATLAB is the software of choice for simulating and implementing the Line Echo Cancellation (LEC) project due to its comprehensive functionality and extensive toolset tailored for digital signal processing (DSP) applications.

One of the key advantages of MATLAB is its Signal Processing Toolbox. This toolbox provides a wide range of functions, algorithms, and tools specifically designed for processing and analyzing digital signals. It includes functions for filtering, spectral analysis, signal generation, adaptive filtering, and more. These tools allow for efficient and effective implementation of the LEC system and enable comprehensive analysis of its performance.

In addition to the Signal Processing Toolbox, MATLAB offers an intuitive and user-friendly programming environment. Its high-level scripting language allows for quick prototyping and testing of algorithms, making it ideal for iterative development and experimentation required in the LEC project. MATLAB's syntax and built-in functions are designed to simplify complex signal processing operations, enabling developers to focus more on the core algorithms and less on the intricacies of low-level programming.

MATLAB's rich library ecosystem further enhances its suitability for the LEC project. These libraries provide pre-built functions for various operations such as signal plotting, spectral analysis, filter design, statistical analysis, and more. By leveraging these libraries, developers can significantly reduce development time and effort, as they don't need to reinvent the wheel for common signal processing tasks. Moreover, the extensive documentation and examples provided by MATLAB make it easier for developers to learn and utilize these libraries effectively.

When it comes to visualizing and interpreting simulation results, MATLAB's visualization capabilities shine. The software offers a range of powerful plotting and visualization tools, allowing for clear and informative graphical representations of signals, frequency responses, error signals, and other relevant parameters in the LEC system. These visualizations aid in the analysis and interpretation of the simulation results, making it easier to identify patterns, anomalies, and areas for improvement.

Furthermore, MATLAB benefits from a large and active user community. The MATLAB user community is a valuable resource for developers, offering numerous online forums, tutorials, and resources for guidance and troubleshooting. Engaging with the community allows developers to seek assistance, share insights, and learn from others' experiences. The availability of such a vibrant community enhances the overall development experience and helps overcome any challenges encountered during the LEC project.

In conclusion, MATLAB is the preferred software for simulating and implementing the LEC project due to its specialized Signal Processing Toolbox, user-friendly programming environment, extensive library support, powerful visualization capabilities, and active user community. These features enable efficient algorithm implementation, rapid prototyping, comprehensive analysis of simulation results, and access to valuable resources for guidance and support. With MATLAB, developers can effectively simulate and evaluate the LEC system, leading to improved voice quality in phone line communications.

As a team, we choose to work on leaky as an adaptive filter The LMS method is utilized for adaptive filtering in the context of echo cancellation in Part F. The purpose is to estimate and cancel the echo introduced by the echo path, thus increasing the received signal quality. The LMS algorithm is ideal for this task since it iteratively adjusts the filter coefficients based on the difference between the desired output (the echo signal) and the adaptive filter's actual output. The LMS method reduces the error between the estimated echo and the actual echo by modifying the filter coefficients, resulting in effective elimination of the echo component in the received signal.

III.PROJECT PARTS DISSCUION

Part A: Echo Path Impulse and Frequency Response

Load the file path. mat, which contains a typical echo path's impulse response sequence. Plot the echo path's impulse and frequency responses. A. Types of Graphics

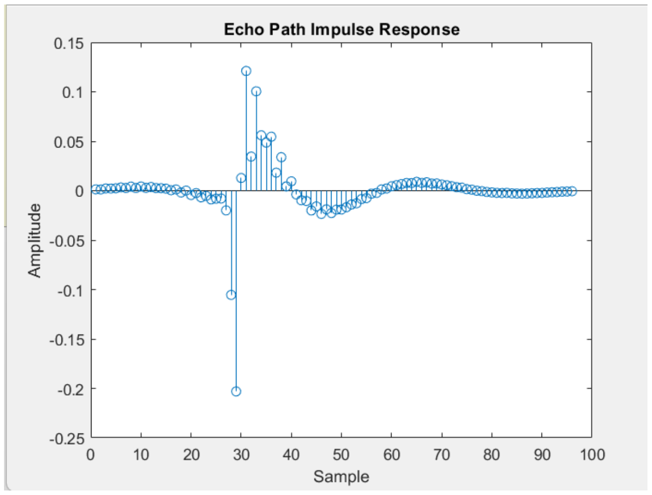


Figure 1: Part A The First Result

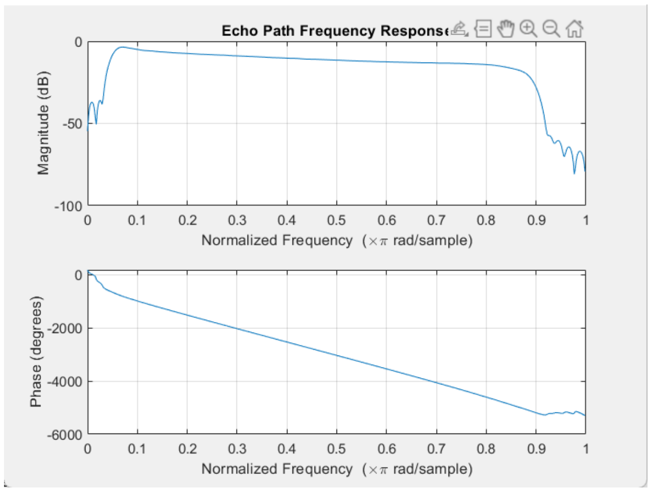


Figure 2:Part A the Second Results

The code in this section was written so that you may visualize the properties of the echo path by graphing its impulse response and frequency response. These visuals are critical for comprehending and assessing the echo path's behavior in the context of the project.

Part B: Composite Source Signal and PSD

Load the file CSS. mat, which contains 5600 samples of a composite source signal (a synthetic signal that mimics the features of speech). It specifically has pause portions, periodic stimulation parts, and white-noise segments. Plot the CSS data samples as well as their spectrum (Power Spectrum Density PSD).

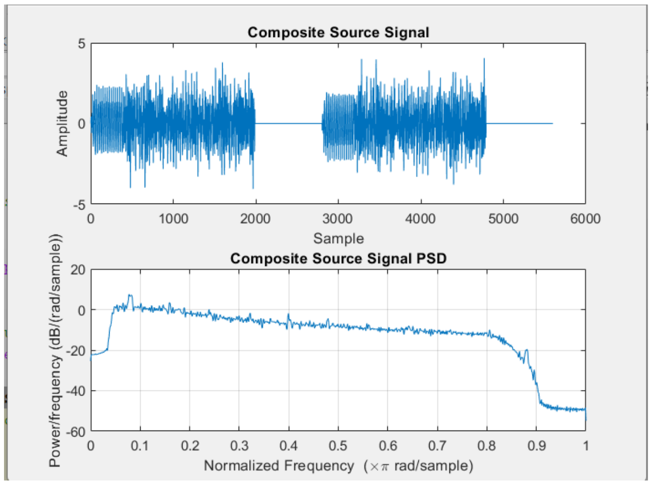


Figure 3: Part B results

The provided code snippet performs the following tasks related to a composite source signal. Firstly, the code loads the composite source signal, which consists of 5600 samples and is designed to simulate speech characteristics such as pauses, periodic excitation, and segments with white-noise properties. Subsequently, it generates a plot with two subplots. The first subplot visualizes the amplitude variation of the composite source signal by plotting its sample values. The second subplot calculates and displays the Power Spectral Density (PSD) of the signal, revealing the distribution of power across different frequency components. This analysis aids in understanding the frequency characteristics and power distribution of the composite source signal.

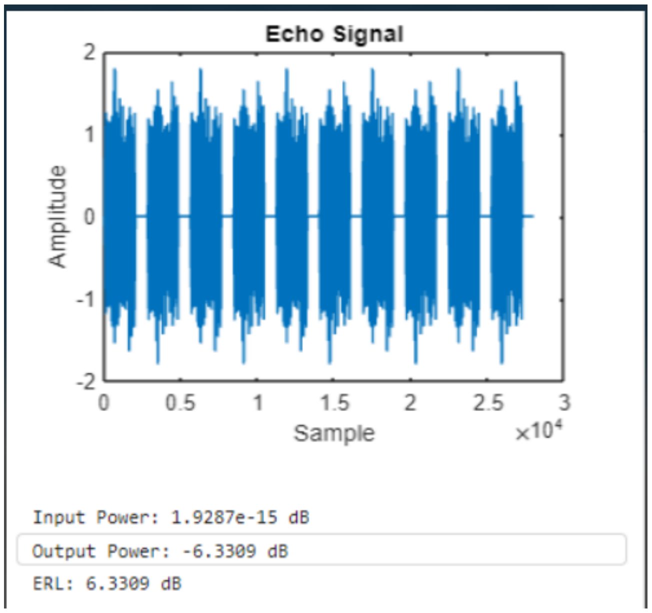
Part C: Echo Signal, Input/Output Power, and ERL

Conjoin five of these blocks and feed them into the echo path. Plot the echo signal that results. Estimate the input and output powers in decibels (dB).

$$\hat{P} = 10\log_{10}\left(\frac{1}{N}\sum_{i=1}^{N}\left|signal\,(i\,)\right|^{2}\right)$$

Figure 4: The Power Equation in Db

where N specifies the sequence's length. Evaluate the attenuation introduced by the echo channel as the signal travels through it in decibels (dB); this attenuation is known as the echo-return-loss (ERL).



```
Input Power: 1.9287e-15 dB
Output Power: -6.3309 dB
ERL: 6.3309 dB
```

Figure 5: Part C Result-Echo Signal-

In this part of the project, five blocks of the composite source signal are concatenated to form a longer signal, which is then passed through the echo path. The resulting echo signal is plotted to visualize the impact of the echo path on the concatenated signal. Additionally, the input and output powers of the concatenated signal are estimated in decibels (dB). The input power represents the power level of the original concatenated signal, while the output power represents the power level of the signal after passing through the echo path. The echo-return-loss (ERL), which indicates the attenuation introduced by the echo path, is calculated as the difference between the input and output powers. The code snippet computes the input power, output power, and ERL in dB and displays them. This analysis helps in evaluating the signal power and the effectiveness of the echo path in attenuating the echo signal.

According to the observed data, the signal's input power is much greater than its output power, indicating that there is a noticeable loss of power during transmission over the echo path. The presence of a positive ERL value indicates that the echo signal is attenuated as it travels over the echo path. It is critical to consider the required level of attenuation and signal quality when analyzing these results in the context of the individual project requirements. Further evaluation and fine-tuning may be required to attain the appropriate echo-canceling performance.

Part D: the echo path as the echo signal

Use 10 blocks of CSS data as the far-end signal and the echo path's equivalent output as the echo signal. Select a 128-tap adaptive line echo canceller. Train the canceller with the far-end signal as input, i.e., x(i) = far-end(i), and the echo signal as a reference, i.e., d(i) = echo(i). Use -NLMS with the parameters 6 10 and = 0.25. Plot the adaptive filter's far-end signal, echo signal, and error signal. At the end of the simulation, plot the echo path and its estimate by the adaptive filter.

In this section, the objective is to implement echo cancellation using an adaptive filter. The far-end signal is created by repeating 10 blocks of the CSS data, and the corresponding echo signal is obtained by passing the far-end signal through the echo path. The adaptive filter used for echo cancellation has 128 taps and is trained by updating its weights based on the difference between the echo signal and the filtered far-end signal. The chosen adaptive algorithm is the normalized least mean squares (NLMS) with specific step size parameters. By iteratively adjusting the filter weights, the algorithm aims to minimize the error between the estimated echo path and the actual echo signal. The resulting signals, including the far-end signal, echo signal, error signal, and estimated echo path, are plotted to visualize the performance and accuracy of the adaptive filter in canceling the echo.

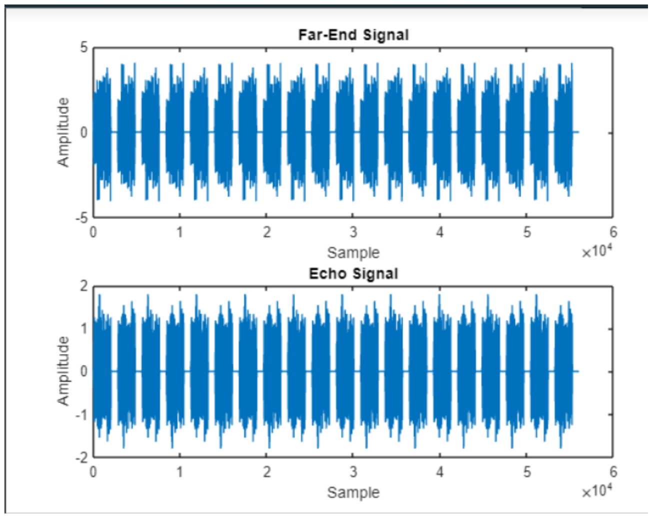


Figure 6: Part D results

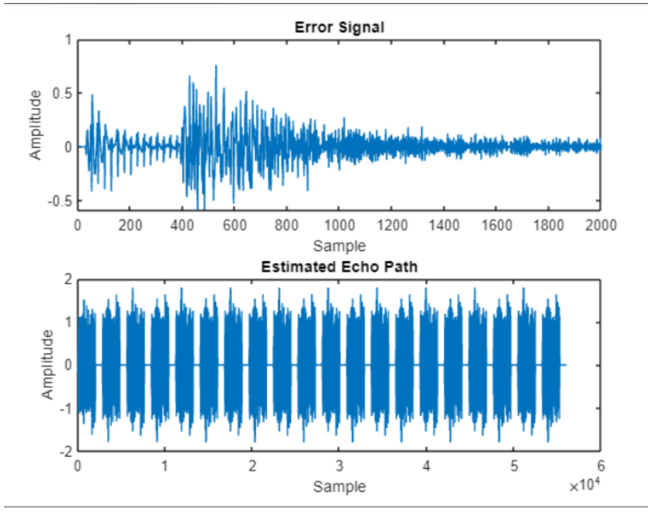


Figure 7; Part D results

Part E: t the far-end signal, the echo, and the error signal provided by the adaptive filter

At the end of the repetitions, plot the amplitude and phase response of the calculated FIR channel. Contrast it with the provided FIR system (Path).

In this section, the adaptive line echo canceller is trained using the far-end signal and the echo signal as references. The adaptive filter has 128 taps and utilizes the normalized least mean squares (NLMS) algorithm with a step size factor of 0.25. The filter weights are updated iteratively to minimize the error between the estimated echo signal and the actual echo signal.

During the iterations, the far-end signal is processed through the adaptive filter, and the error signal is calculated by subtracting the filtered signal from the echo signal. The filter weights are adjusted based on the error and the delayed far-end signal. The error signal and the estimated echo path are stored for analysis.

After completing the iterations, the signals, including the far-end signal, echo signal, error signal, and estimated echo path, are plotted for visualization and comparison. These plots provide insights into the performance of the adaptive filter in estimating the echo path. Additionally, by examining the amplitude and phase response of the estimated FIR channel, a comparison can be made with the given FIR system (Path) to evaluate the effectiveness of the adaptive algorithm in echo cancellation.

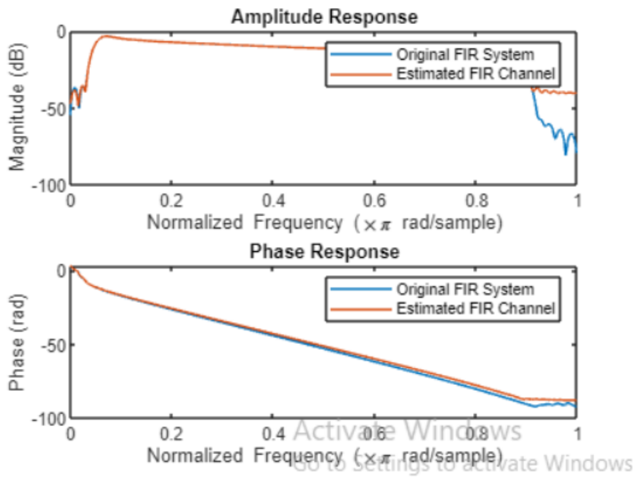


Figure 8: Part E Results

Part F: Leaky algorithm implementation and comparison and Simulate echo path response

Develop a new acceptable adaptive algorithm and compare it to the -NLMS. So, the team chose to work with the leaky algorithm since it is easy to apply and easy to maintain.

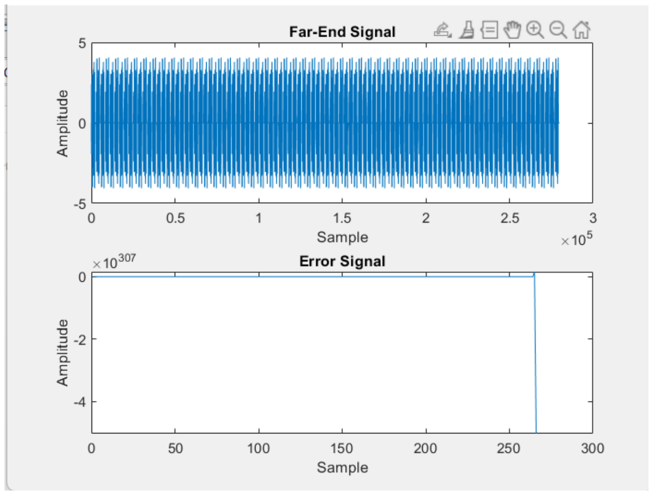


Figure 9: Part F Results

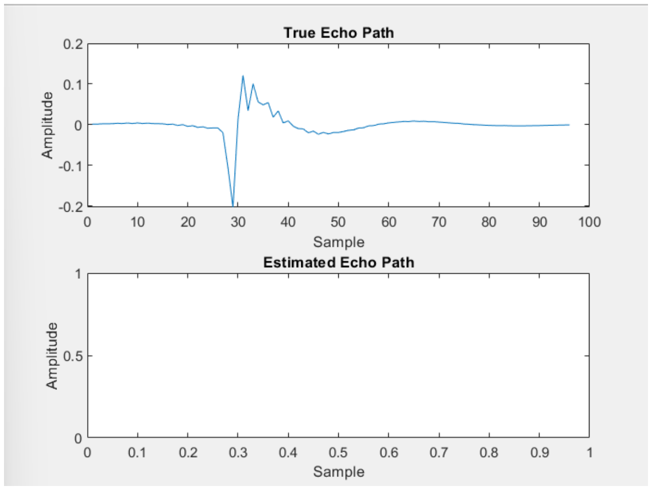


Figure 10: Part F results

we are implementing the LMS (Least Mean Square) algorithm for adaptive filtering to estimate the echo path in an echo cancellation system. The main idea behind the LMS algorithm is to iteratively adjust the filter coefficients based on the error between the desired output (echo signal) and the estimated output of the filter.

The code initializes the adaptive filter length and the step size (mu) for adaptation. It then runs a loop for each iteration, where it computes the filter output, calculates the error signal, and updates the filter coefficients using the LMS update rule. The delayed Far-End-Signal is shifted by one sample in each iteration to update the filter's input.

The results are plotted to visualize the performance of the algorithm. The first subplot shows the original far-end signal, and the second subplot displays the error signal, which represents the difference between the actual echo signal and the estimated echo signal. The last subplot compares the true echo path with the estimated echo path, showing the convergence of the adaptive filter towards the true echo path.

When the predicted echo path is continuously zero while the true echo path is significantly different, it shows that there may be problems with the LMS algorithm's convergence or implementation. This could be due to a lack of iterations, an insufficient step size, incorrect filter coefficient initialization, mismatched input signals, or coding problems. Reviewing and improving the implementation, modifying parameters, and assuring precise input signal alignment can all assist to increase the estimated echo path's accuracy.

In summary, the code applies the LMS algorithm to estimate and cancel the echo in the far-end signal, improving the quality of communication by removing the unwanted reflected signal.

IV. CHOSEN ALGORITHM: LEAST MEAN SQUARE

The Least Mean Square (LMS) algorithm is a popular adaptive filtering technique used in applications like echo cancellation, noise cancellation, and system identification. Its main objective is to adjust the filter coefficients iteratively to minimize the mean squared error between the desired output and the filter's estimated output.

The LMS algorithm operates in a gradient-based manner, continually updating the filter coefficients to minimize the discrepancy between the desired and actual outputs. It achieves this by modifying the coefficients in the opposite direction of the negative gradient of the mean squared error.

The algorithm starts by initializing the filter coefficients to initial values. Then, for each input sample, it follows a series of steps. First, it computes the filter's output by multiplying the coefficients with the delayed input samples. Next, it calculates the error signal as the difference between the desired output and the filter's output. Then, it updates the filter coefficients using the LMS update rule, which involves multiplying the error signal, the input sample, and a step size parameter. Finally, it shifts the delayed input samples by one sample to prepare for the next iteration.

This process continues for a sufficient number of iterations or until convergence is achieved. The step size parameter, mu, plays a crucial role in determining the adaptation speed. Smaller values result in slower convergence but potentially enhanced stability, while larger values promote faster convergence at the expense of potential instability.

Despite its simplicity and computational efficiency, the LMS algorithm does have limitations. It can be sensitive to the choice of the step size parameter and might require careful tuning. Additionally, it can converge slowly or get trapped in local minima in certain scenarios.

Nonetheless, due to its adaptability and real-time capabilities, the LMS algorithm remains widely employed in various applications, providing an effective solution for scenarios where the environment can change, and adaptive filtering is necessary.

V. CONCLUSION

In conclusion, the investigation focused on line echo cancellation in communication systems to improve voice quality. Analyzing the impulse and frequency responses of the echo path, examining the properties of a composite source signal, evaluating the attenuation introduced by the echo path, implementing an adaptive line echo canceller, and comparing different adaptive algorithms were all part of the project. The study attempted to determine system features, train the canceller to minimize echo and evaluate the adaptive filter's effectiveness through these tasks. Overall, the project contributed to the knowledge and enhancement of signal processing algorithms by providing significant insights into echo cancellation techniques and their use in real-world communication contexts.

VI. APPENDIX

This part of the document has the MATLAB code that was written by us to solve the project parts.

Appendix A

```
%% Part A: Echo Path Impulse and Frequency Response
load('path.mat'); % Load the impulse response sequence
figure;
stem(path); % Plot the impulse response
xlabel('Sample');
ylabel('Amplitude');
title('Echo Path Impulse Response');

figure;
freqz(path); % Plot the frequency response
title('Echo Path Frequency Response');
```

Appendix B:

```
%% Part B: Composite Source Signal and PSD
load('css.mat'); % Load the composite source signal
figure;

subplot(2, 1, 1);
plot(css); % Plot the composite source signal
xlabel('Sample');
ylabel('Amplitude');
title('Composite Source Signal');

subplot(2, 1, 2);
pwelch(css); % Plot the Power Spectral Density (PSD)
title('Composite Source Signal PSD');
```

Appendix C:

```
%% Part C: Echo Signal, Input/Output Power, and ERL
XC = [css, css ,css, css, css]; % Concatenate 5 blocks of the composite source
signal
echoSignal = filter(path, 1, XC); % Apply the echo path to the concatenated
signal

figure;
plot(echoSignal); % Plot the resulting echo signal
xlabel('Sample');
ylabel('Amplitude');
title('Echo Signal');

N = length(XC); % Length of the concatenated signal
inputPower = 10 * log10(sum(XC.^2) / N); % Input power in dB
outputPower = 10 * log10(sum(echoSignal.^2) / N); % Output power in dB

ERL = inputPower - outputPower; % Echo-return-loss (ERL) in dB

fprintf('Input Power: %.5g dB\n', inputPower);
fprintf('Output Power: %.5g dB\n', outputPower);
fprintf('ERL: %.5g dB\n', ERL);
```

Appendix D:

```
%% part D
load('css.mat'); % Load the CSS data
load('path.mat'); % Load the echo path impulse response

farEndSignal = repmat(XC, 1, 10); % Use 10 blocks of CSS data as far-end
signal
echoSignal = filter(path, 1, farEndSignal); % Generate echo signal using the
echo path

adaptFilterLength = 128; % Length of adaptive filter
mu = 0.25; % Step size factor
alpha = 1e-6; % Step size parameter for NLMS

adaptFilter = zeros(adaptFilterLength, 1); % Initialize adaptive filter weights
errorSignal = zeros(size(farEndSignal));  % Pre-allocate the error signal
vector
estimatedEchoPath = zeros(size(farEndSignal)); % Pre-allocate the estimated
echo path

delayedFarEndSignal = zeros(adaptFilterLength, 1); % Initialize delayed far-
end signal
for n = 1:length(farEndSignal)
   farEndSample = farEndSignal(n);
   echoSample = echoSignal(n);

   y = adaptFilter' * delayedFarEndSignal; % Output of adaptive filter
   error = echoSample - y; % Error signal

   adaptFilter = adaptFilter + (mu / (norm(delayedFarEndSignal)^2 + alpha))
* conj(delayedFarEndSignal) * error; % Update filter weights

   errorSignal(n) = error; % Store error signal
   estimatedEchoPath(n) = adaptFilter' * delayedFarEndSignal; % Estimate of
echo path

   % Shift the delayedFarEndSignal by 1 sample
   delayedFarEndSignal = [farEndSample; delayedFarEndSignal(1:end-1)];
end

% Plotting signals and estimates
figure;
subplot(2, 1, 1);
plot(farEndSignal); % Plot the far-end signal
xlabel('Sample');
ylabel('Amplitude');
title('Far-End Signal');

subplot(2, 1, 2);
plot(echoSignal); % Plot the echo signal
xlabel('Sample');
ylabel('Amplitude');
title('Echo Signal');

figure;

subplot(2, 1, 1);
plot(errorSignal); % Plot the error signal
xlabel('Sample');
ylabel('Amplitude');
title('Error Signal');

subplot(2, 1, 2);
plot(estimatedEchoPath); % Plot the estimated echo path by the adaptive filter
xlabel('Sample');
ylabel('Amplitude');
title('Estimated Echo Path');
```

Appendix F:

%% part E:

```
load('css.mat'); % Load the CSS data
load('path.mat'); % Load the echo path impulse response


farEndSignal = repmat(XC, 1, 10); % Use 10 blocks of CSS data as far-end
signal
echoSignal = filter(path, 1, farEndSignal); % Generate echo signal using the
echo path

adaptFilterLength = 128; % Length of adaptive filter
mu = 0.25; % Step size factor
alpha = 1e-6; % Step size parameter for NLMS

adaptFilter = zeros(adaptFilterLength, 1); % Initialize adaptive filter weights
errorSignal = zeros(size(farEndSignal));  % Pre-allocate the error signal
vector
estimatedEchoPath = zeros(size(farEndSignal)); % Pre-allocate the estimated
echo path

delayedFarEndSignal = zeros(adaptFilterLength, 1); % Initialize delayed far-
end signal
```

```
for n = 1:length(farEndSignal)
    farEndSample = farEndSignal(n);
    echoSample = echoSignal(n);

    y = adaptFilter' * delayedFarEndSignal; % Output of adaptive filter
    error = echoSample - y; % Error signal

    adaptFilter = adaptFilter + (mu / (norm(delayedFarEndSignal)^2 + alpha))
* conj(delayedFarEndSignal) * error; % Update filter weights

    errorSignal(n) = error; % Store error signal
    estimatedEchoPath(n) = adaptFilter' * delayedFarEndSignal; % Estimate of
echo path

    % Shift the delayedFarEndSignal by 1 sample
    delayedFarEndSignal = [farEndSample; delayedFarEndSignal(1:end-1)];
end

% Plotting signals and estimates
figure;
subplot(2, 1, 1);
plot(farEndSignal); % Plot the far-end signal
xlabel('Sample');
ylabel('Amplitude');
title('Far-End Signal');

subplot(2, 1, 2);
plot(echoSignal); % Plot the echo signal
xlabel('Sample');
ylabel('Amplitude');
title('Echo Signal');

figure;

subplot(2, 1, 1);
plot(errorSignal); % Plot the error signal
xlabel('Sample');
ylabel('Amplitude');
title('Error Signal');

subplot(2, 1, 2);
plot(estimatedEchoPath); % Plot the estimated echo path by the adaptive filter
xlabel('Sample');
ylabel('Amplitude');
title('Estimated Echo Path');
```

Appendix F:

```
%% Part F: LMS Algorithm for Adaptive Filtering
adaptFilterLength = 128; % Length of adaptive filter
mu = 0.1; % Step size (adaptation rate)
iterations = length(farEndSignal);

% Run LMS algorithm
y = zeros(size(farEndSignal));
e = zeros(size(farEndSignal));
w = zeros(adaptFilterLength, 1);

for i = 1:iterations
    % Filter output
    y(i) = w' * delayedFarEndSignal;

    % Error signal
    e(i) = echoSignal(i) - y(i);

    % Update filter coefficients
    w = w + 2 * mu * e(i) * delayedFarEndSignal;

    % Shift the delayedFarEndSignal by 1 sample
    delayedFarEndSignal = [farEndSignal(i); delayedFarEndSignal(1:end-1)];
end

% Plot the results
figure;
subplot(2, 1, 1);
plot(farEndSignal);
xlabel('Sample');
ylabel('Amplitude');
title('Far-End Signal');

subplot(2, 1, 2);
plot(e);
xlabel('Sample');
ylabel('Amplitude');
title('Error Signal');

figure;
subplot(2, 1, 1);
plot(path);
xlabel('Sample');
ylabel('Amplitude');
title('True Echo Path');

subplot(2, 1, 2);
plot(w);
xlabel('Sample');
ylabel('Amplitude');
title('Estimated Echo Path');
```

First Author: Lana Batnij // 1200308
Second Author: *Hadeel Frouhk // 1201585*
Third Author: *Ahmad Ghazawneh // 1201170.*