University of Pennsylvania

School of Engineering and Applied Science

Department of Electrical and Systems Engineering

ESE 5310 — Spring 2025

Digital Signal Processing

Project 2

Adaptive Filtering

Hesham Nabil Hosny Maher 58309713

Submission Date: Monday, April 28, 2025

Contents

1 Part A1: Adaptive Notch Filter for Interference Cancellation

The goal of this part is to design an adaptive notch filter capable of suppressing an unknown sinusoidal interference contaminating a desired signal. The filter should adaptively locate and eliminate the interference frequency without prior knowledge.

The notch filter is modeled using a second-order Infinite Impulse Response (IIR) structure defined by

$$H(z) = \frac{1 + az^{-1} + z^{-2}}{1 + raz^{-1} + r^2 z^{-2}}$$

where $a = -2\cos(\omega_0)$ defines the notch center frequency ω_0 and r controls the sharpness of the notch. A larger r places the poles closer to the unit circle, creating a sharper and deeper notch.

An adaptive algorithm is required to track the unknown interference frequency. The Least Mean Squares (LMS) method is used to update the parameter $a[n]$ iteratively as follows:

$$a[n+1] = a[n] - \mu\, y[n]\, x[n-1]$$

where μ is a small positive step size, $y[n]$ is the filter output, and $x[n-1]$ is the delayed input sample. The goal is to minimize the output power $y[n]^2$, forcing the notch to adapt to the interference. The true target value of a corresponding to the interference frequency f_{noise} is given by

$$a_{\text{true}} = -2\cos\left(2\pi \frac{f_{\text{noise}}}{f_s}\right)$$

where f_s is the sampling frequency.

The following parameters were used during simulation:

- Sampling frequency: $f_s = 200$ Hz | Desired signal frequency: $f_{\text{signal}} = 3$ Hz
- Desired signal amplitude: 1 | Interference amplitude: 10
- Number of samples: 500,000

Several experiments were performed to analyze the filter behavior:

- **Experiment 1**: $\mu = 10^{-6}$, $r = 0.85$, $f_{\text{noise}} = 30$ Hz
- **Experiment 2**: $\mu = 10^{-5}$, $r = 0.85$, $f_{\text{noise}} = 30$ Hz (Changing μ)
- **Experiment 3**: $\mu = 10^{-6}$, $r = 0.95$, $f_{\text{noise}} = 30$ Hz (Changing r)
- **Experiment 4**: $\mu = 10^{-6}$, $r = 0.85$, $f_{\text{noise}} = 40$ Hz (Changing f_{noise})

For each configuration, the desired and interference signals were summed to form the input, and the LMS algorithm was applied to adaptively update $a[n]$ over time.

Experiment 1:

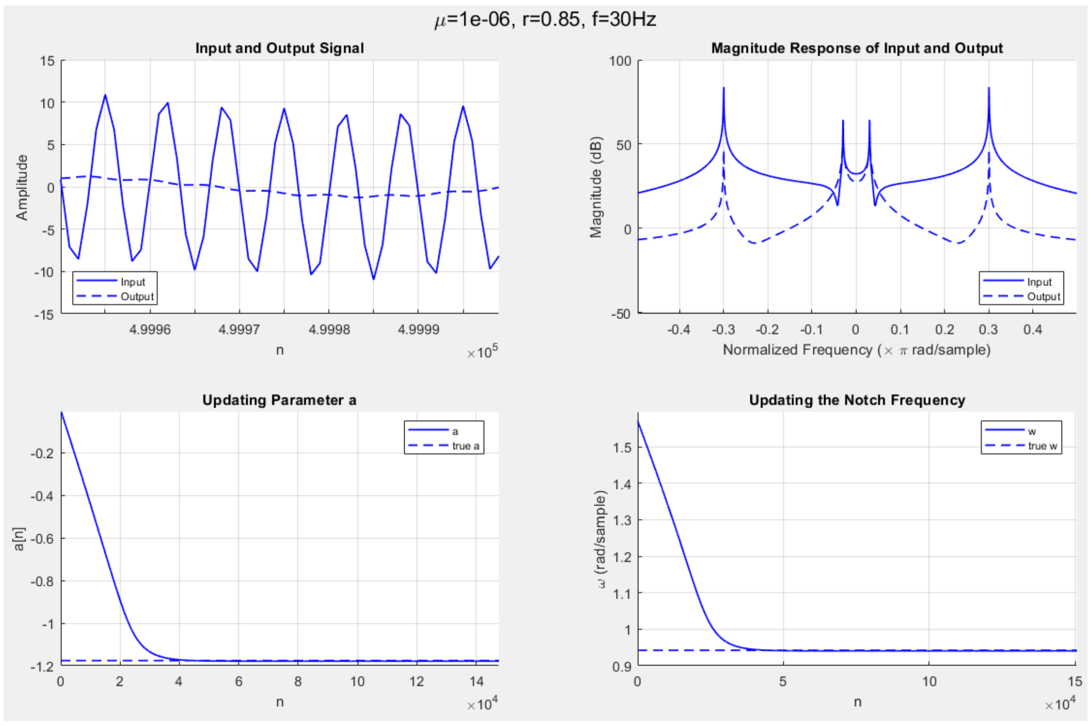

Figure 1: Experiment 1: $\mu = 10^{-6}$, $r = 0.85$, $f_{\text{noise}} = 30$ Hz

Experiment 2:

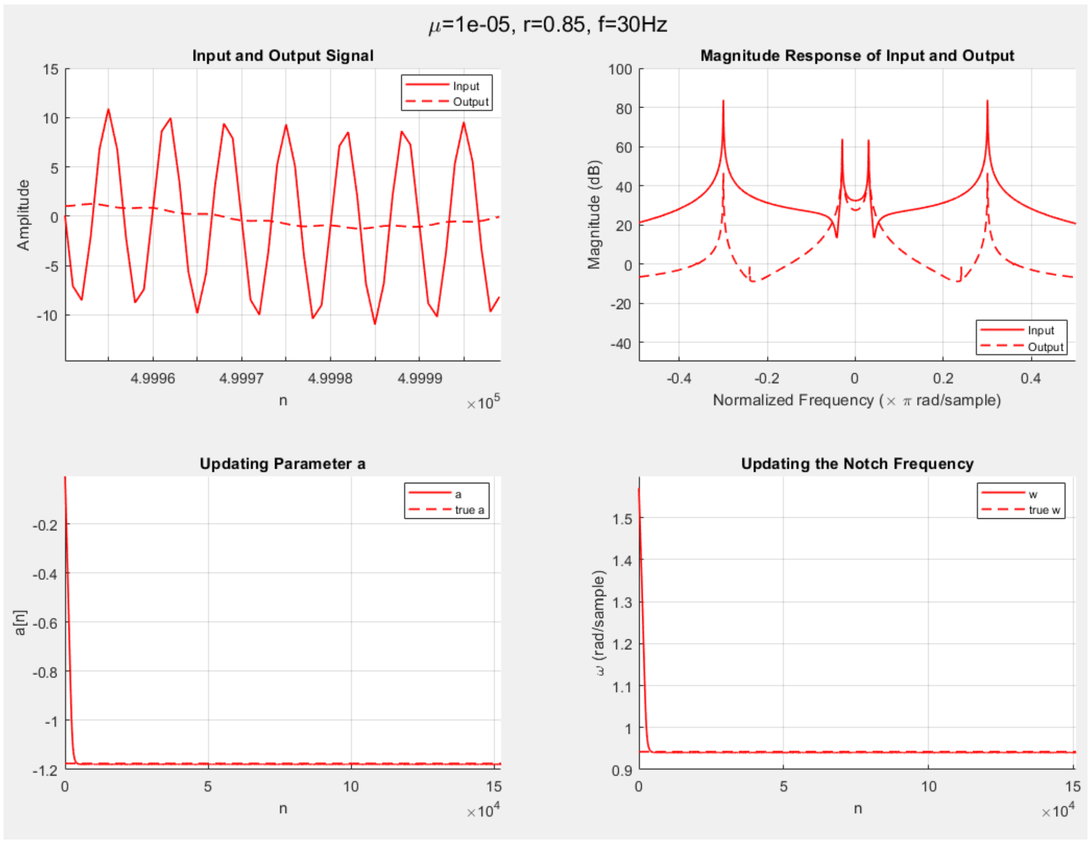

Figure 2: Experiment 2: $\mu = 10^{-5}$, $r = 0.85$, $f_{\text{noise}} = 30$ Hz

Experiment 3:

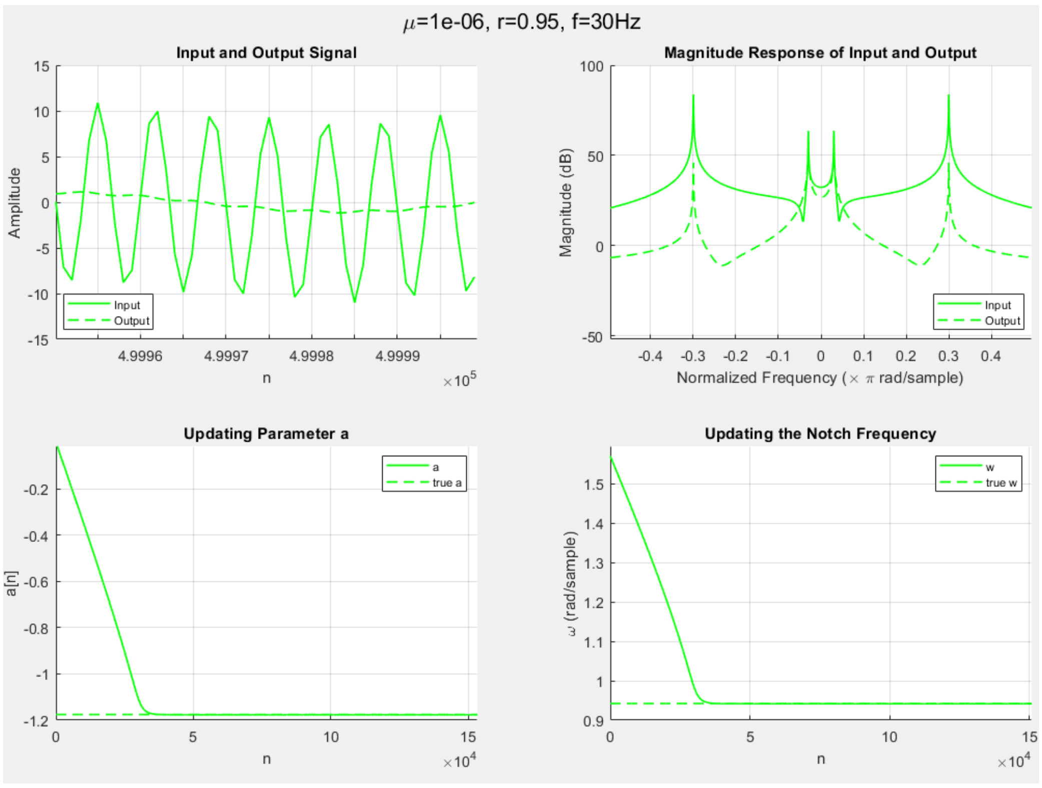

Figure 3: Experiment 3: $\mu = 10^{-6}$, $r = 0.95$, $f_{\text{noise}} = 30$ Hz

Experiment 4:

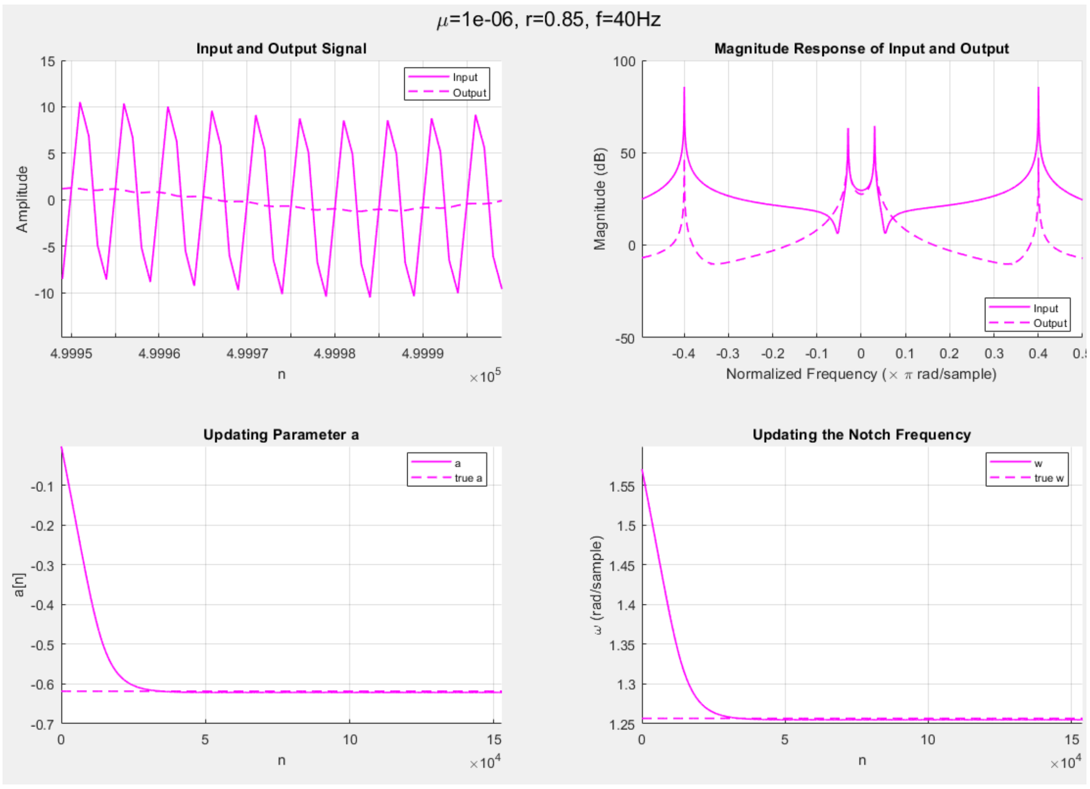

Figure 4: Experiment 4: $\mu = 10^{-6}$, $r = 0.85$, $f_{\text{noise}} = 40$ Hz

From the results of the four experiments, it is evident that the adaptive notch filter is functioning correctly and robustly. In all cases, the filter successfully tracked the unknown interference frequency without prior information and adjusted its notch location adaptively over time. The experiments demonstrated that with appropriate tuning of the step size μ and pole radius r, the filter can achieve stable convergence, rapid adaptation when needed, and effective suppression of the interfering sinusoid. Furthermore, the ability of the filter to track changes in the interference frequency, as seen in the fourth experiment, highlights its flexibility and reliability in dynamic environments. Overall, the adaptive algorithm consistently minimized the output power and accurately placed the notch at the interference frequency, validating the theoretical design.

Overall, the results confirm that:

- Smaller μ leads to slower but more stable convergence, while larger μ speeds up convergence at the cost of stability.
- Increasing r creates a sharper, deeper notch for more selective interference cancellation.
- The LMS-based adaptive notch filter effectively tracks different interference frequencies without prior knowledge, ensuring robust interference cancellation.

2 Part A2: Adaptive Notch Filter Tracking a Slowly Changing Frequency

After validating the adaptive notch filter under fixed interference conditions, we next examine its ability to track a sinusoidal interference whose frequency changes slowly over time. In this experiment, the interference frequency was designed to vary linearly from 30 Hz to 40 Hz across the simulation duration.

The input signal consisted of the original desired 3 Hz sinusoid combined with a strong time-varying interference component. The instantaneous phase of the interference was generated by integrating the varying frequency, ensuring a smooth and continuous sweep. The LMS-based adaptive algorithm was then applied to dynamically update the filter parameter $a[n]$ in real time as the interference frequency changed.

The goal of this part is to assess whether the adaptive filter can continuously adjust its notch location to match the slowly drifting interference without prior knowledge of the changing frequency profile.

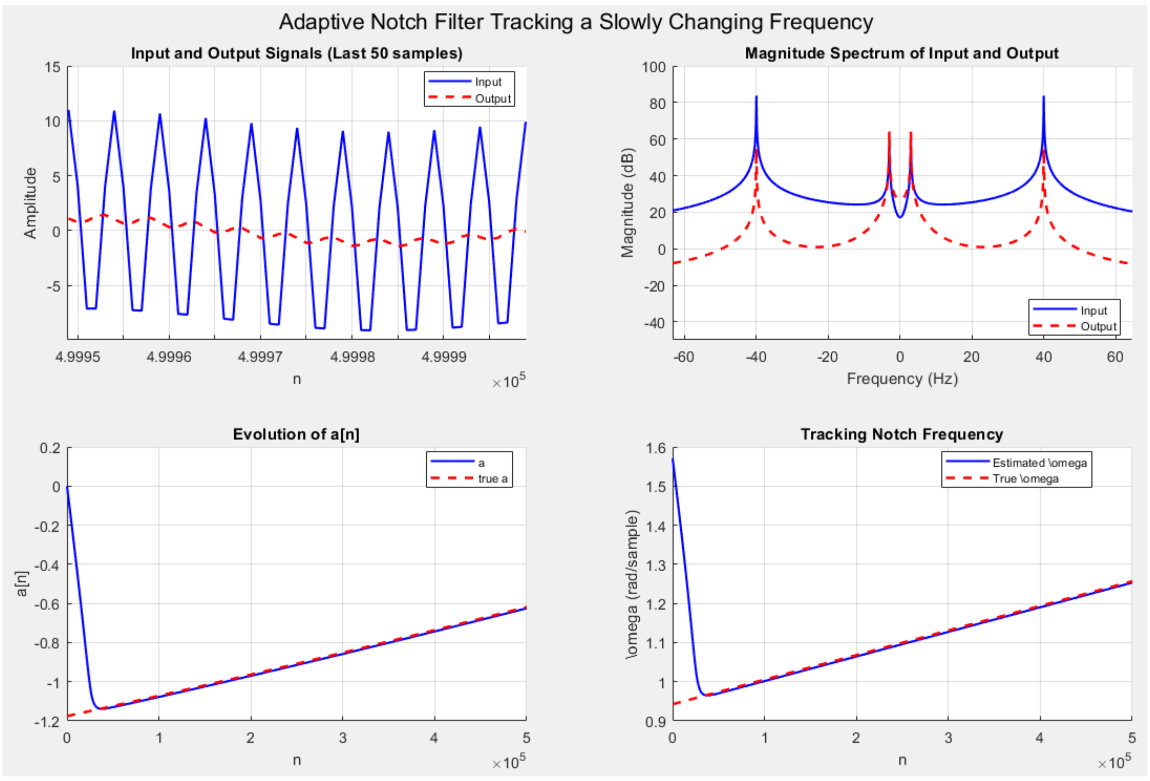

Figure 5: Adaptive notch filter performance while tracking a slowly changing interference frequency from 30 Hz to 40 Hz.

The top-left plot shows the input and output signals over the last 50 samples, highlighting the filter's effectiveness in attenuating the interference even near the end of the sweep. The top-right plot displays the magnitude spectra, where the output spectrum clearly shows strong suppression of the varying interference across a wide frequency band. The bottom-left plot presents the evolution of the adaptive parameter $a[n]$ as it smoothly transitions to follow the new interference conditions. The bottom-right plot compares the estimated notch frequency $\omega[n]$ to the true interference frequency over time, confirming precise and consistent tracking.

Overall, the results demonstrate that the adaptive notch filter not only converges effectively under static interference, but also robustly tracks slow frequency variations. The LMS-based adaptation successfully maintains suppression performance even in the presence of gradual environmental changes.

3 Part A3: Adaptive Notch Filter for Two Sinusoidal Interferences

In this part, we extend the adaptive notch filter design to simultaneously cancel two unknown sinusoidal interferences present in the desired signal. The idea is to cascade two second-order notch filters, each tuned adaptively to one interference frequency. This structure allows the system to independently track and eliminate each interfering tone without prior frequency knowledge.

The transfer function of each individual notch filter remains

$$H(z) = \frac{1 + az^{-1} + z^{-2}}{1 + raz^{-1} + r^2 z^{-2}}$$

where a is updated adaptively to center the notch at the correct interference frequency, and r determines the notch sharpness.

The cascade structure is organized as follows: the input signal $x[n]$ passes first through the first adaptive notch filter, producing an intermediate signal $y_1[n]$. This output is then fed into the second adaptive notch filter, resulting in the final output $y_2[n]$. Each stage has its own independent LMS adaptation loop to update the corresponding parameter $a_1[n]$ and $a_2[n]$ separately.

Cascade Structure:

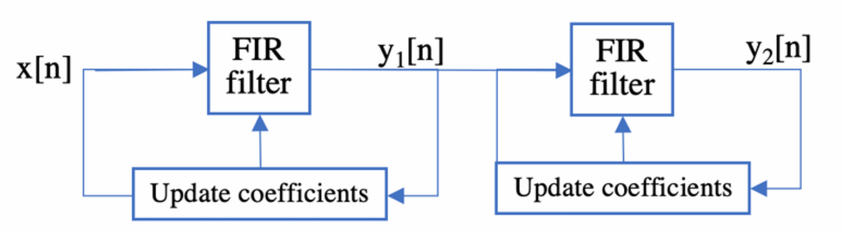

Figure 6: Cascade structure for two adaptive notch filters.

In this experiment, the desired signal is a low-frequency sinusoid at 5 Hz, contaminated by two interfering sinusoids at 45 Hz and 60 Hz respectively. The system is tasked with canceling both interferences adaptively using two cascaded adaptive notch filters. The parameters used were:

- Sampling frequency: $f_s = 200$ Hz
- Desired signal frequency: $f_{\text{signal}} = 5$ Hz
- Interference frequencies: $f_{\text{noise1}} = 45$ Hz, $f_{\text{noise2}} = 60$ Hz
- Pole radius: $r = 0.95$
- Step size: $\mu_1 = 10^{-5}$ for the first filter, $\mu_2 = 2 \times 10^{-6}$ for the second filter

The LMS update equations for each filter are applied independently based on the corresponding filter output and input sample history. Special care was taken to reduce the step size for the second filter, as the residual signal after the first notch filter becomes weaker, affecting the adaptation dynamics.

Figure 7 summarizes the performance of the cascaded structure. The top-left subplot shows the input and output signals over the last 50 samples, demonstrating substantial interference cancellation. The top-right subplot presents the magnitude spectra of the input and output signals, where the interference components are strongly suppressed. The bottom-left subplot illustrates the evolution of the adaptive parameters $a_1[n]$ and $a_2[n]$ compared to their theoretical true values. The bottom-right subplot shows the estimated notch frequencies $\omega_1[n]$ and $\omega_2[n]$ versus their true target frequencies, confirming successful convergence.

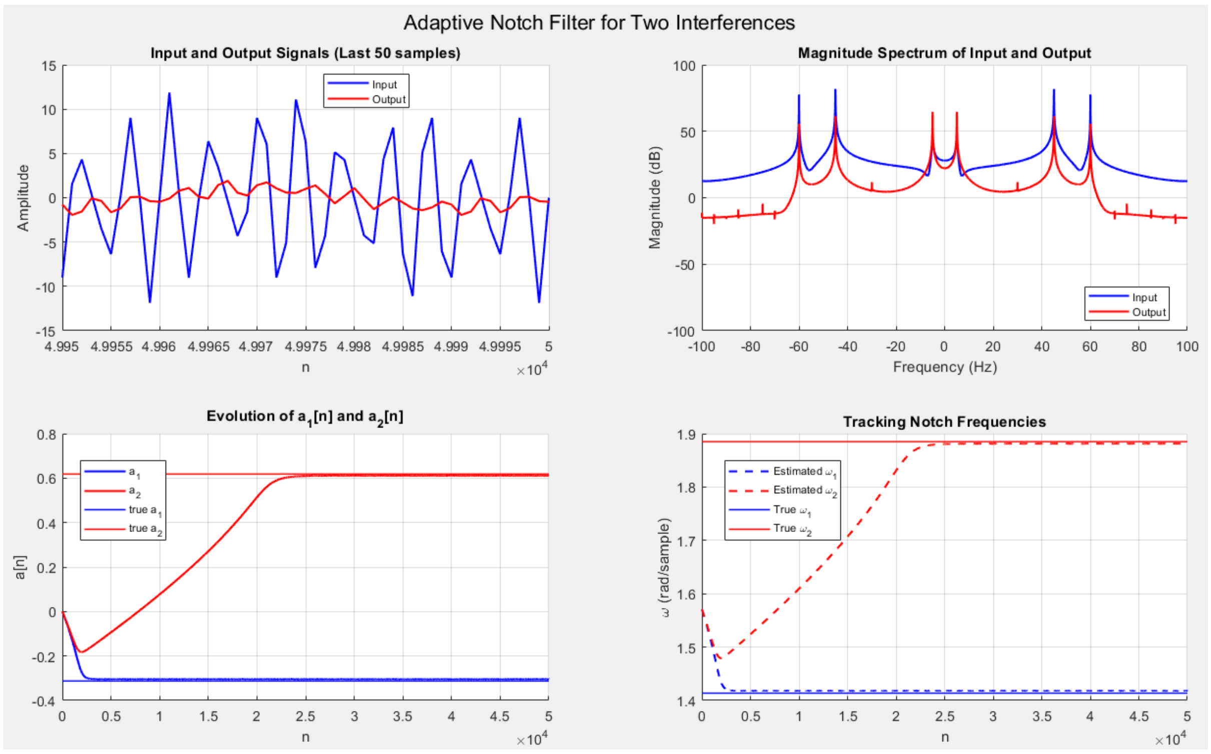

Figure 7: Performance of the cascaded adaptive notch filter tracking two fixed interference frequencies.

Figure 8 provides additional insight by displaying the magnitude response at different stages of filtering. It highlights the progressive attenuation of interference tones after each notch filter stage, and clearly visualizes how $y_1[n]$ and $y_2[n]$ progressively clean the signal from interference components.

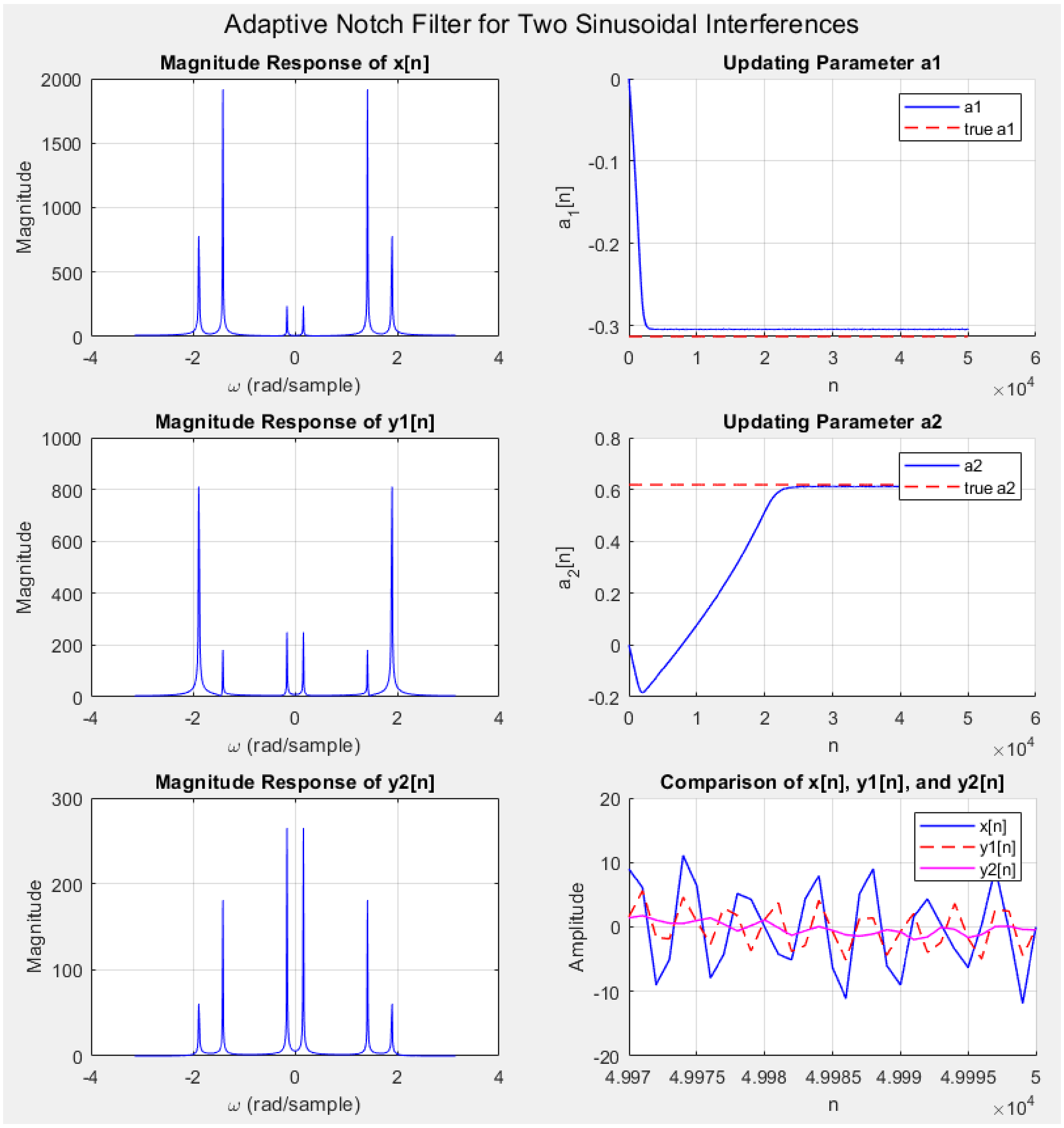

Figure 8: Magnitude response of $x[n]$, $y_1[n]$, and $y_2[n]$ along with the evolution of adaptive parameters.

4 Part B: Adaptive Equalization

Theory and Design

Data transmission over physical channels—such as copper cables, fiber optics, or wireless mediums—can cause signal distortion, especially over long distances. These distortions lead to **inter-symbol interference (ISI)**, where multiple transmitted symbols interfere with each other, degrading the received signal quality. To counteract ISI, **adaptive equalizers** are used at the receiver to learn and invert the unknown channel distortion without requiring prior knowledge of the channel characteristics.

The basic structure of the adaptive equalization system is depicted in Figure 9, where the transmitted signal $s[n]$ passes through an unknown channel and noise $w[n]$ is added. The received signal $x[n]$ is then processed by the adaptive filter to produce the estimate $y[n]$.

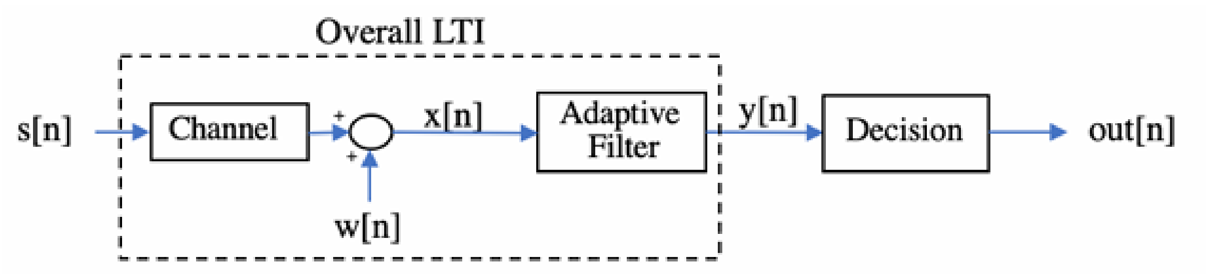

Figure 9: Basic adaptive equalization system.

A more detailed view of the internal adaptive filtering process is shown in Figure 10. This schematic illustrates how the LMS algorithm iteratively updates the filter coefficients based on the error signal.

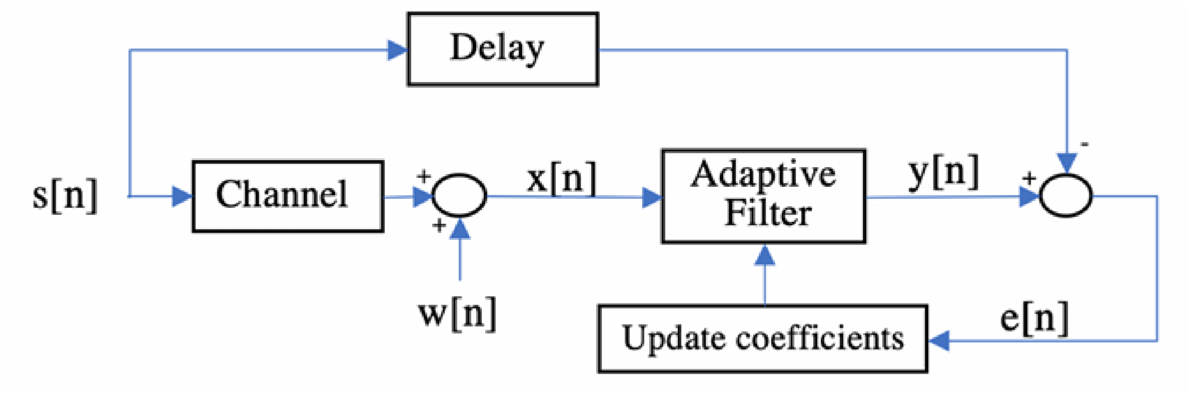

Figure 10: Detailed LMS-based adaptive equalizer structure.

The received signal model is:

$$x[n] = h_{\text{channel}}[n] * s[n] + w[n]$$

where $h_{\text{channel}}[n] = [0.3,\ 1,\ 0.7,\ 0.3,\ 0.2]$ represents the unknown channel impulse response, and $w[n]$ is zero-mean additive Gaussian noise.

The adaptive filter attempts to recover $s[n]$ by producing:

$$y[n] = x[n] * h[n]$$

The error signal is computed as:

$$e[n] = s[n - n_d] - y[n]$$

where n_d is the appropriate delay to align the transmitted and received signals, accounting for channel and filter delay.

The filter taps are updated using the **Least Mean Squares (LMS)** algorithm:

$$h_{n+1} = h_n + 2\mu e[n] x_n^*$$

where:

- μ is the LMS step size (controls adaptation speed and stability)
- $x_n^* = [x[n], x[n-1], x[n-2], \ldots, x[n-M]]^T$ is the reversed window of past $M+1$ samples

By minimizing the mean squared error (MSE), the adaptive equalizer converges to an inverse model of the channel, effectively canceling ISI and recovering the transmitted signal.

Simulation Setup

The parameters for simulation are summarized below:

Parameter	**Value**
Channel Impulse Response	[0.3, 1, 0.7, 0.3, 0.2]
Adaptive Filter Order M	11
Step Size μ	0.01
Initial Coefficients	All ones
Delay n_d	8 samples
SNR	25 dB (initial), varied later

Table 1: Simulation parameters used in the adaptive filter setup.

Results and Analysis

Training Performance During the training phase, the filter learns to compensate for the distortion. Figure 11 shows the convergence of the error signal and the comparison between the input and equalizer output.

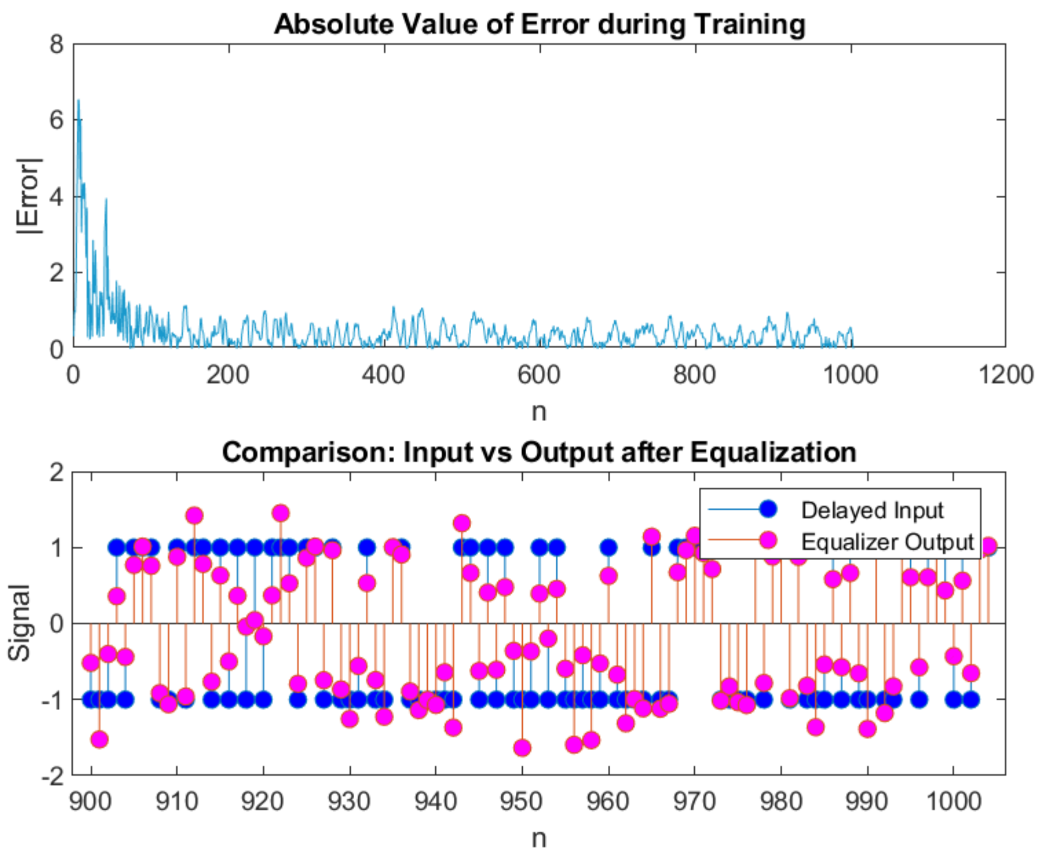

Figure 11: Training error (top) and equalizer output vs delayed input (bottom).

Tap Convergence : The convergence of the 12 adaptive taps is shown in Figure 12. These taps adjust dynamically until the optimal inverse filter is approximated.

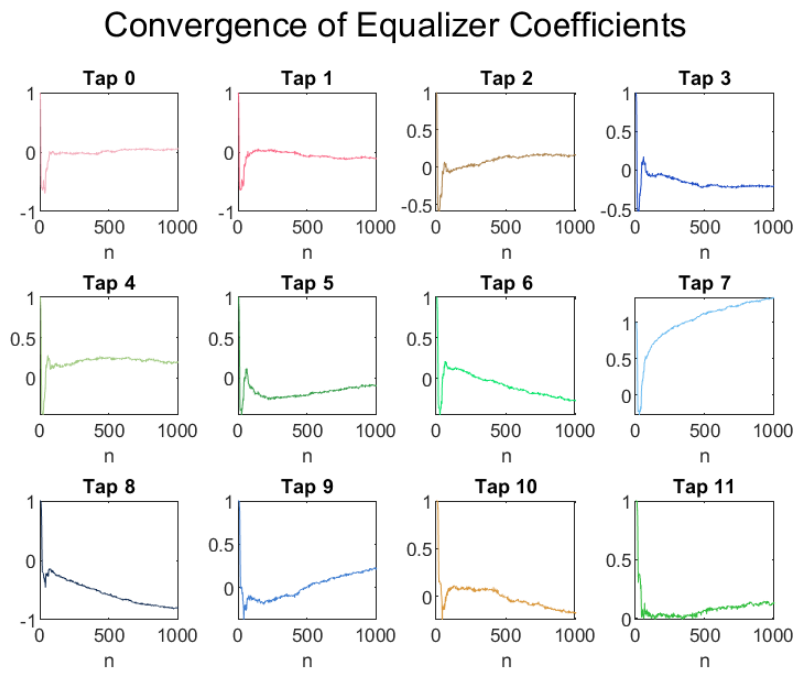

Figure 12: Evolution of equalizer filter coefficients during training.

Overall LTI System : To visualize the convergence of the combined system (channel + equalizer), we observe the convolution result in Figure 13.

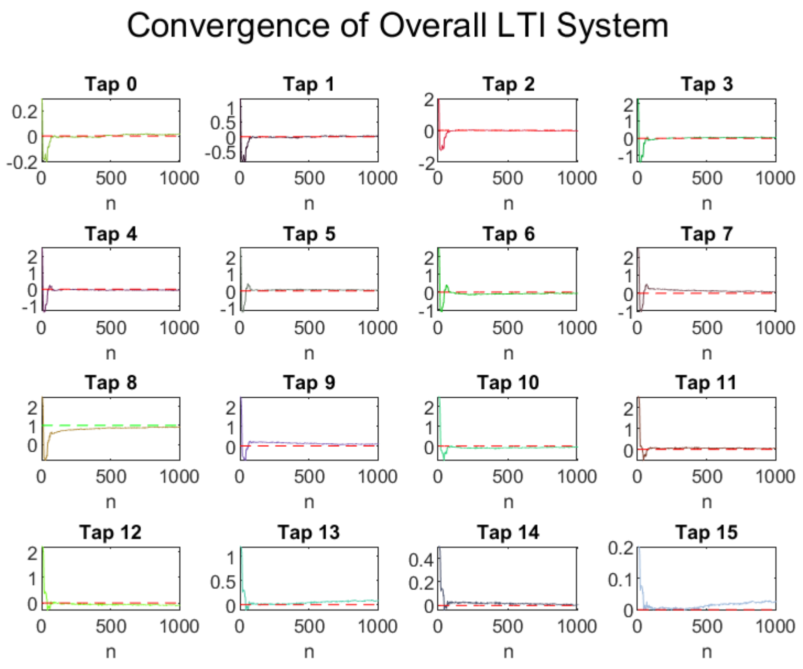

Figure 13: Convergence of the overall LTI system (equalizer * channel).

Frequency and Impulse Responses The frequency and impulse responses of the channel, equalizer, and full system are provided in Figures 14 and 28.

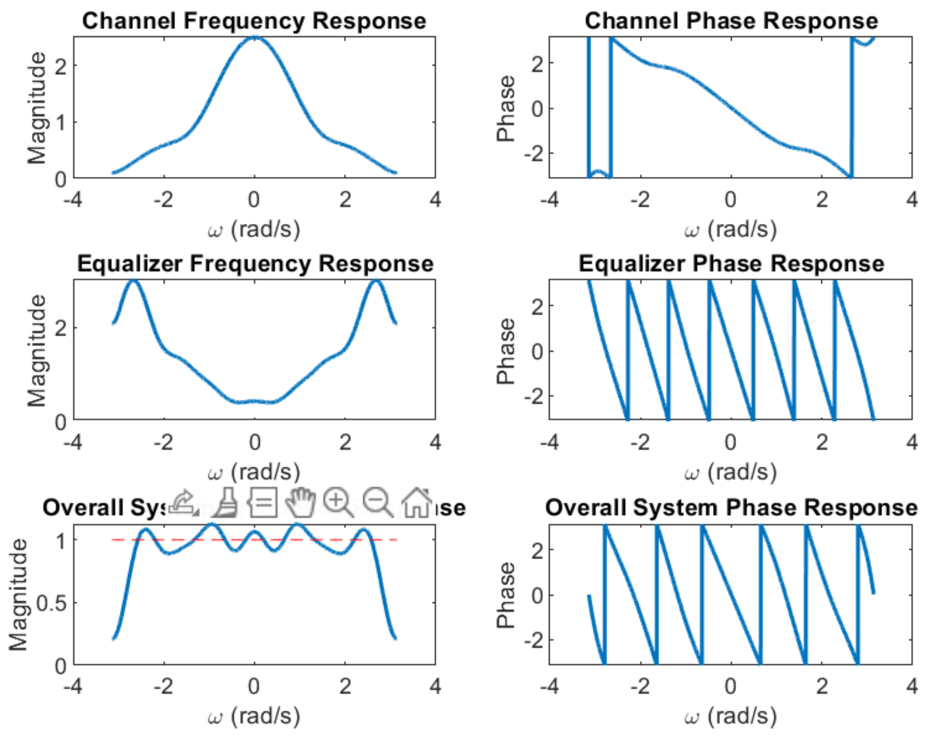

Figure 14: Frequency responses: channel, equalizer, and overall system.

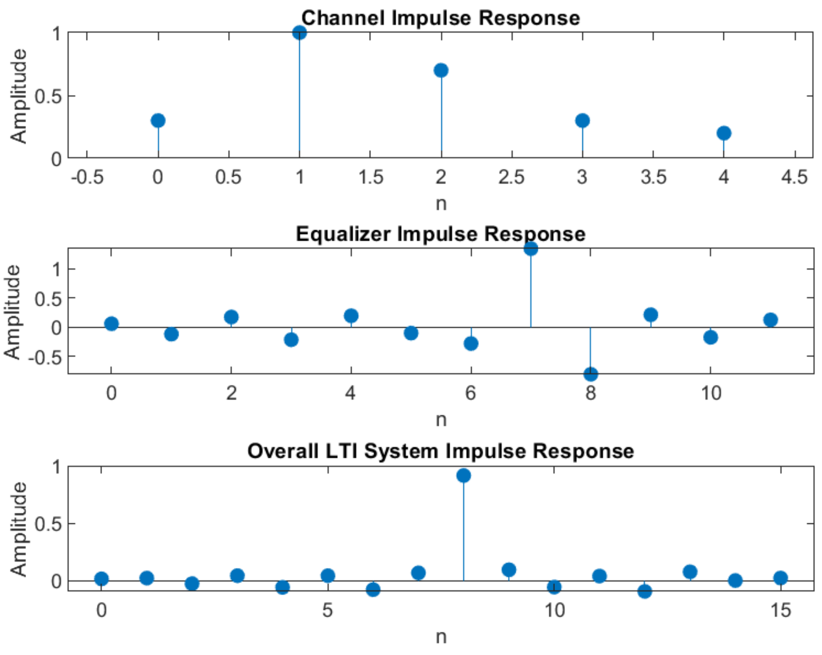

Figure 15: Impulse responses: channel, equalizer, and overall system.

Pole-Zero Analysis To further understand the system characteristics, pole-zero plots are shown in Figure 16.

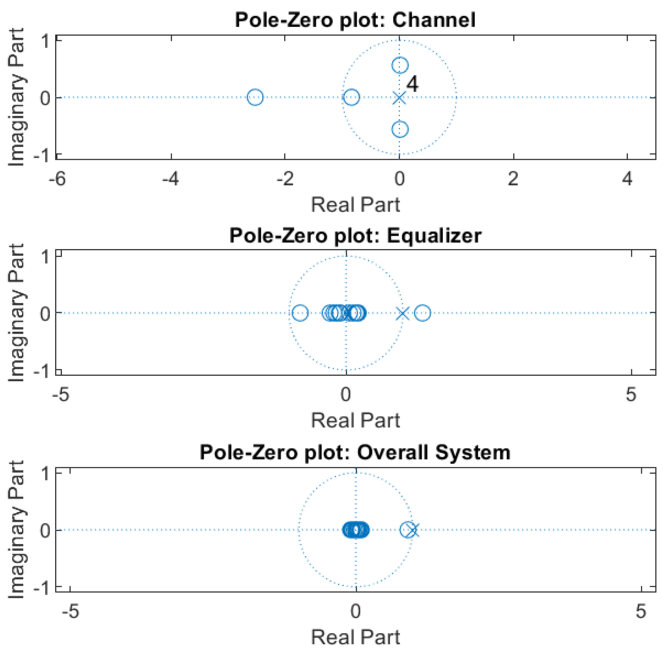

Figure 16: Pole-zero diagrams of channel, equalizer, and overall system.

Equalizer Testing Once the adaptive filter converges, we test it on new input data. Figure 17 compares the equalized vs unequlized outputs.

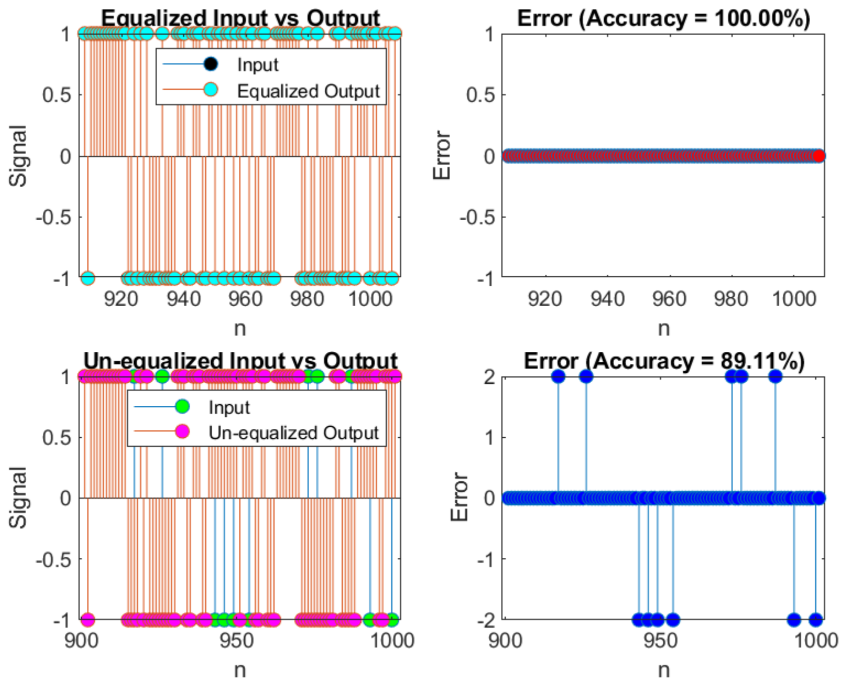

Figure 17: Comparison between equalized and un-equalized signal.

Performance Analysis

now lets analyze how the equalizer performs under varying conditions.

Performance vs SNR Figure 18 shows how increasing SNR improves both error and accuracy.

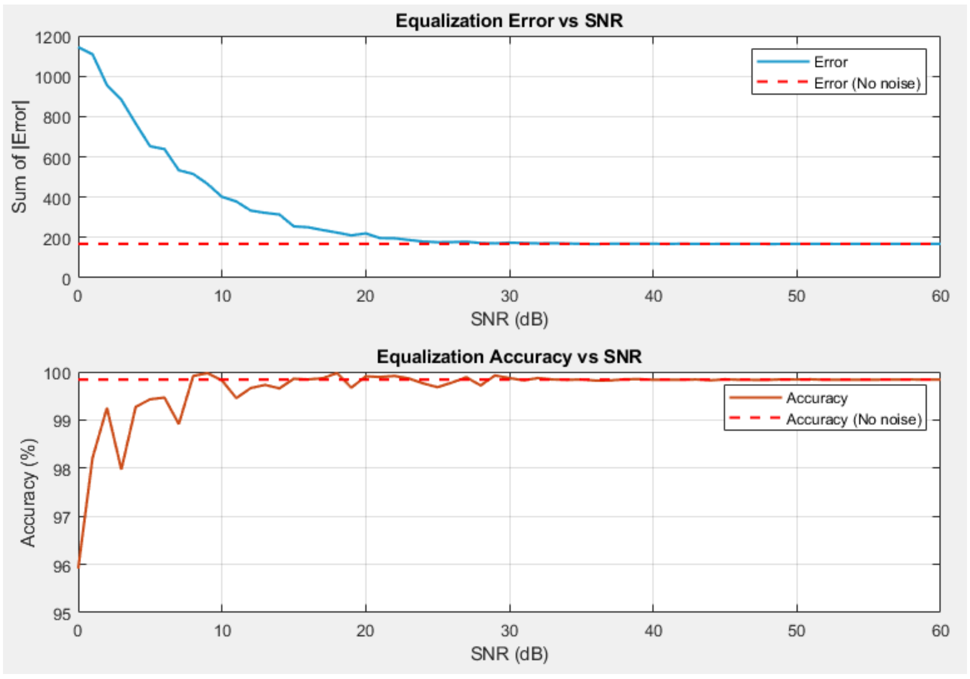

Figure 18: Equalization performance with varying SNR.

Performance vs Filter Order Increasing the number of taps improves performance up to a point, as shown in Figure 19.

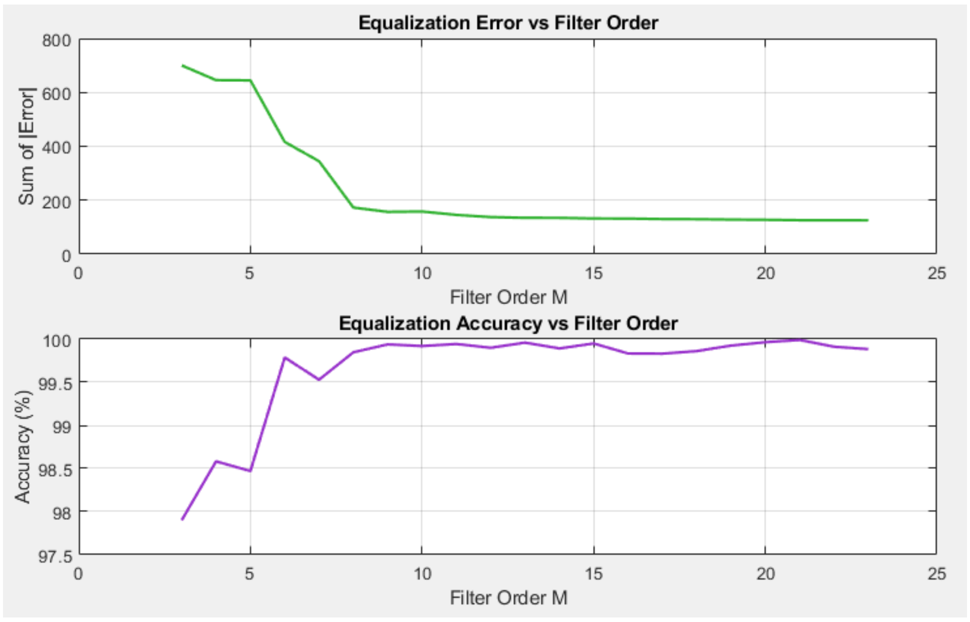

Figure 19: Equalization error and accuracy vs filter order.

Performance vs Initialization Finally, Figure 20 shows how initial filter coefficient values affect convergence.

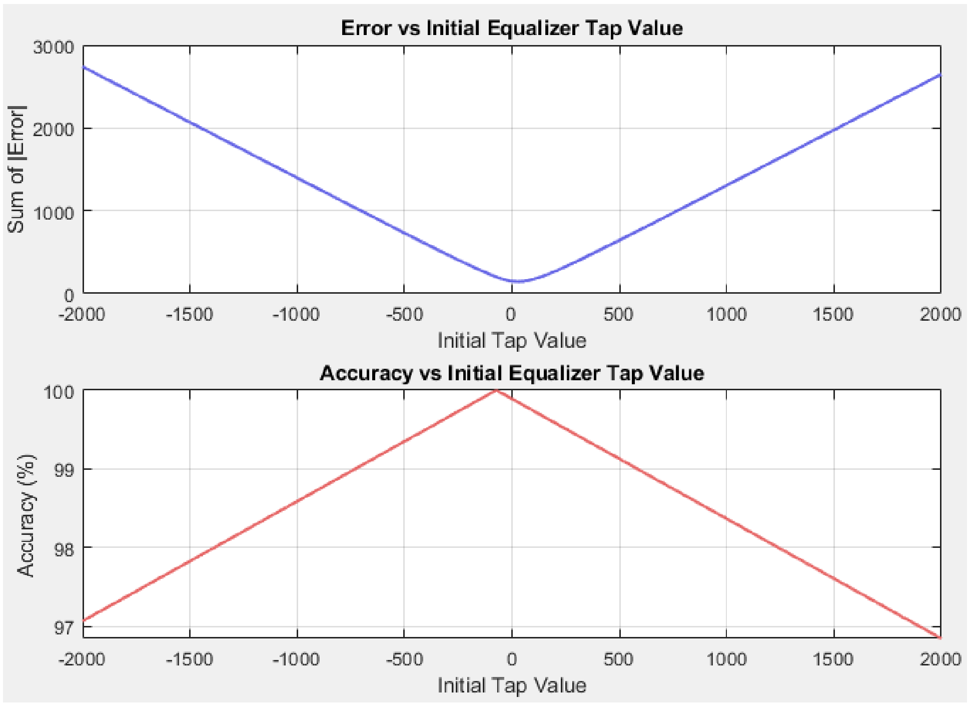

Figure 20: Performance vs equalizer initialization value.

Conclusion

In this part of the project, an LMS-based adaptive equalizer was implemented to mitigate inter-symbol interference from an unknown channel. The filter coefficients were adapted iteratively to minimize the error between the desired and actual outputs.
It was observed that the adaptive filter approximated the inverse of the channel effectively. Testing with new input sequences confirmed that performance was maintained after training. Performance was also evaluated under different signal-to-noise ratios (SNR), filter lengths, and initializations, demonstrating the equalizer's robustness with proper parameter tuning.

Summary: The equalizer was shown to identify and invert the distortion caused by a noisy channel with an accuracy of 100%. It was also found that the required delay for learning can vary, and the filter is capable of adapting to different values; delays between 4 and 12 samples were tested and still resulted in perfect reconstruction. However, sensitivity to the LMS step size was observed—large step sizes caused instability, while very small values led to slow convergence. A step size of $\mu = 0.01$ provided a stable and fast-converging solution.
When prior knowledge of the input signal was available, a decision block using a sign function could be applied to enhance accuracy. Using the equalizer, the system achieved 100% accuracy, while the un-equalized case yielded only 89%. The overall impulse response of the equalized system resembled a delay line, confirming that the input could be fully reconstructed with a fixed delay. The frequency response was approximately unity up to 2.5 rad/s, indicating the bandwidth over which the equalizer was most effective.

5 Part C: ADAPTIVE BLIND EQUALIZATION

Theory and Design

In many practical scenarios, an initial training phase may not be feasible for adaptive filtering. Nonetheless, if certain characteristics of the transmitted signal are known—such as a constant amplitude of ±1—blind equalization techniques can be applied. Under such conditions, the equalizer can function without access to the original transmitted sequence, operating instead in what is known as the **blind mode**. This approach relies on the *constant modulus* property of the input signal.

The **Constant Modulus (CM)** blind equalization algorithm exploits this property by minimizing the deviation of the equalizer output magnitude from a fixed value (in our case, 1). The schematic of the blind equalization system is shown in Figure 21.

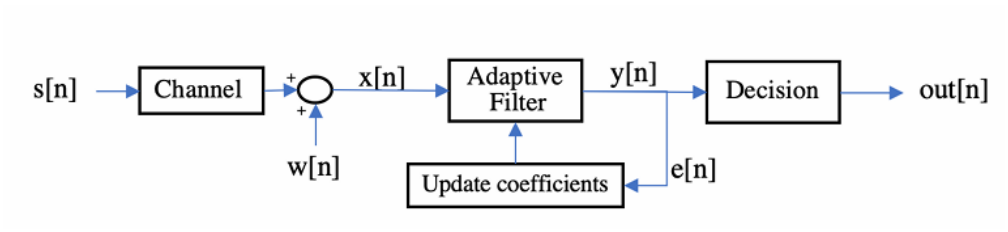

Figure 21: CM Blind Equalization System Structure

In this system, the error is computed using the square deviation of the output's magnitude from 1:

$$e[n] = |y[n]|^2 - 1 = (h_n^T x_n)^2 - 1$$

The gradient of the error with respect to the filter coefficients is:

$$\frac{d(e[n]^2)}{dh_n} = 4(|y[n]|^2 - 1)y[n]x_n$$

The update rule follows the negative gradient direction with step size μ, resulting in:

$$h_{n+1} = h_n - \mu \cdot 4e[n]y[n]x_n$$

Simulation Setup

The parameters used for this experiment are shown in Table 2.

Channel Impulse Response	$[0.3, 1, 0.7, 0.3, 0.2]$
Equalizer Filter Order M	11
LMS Step Size μ	0.0002
Initial Coefficients h_0	All ones
SNR	25 dB

Table 2: Parameters Used in Blind Equalization

Results and Analysis

The absolute value of the CM error over time is shown in Figure 22. As expected, the error decreases and stabilizes, indicating convergence of the adaptive filter.

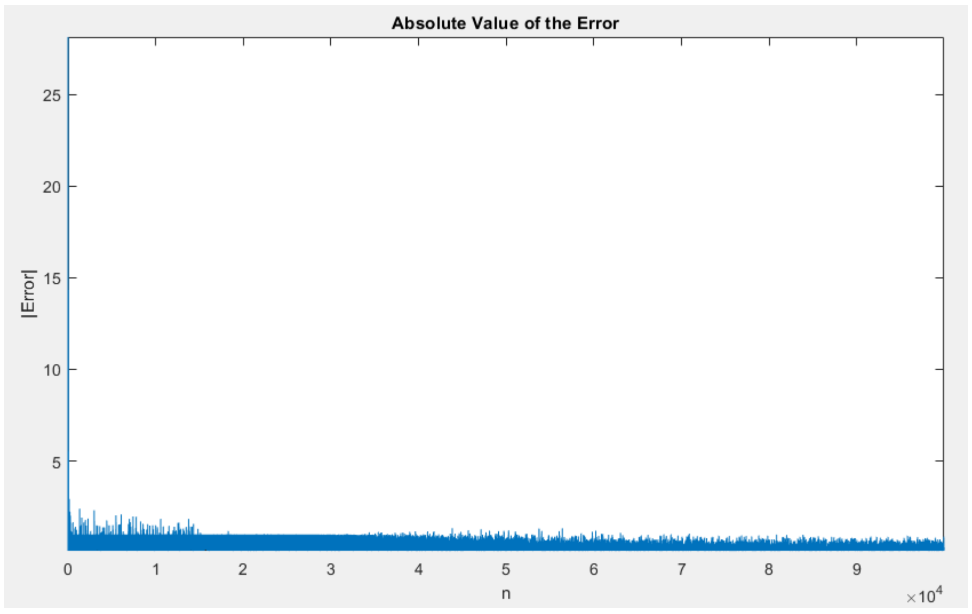

Figure 22: Absolute Value of the Error During CMA Training

The output amplitude across time is shown in Figure 23. A red dashed line is plotted at amplitude = 1 to represent the target constant modulus. The output stabilizes around this line as the equalizer converges.

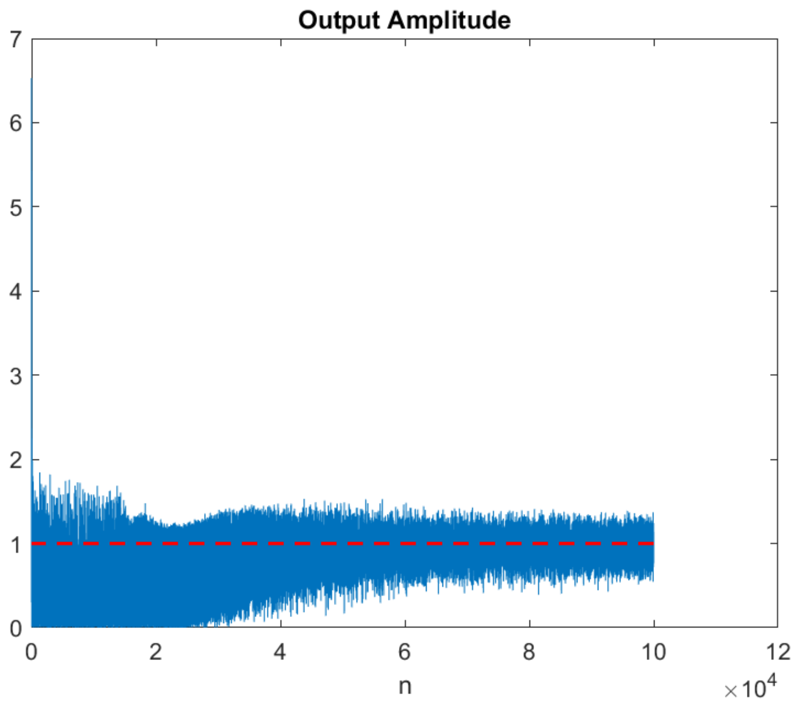

Figure 23: Magnitude of Equalizer Output Signal

Figure 24 shows the evolution of the equalizer coefficients over time. Each subplot corresponds to one tap of the equalizer. The taps converge to stable values, reflecting the adaptive filter's ability to invert the channel response blindly.

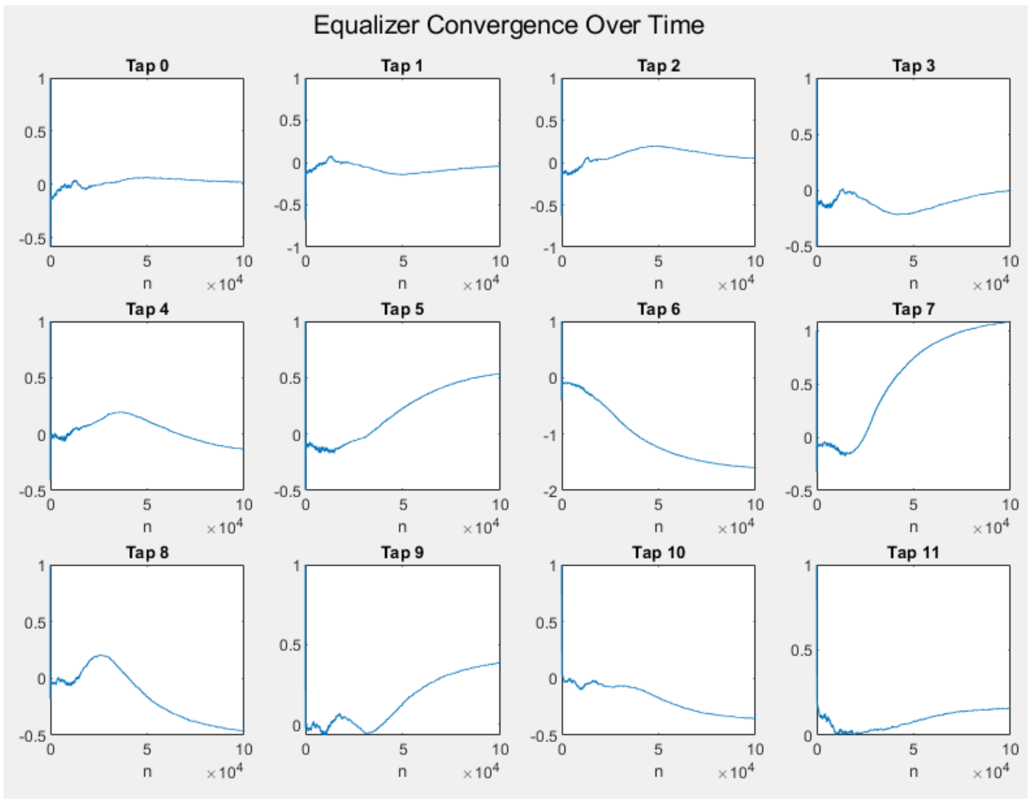

Figure 24: Equalizer Coefficients Convergence Over Time

The output is compared against the desired signal using a decision block (sign function). As shown in Figure 25, after applying the decision block, the output matches the original delayed signal with 100% accuracy.

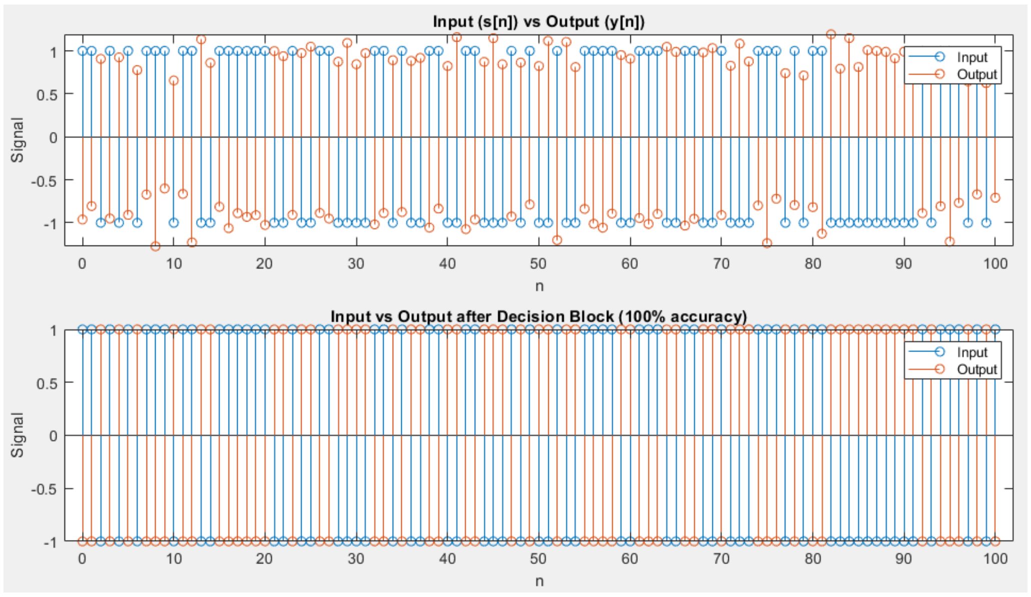

Figure 25: Input vs Output Before and After Decision Block (100% accuracy)

A comparison between the blind-equalized and un-equalized output is illustrated in Figure 26. While the blind equalizer achieves 100% accuracy, the un-equalized output only matches the input about 49% of the time.

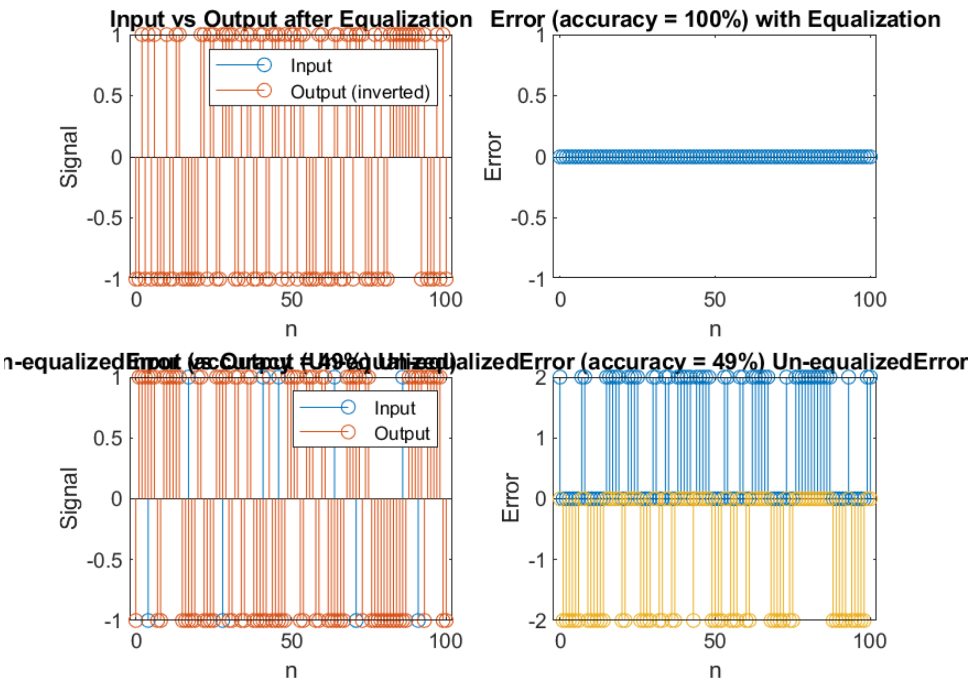

Figure 26: Equalized vs Un-equalized Output Comparison

The frequency response of the channel, the equalizer, and their combined effect is shown in Figure 27. The overall system exhibits a near-flat frequency response across most of the spectrum, confirming effective equalization.

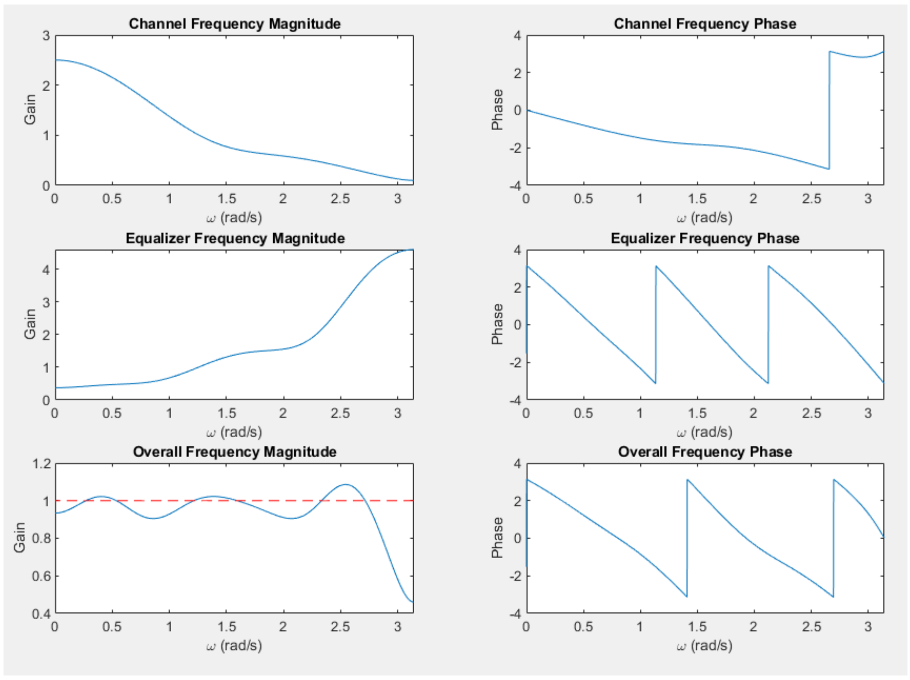

Figure 27: Frequency Responses: Channel, Equalizer, and Combined LTI System

Figure 28 shows the impulse responses. The overall LTI system resembles a delayed delta function, indicating that the combined channel and equalizer system behaves as a pure delay.

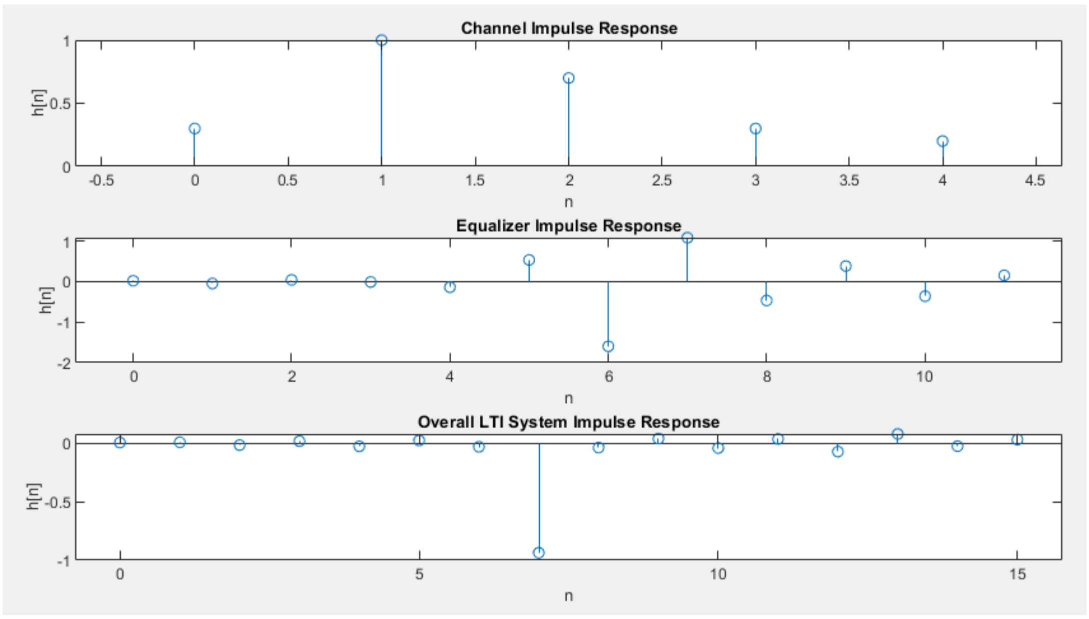

Figure 28: Impulse Responses: Channel, Equalizer, and Overall System

Lastly, the tap weights of the overall LTI system are shown in Figure 29. Most taps converge to zero, except for a strong peak at a certain index, supporting that the system behaves as a delay.

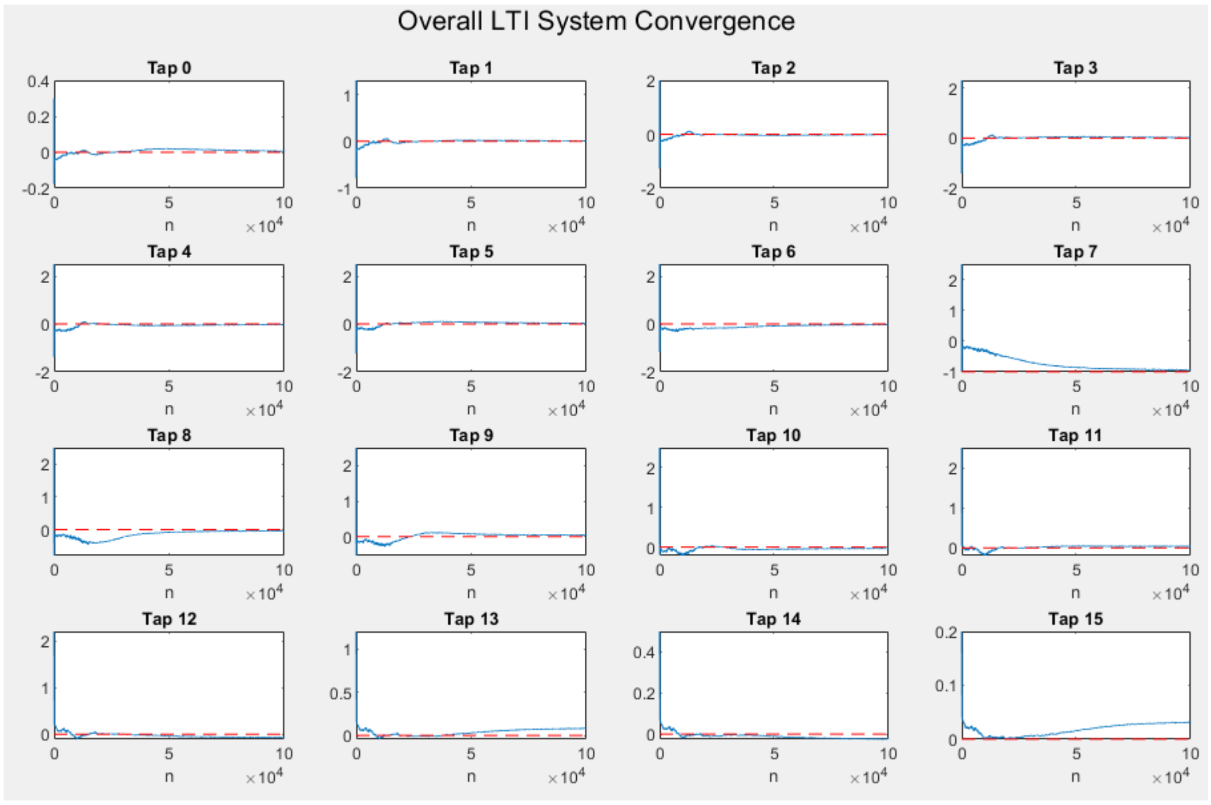

Figure 29: Overall LTI System Tap Convergence

Appendix: MATLAB Code

i. Part A1: Adaptive Notch Filter — Static Interference

This script runs four experiments using an LMS-based adaptive notch filter to cancel a fixed-frequency sinusoid. It evaluates the effect of varying μ, r, and f_{noise} on convergence and accuracy.
Output Figures: Figure 1 – Experiment 1 — **Figure 2** – Experiment 2 — **Figure 3** – Experiment 3 — **Figure 4** – Experiment 4

```
clc; clear all; close all;
% Parameters
fs = 200;                  % Sampling frequency (Hz)
f_signal = 3;              % Desired signal frequency (Hz)
N = 500000;                % Number of samples
n = 0:N-1;                 % Time index
N_fft = 4096;              % FFT size

% Define experiments
experiments = [
    struct('mu',1e-6,'r',0.85,'f_noise',30),
    struct('mu',1e-5,'r',0.85,'f_noise',30),
    struct('mu',1e-6,'r',0.95,'f_noise',30),
    struct('mu',1e-6,'r',0.85,'f_noise',40)
];
legend_entries = {
    '\mu=1e-6, r=0.85, f=30Hz',
    '\mu=1e-5, r=0.85, f=30Hz',
    '\mu=1e-6, r=0.95, f=30Hz',
    '\mu=1e-6, r=0.85, f=40Hz'
};
colors = {'b', 'r', 'g', 'm'};
% Storage init
X_f = cell(1,4); Y_f = cell(1,4); a_all = cell(1,4);
omega_est_all = cell(1,4); omega_true_all = zeros(1,4);
y_all = cell(1,4); x_all = cell(1,4);

% Experiment loop
for idx = 1:length(experiments)
    mu = experiments(idx).mu;
    r = experiments(idx).r;
    f_noise = experiments(idx).f_noise;
    desired_signal = 1 * sin(2*pi*f_signal/fs * n);
    interference = 10 * sin(2*pi*f_noise/fs * n);
    x = desired_signal + interference;
    a = zeros(N+1,1); y = zeros(N,1); e = zeros(N,1);
    for k = 1:N
        if k == 1, x_n1 = 0; x_n2 = 0; y_n1 = 0; y_n2 = 0;
        elseif k == 2, x_n1 = x(k-1); x_n2 = 0; y_n1 = y(k-1); y_n2 =
            0;
        else, x_n1 = x(k-1); x_n2 = x(k-2); y_n1 = y(k-1); y_n2 = y(k
            -2);
        end
        e(k) = x(k) + a(k)*x_n1 + x_n2;
        y(k) = e(k) - r*a(k)*y_n1 - r^2*y_n2;
        a(k+1) = a(k) - mu*y(k)*x_n1;
        if (a(k+1) > 2) || (a(k+1) < -2), a(k+1) = 0; end
    end
    a = a(1:N);
    x_all{idx} = x; y_all{idx} = y; a_all{idx} = a;
    X_f{idx} = fftshift(fft(x(end-4096+1:end), N_fft));
    Y_f{idx} = fftshift(fft(y(end-4096+1:end), N_fft));
    omega_est_all{idx} = acos(-a/2);
    omega_true_all(idx) = 2*pi*f_noise/fs;
end

w = linspace(-pi, pi, N_fft);

% Plotting results
for idx = 1:4
    figure;
    sgtitle(["\mu=", num2str(experiments(idx).mu), ", r=", num2str(
        experiments(idx).r), ", f=", num2str(experiments(idx).f_noise),
        "Hz"]);

    % 1. Input vs Output
    subplot(2,2,1); hold on;
    plot(n(end-50:end), x_all{idx}(end-50:end), '-', 'Color', colors{
        idx}, 'LineWidth', 1.2);
    plot(n(end-50:end), y_all{idx}(end-50:end), '--', 'Color', colors{
        idx}, 'LineWidth', 1.2);
    xlabel('n'); ylabel('Amplitude');
    title('Input and Output Signal'); legend('show'); grid on;

    % 2. Magnitude Spectrum
    subplot(2,2,2); hold on;
    plot(w/pi, 20*log10(abs(X_f{idx})+1e-8), '-', 'Color', colors{idx},
         'LineWidth', 1.2);
    plot(w/pi, 20*log10(abs(Y_f{idx})+1e-8), '--', 'Color', colors{idx
        }, 'LineWidth', 1.2);
    xlabel('Normalized Frequency (\times \pi rad/sample)'); ylabel('
        Magnitude (dB)');
    title('Magnitude Response of Input and Output'); ylim([-100 100]);
        legend('show'); grid on;

    % 3. Evolution of a[n]
    subplot(2,2,3); hold on;
    plot(n, a_all{idx}, '-', 'Color', colors{idx}, 'LineWidth', 1.2);
    plot(n, -2*cos(2*pi*experiments(idx).f_noise/fs)*ones(size(n)), '--
        ', 'Color', colors{idx}, 'LineWidth', 1.2);
    xlabel('n'); ylabel('a[n]'); title('Updating Parameter a'); legend(
        'show'); grid on;

    % 4. Notch Frequency Tracking
    subplot(2,2,4); hold on;
    plot(n, omega_est_all{idx}, '-', 'Color', colors{idx}, 'LineWidth',
         1.2);
    plot(n, omega_true_all(idx)*ones(size(n)), '--', 'Color', colors{
        idx}, 'LineWidth', 1.2);
    xlabel('n'); ylabel('\omega (rad/sample)');
    title('Updating the Notch Frequency'); legend('show'); grid on;
end
```

ii. Part A2: Adaptive Notch Filter — Slowly Changing Interference

This script applies an LMS-based adaptive notch filter to cancel an interference whose frequency drifts slowly from 30 Hz to 40 Hz over time. The filter dynamically tracks the changing notch location without prior knowledge of the frequency profile.

Output Figure: Figure 5 – Adaptive filter tracking performance.

```
clc; clear all; close all;
% Parameters
fs = 200;                    % Sampling frequency (Hz)
f_signal = 3;                % Desired signal frequency (Hz)
N = 500000;                  % Number of samples
n = 0:N-1;                   % Time index
N_fft = 4096;                % FFT size
mu = 1e-6;                   % Step size
r = 0.9;                     % Pole radius

% Define slowly changing interference frequency
f_start = 30;
f_end = 40;
f_noise = linspace(f_start, f_end, N);

% Desired signal
desired_signal = 1 * sin(2*pi*f_signal/fs * n);

% Interference signal with varying frequency
phi = cumsum(2*pi*f_noise/fs);
interference = 10 * cos(phi);

% Combined input signal
x = desired_signal + interference;

% Adaptive filtering initialization
a = zeros(N+1,1);
y = zeros(N,1);
e = zeros(N,1);

for k = 1:N
    if k == 1
        x_n1 = 0; x_n2 = 0; y_n1 = 0; y_n2 = 0;
    elseif k == 2
        x_n1 = x(k-1); x_n2 = 0; y_n1 = y(k-1); y_n2 = 0;
    else
        x_n1 = x(k-1); x_n2 = x(k-2); y_n1 = y(k-1); y_n2 = y(k-2);
    end

    e(k) = x(k) + a(k)*x_n1 + x_n2;
    y(k) = e(k) - r*a(k)*y_n1 - r^2*y_n2;
    a(k+1) = a(k) - mu * y(k) * x_n1;

    if (a(k+1) > 2) || (a(k+1) < -2)
        a(k+1) = 0;
    end
end
```

```
a = a(1:N);

% Estimate omega
omega_est = acos(-a/2);
omega_true = 2*pi*f_noise/fs;

% Plotting
figure;
subplot(2,2,1); hold on;
plot(n(end-50:end), x(end-50:end), '-', 'Color', 'b', 'LineWidth', 1.5)
   ;
plot(n(end-50:end), y(end-50:end), '--', 'Color', 'r', 'LineWidth',
   1.8);
xlabel('n'); ylabel('Amplitude');
title('Input and Output Signals (Last 50 samples)');
legend('show', 'FontSize',8, 'Location', 'best');
grid on;

subplot(2,2,2); hold on;
plot(linspace(-fs/2, fs/2, N_fft), 20*log10(abs(fftshift(fft(x(end
   -4096+1:end),N_fft)))+1e-8), '-', 'Color', 'b', 'LineWidth', 1.5);
plot(linspace(-fs/2, fs/2, N_fft), 20*log10(abs(fftshift(fft(y(end
   -4096+1:end),N_fft)))+1e-8), '--', 'Color', 'r', 'LineWidth', 1.8);
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Magnitude Spectrum of Input and Output');
ylim([-100 100]);
legend('show', 'FontSize',8, 'Location', 'best');
grid on;

subplot(2,2,3); hold on;
plot(n, a, '-', 'Color', 'b', 'LineWidth', 1.5);
plot(n, -2*cos(2*pi*f_noise/fs), '--', 'Color', 'r', 'LineWidth', 1.8);
xlabel('n'); ylabel('a[n]');
title('Evolution of a[n]');
legend('show', 'FontSize',8, 'Location', 'best');
grid on;

subplot(2,2,4); hold on;
plot(n, omega_est, '-', 'Color', 'b', 'LineWidth', 1.5);
plot(n, omega_true, '--', 'Color', 'r', 'LineWidth', 1.8);
xlabel('n'); ylabel('\\omega (rad/sample)');
title('Tracking Notch Frequency');
legend('show', 'FontSize',8, 'Location', 'best');
grid on;
```

iii. Part A3: Adaptive Notch Filter — Two Sinusoidal Interferences

This script implements a cascaded adaptive notch filter to cancel two independent sinusoidal interferences at 45 Hz and 60 Hz. Two LMS loops update the filter parameters $a_1[n]$ and $a_2[n]$ independently to suppress both tones.

Output Figures: Figure 7 – Tracking performance of the cascaded filters, **Figure 8** – Magnitude responses and parameter evolutions.

```
clc; clear all; close all;
% Parameters
fs = 200;
N = 50000;
n = 0:N;
u = 1e-5;
r = 0.95;

% Signal definition
fsig = 5;
fn1 = 45;
fn2 = 60;

desired = sin(2*pi*fsig/fs * n);
noise1 = 8*sin(2*pi*fn1/fs * n);
noise2 = 4*sin(2*pi*fn2/fs * n);
x = desired + noise1 + noise2;

% True a values
wnoise1 = 2*pi*fn1/fs;
wnoise2 = 2*pi*fn2/fs;
atrue1 = -2*cos(wnoise1);
atrue2 = -2*cos(wnoise2);

% Initialization
a1 = zeros(length(x),1);
a2 = zeros(length(x),1);
yn1 = zeros(length(x),1);
yn2 = zeros(length(x),1);

% Adaptive Filtering
for k = 1:length(x)
    [yn1, xn1_1] = adaptive_filter_OH(x, a1, yn1, u, r, k);
    [yn2, xn2_1] = adaptive_filter_OH(yn1, a2, yn2, u, r, k);

    a1(k+1) = a1(k) - u * yn1(k) * xn1_1;
    if (a1(k+1) > 2) || (a1(k+1) < -2), a1(k+1) = 0; end

    a2(k+1) = a2(k) - u * yn2(k) * xn2_1;
    if (a2(k+1) > 2) || (a2(k+1) < -2), a2(k+1) = 0; end
end

a1 = a1(1:end-1);
a2 = a2(1:end-1);
```

```
% Frequency Analysis
N_fft = 512;
w = linspace(-pi, pi, N_fft);
X_f = fftshift(fft(x, N_fft));
Y1_f = fftshift(fft(yn1(end-511:end), N_fft));
Y2_f = fftshift(fft(yn2(end-511:end), N_fft));

% Plotting
figure;

subplot(3,2,1);
plot(w, abs(X_f), 'b');
title('Magnitude Response of x[n]');
xlabel('\omega (rad/sample)');
ylabel('Magnitude');
grid on;

subplot(3,2,3);
plot(w, abs(Y1_f), 'b');
title('Magnitude Response of y1[n]');
xlabel('\omega (rad/sample)');
ylabel('Magnitude');
grid on;

subplot(3,2,5);
plot(w, abs(Y2_f), 'b');
title('Magnitude Response of y2[n]');
xlabel('\omega (rad/sample)');
ylabel('Magnitude');
grid on;

subplot(3,2,6); hold on;
plot(n(end-30:end), x(end-30:end), 'b-', 'LineWidth', 1);
plot(n(end-30:end), yn1(end-30:end), 'r--', 'LineWidth', 1);
plot(n(end-30:end), yn2(end-30:end), 'm-', 'LineWidth', 1);
legend('x[n]', 'y1[n]', 'y2[n]');
title('Comparison of x[n], y1[n], and y2[n]');
xlabel('n');
ylabel('Amplitude');
grid on;
ylim([-20 20]);

subplot(3,2,2); hold on;
plot(a1, 'b-', 'LineWidth', 1);
plot(atrue1*ones(length(a1),1), 'r--', 'LineWidth', 1);
legend('a1', 'true a1');
title('Updating Parameter a1');
xlabel('n');
ylabel('a_1[n]');
grid on;

subplot(3,2,4); hold on;
plot(a2, 'b-', 'LineWidth', 1);
plot(atrue2*ones(length(a2),1), 'r--', 'LineWidth', 1);
legend('a2', 'true a2');
```

```
title('Updating Parameter a2');
xlabel('n');
ylabel('a_2[n]');
grid on;

sgtitle('Adaptive Notch Filter for Two Sinusoidal Interferences');

% Second figure (input vs output, tracking)
input_color = 'b';
output_color = 'r';

figure;
subplot(2,2,1); hold on;
plot(n(end-50:end), x(end-50:end), input_color, 'LineWidth', 1.5);
plot(n(end-50:end), yn2(end-50:end), output_color, 'LineWidth', 1.5);
xlabel('n'); ylabel('Amplitude');
title('Input and Output Signals (Last 50 samples)');
legend('show', 'FontSize',8, 'Location', 'best');
grid on;

subplot(2,2,2); hold on;
N_fft = 4096;
plot(linspace(-fs/2, fs/2, N_fft), 20*log10(abs(fftshift(fft(x(end
    -4096+1:end),N_fft)))+1e-8), input_color, 'LineWidth', 1.5);
plot(linspace(-fs/2, fs/2, N_fft), 20*log10(abs(fftshift(fft(yn2(end
    -4096+1:end),N_fft)))+1e-8), output_color, 'LineWidth', 1.5);
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Magnitude Spectrum of Input and Output');
ylim([-100 100]);
legend('show', 'FontSize',8, 'Location', 'best');
grid on;

subplot(2,2,3); hold on;
plot(n, a1, input_color, 'LineWidth', 1.5);
plot(n, a2, output_color, 'LineWidth', 1.5);
plot(n, ones(size(n))*atrue1, input_color, 'LineWidth', 1);
plot(n, ones(size(n))*atrue2, output_color, 'LineWidth', 1);
xlabel('n');
ylabel('a[n]');
title('Evolution of a_1[n] and a_2[n]');
legend('show', 'FontSize',8, 'Location', 'best');
grid on;

subplot(2,2,4); hold on;
omega1_est = acos(-a1/2);
omega2_est = acos(-a2/2);
omega1_true = 2*pi*fn1/fs;
omega2_true = 2*pi*fn2/fs;

plot(n, omega1_est, '--', 'Color', 'b', 'LineWidth', 1.5);
plot(n, omega2_est, '--', 'Color', 'r', 'LineWidth', 1.5);
plot(n, omega1_true*ones(size(n)), input_color, 'LineWidth', 1);
plot(n, omega2_true*ones(size(n)), output_color, 'LineWidth', 1);
xlabel('n');
ylabel('\omega (rad/sample)');
title('Tracking Notch Frequencies');
legend('show', 'FontSize',8, 'Location', 'best');
```

```
grid on;
sgtitle('Adaptive Notch Filter for Two Interferences');
```

iv. Part B: Adaptive Notch Filtering

This script implements an LMS-based adaptive equalizer to mitigate inter-symbol interference caused by an unknown channel. It trains on distorted data, adapts filter coefficients, and tests the equalizer under varying SNR, filter order, and initialization conditions.

Output Figures:

- **Figure 9** – System Block Diagram
- **Figure 11** – Training Error and Equalizer Output vs Delayed Input
- **Figure 12** – Equalizer Coefficient Convergence
- **Figure 13** – Overall LTI System Convergence
- **Figure 14** – Frequency Responses
- **Figure 28** – Impulse Responses
- **Figure 16** – Pole-Zero Plots
- **Figure 17** – Equalized vs Un-equalized Testing
- **Figure 18** – Performance vs SNR
- **Figure 19** – Performance vs Filter Order
- **Figure 20** – Performance vs Initialization

```
    clc; clear all; close all;
    % Generate Training Data
    rng(1, 'twister');
    sn = randi([0, 1], 1, 1000);
    sn(sn == 0) = -1;

    hn = [0.3, 1, 0.7, 0.3, 0.2]; % Channel impulse response
    noise = wgn(1, length(sn) + length(hn) - 1, -25); % Additive
        Gaussian noise
    xn = (conv(sn, hn) + noise)';

    % Parameters
    M = 11; % Equalizer order
    delays = 8; % Fixed delay
    h = ones(M+1, 1); % Initial filter coefficients
    u = 0.01; % Step size

    hd = [zeros(1, delays), 1]; % Delay line
    sn_delayed = conv(hd, sn)';

    e = zeros(length(sn), 1);
```

```
y = zeros(length(sn), 1);
h_track = zeros(length(sn), M+1);
overall_lti_track = zeros(length(sn), M+5);


% Adaptive LMS Training

for n = 1:length(xn)
    xf = flip(xn(max(1, n - M):n));
    xf = [xf; zeros(M+1-length(xf), 1)];
    y(n) = sum(xf .* h);
    e(n) = sn_delayed(n) - y(n);
    h = h + 2 * u * e(n) .* xf;
    h_track(n, :) = h';
    overall_lti_track(n, :) = conv(h, hn);
end

overall_lti = conv(h, hn);

% Training Error and Output Comparison

figure;
subplot(2,1,1);
plot(abs(e), 'Color', [0.1, 0.6, 0.8]);
xlabel('n'); ylabel('|Error|');
title('Absolute Value of Error during Training');

subplot(2,1,2);
stem(900:1004, sn_delayed(900:1004), 'filled', 'MarkerFaceColor', '
   b');
hold on;
stem(900:1004, y(900:1004), 'filled', 'MarkerFaceColor', 'm');
legend('Delayed Input', 'Equalizer Output');
xlabel('n'); ylabel('Signal');
title('Comparison: Input vs Output after Equalization');
saveas(gcf, 'figures/training_error_comparison.png');

% Accuracy Calculation

in = sn_delayed(900:1004);
out = y(900:1004);
out(out > 0) = 1;
out(out < 0) = -1;
out(out == 0) = -1;
acc = (1 - abs(sum(in' - out)) / length(in)) * 100;
sprintf('Accuracy = %.2f%%', acc)

% Filter Convergence Plots

figure;
for i = 1:M+1
    subplot(3,4,i);
    plot(h_track(:,i), 'Color', [rand, rand, rand]);
    xlabel('n');
    title(sprintf('Tap %d', i-1));
end
sgtitle('Convergence of Equalizer Coefficients');
saveas(gcf, 'figures/equalizer_convergence.png');
```

```
figure;
for i = 1:M+5
    subplot(4,4,i);
    plot(overall_lti_track(:,i), 'Color', [rand, rand, rand]);
    hold on;
    if i ~= 9
        plot(zeros(1,length(sn)),'--r');
    else
        plot(ones(1,length(sn)),'--g');
    end
    xlabel('n');
    title(sprintf('Tap %d', i-1));
end
sgtitle('Convergence of Overall LTI System');
saveas(gcf, 'figures/overall_lti_convergence.png');

% Frequency Response and Impulse Response
figure;
subplot(3,2,1);
[channel_f, w] = DTFT(hn);
plot(w, abs(channel_f), 'LineWidth', 1.5);
xlabel('\omega (rad/s)'); ylabel('Magnitude');
title('Channel Frequency Response');

subplot(3,2,2);
plot(w, angle(channel_f), 'LineWidth', 1.5);
xlabel('\omega (rad/s)'); ylabel('Phase');
title('Channel Phase Response');

subplot(3,2,3);
[h_f, ~] = DTFT(h);
plot(w, abs(h_f), 'LineWidth', 1.5);
xlabel('\omega (rad/s)'); ylabel('Magnitude');
title('Equalizer Frequency Response');

subplot(3,2,4);
plot(w, angle(h_f), 'LineWidth', 1.5);
xlabel('\omega (rad/s)'); ylabel('Phase');
title('Equalizer Phase Response');

subplot(3,2,5);
plot(w, abs(h_f .* channel_f'), 'LineWidth', 1.5);
hold on;
plot(w, ones(1,length(w)), '--r');
xlabel('\omega (rad/s)'); ylabel('Magnitude');
title('Overall System Frequency Response');

subplot(3,2,6);
plot(w, angle(h_f .* channel_f'), 'LineWidth', 1.5);
xlabel('\omega (rad/s)'); ylabel('Phase');
title('Overall System Phase Response');
saveas(gcf, 'figures/frequency_response.png');
```

```
figure;
subplot(3,1,1);
stem(0:length(hn)-1, hn, 'filled');
xlabel('n'); ylabel('Amplitude');
title('Channel Impulse Response');

subplot(3,1,2);
stem(0:length(h)-1, h, 'filled');
xlabel('n'); ylabel('Amplitude');
title('Equalizer Impulse Response');

subplot(3,1,3);
stem(0:length(overall_lti)-1, overall_lti, 'filled');
xlabel('n'); ylabel('Amplitude');
title('Overall LTI System Impulse Response');
saveas(gcf, 'figures/impulse_responses.png');

figure;
subplot(3,1,1);
zplane(hn, [1]);
title('Pole-Zero plot: Channel');

subplot(3,1,2);
zplane(h, [1]);
title('Pole-Zero plot: Equalizer');

subplot(3,1,3);
zplane(conv(h, hn), [1]);
title('Pole-Zero plot: Overall System');
saveas(gcf, 'figures/pole_zero_plots.png');

% Testing the Equalizer (new input)
rng(5, 'twister');
sn = randi([0,1],1,1000);
sn(sn==0) = -1;
noise = wgn(1,length(sn)+length(hn)-1,-25);
xn = (conv(sn,hn)+noise)';
yn = conv(h,xn);

% Hard Decision
yn(yn>0) = 1; yn(yn<0) = -1; yn(yn==0) = -1;

% For comparison
hd = [zeros(1,8),1];
sn_delayed = conv(hd, sn)';

% Plot Results
figure;
subplot(2,2,1);
stem(908:1008, sn_delayed(908:1008), 'filled', 'MarkerFaceColor', '
   k');
hold on;
stem(908:1008, yn(908:1008), 'filled', 'MarkerFaceColor', 'c');
```

```
legend('Input','Equalized Output');
xlabel('n'); ylabel('Signal'); title('Equalized Input vs Output');



subplot(2,2,2);
err = sn_delayed(908:1008) - yn(908:1008);
stem(908:1008, err, 'MarkerFaceColor', 'r');
c = sum(err == 0) / length(err) * 100;
title(sprintf('Error (Accuracy = %.2f%%)', c));
xlabel('n'); ylabel('Error');

subplot(2,2,3);
sn_delayed2 = conv([0 1], sn)';
xn(xn>0) = 1; xn(xn<0) = -1; xn(xn==0) = -1;
stem(901:1001, sn_delayed2(901:1001), 'filled', 'MarkerFaceColor',
    'g');
hold on;
stem(901:1001, xn(901:1001), 'filled', 'MarkerFaceColor', 'm');
legend('Input','Un-equalized Output');
xlabel('n'); ylabel('Signal'); title('Un-equalized Input vs Output'
    );

subplot(2,2,4);
err = sn_delayed2(901:1001) - xn(901:1001);
stem(901:1001, err, 'MarkerFaceColor', 'b');
c = sum(err == 0) / length(err) * 100;
title(sprintf('Error (Accuracy = %.2f%%)', c));
xlabel('n'); ylabel('Error');
saveas(gcf, 'figures/test_equalizer_performance.png');
```

```
%Performance Analysis - SNR, Filter Order, Initialization

clc; clear all; close all;

% Generate Training Data

rng(1,'twister');
sn = randi([0,1], 1,1000);
sn(sn==0) = -1;
hn = [0.3, 1, 0.7, 0.3, 0.2];

noise = wgn(1,length(sn)+length(hn)-1,-25);
xn = (conv(sn,hn) + noise)';
[h,e,sn_delayed,out] = equalizer(sn,xn,hn,11);

% Test Setup

rng(5,'twister');
sn = randi([0,1], 1,1000);
sn(sn==0) = -1;
delays = 8;
hd = [zeros(1,delays), 1];
sn_delayed = conv(hd,sn);
```

```
% Performance vs SNR

err1 = zeros(61,1);
acc = zeros(61,1);
for i = 0:60
    noise = wgn(1,length(sn)+length(hn)-1,-i);
    xn = (conv(sn,hn) + noise)';
    out = conv(h,xn);
    outd = sign(out);
    outd(outd==0) = -1;
    err1(i+1) = sum(abs(sn_delayed(8:900)' - out(8:900)));
    acc(i+1) = (1 - abs(sum(sn_delayed(8:900)' - out(8:900))) /
       length(8:900)) * 100;
end

figure;
subplot(2,1,1);
plot(0:60, err1, 'Color', [0.1 0.6 0.8], 'LineWidth', 1.5);
hold on;
plot(0:60, ones(1,61)*167.94, '--r', 'LineWidth', 1.5);
xlabel('SNR (dB)'); ylabel('Sum of |Error|');
title('Equalization Error vs SNR');
legend('Error', 'Error (No noise)');
grid on;

subplot(2,1,2);
plot(0:60, acc, 'Color', [0.8 0.3 0.1], 'LineWidth', 1.5);
hold on;
plot(0:60, ones(1,61)*99.84, '--r', 'LineWidth', 1.5);
xlabel('SNR (dB)'); ylabel('Accuracy (%)');
title('Equalization Accuracy vs SNR');
legend('Accuracy', 'Accuracy (No noise)');
grid on;

% Performance vs Filter Order
clc; clear err1; clear acc;
noise = wgn(1,length(sn)+length(hn)-1,-25);
xn = (conv(sn,hn) + noise)';

err1 = zeros(21,1);
acc = zeros(21,1);
for i = 0:20
    [h,e,sn_delayed,out] = equalizer(sn,xn,hn,i+3);
    out = conv(h,xn);
    outd = sign(out);
    outd(outd==0) = -1;
    err1(i+1) = sum(abs(sn_delayed(200:900)' - out(200:900)));
    acc(i+1) = (1 - abs(sum(sn_delayed(200:900)' - out(200:900))) /
        length(8:900)) * 100;
end

figure;
subplot(2,1,1);
```

```
plot(3:23, err1, 'Color', [0.2 0.7 0.2], 'LineWidth', 1.5);
xlabel('Filter Order M'); ylabel('Sum of |Error|');
title('Equalization Error vs Filter Order');
grid on;

subplot(2,1,2);
plot(3:23, acc, 'Color', [0.6 0.2 0.8], 'LineWidth', 1.5);
xlabel('Filter Order M'); ylabel('Accuracy (%)');
title('Equalization Accuracy vs Filter Order');
grid on;

% Performance vs Initialization
clc; clear err1; clear acc;
noise = wgn(1,length(sn)+length(hn)-1,-25);
xn = (conv(sn,hn) + noise)';

c = 1;
err1 = zeros(4001,1);
acc = zeros(4001,1);
for i = -2000:2000
    [h,e,sn_delayed,out] = equalizer(sn,xn,hn,11,ones(12,1)*i);
    out = conv(h,xn);
    outd = sign(out);
    outd(outd==0) = -1;
    err1(c) = sum(abs(sn_delayed(200:900)' - out(200:900)));
    acc(c) = (1 - abs(sum(sn_delayed(200:900)' - out(200:900))) /
       length(8:900)) * 100;
    c = c + 1;
end

figure;
subplot(2,1,1);
plot(-2000:2000, err1, 'Color', [0.4 0.4 0.9], 'LineWidth', 1.5);
xlabel('Initial Tap Value'); ylabel('Sum of |Error|');
title('Error vs Initial Equalizer Tap Value');
grid on;

subplot(2,1,2);
plot(-2000:2000, acc, 'Color', [0.9 0.4 0.4], 'LineWidth', 1.5);
xlabel('Initial Tap Value'); ylabel('Accuracy (%)');
title('Accuracy vs Initial Equalizer Tap Value');
grid on;
```

v. Part C: Adaptive Blind Equalization

This script implements a blind adaptive equalizer using the Constant Modulus Algorithm (CMA) to recover a transmitted signal without access to a training sequence. It minimizes the deviation of the output magnitude from a constant value and demonstrates complete system inversion without prior information.

Output Figures: Figure 22, Figure 23, Figure 24, Figure 25, Figure 26, Figure 27, Figure 28, Figure 29.

```
clc; clear all; close all;

% Input Signal
rng(1,'twister');
sn = randi([0,1], 1,100000);
sn(sn==0) = -1;

% Channel and Noise
h_channel = [0.3, 1, 0.7, 0.3, 0.2];
noise = wgn(1,length(sn)+length(h_channel)-1,-25);
xn = conv(sn,h_channel)';

% Initialization
M = 11;
h = ones(M+1,1);
u = 0.0002;
e = zeros(length(sn),1);
y = zeros(length(sn),1);
h_track = zeros(length(sn),M+1);
overall_lti_track = zeros(length(sn),M+5);

% CMA Equalizer Update
for n = 1:length(xn)
    xf = flip(xn(1:n));
    if length(xf) < M+1
        xf = [xf; zeros(M+1-length(xf),1)];
    end
    xf = xf(1:M+1);
    y(n) = sum(xf .* h);
    e(n) = y(n)^2 - 1;
    h = h - 4 * u * e(n) * y(n) .* xf;
    h_track(n,:) = h';
    overall_lti_track(n,:) = conv(h,h_channel);
end

overall_lti = conv(h,h_channel);

% Accuracy Evaluation
mstart = length(sn)-1000;
mend = length(sn)-900;
delays = M-4;
hd = [zeros(1,delays), 1];
sn_delayed = conv(hd,sn);
in = sn_delayed(mstart:mend);
out = sign(y(mstart:mend))';
a = in + out;
acc = sum(a==0) / length(a) * 100;
sprintf('Accuracy = %d%%', round(acc, 0))
```

```
% Error Plot
figure;
plot(abs(e));
xlabel('n'); ylabel('|Error|');
title('Absolute Value of the Error');
saveas(gcf, 'figures2/error_abs.png');

% Input vs Output Comparison
figure;
subplot(2,1,1);
stem(0:mend-mstart, sn_delayed(mstart:mend)); hold on;
stem(0:mend-mstart, y(mstart:mend));
legend('Input','Output');
title('Input (s[n]) vs Output (y[n])'); xlabel('n'); ylabel('Signal');

subplot(2,1,2);
stem(0:mend-mstart, sn_delayed(mstart:mend)); hold on;
stem(0:mend-mstart, sign(y(mstart:mend)));
legend('Input','Output');
title(sprintf('Input vs Output after Decision Block (%d%% accuracy)',
    round(acc, 0)));
xlabel('n'); ylabel('Signal');
saveas(gcf, 'figures2/input_vs_output.png');

% Output Amplitude
figure;
plot(abs(y)); hold on;
plot(ones(1,length(y)), '--r','LineWidth',1.5);
title('Output Amplitude'); xlabel('n');
saveas(gcf, 'figures2/output_amplitude.png');

% Equalization vs No Equalization
figure;
subplot(2,2,1);
delays = M-4;
hd = [zeros(1,delays), 1];
sn_delayed = -1 * conv(hd,sn);
stem(0:mend-mstart, sn_delayed(mstart:mend)); hold on;
stem(0:mend-mstart, sign(y(mstart:mend)));
legend('Input', 'Output (inverted)');
title('Input vs Output after Equalization'); xlabel('n'); ylabel('
    Signal');

subplot(2,2,2);
a = sn_delayed(mstart:mend) - sign(y(mstart:mend))';
stem(0:mend-mstart, a);
c = sum(a==0) / length(a) * 100;
title(sprintf('Error (accuracy = %d%%) with Equalization', round(c, 0))
    );
xlabel('n'); ylabel('Error');

subplot(2,2,3);
delays = 1;
hd = [zeros(1,delays), 1];
sn_delayed = conv(hd,sn);
stem(0:mend-mstart, sn_delayed(mstart:mend)); hold on;
stem(0:mend-mstart, sign(xn(mstart:mend)));
legend('Input','Output');
```

```
title('Input vs Output (Un-equalized)'); xlabel('n'); ylabel('Signal');

subplot(2,2,4);
stem(0:mend-mstart, sn_delayed(mstart:mend) - sign(xn(mstart:mend)));
c = sum(sn_delayed(mstart:mend) == sign(xn(mstart:mend))) / length(a) *
    100;
title(sprintf('Error (accuracy = %d%%) Un-equalized', round(c, 0)));
xlabel('n'); ylabel('Error');
saveas(gcf, 'figures2/equalization_comparison.png');

% Convergence of Equalizer
figure;
for i = 1:M+1
    subplot(3,4,i);
    plot(h_track(:,i));
    xlabel('n'); title(sprintf('Tap %d', i-1));
end
sgtitle('Equalizer Convergence Over Time');
saveas(gcf, 'figures2/equalizer_convergence.png');

% Convergence of Overall System
figure;
for i = 1:M+5
    subplot(4,4,i);
    plot(overall_lti_track(:,i)); hold on;
    if i ~= 8
        plot(zeros(1,length(sn)), '--r');
    else
        plot(-1*ones(1,length(sn)), '--r');
    end
    xlabel('n'); title(sprintf('Tap %d', i-1));
end
sgtitle('Overall LTI System Convergence');
saveas(gcf, 'figures2/lti_convergence.png');

% Frequency and Impulse Response
figure;
subplot(3,2,1);
[channel_f, w] = DTFT(h_channel);
plot(w, abs(channel_f));
xlabel('\omega (rad/s)'); ylabel('Gain'); xlim([0, pi]);
title('Channel Frequency Magnitude');

subplot(3,2,2);
plot(w, angle(channel_f));
xlabel('\omega (rad/s)'); ylabel('Phase'); xlim([0, pi]);
title('Channel Frequency Phase');

subplot(3,2,3);
[h_f, ~] = DTFT(h);
plot(w, abs(h_f));
xlabel('\omega (rad/s)'); ylabel('Gain'); xlim([0, pi]);
title('Equalizer Frequency Magnitude');

subplot(3,2,4);
plot(w, angle(h_f));
xlabel('\omega (rad/s)'); ylabel('Phase'); xlim([0, pi]);
title('Equalizer Frequency Phase');
```

```
subplot(3,2,5);
plot(w, abs(h_f .* channel_f'));
hold on;
plot(w, ones(1,length(w)), '--r');
xlabel('\omega (rad/s)'); ylabel('Gain'); xlim([0, pi]);
title('Overall Frequency Magnitude');

subplot(3,2,6);
plot(w, angle(h_f .* channel_f'));
xlabel('\omega (rad/s)'); ylabel('Phase'); xlim([0, pi]);
title('Overall Frequency Phase');
saveas(gcf, 'figures2/frequency_response.png');

% Impulse Responses
figure;
subplot(3,1,1);
stem(0:length(h_channel)-1, h_channel);
xlabel('n'); ylabel('h[n]');
title('Channel Impulse Response');

subplot(3,1,2);
stem(0:length(h)-1, h);
xlabel('n'); ylabel('h[n]');
title('Equalizer Impulse Response');

subplot(3,1,3);
a_lti = conv(h,h_channel);
stem(0:length(a_lti)-1, a_lti);
xlabel('n'); ylabel('h[n]');
title('Overall LTI System Impulse Response');
saveas(gcf, 'figures2/impulse_response.png');
```