Adaptive filter algorithms for system identification

[[1]](#footnote-1)

***Abstract* —** The project focuses on solving the problem of echo in phone line communication. Echo occurs when a signal reflects back from the near-end to the far-end, degrading the communication quality. The project aims to develop a system that cancels out the echo using an adaptive filter. By analyzing the incoming signal and the corresponding echo, the system learns to remove the unwanted echo and improve communication clarity. The performance of the system is evaluated by comparing the input signal, echo signal, and the error signal. The goal is to enhance the quality of phone line communication by effectively eliminating echo.

# INTRODUCTION

Objective: The objective is to explain the occurrence of line echo in phone line communication, where signals reflecting at the near-end result in an echo received by the far-end speaker. The objective is to highlight the need for a line echo cancellation system to address this issue and improve communication quality.

# Problem Specification

The problem is the presence of echo in phone line communication, where the signal from the far-end is reflected at the near-end and results in an attenuated replica reaching the far-end speaker. This echo interferes with the desired signal and degrades the quality of communication. The objective is to develop an adaptive line echo canceller (LEC) system that utilizes the far-end signal as input and its reflected version as the reference. The goal is to effectively eliminate the echo and enhance voice quality in phone line communication.

# Evaluation Criteria

The evaluation criteria for the line echo cancellation system include assessing its echo cancellation performance, voice quality improvement, robustness to circuit mismatches, adaptability to varying echo conditions, and computational efficiency. These criteria are used to determine the system's effectiveness, its ability to enhance voice quality, handle different scenarios, adapt to changes, and operate in real-time.

# Approach

To meet the requirements in this scenario, Matlab is utilized. The input signal is transformed into discrete values using real integer indices. The desired output signal is calculated by performing a convolution of the input signal with the transfer function represented by the FIR system (h[n] \* x[n]). This can be achieved either by doing the convolution in the time domain or by using the Z-transform to convert both signals to the Z-domain and then performing the multiplication using Z-transform properties. The Matlab filter () function can be used for this. A signal of the same length as the input signal is generated. The learning factor must first be chosen within its valid range, followed by defining the filter length, which is crucial for the estimation process. To estimate the coefficients, a loop is used to iteratively apply a new window size to each value of the input signal. The loop repeats as many times as the length of the input or desired signal. The transposed windowed data from the input signal is multiplied with the estimated matrix, which has M columns and one row, each time. Alternatively, an estimated values matrix with one column and M rows can be used. In this case, the estimated values vector is transposed (w). The result of the multiplication is used to calculate the error, which is then used to update the estimated values for the next iteration.

# Algorithms used

The Least Mean Squares (LMS) algorithm is a method used for adapting the coefficients in a linear filter to minimize the mean square error between the desired output and the actual output of the filter. The algorithm works by using a sliding window of input data to update the estimated coefficients at each sample. The process starts with initial estimates for the coefficients, which are stored in a matrix called "w(n)". This matrix is then multiplied by the current window of input data "x(n)" to produce an output "y(n)". The difference between the desired output "d(n)" and the actual output "y(n)" is calculated to give the error "e(n)”. The LMS algorithm then updates the estimated coefficients by adding a factor of "2 × μ × e(n) × x(n)" to the current coefficients. This factor is determined by the error and the current window of input data, and helps to adjust the coefficients so that the mean square error is reduced. The process is repeated at each sample until the coefficients converge to the optimal values.

M: filter length,

µ:step-size factor,

x(n): input data to the adaptive filter of length N (Vector),

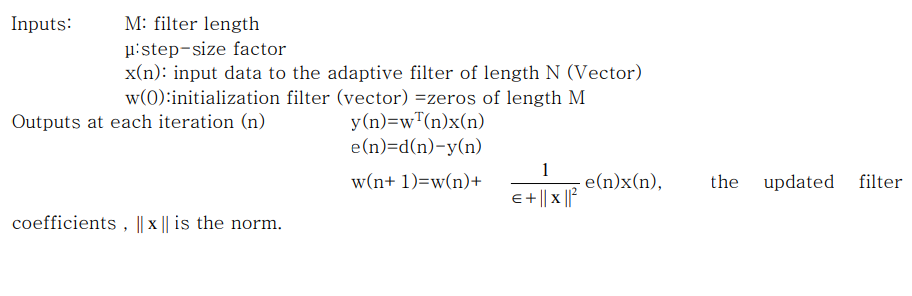
w(0) :initialization filter (vector) =zeros of length M,

Outputs at each iteration (n) y(n) = (n) x(n),

calculate error : 𝑒(𝑛) = 𝑑(𝑛) − 𝑦(𝑛) ,

the error at n sample 𝑤(𝑛 + 1) = 𝑤(𝑛) + e(n) x(n) the updated filter coefficients , || x || is the norm.

Figure 1



2- The Recursive Least Squares (RLS) algorithm is a type of adaptive filtering algorithm used to estimate the parameters of a linear system in real-time. The RLS algorithm is an adaptive filter that is well suited for tracking slowly changing system parameters, as it has faster convergence and better stability compared to the Least Mean Squares (LMS) algorithm. The basic idea behind the RLS algorithm is to estimate the parameters of the system using a recursive procedure. The algorithm uses a matrix called the "inverse correlation matrix" to track the parameters of the system and update the estimates at each time step. The inverse correlation matrix is updated recursively based on the current input and output data, and the estimates of the parameters are obtained by multiplying this matrix with the current output data. The RLS algorithm uses a forgetting factor, which determines the importance of older data in the calculation of the inverse correlation matrix. This factor allows the algorithm to adapt to changes in the system over time, and to effectively track slowly changing system parameters. In summary, the RLS algorithm is an effective method for estimating the parameters of a linear system in real-time, and it is especially well suited for tracking slowly changing system parameters.

M: filter length.

where w is the estimated values vector, the transpose is to make the matrix multiplication possible

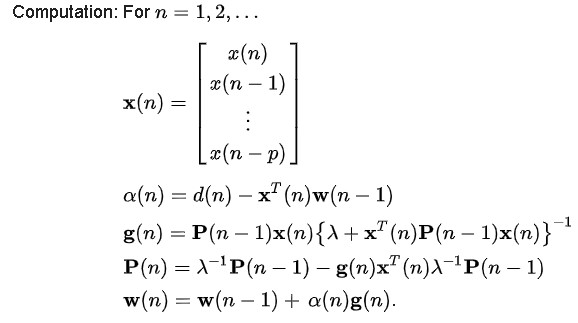
, where j is index starts from 1

, updates the estimated values.

As was already mentioned, the RLS algorithm adds a new component known as the Kalman gain (k), which varies depending on the value of the current window and some mathematical operations. The estimated values are directly impacted by this factor.

Figure 2

LMS coefficients update criteria

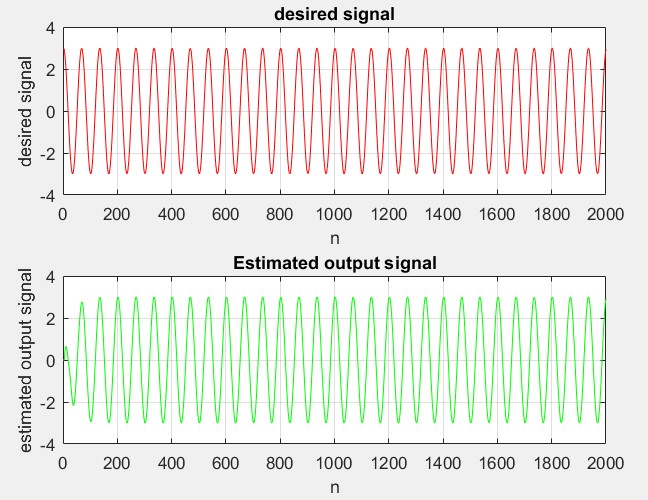


# results and analysis

Although there will undoubtedly be some error between the two signals, the estimated signal should resemble the desired signal for evaluation purposes. It's important to check this difference in the frequency domain because it might not be recognizable in the time domain. The estimation's yielded coefficients might not exactly match the desired coefficients. The desired signal and the estimated signal are shown in the time domain in the following figure. :

Figure 3

Estimated and desired signals in time domain



The Bode plot for the desired signal response is displayed above. The plot depicts that the estimated signal is almost equal to the desired signal, but with some noise or error present. To analyze the frequency response of the estimated signal, one can either use the freqz() function or create the transfer function in the Z-domain and plot it on a Bode plot. The transfer function can be expressed as transferFunc=tf(w,[1,…0],[]), where "w" stands for the estimated values and the numerator coefficients in the Z-domain, while the second parameter symbolizes the denominator and depends on the length of the filter.

## **PART 1**

1. in this point the input x[n] = cos(0.03πn) has been generated and plotted for 2000 samples.

Figure 4

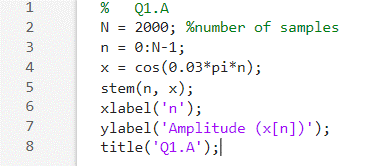
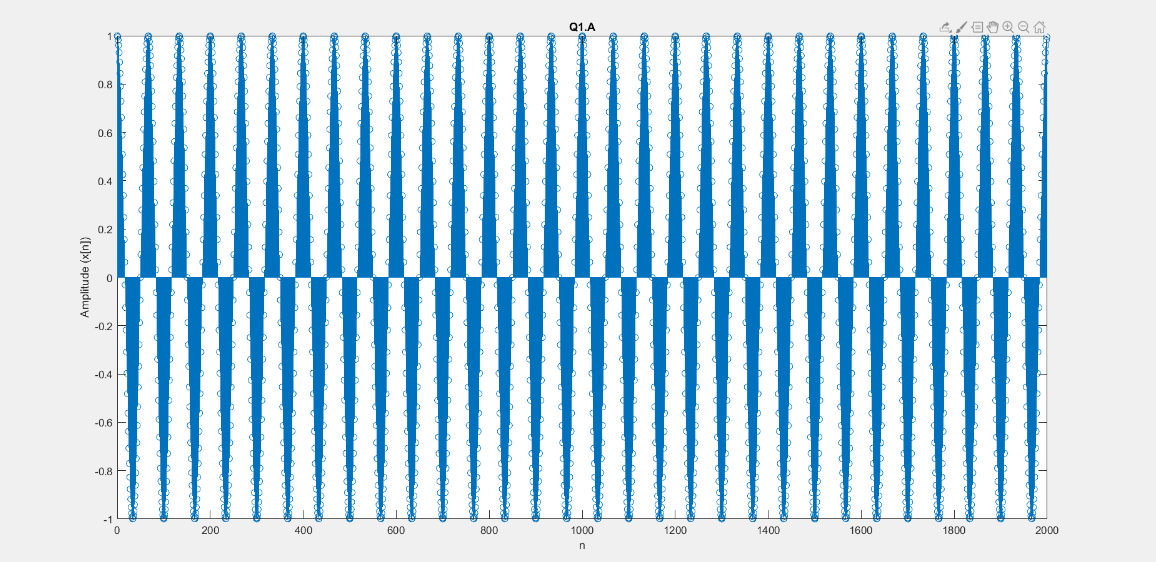
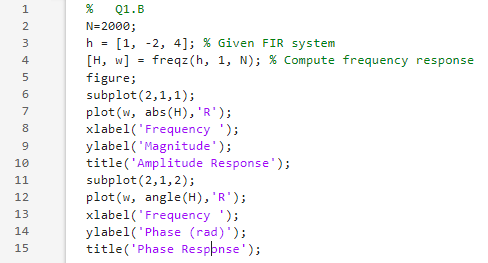


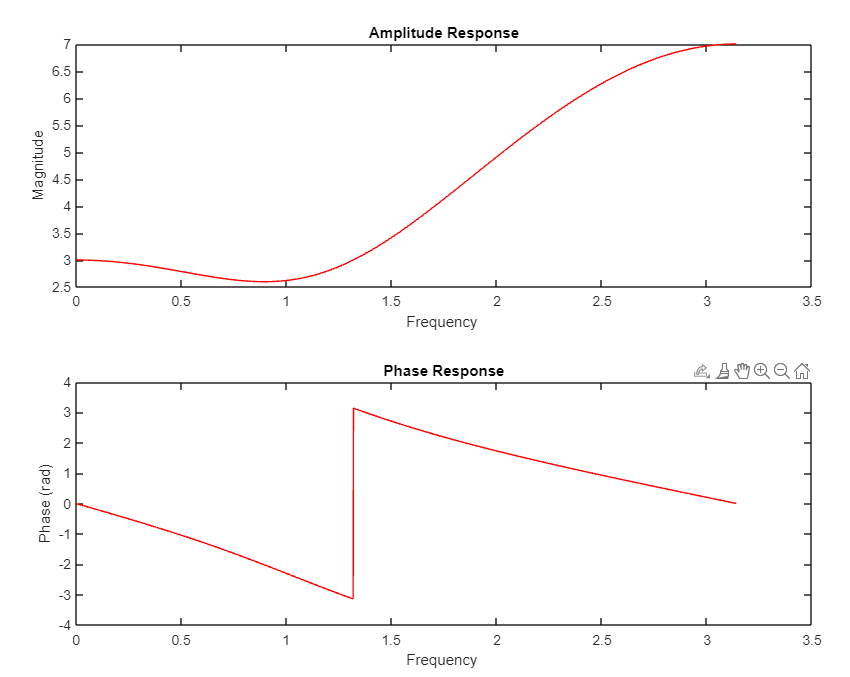
Figure 5



1. from the given FIR system, we can notice that the coefficient are (1,-2,4) and the filter length is 3 so we can implement the below code to plot the amplitude and phase response for the FIR

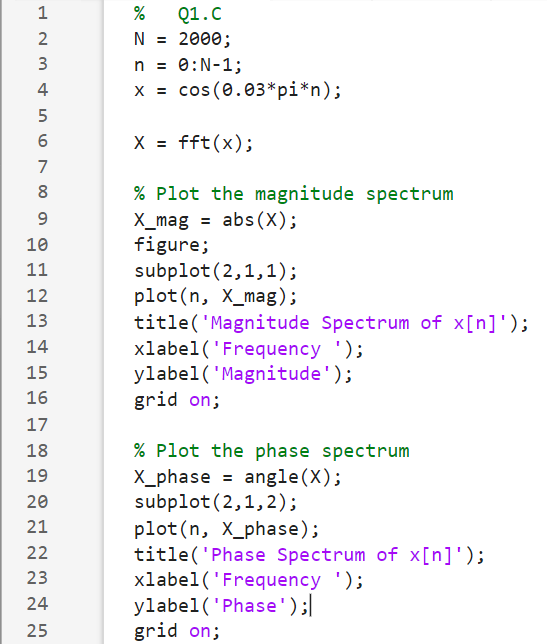
Figure 6





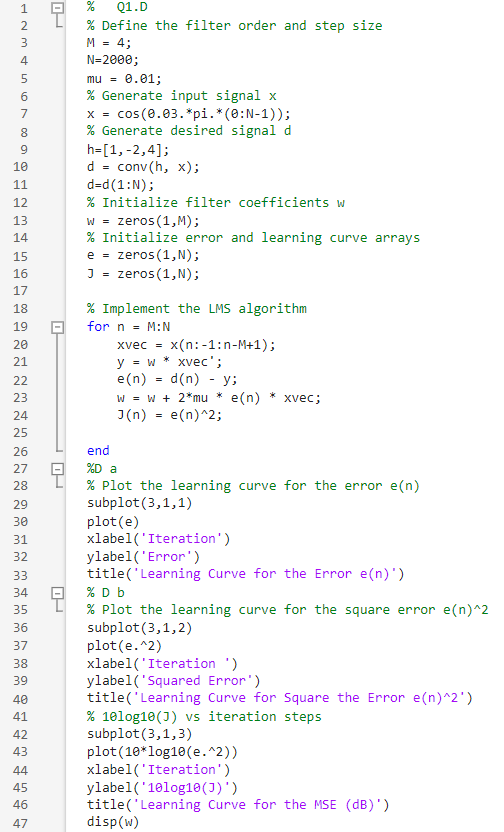
1. by taking the Fourier transform for the input signal x[n] we can plot the spectrum for the signal as shown below

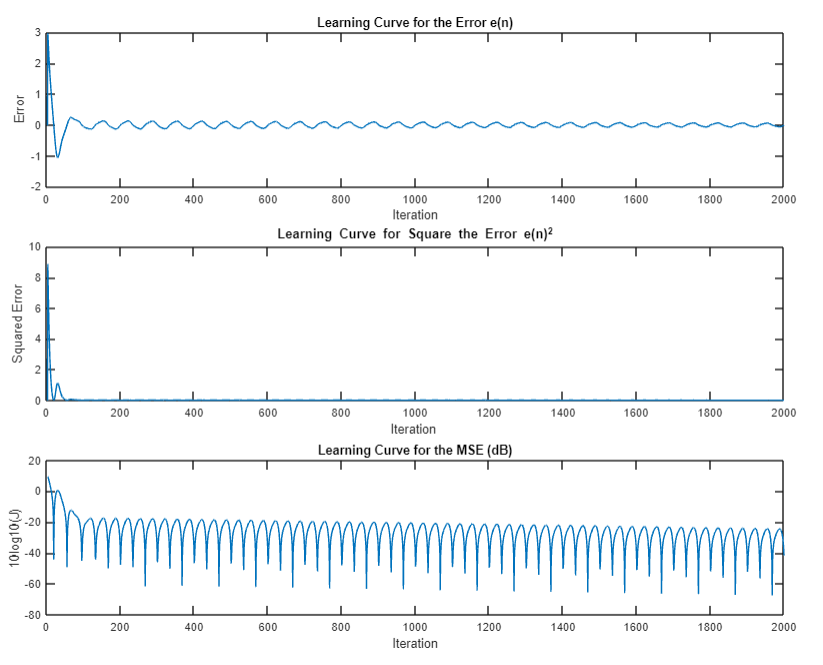
Figure 7



1. in the figure below LMS algorithm has been implemented with µ factor=0.01 then learning curves drawn.

Figure 8, 9

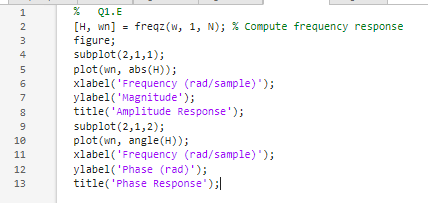


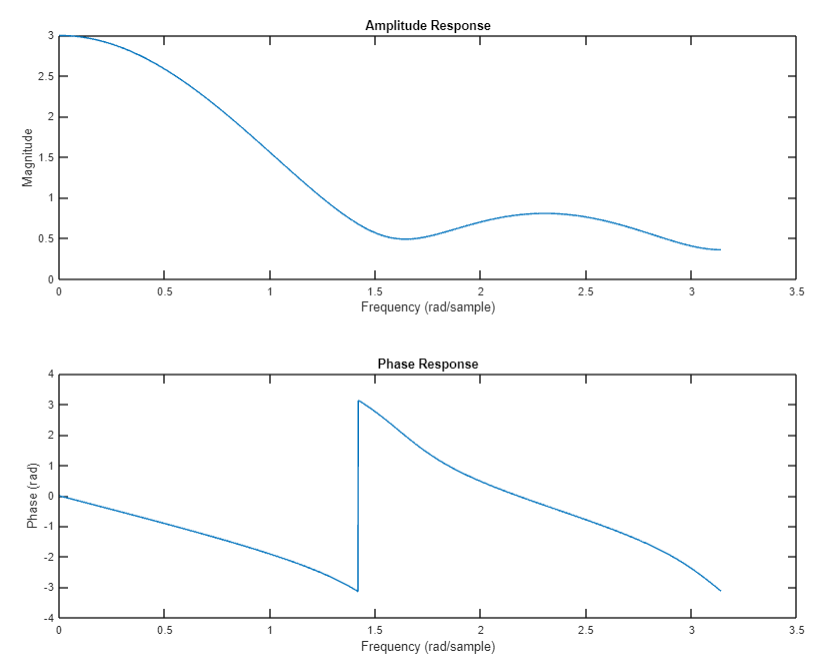


Here we notice that the error is decreases after n samples to be almost zero which means the leaning is done and the unknown filter coefficients is known.

1. Here in this part we calculate the frequency response to plot the amplitude and phase response to filter coefficients found in part d.

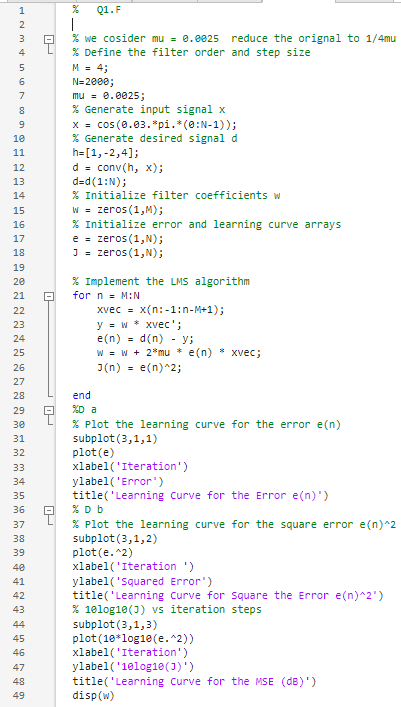
Figure 10, 11

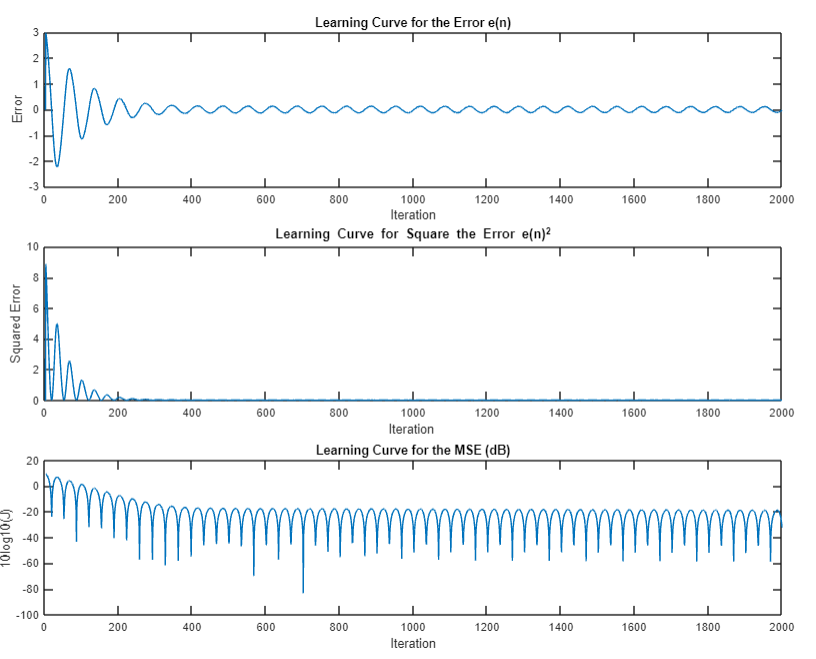




1. In the figure below we have two shots the first when µ equal=0.1and the other we decrease it to 0.001 then we noticed that when it decreased the steady sate error will increased and the learning process will be slow.

Figure 12, 13



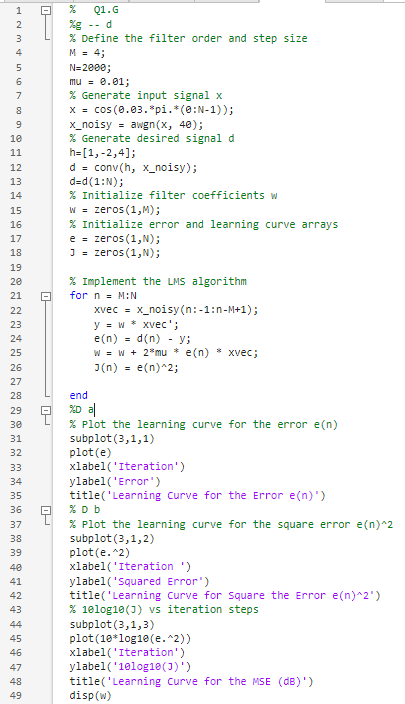


we noticed that when it decreased the steady sate error will have increased and the learning process will be slow.

1. to estimate the filter coefficients for a given input signal and desired signal.

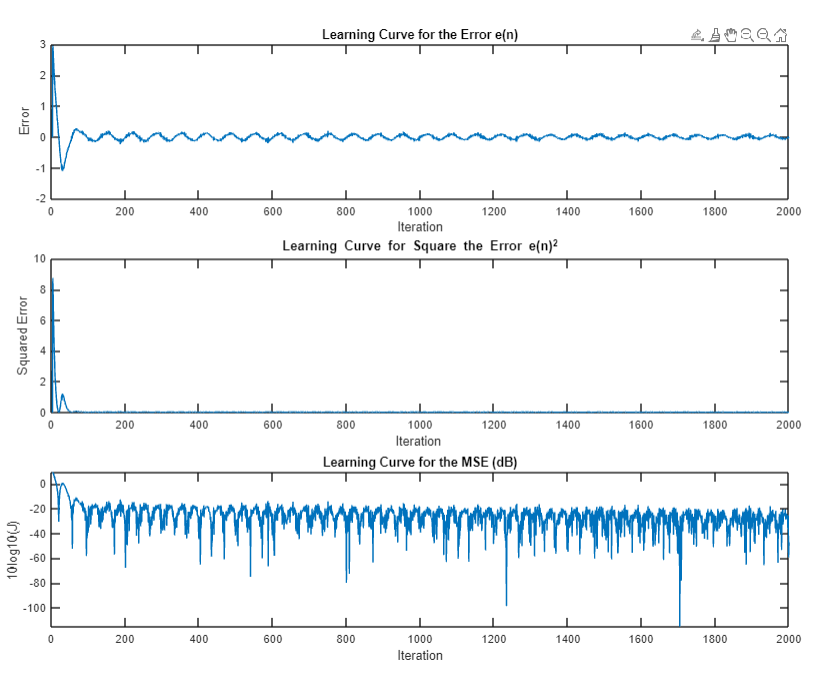
* When we repeat D , mu=0.01

Figure 14



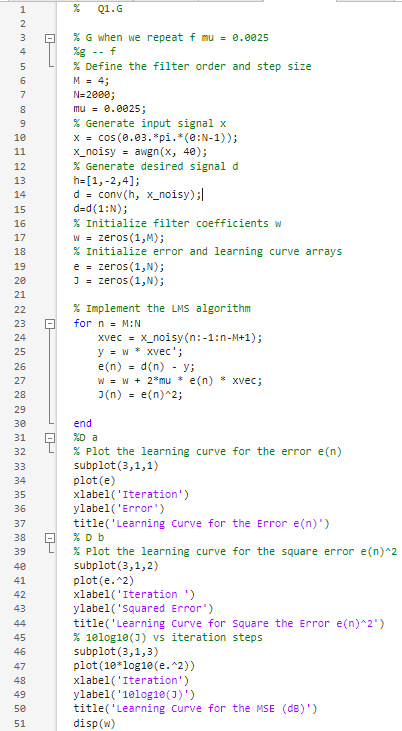
And the results are depicted in the figure 15

Figure 15



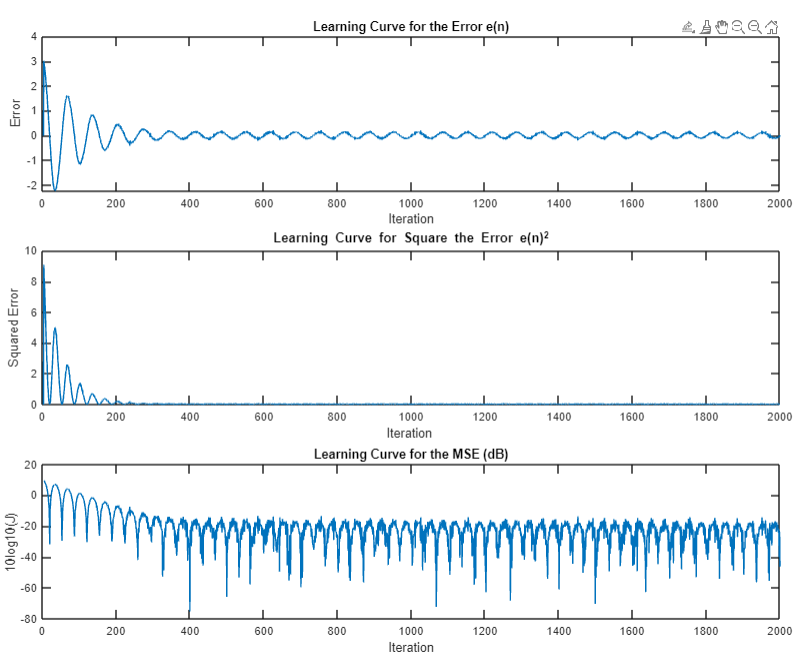
* When we repeat f, mu=0.0025

Figure 16



The results will be as shown in Figure 17

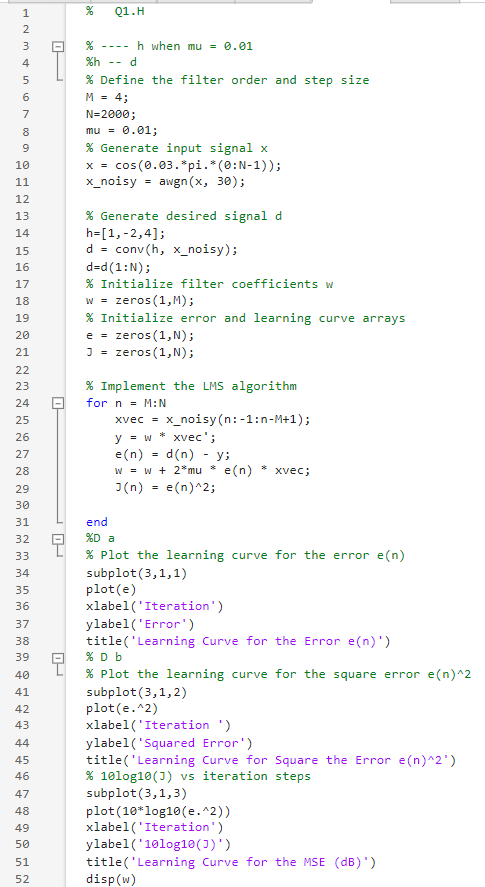
Figure 17



1. to modify the step size value

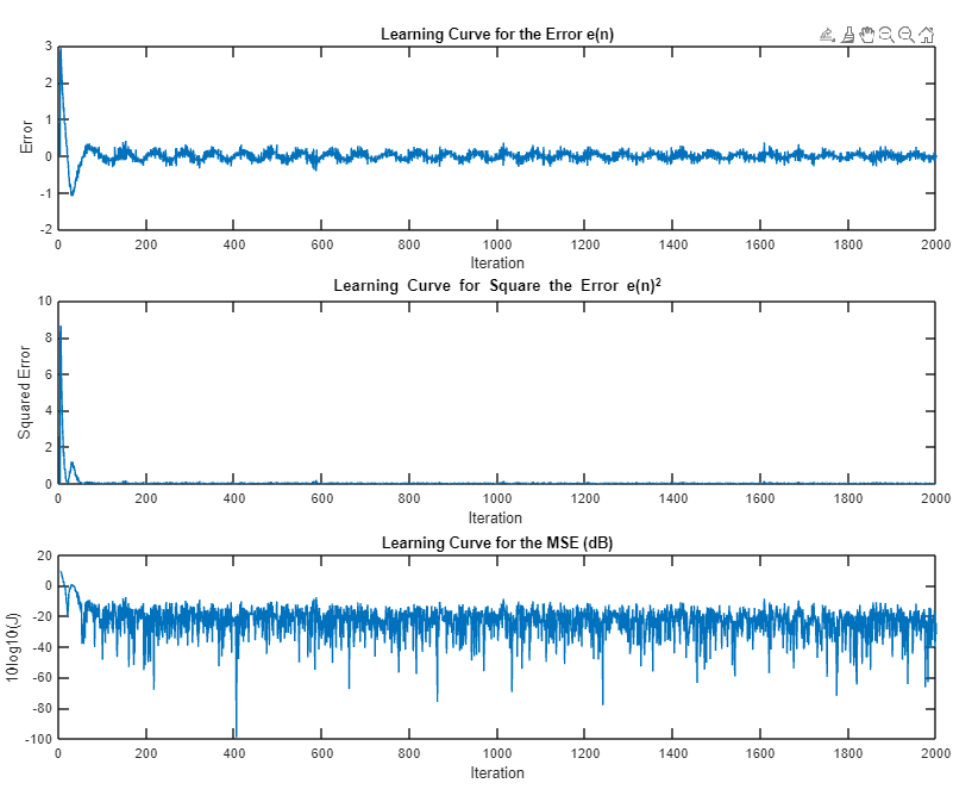
* When we repeat D , mu=0.01 for 30dB

Figure 18



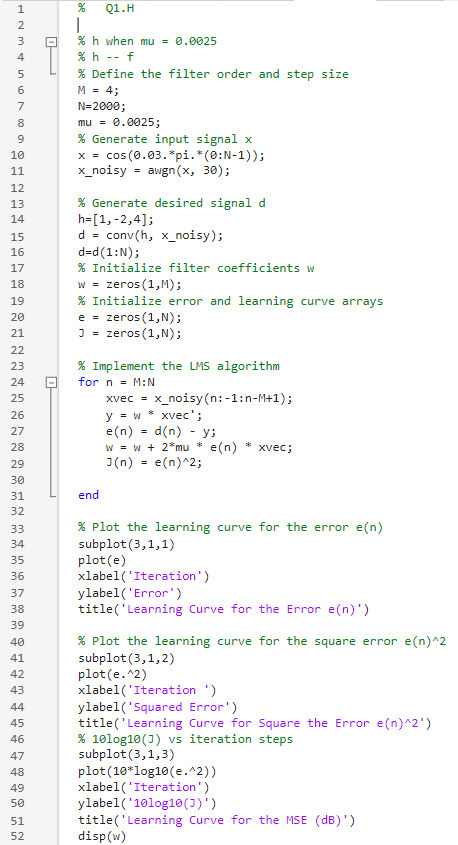
And the results are depicted in the figure 19

Figure 19



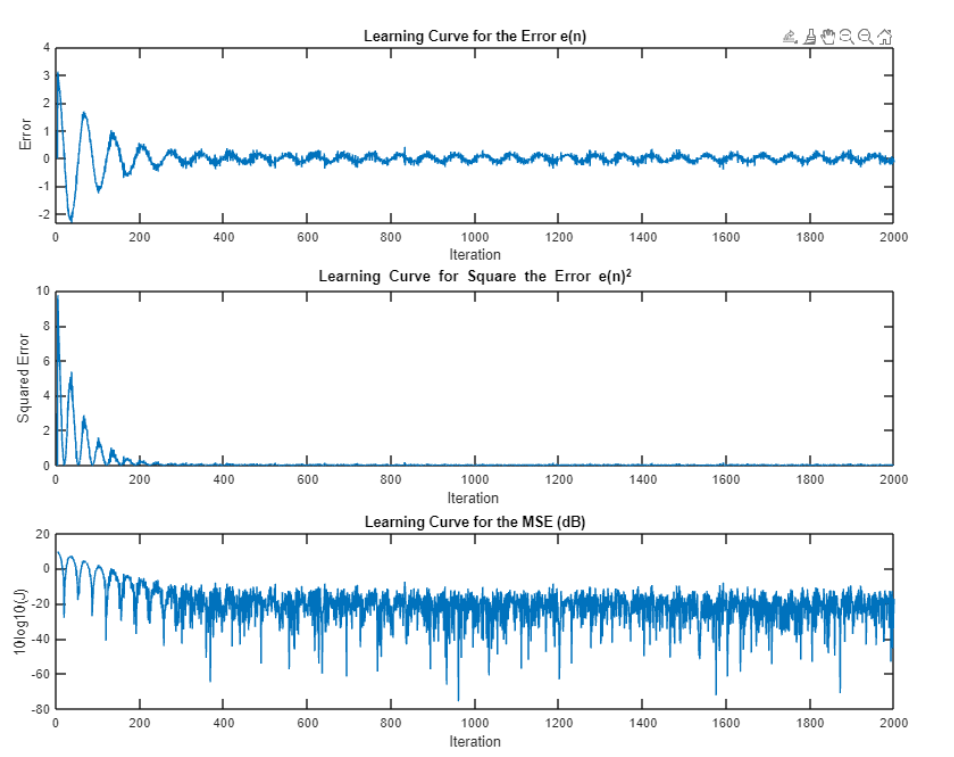
* When we repeat f , mu=0.0025 for 30dB

Figure 20



The results will be as shown in Figure 21

Figure 21



1. After introducing noise to the signal, an error must be read 1000 times to estimate the best error (ensemble average). There are various errors in computing the average:  
    Figure 22

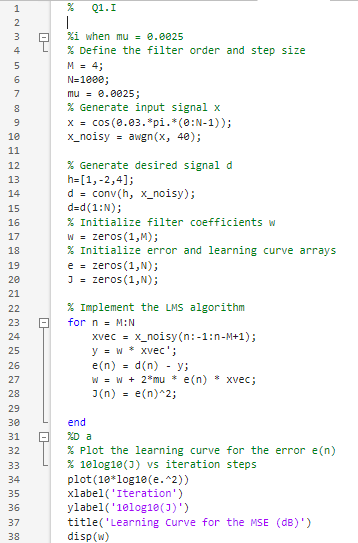
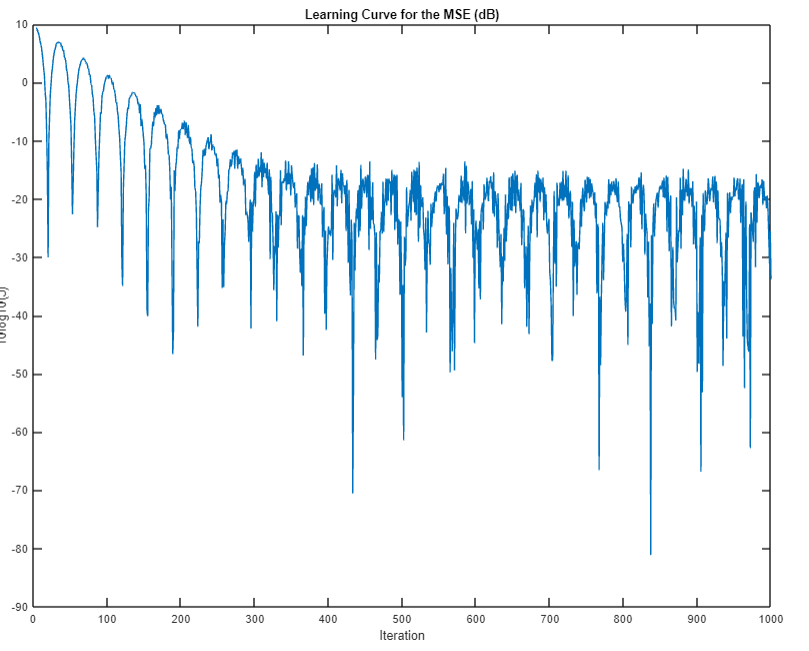


Figure 22

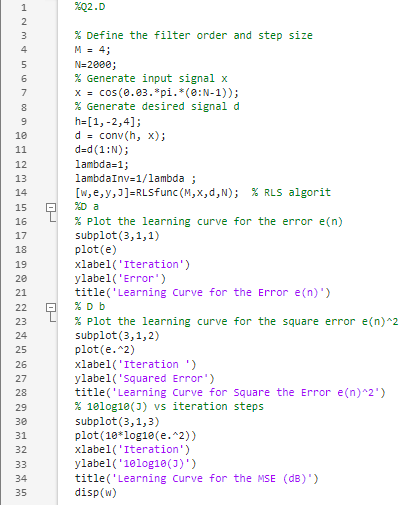


## **PART 2**

**Here we re-solved the questions using RLS algorithm**

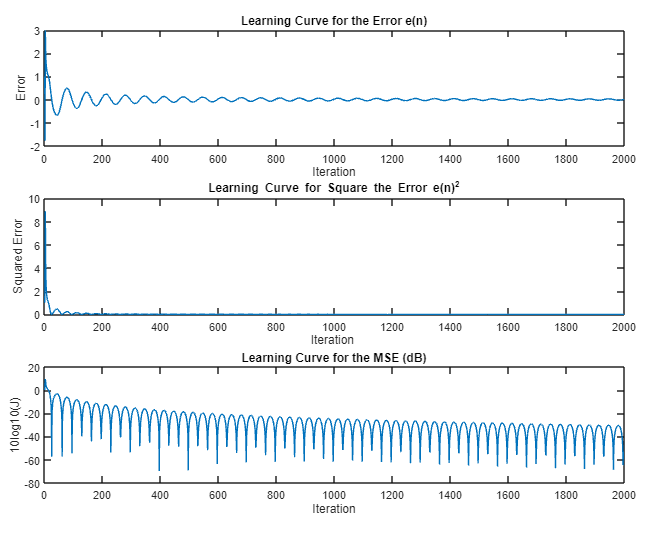
1. For the algorithm RLS , the value of forgetting factor Lambda (λ) should be 0.998 ≤ λ ≤ 1 , here the error signal when RLS algorithm used when λ=1:

Figure 23



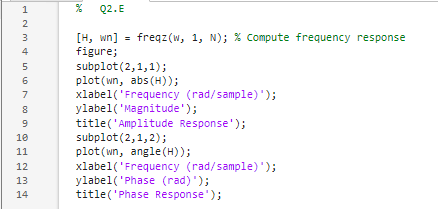
The results will be as shown in Figure 24

Figure 24



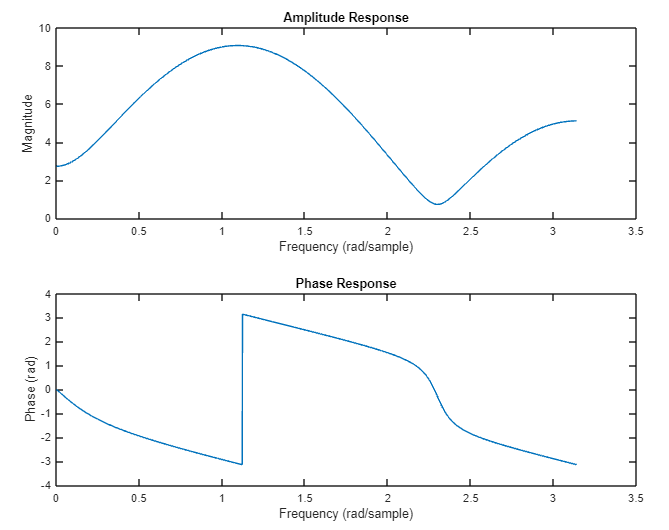
1. the frequency response when RLS algorithm is used

Figure 25



Next figure shows the frequency response when RLS algorithm is used :

Figure 26



1. To estimate the filter coefficients for a given input signal and desired signal. Using RLS algorithm.

Figure 27

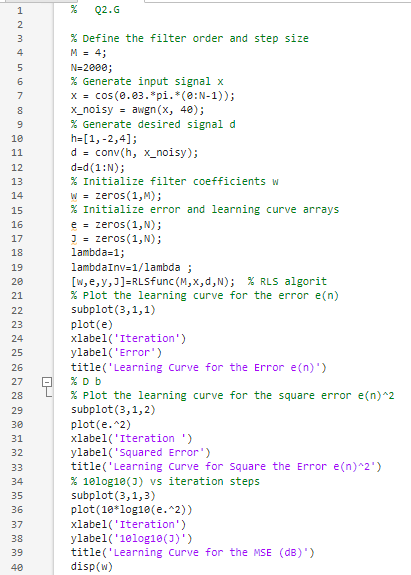
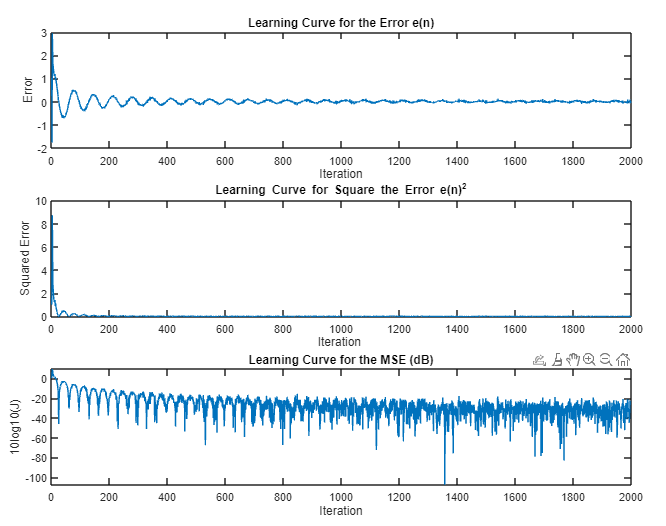


Figure 28



1. To modify the step size value. Using RLS algorithm.

Figure 29

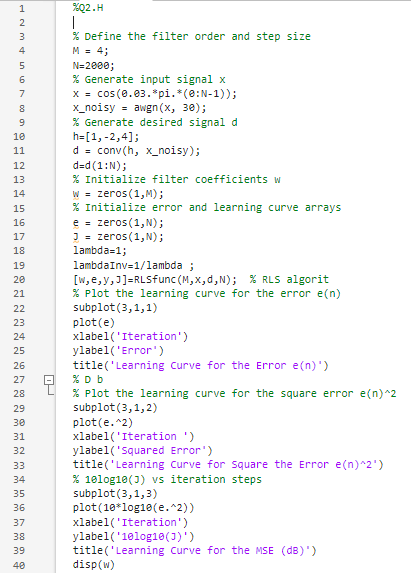
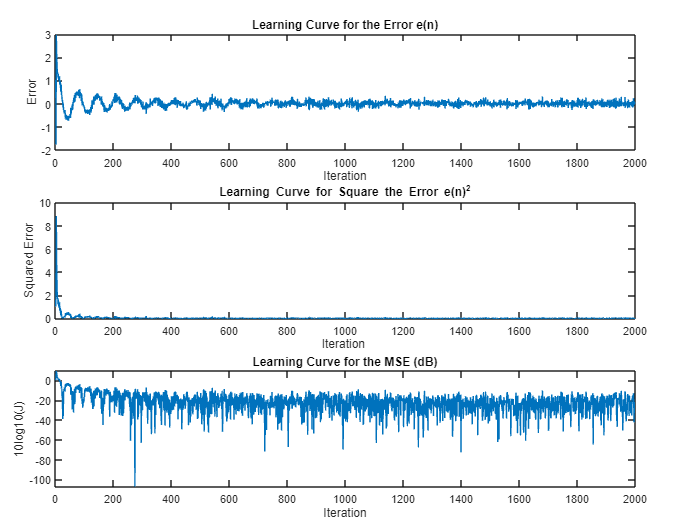


Figure 30



1. Using RLS

Figure 31

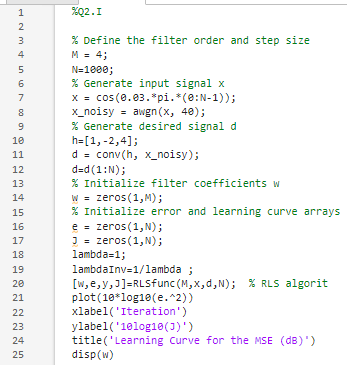
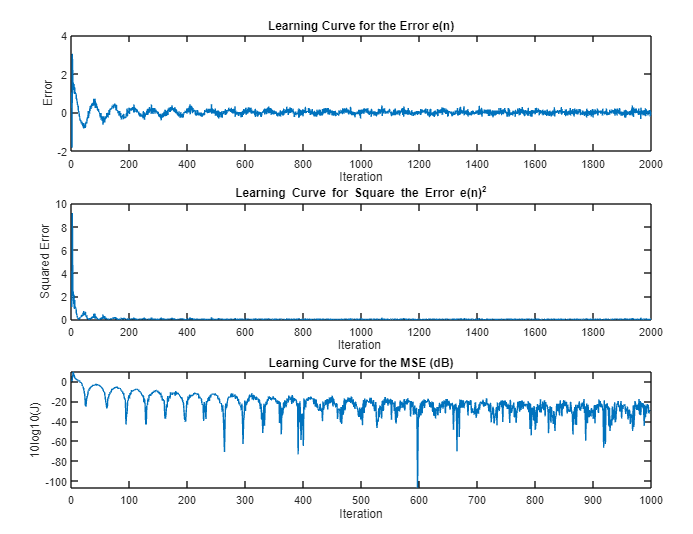


Figure 32



1. Conclusion

In this project we learned more about how to find the coefficients of an unknown filter using adaptive filters which try to generate the frequency response of the unknown filter while minimizing the error, by implementing two important and widely used algorithms in real life.

Adaptive filters are critical in system identification. They are used to calculate the coefficients and build the predicted signal for the system. For signal estimating, different algorithms can be used, including least mean squares (LMS) and recursive least squares (RLS), among others.

The LMS algorithm's performance is hugely affected by the value of the learning factor, which influences the estimate of the coefficients at each step of the algorithm's implementation. Changing its value can have an impact on the estimating process and the incorrect output. Similarly, the Kalman gain controls the estimation mechanism in the RLS method, and the forgetting factor, denoted by lambda (), influences the estimating process.

1. References

[1]<https://www.diva-portal.org/smash/get/diva2:280596/fulltext01>

[2] <https://en.wikipedia.org/wiki/Least_mean_squares_filter>

[3]<https://www.keil.com/pack/doc/CMSIS/DSP/html/group__LMS.html>

1. Appendix

**The Full Code of Part 1:**

% Q1.A

N = 2000; %number of samples

n = 0:N-1;

x = cos(0.03\*pi\*n);

stem(n, x);

xlabel('n');

ylabel('Amplitude (x[n])');

title('Q1.A');

% -----------------------------------------------B

% Q1B

N=2000;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

h = [1, -2, 4]; % Given FIR system

[H, wn] = freqz(h, 1, N);

figure;

subplot(2,1,1);

plot(wn, abs(H),'R');

xlabel('Frequency ');

ylabel('Magnitude');

title('Amplitude Response');

subplot(2,1,2);

plot(wn, angle(H),'R');

xlabel('Frequency ');

ylabel('Phase (rad)');

title('Phase ')

%-------------------------------------------------

%-----------------------------D&E in same script

% Define the filter order and step size

M = 4;

N=2000;

mu = 0.01;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

% Implement the LMS algorithm

for n = M:N

xvec = x(n:-1:n-M+1);

y = w \* xvec';

e(n) = d(n) - y;

w = w + 2\*mu \* e(n) \* xvec;

J(n) = e(n)^2;

end

%D a

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% D b

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

%E

[H, wn] = freqz(w, 1, N); % Compute frequency response

figure;

subplot(2,1,1);

plot(wn, abs(H));

xlabel('Frequency (rad/sample)');

ylabel('Magnitude');

title('Amplitude Response');

subplot(2,1,2);

plot(wn, angle(H));

xlabel('Frequency (rad/sample)');

ylabel('Phase (rad)');

title('Phase Response');

%-----------------------end of D&E-------------

%--- F -- we cosider mu = 0.0025 reduce the orignal to 1/4mu

% Define the filter order and step size

M = 4;

N=2000;

mu = 0.0025;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

% Implement the LMS algorithm

for n = M:N

xvec = x(n:-1:n-M+1);

y = w \* xvec';

e(n) = d(n) - y;

w = w + 2\*mu \* e(n) \* xvec;

J(n) = e(n)^2;

end

%D a

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% D b

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

%% Now part G when we repeat D mu = 0.01

%g -- d

% Define the filter order and step size

M = 4;

N=2000;

mu = 0.01;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 40);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

% Implement the LMS algorithm

for n = M:N

xvec = x\_noisy(n:-1:n-M+1);

y = w \* xvec';

e(n) = d(n) - y;

w = w + 2\*mu \* e(n) \* xvec;

J(n) = e(n)^2;

end

%D a

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% D b

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

% ------------------ G when we repeat f mu = 0.0025

%g -- f

% Define the filter order and step size

M = 4;

N=2000;

mu = 0.0025;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 40);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

% Implement the LMS algorithm

for n = M:N

xvec = x\_noisy(n:-1:n-M+1);

y = w \* xvec';

e(n) = d(n) - y;

w = w + 2\*mu \* e(n) \* xvec;

J(n) = e(n)^2;

end

%D a

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% D b

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

%% ----- the end of f

% ---- h when mu = 0.01

%h -- d

% Define the filter order and step size

M = 4;

N=2000;

mu = 0.01;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 30);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

% Implement the LMS algorithm

for n = M:N

xvec = x\_noisy(n:-1:n-M+1);

y = w \* xvec';

e(n) = d(n) - y;

w = w + 2\*mu \* e(n) \* xvec;

J(n) = e(n)^2;

end

%D a

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% D b

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

%% --------------------------h when mu = 0.0025

%h -- f

% Define the filter order and step size

M = 4;

N=2000;

mu = 0.0025;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 30);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

% Implement the LMS algorithm

for n = M:N

xvec = x\_noisy(n:-1:n-M+1);

y = w \* xvec';

e(n) = d(n) - y;

w = w + 2\*mu \* e(n) \* xvec;

J(n) = e(n)^2;

end

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

% ---------------------- i

%i when mu = 0.01

% Define the filter order and step size

M = 4;

N=1000;

mu = 0.01;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 40);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

% Implement the LMS algorithm

for n = M:N

xvec = x\_noisy(n:-1:n-M+1);

y = w \* xvec';

e(n) = d(n) - y;

w = w + 2\*mu \* e(n) \* xvec;

J(n) = e(n)^2;

end

%D a

% Plot the learning curve for the error e(n)

% 10log10(J) vs iteration steps

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

%%% --------------------------------- i when mu = 0.0025

%i when mu = 0.0025

% Define the filter order and step size

M = 4;

N=1000;

mu = 0.0025;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 40);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

% Implement the LMS algorithm

for n = M:N

xvec = x\_noisy(n:-1:n-M+1);

y = w \* xvec';

e(n) = d(n) - y;

w = w + 2\*mu \* e(n) \* xvec;

J(n) = e(n)^2;

end

%D a

% Plot the learning curve for the error e(n)

% 10log10(J) vs iteration steps

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

**The Full Code of Part 2:**

%Part2 D

% Define the filter order and step size

M = 4;

N=2000;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x);

d=d(1:N);

lambda=1;

lambdaInv=1/lambda ;

[w,e,y,J]=RLSfunc(M,x,d,N); % RLS algorit

%D a

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% D b

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

%---------------------------

%Part2 E

[H, wn] = freqz(w, 1, N); % Compute frequency response

figure;

subplot(2,1,1);

plot(wn, abs(H));

xlabel('Frequency (rad/sample)');

ylabel('Magnitude');

title('Amplitude Response');

subplot(2,1,2);

plot(wn, angle(H));

xlabel('Frequency (rad/sample)');

ylabel('Phase (rad)');

title('Phase Response');

%---------------------------

%part2 G

% Define the filter order and step size

M = 4;

N=2000;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 40);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

lambda=1;

lambdaInv=1/lambda ;

[w,e,y,J]=RLSfunc(M,x,d,N); % RLS algorit

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% D b

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

%---------------------------

%part2 H

% Define the filter order and step size

M = 4;

N=2000;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 30);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

lambda=1;

lambdaInv=1/lambda ;

[w,e,y,J]=RLSfunc(M,x,d,N); % RLS algorit

% Plot the learning curve for the error e(n)

subplot(3,1,1)

plot(e)

xlabel('Iteration')

ylabel('Error')

title('Learning Curve for the Error e(n)')

% D b

% Plot the learning curve for the square error e(n)^2

subplot(3,1,2)

plot(e.^2)

xlabel('Iteration ')

ylabel('Squared Error')

title('Learning Curve for Square the Error e(n)^2')

% 10log10(J) vs iteration steps

subplot(3,1,3)

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

%---------------------------

%Part2 I

% Define the filter order and step size

M = 4;

N=1000;

% Generate input signal x

x = cos(0.03.\*pi.\*(0:N-1));

x\_noisy = awgn(x, 40);

% Generate desired signal d

h=[1,-2,4];

d = conv(h, x\_noisy);

d=d(1:N);

% Initialize filter coefficients w

w = zeros(1,M);

% Initialize error and learning curve arrays

e = zeros(1,N);

J = zeros(1,N);

lambda=1;

lambdaInv=1/lambda ;

[w,e,y,J]=RLSfunc(M,x,d,N); % RLS algorit

plot(10\*log10(e.^2))

xlabel('Iteration')

ylabel('10log10(J)')

title('Learning Curve for the MSE (dB)')

disp(w)

1. [↑](#footnote-ref-1)