EECS 241B: DIGITAL COMMUNICATION - II

PROJECT

Jyotica Yadav; SID: 48084242

Adaptive Filtering:

An adaptive filter is a filter that self-adjusts its tap weights or the overall transfer function according to an optimization algorithm in order to accurately track the desired response. The main objective of the filtering algorithm is to reduce the error between the output produced by the filter and the desired response of the signal. There are different approaches used in adaptive filtering of which few are implemented in the project using MATLAB.

[1] Stochastic gradient approach:

In this approach, gradient vector is calculated which is defined as the derivative of the mean squared error with respect to tap weight vector and based on that, a guess is made on the tap weight vector in a direction opposite to that of the gradient vector. Least mean square estimate and Normalized least mean square approach falls under the category of stochastic gradient algorithms.

<u>Least Mean Square Estimation (LMS):</u>

LMS algorithm traces a desired response by adjusting its tap weights with an aim of reducing the error (difference between the desired response and the output produced by the filter). In the case of negative gradient, tap weights have to be increased by a certain amount and in the case of positive gradient, tap weights have to be reduced.

Approach:

1. First step involves the generation of transmitted symbols which are a sequence of 1's and -1's. Number of transmitted symbols is given to be 500. A loop is placed on a random sequence of 0s and 1s generated using MATLAB and the formula $2u - 1$ is used for the data conversion into the specified format.
2. Next step is to generate the sequence of 'v' which is the input to the equalizer. Channel coefficients and noise power is given in the question.
   Formulas used for generating v are:
$$x(n) = h(n) * h^*(-n)$$

Where h are the channel coefficients

$$v(n) = f_n * u(n) + \eta$$

f_n – Coefficients of F(z) where F(z) is the causal and stable part of X(z)

$u(n)$- Input data

η= Noise power

3. Implementation of the LMS algorithm:

Desired response is obtained by shifting the input signal by 7 and padding the vacant spots with zeros. The number of tap weights is given to be 11. Initial tap vector is taken to be all zeros except the middle element which is assumed to be 1.
Formulas:

$$y(n) = w^H(n)v(n)$$

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

$$w(n+1) = w(n) + \mu v(n)e^*(n)$$

Where $y(n)$ is the output of the filter

$e(n)$ is the error generated

$d(n)$ is the desired signal

$w(n)$ is the tap weight vector

μ is the step size (0.0550, 0.0275, 0.0138)

Code:

a) Calling function of equalization algorithm

```
clc;

clear all;
close all;

Iterations = 200;                 % Number of Iterations
number_of_Symbols= 500;           % Number of Transmitted symbols
h = [0.2194, 1.000, 0.2194];      % Channel Coefficients
noise_power = 0.001;              % Defined noise power

u = randn(Iterations, number_of_Symbols);
% Initializations for u and v

v = randn(Iterations, number_of_Symbols + length(h)-1);
% Length of the convolutional sum is N+M-1 where N and M is the length of the
%individual sequences

N = 7;                            % Desired response is N shifted version of u
tap_weights = 11;                 % Number of tap weights defined
step_size_LMS = 0.0550;           % Step size defined

for i=1:Iterations
        u(i,:) = Gen_trans_sym(number_of_Symbols);
        % Creating an array of transmitted symbols [1,-1]

        v(i,:) = Gen_v(h, noise_power, u(i,:));
        % Generating the sequence vn

        % Least Mean Square (LMS) equalization algorithm
        [y_LMS(i,:), e_LMS(i,:)]= LMS_algorithm_check(step_size_LMS, tap_weights,
        number_of_Symbols, u(i,:), v(i,:), N);
```

```
        plot(abs(e_LMS(i,:)),'b')
        % Plotting the absolute value of error for each iteration

        hold on;
  end

  title("Convergence plot of error in LMS algorithm with step size 0.0550");

  y_LMS_mean = mean(y_LMS);
  % Calculating the mean of output obtained after 200 iterations

  e_LMS_mean = mean(e_LMS);
  % Calculating the mean of the error generated after 200 iterations

  range_plot = 475:500;
  % Observing the output for only fixed data points

  figure
  plot(e_LMS_mean);                   % Plotting the mean error in LMS
  title("Average value of error in LMS algorithm with step size 0.0550");

  Plotting(y_LMS, u, v);              % Plots for LMS algorithm
```

b) Function to generate input symbols

```
function u = Gen_trans_sym(number_of_Symbols)
    u = randi([0,1], 1, number_of_Symbols);
    % Random array of transmitted symbols generated for integers between 0 and 1

    equation = @(i)(2*i-1);          % Equation to convert 0 and 1 into -1 and 1

    for iterations = 1:number_of_Symbols

       % Performing data manipulation on each entry
       u(iterations) = equation(u(iterations));
    end
end
```

c) Function to generate signal after channel

```
function v = Gen_v(h, noise_power, u)
       conj_h = conj(fliplr(h));                    % Generating h*(-n)
       x = conv(h, conj_h);                         % x(n)=h(n)*(h*(-n))

       syms z;
       len = length(x);

       % Calculating the length of the convolutional sum x
       for i = 1:len

             z_elements(i)= x(i)*z^[(i-1)];
             % Determining the z transform
       end

       zTransNum = sum(z_elements);

       % Representation of z transform on the command prompt
       roots_zTrans = roots(x);
       % Determining the roots of Z transform

       j=1;
```

```
        for i=1:length(roots_zTrans)        % Determining the roots of Z transform
              if (abs(roots_zTrans(i))< 1)
                    roots_Fz(j) = roots_zTrans(i);
                    j= j+1;
              end
        end

        coeff_Fz = poly(roots_Fz);            % Determining fn (coefficients of Fz)

        v = conv(coeff_Fz, u)+ noise_power;
        % Calculation of vn which is input to the equalizer
end
```

d) Function to implement LMS algorithm

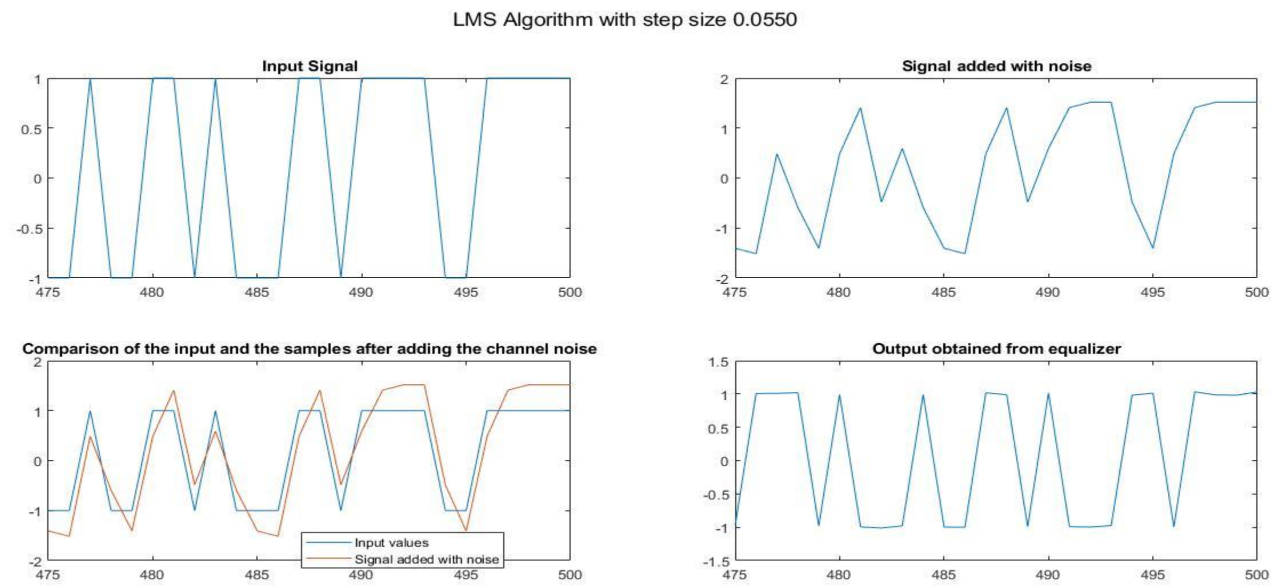
```
function [y,e]= LMS_algorithm(step_size_LMS, tap_weights, number_of_Symbols, u, v,
N)
  y = zeros(1,number_of_Symbols);                    % Output row vector defined
  e = zeros(1, number_of_Symbols);
  % Row vector representing error defined

  w = zeros(number_of_Symbols,tap_weights);
  % Matrix for filter weights defined where each row is designated for each
  %iteration

  w(tap_weights, (tap_weights + 1)/2)= 1;          % tap weights initialised
  % Desired response is N sampled delayed version of channel's input where N =7

  d = circshift(u, N);
  % Delayed version of u generated by padding zeros in the beginning
  d(1:N)=0;

  for k=1:iterations
     for i=1:length(step_size_LMS)
         for j = tap_weights:number_of_Symbols
                range = j:-1 : j-tap_weights+1;
                y(j) = w(j,:)*v(range)';
                % Output generated (1*11)*(11*1)

                e(j) = d(j)-y(j);
                % Error calculation
                w(j+1,:) = w(j,:) + step_size_LMS(i)*v(range)*e(j);
                % Weight adaptation with each iteration
         end
         err_wrt_step(i) = e(j);
     end
  end
end
```

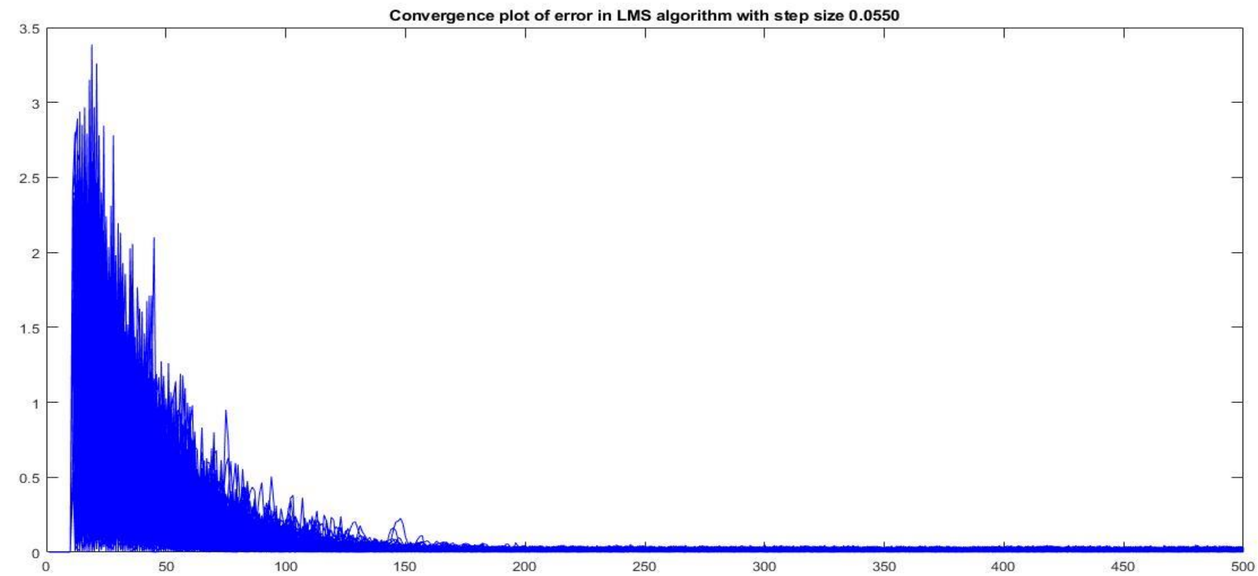
e) Function to plot the samples

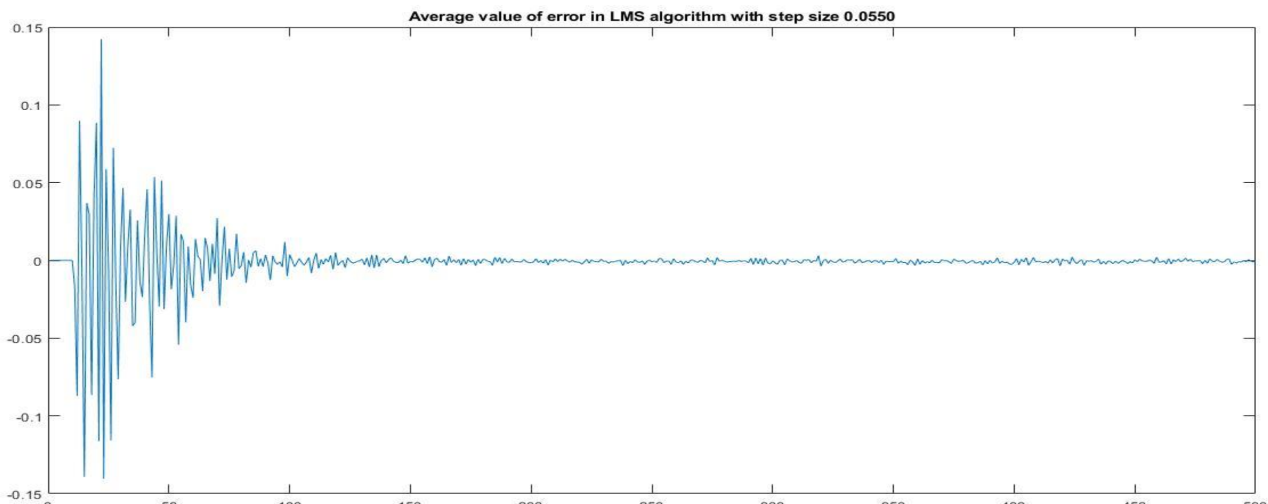
```
function Plotting(y, u, v)
        range_plot = 475:500;

        figure

        subplot(2,2,1)

        plot(range_plot, u(200,475:500));
        % Plot of input values from n =475:500
```

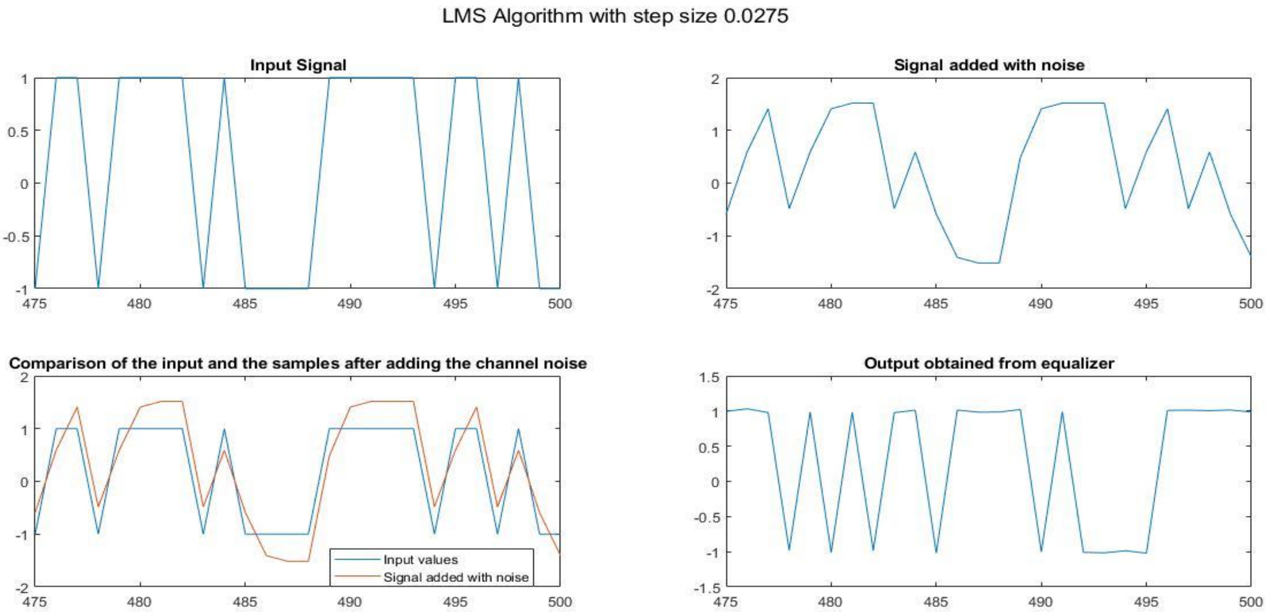
```
    title("Input Signal");

    subplot(2,2,2)

    plot(range_plot, v(200,475:500));
    % Plot of noisy signal from n =475:500

    title("Signal added with noise");

    subplot(2,2,3)

    % Plot of input values along with the signal added with noise
    plot(range_plot, u(200,475:500));
    hold on;

    plot(range_plot, v(200,475:500));

    title("Comparison of the input and the samples after adding the channel
    noise");

    legend('Input values','Signal added with noise')

    subplot(2,2,4)

    % Plot of output values obtained after equalization
    plot(400:500,y(200, 400:500));

    title("Output obtained from equalizer");


    suptitle('LMS Algorithm with step size 0.0550');

end
```

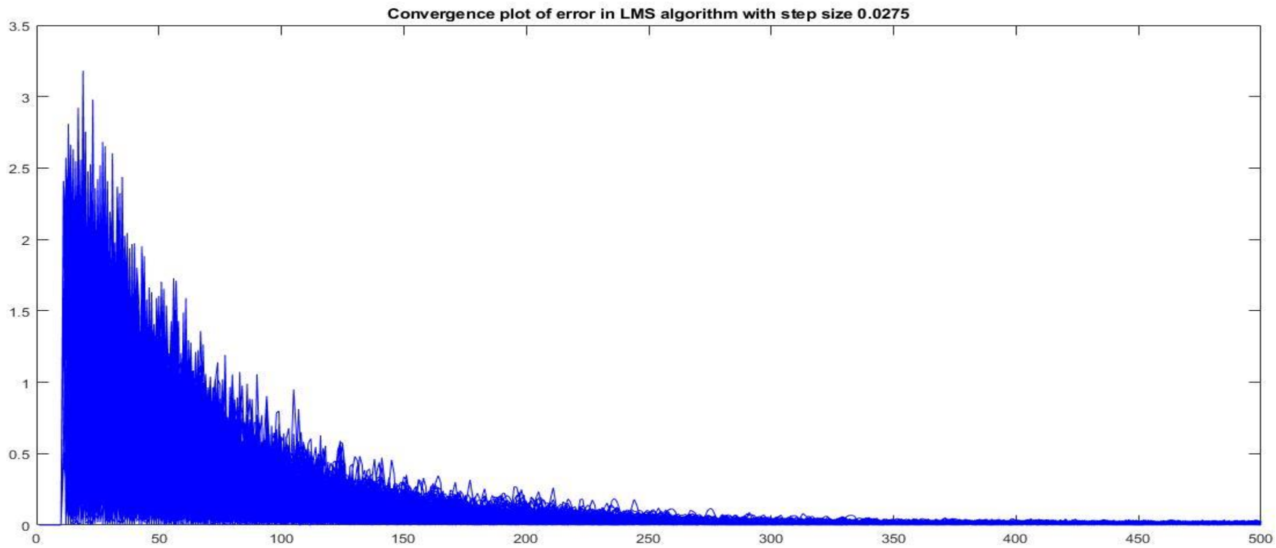
Plots:

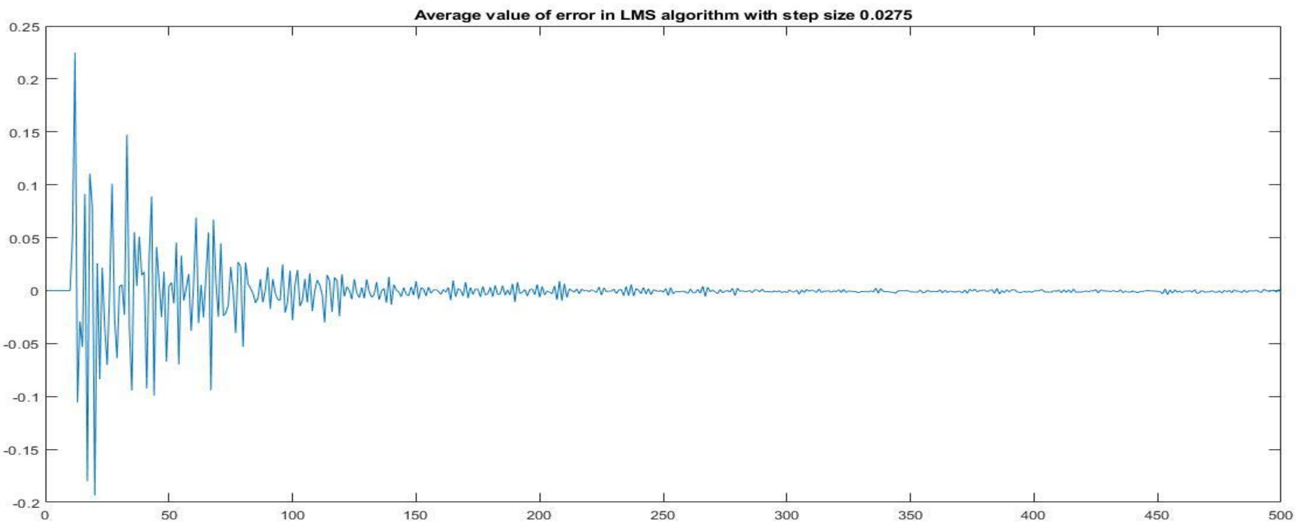

LMS Algorithm with step size 0.0550
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer

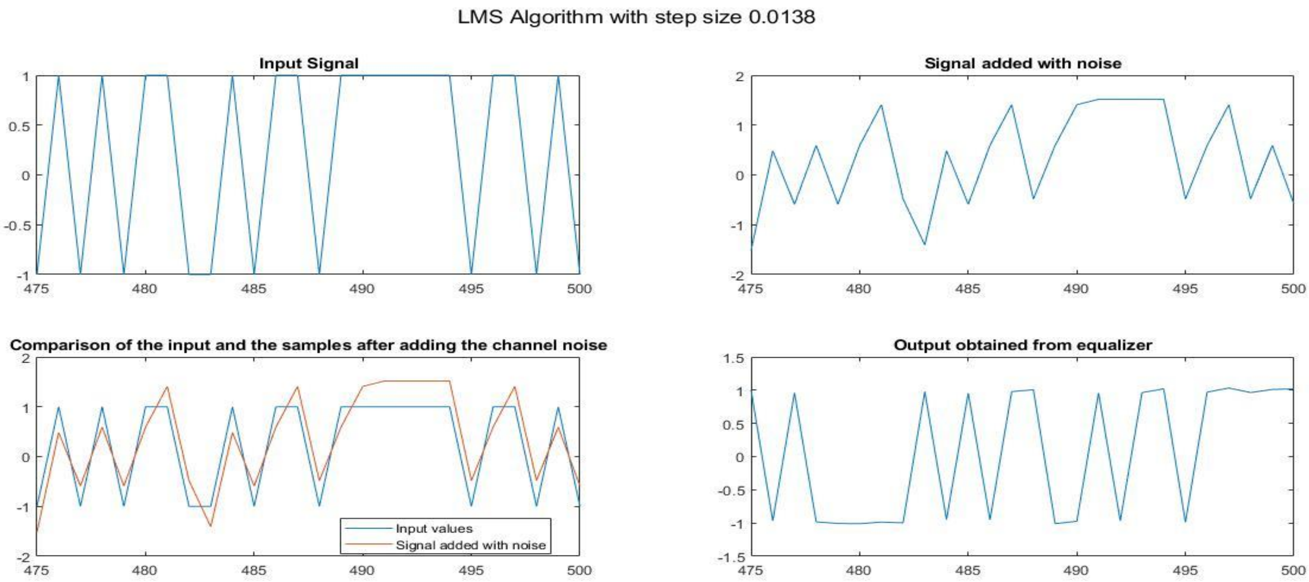

Convergence plot of error in LMS algorithm with step size 0.0550

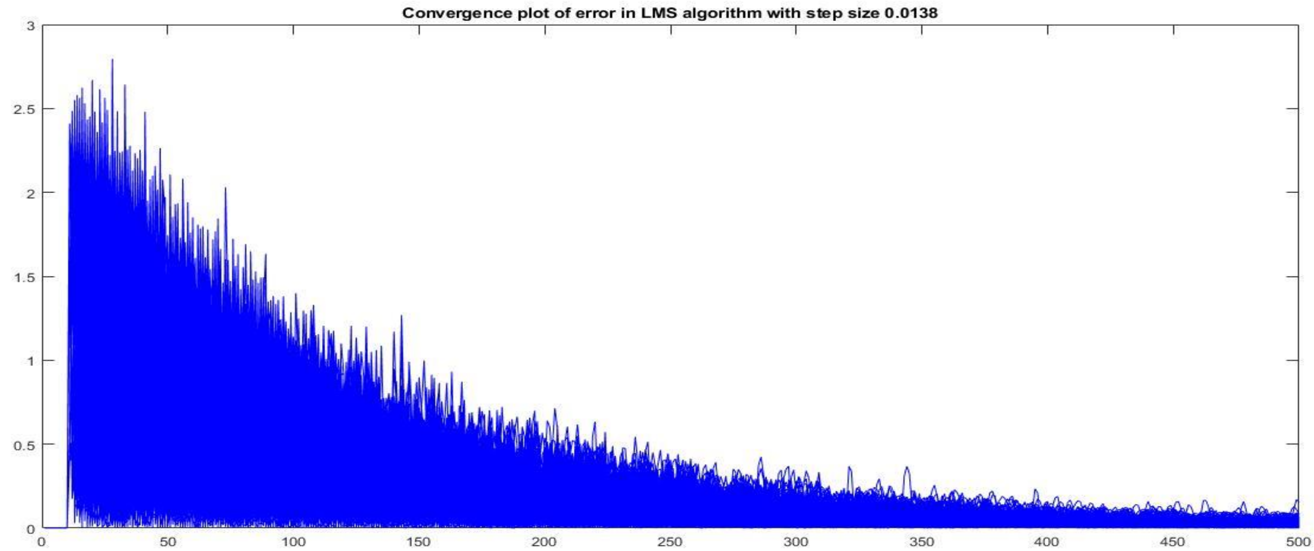

Average value of error in LMS algorithm with step size 0.0550

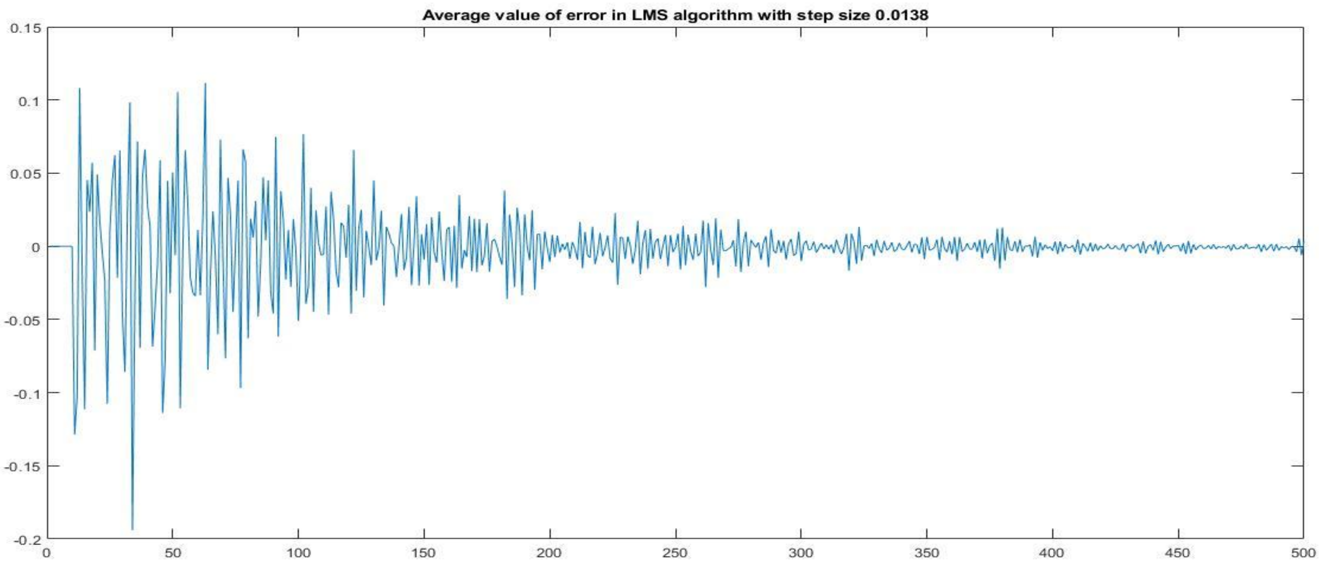

LMS Algorithm with step size 0.0275
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer

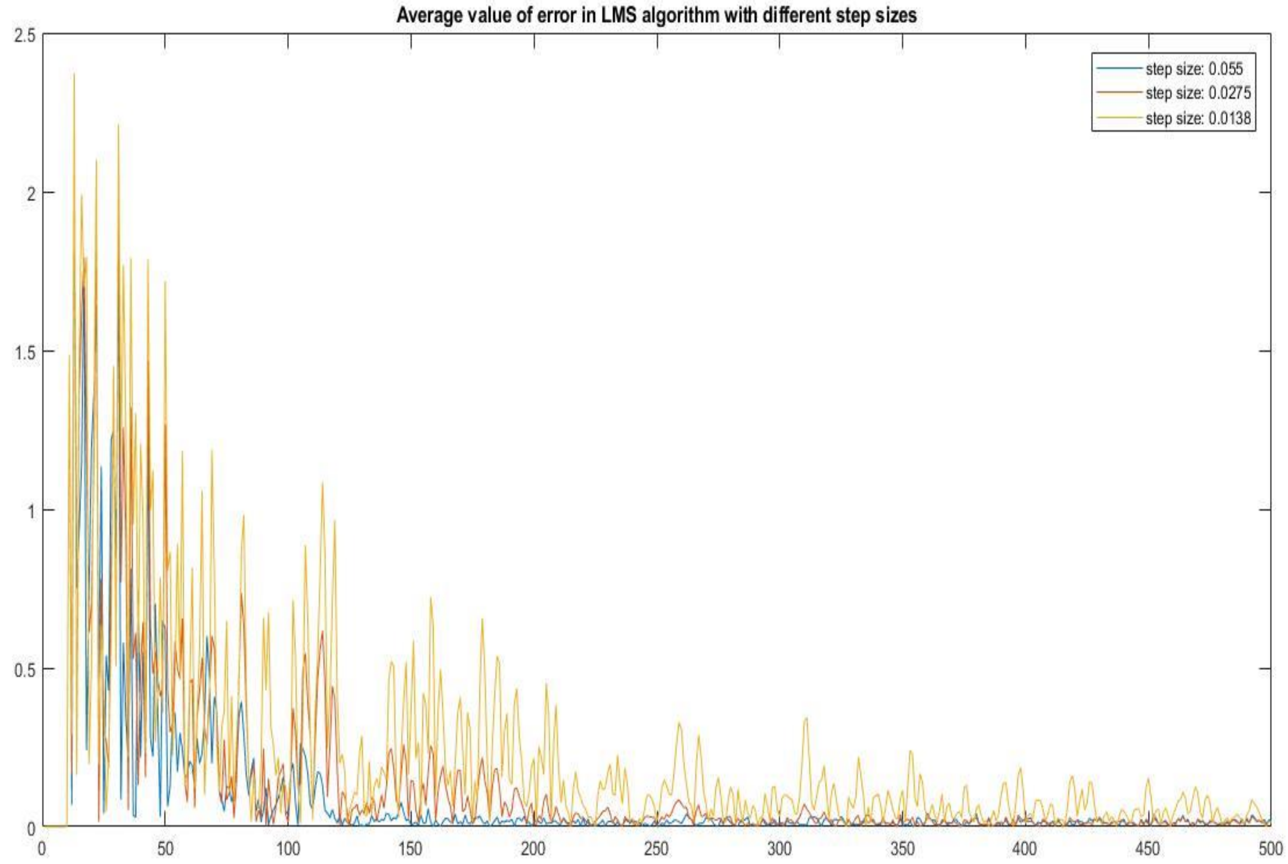

Convergence plot of error in LMS algorithm with step size 0.0275


Average value of error in LMS algorithm with step size 0.0275


LMS Algorithm with step size 0.0138
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer
475
480
485
490
495
500


Convergence plot of error in LMS algorithm with step size 0.0138
0
50
100
150
200
250
300
350
400
450
500


Average value of error in LMS algorithm with step size 0.0138
0
50
100
150
200
250
300
350
400
450
500



Conclusion:

Convergence rate of LMS algorithm increases with increase in the value of step size.

Normalized Least Mean Square Estimation (NLMS):

The main drawback of LMS algorithm is that it is very sensitive to input scaling therefore a version of LMS algorithm entitled "Normalized Least Mean Square Algorithm (NLMS Algorithm)" was developed. Under this algorithm, the power of the input is normalized using a scale factor which in turn reduces the algorithm's sensitivity to variations in input.

Implementation of NLMS algorithm:

Desired response is obtained by shifting the input signal by 7 and padding the vacant spots with zeros. The number of tap weights is given to be 11. Initial tap vector is taken to be all zeros except the middle element which is assumed to be 1.
Formulas:

$$y(n) = w^H(n)v(n)$$

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

$$w(n+1) = w(n) + \frac{\mu}{a + ||w(n)||^2 v(n)e*(n)}$$

Where $y(n)$ is the output of the filter

$e(n)$ is the error generated

$d(n)$ is the desired signal

$w(n)$ is the tap weight vector

μ is the step size (0.11, 0.055, 0.0275)

a is a small constant of value larger than 0

Code:

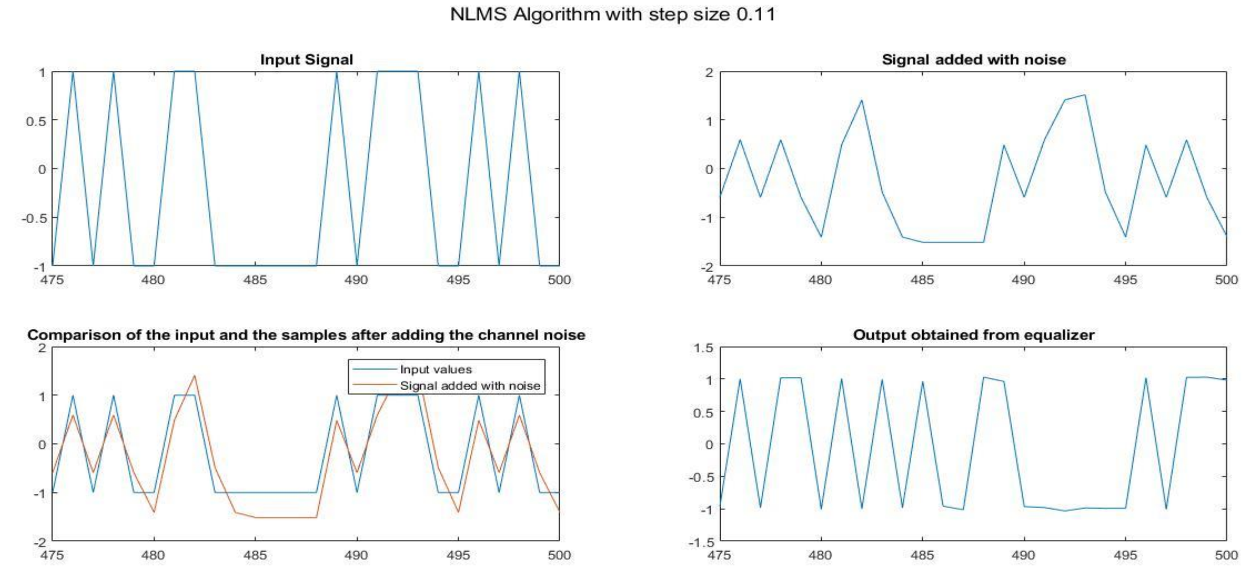
```
clc;
clear all;
close all;

Iterations = 200;                    % Number of Iterations
number_of_Symbols= 500;              % Number of Transmitted symbols
h = [0.2194, 1.000, 0.2194];         % Channel Coefficients
noise_power = 0.001;                 % Defined noise power

u = randn(Iterations, number_of_Symbols);              % Initializations for u and v

v = randn(Iterations, number_of_Symbols + length(h)-1);
% Length of the convolutional sum is N+M-1 where N and M is the length of the
%individual sequences

N = 7;                               % Desired response is N shifted version of u
tap_weights = 11;                    % Number of tap weights defined
step_size_NLMS = 0.11;               % Step size defined
a = 0.05;                            % parameter for NLMS algorithm

for i=1:Iterations
     u(i,:) = Gen_trans_sym(number_of_Symbols);
```

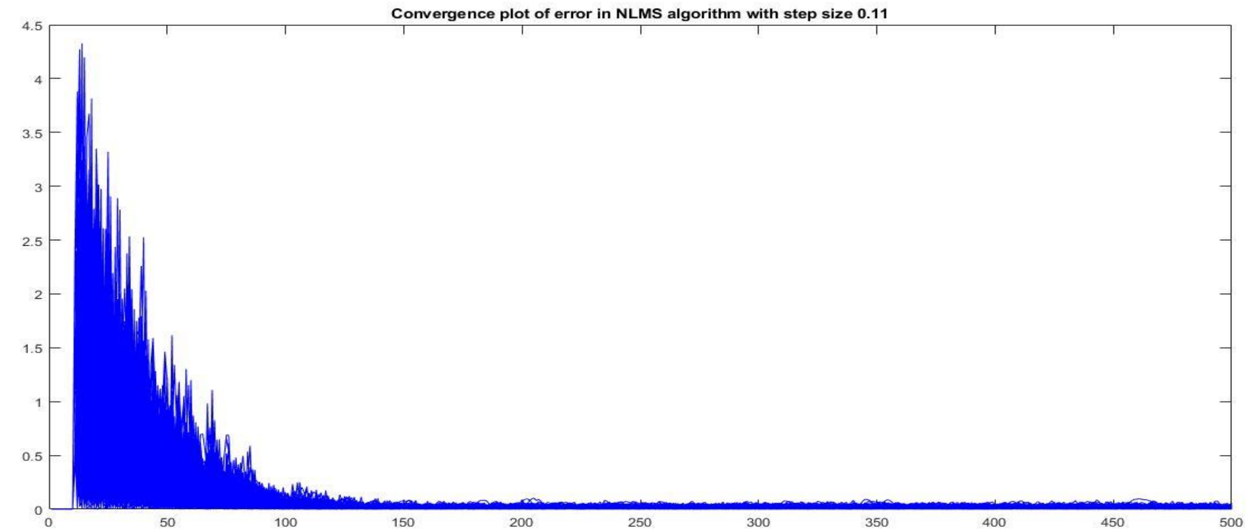
```
    % Creating an array of transmitted symbols [1,-1]
    v(i,:) = Gen_v(h, noise_power, u(i,:));
    % Generating the sequence vn

    % Normalized Least Mean Square (NLMS) equalization algorithm

    [y_NLMS(i,:), e_NLMS(i,:)]= NLMS_algorithm_check(step_size_NLMS, tap_weights,
    number_of_Symbols, u(i,:), v(i,:), a, N);

    plot(abs(e_NLMS(i,:)),'b')
    % Plotting the absolute value of error for each iteration
    hold on;
end
title("Convergence plot of error in NLMS algorithm with step size 0.11");

y_NLMS_mean = mean(y_NLMS);
% Calculating the mean of output obtained after 200 iterations
e_NLMS_mean = mean(e_NLMS);
% Calculating the mean of the error generated after 200 iterations

range_plot = 475:500;
% Observing the output for only fixed data points
figure
plot(e_NLMS_mean);                              % Plotting the mean error in NLMS
title("Average value of error in NLMS algorithm with step size 0.11");

Plotting(y_NLMS, u, v);                         % Plots for NLMS algorithm
```

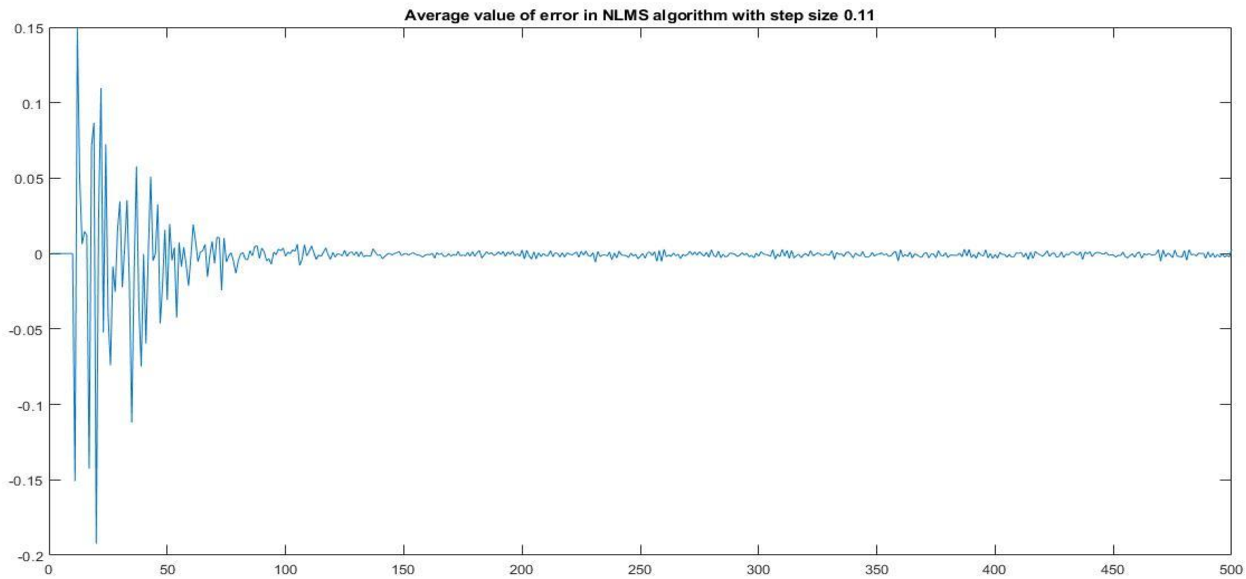
<u>Function to implement NLMS algorithm</u>

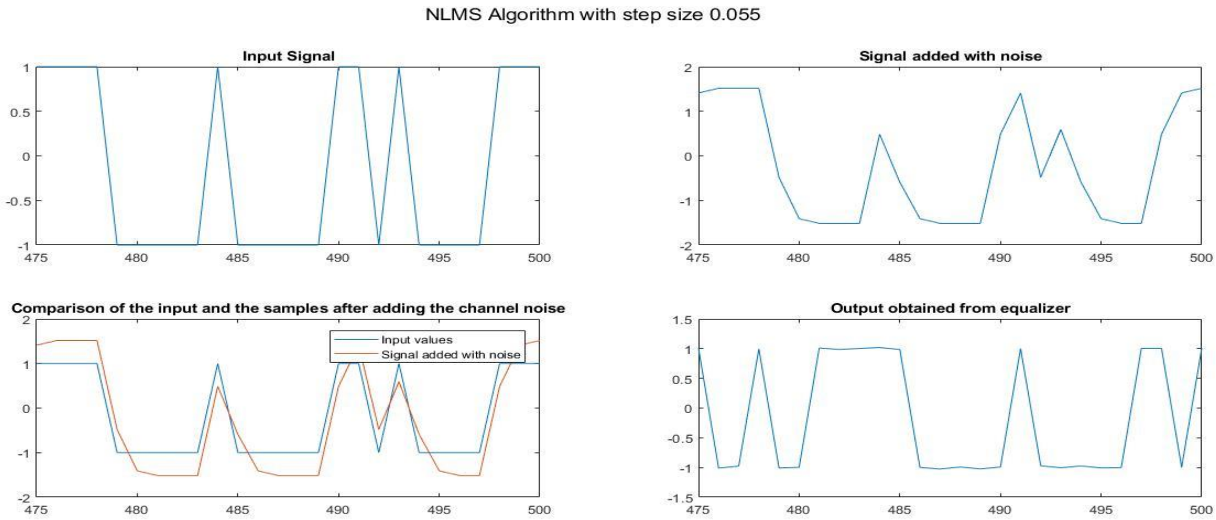
```
 function [y,e]= NLMS_algorithm(step_size, tap_weights, number_of_Symbols, u, v,
 a, N)
      y = zeros(1,number_of_Symbols);          % Output row vector defined
      e = zeros(1, number_of_Symbols);         % Error vector defined

      w = zeros(number_of_Symbols,tap_weights);
      % Matrix for filter weights defined where each row is designated for each %
      %iteration
      w(tap_weights, (tap_weights + 1)/2)= 1;  % tap weights initialised

      % Delayed version of u generated by padding zeros in the beginning
      d = circshift(u, N);
      d(1:N)=0;

      for i = tap_weights:number_of_Symbols
            range = i:-1 : i-tap_weights+1;
            y(i) = w(i,:)*v(range)';            % Output generated(1*11)*(11*1)
            e(i) = d(i)-y(i);                   % Error Calculation

            w(i+1,:) = w(i,:) + (step_size/(a + (norm(w(i,:))^2)))*v(range)*e(i);
            % Weight adaptation with each iteration
      end;
 end
```

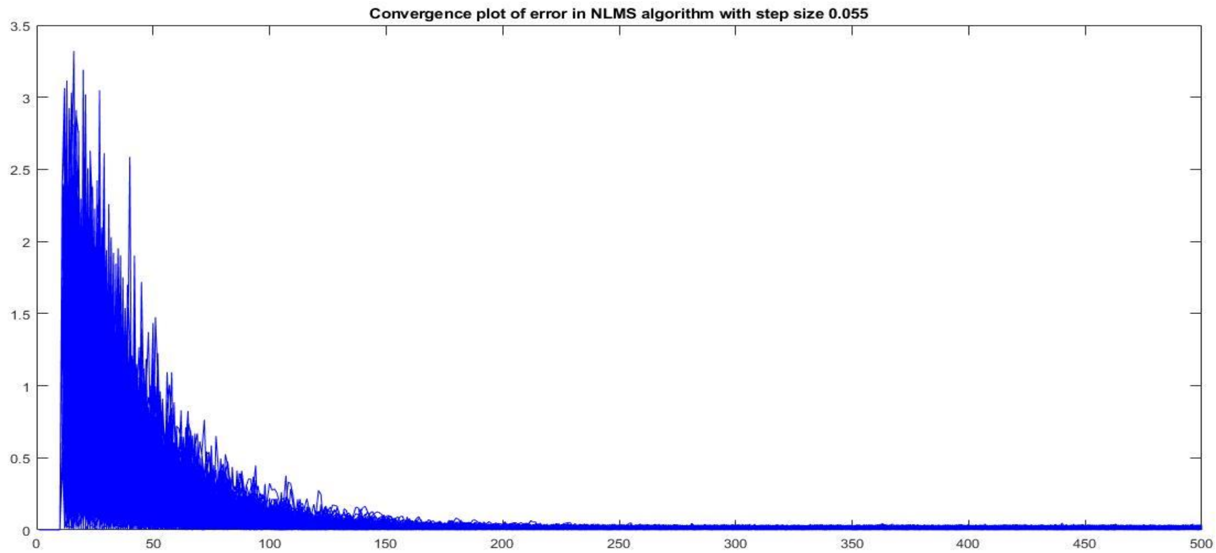
Plots:

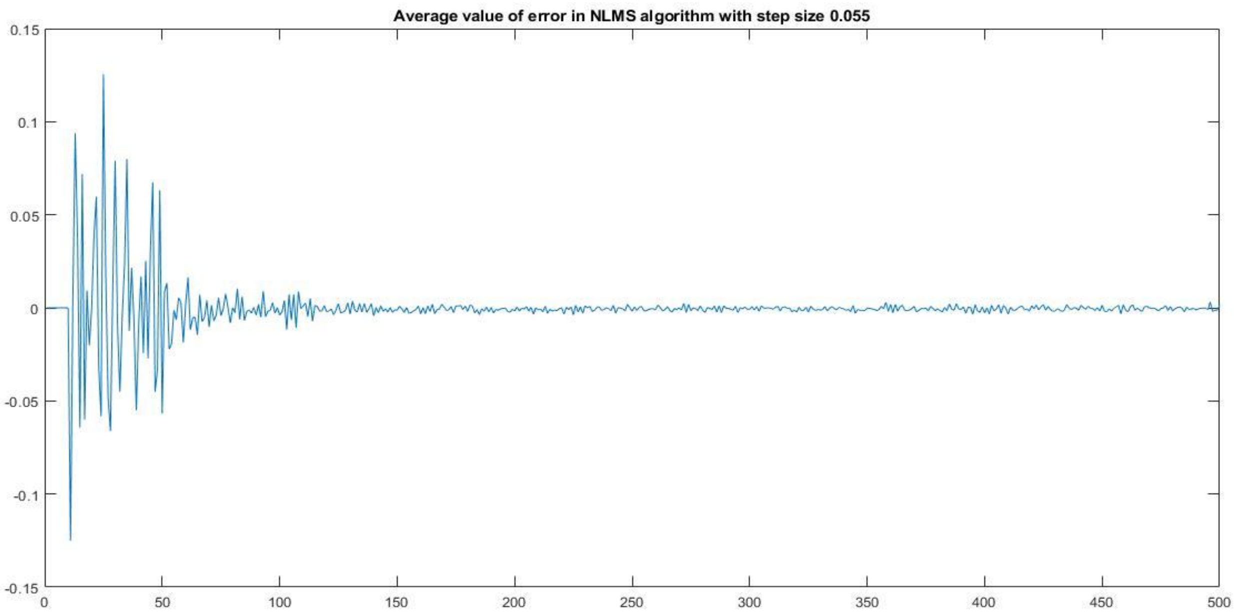

NLMS Algorithm with step size 0.11
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer

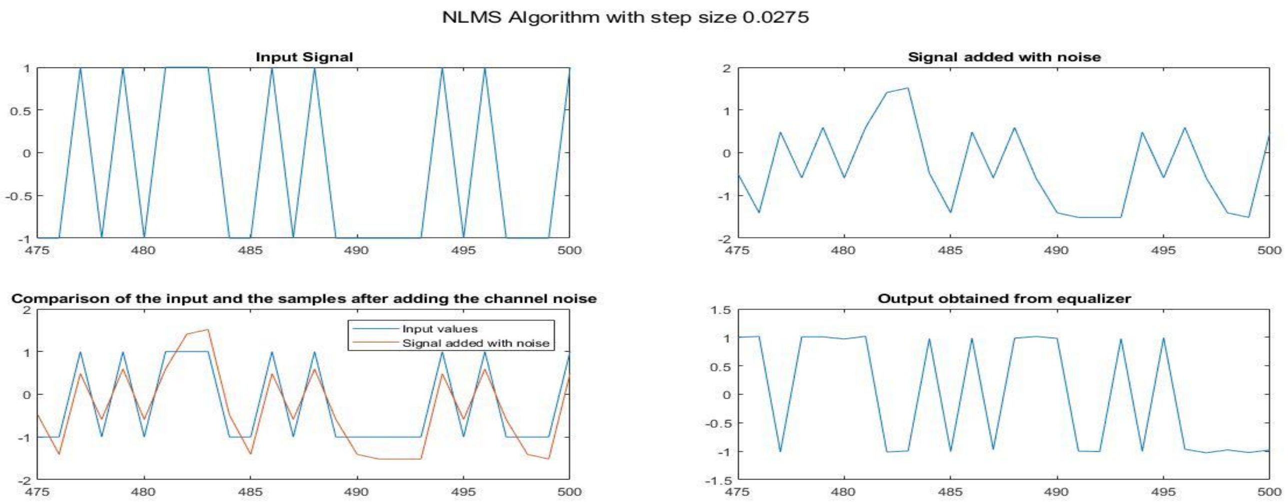

Convergence plot of error in NLMS algorithm with step size 0.11

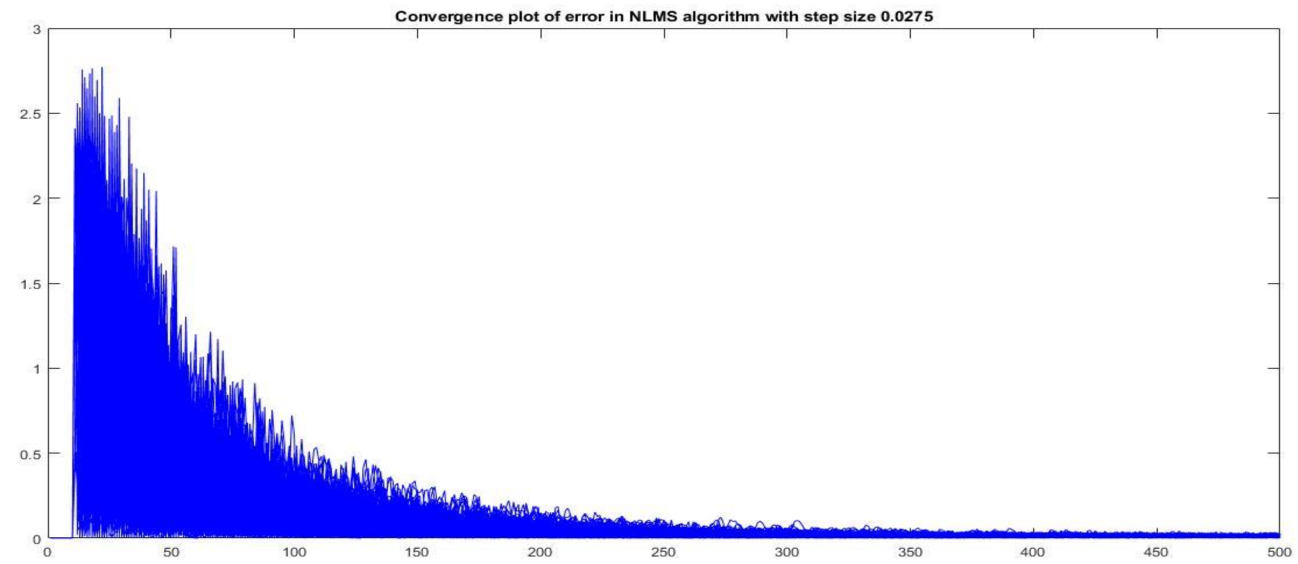

Average value of error in NLMS algorithm with step size 0.11

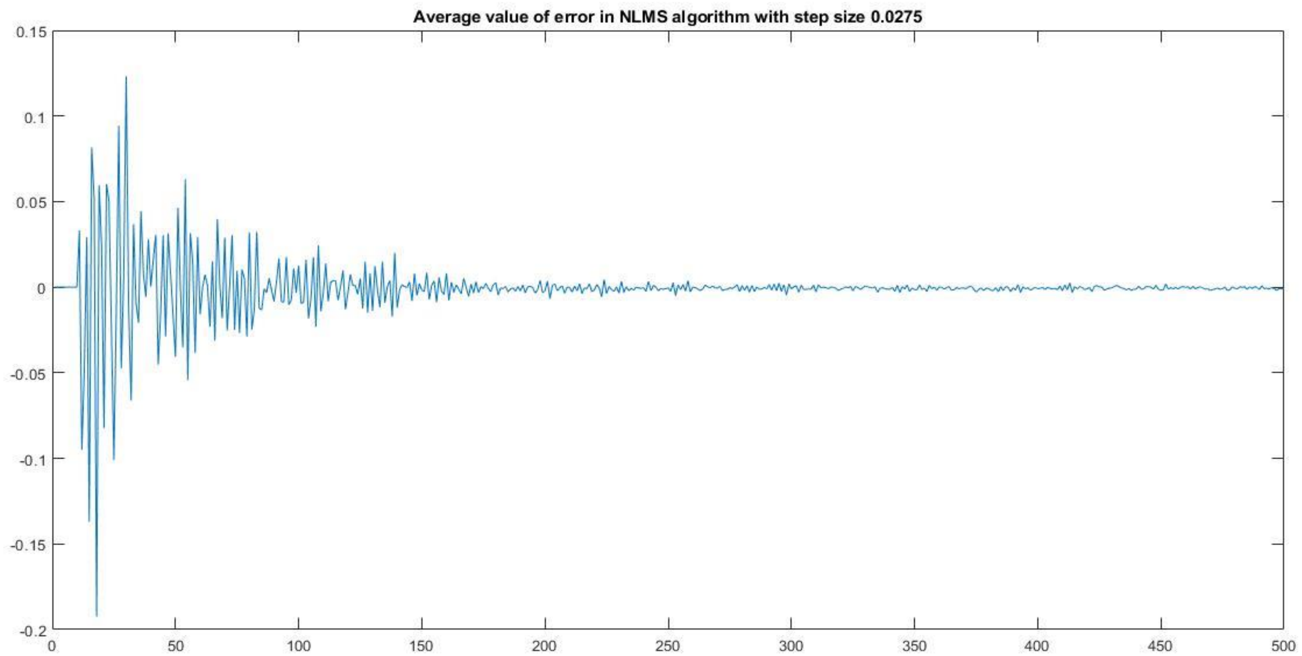

NLMS Algorithm with step size 0.055
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer


Convergence plot of error in NLMS algorithm with step size 0.055


Average value of error in NLMS algorithm with step size 0.055


NLMS Algorithm with step size 0.0275
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer


Convergence plot of error in NLMS algorithm with step size 0.0275


Average value of error in NLMS algorithm with step size 0.0275

Recursive Least Square Estimation (RLS):

RLS algorithm falls under the category of least square estimation algorithms. The main baseline of this algorithm is to perform a recursive operation to determine the filter coefficients so as to minimize the weighted linear least squares cost function relating to the input signals. The major drawback of using this algorithm is the huge complexity associated with it, therefore, introducing a tradeoff between better rate of convergence and complexity.

Implementation of RLS algorithm:

Formulas:

$$k(n) = \frac{\lambda^{-1}P(n-1)u(n)}{1+\lambda^{-1}u(n)P(n-1)u(n)}$$

$$\alpha(n) = d(n) - w^H(n-1)v(n)$$

$$w(n) = w(n-1) + k(n)\,\alpha^*(n)$$

$$y(n) = w^H(n)v(n)$$

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

$$P(n) = \lambda^{-1}P(n-1) - \lambda^{-1}k(n)v^H(n)P(n-1)$$

Where $y(n)$ is the output of the filter

$e(n)$ is the error generated

$d(n)$ is the desired signal

$w(n)$ is the tap weight vector

k is the gain vector

λ can take values of 0.9, 0.7, and 0.5

Code:

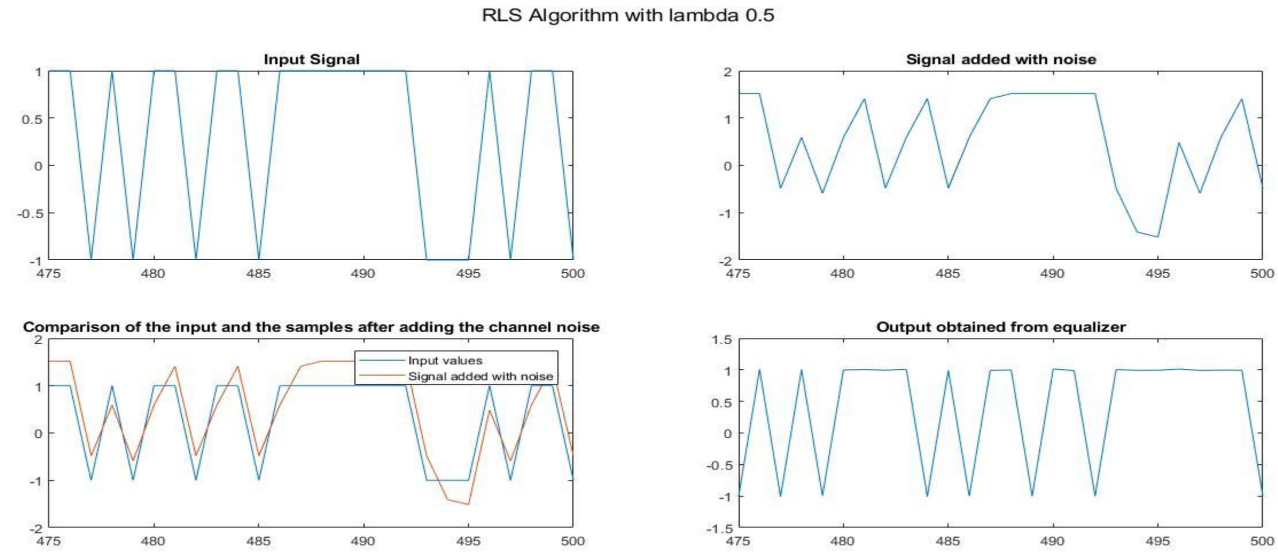
```
clc;
clear all;
close all;

Iterations = 200;                 % Number of Iterations
number_of_Symbols= 500;           % Number of Transmitted symbols
h = [0.2194, 1.000, 0.2194];      % Channel Coefficients
noise_power = 0.001;              % Defined noise power

u = randn(Iterations, number_of_Symbols);             % Initializations for u and v
v = randn(Iterations, number_of_Symbols + length(h)-1);

N = 7;                            % Desired response is N shifted version of u
tap_weights = 11;                 % Number of tap weights defined
lambda = 0.5;                     % Step size defined

u = Gen_trans_sym(number_of_Symbols);
% Creating an array of transmitted symbols [1,-1]
v = Gen_v(h, noise_power, u);           % Generating the sequence vn
```

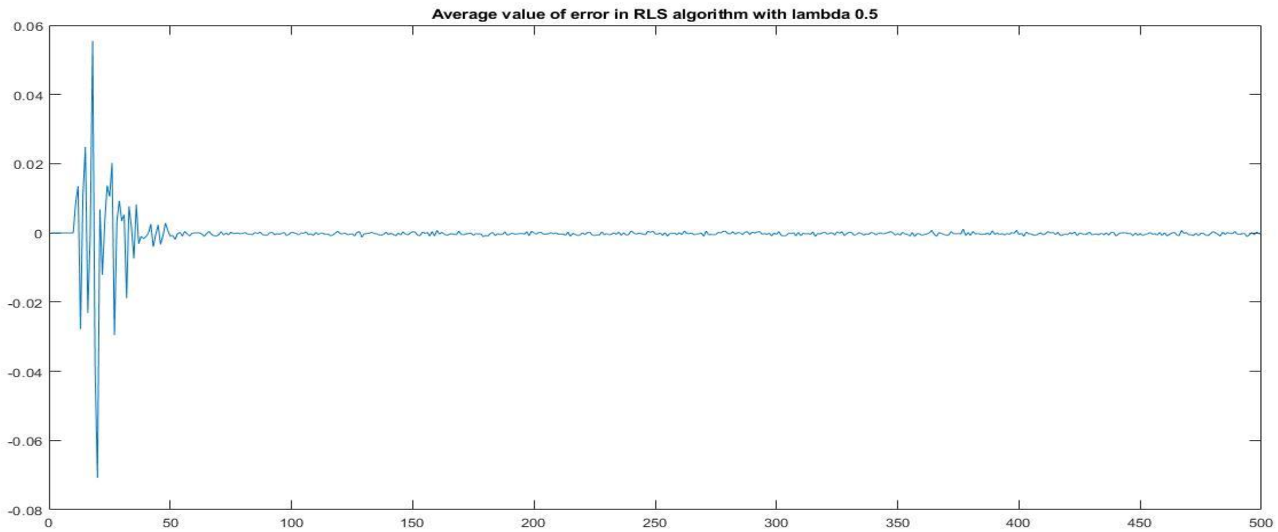
```
for i=1:Iterations
    u(i,:) = Gen_trans_sym(number_of_Symbols);  % Creating transmitted symbols
    v(i,:) = Gen_v(h, noise_power, u(i,:));     % Generating the sequence vn

    % Recursive Least Square (RLS) equalization algorithm
    [y_RLS(i,:), e_RLS(i,:)]= RLS_algorithm_check(tap_weights, u(i,:), v(i,:),
    lambda, number_of_Symbols, N);

    plot(abs(e_RLS(i,:)),'b')
    %Plotting the absolute value of error for each iteration
    hold on;
end

title("Convergence plot of error in RLS algorithm with lambda 0.5");

y_RLS_mean = mean(y_RLS);
e_RLS_mean = mean(e_RLS);

figure
plot(e_RLS_mean);                                % Plotting the mean error in RLS
title("Average value of error in RLS algorithm with lambda 0.5");
Plotting(y_RLS, u, v);                           % Plots for RLS algorithm
```

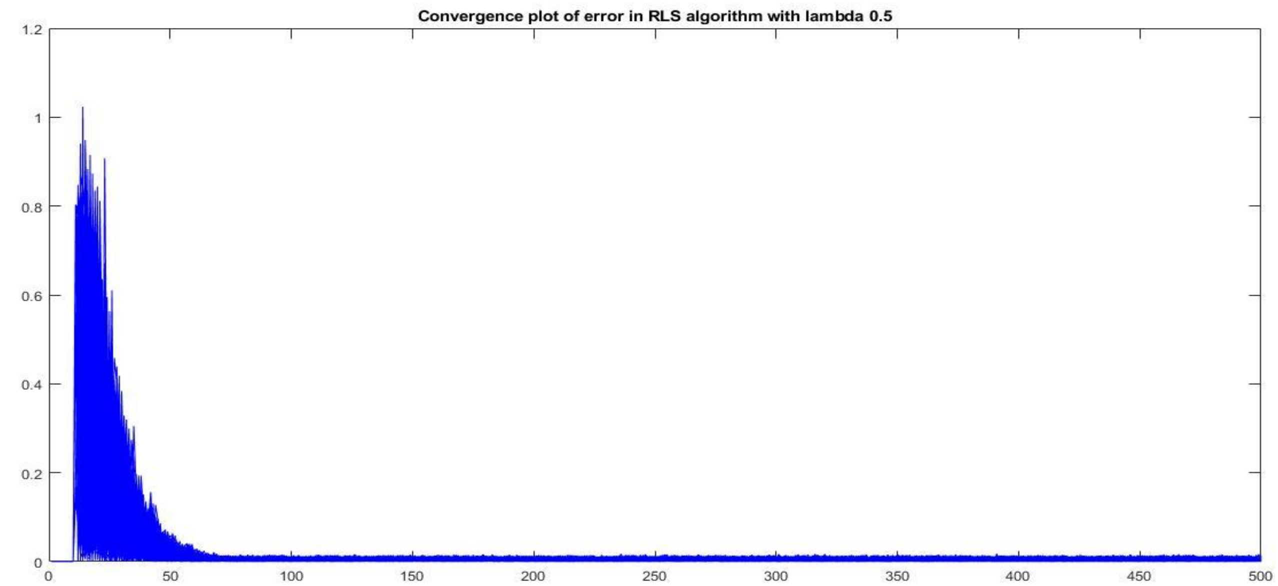
<u>Function to implement RLS algorithm</u>

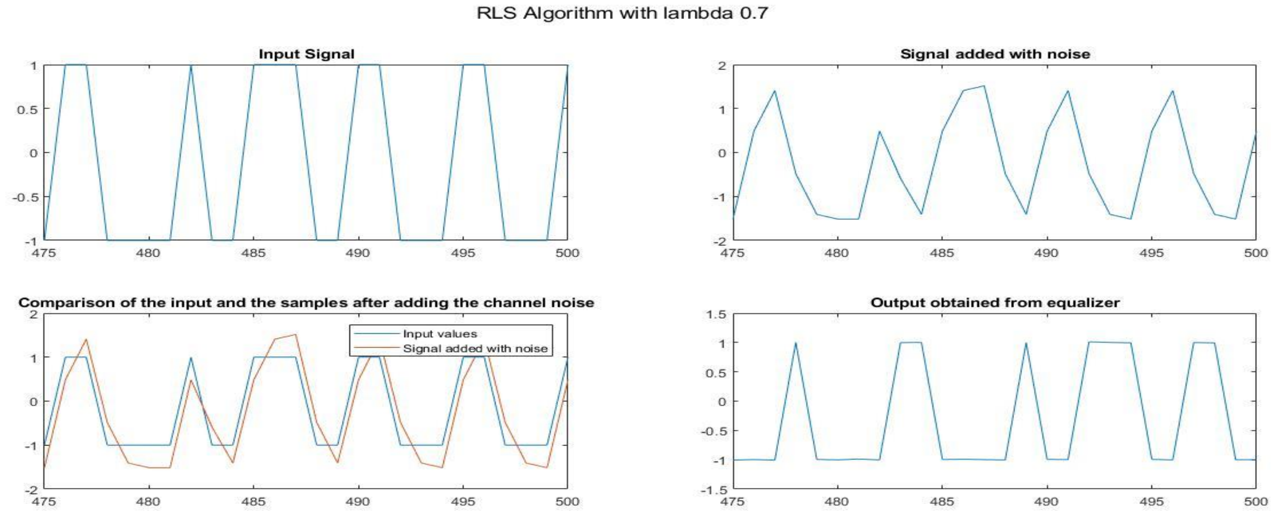
```
function [y, e] = RLS_algorithm(tap_weights, u, v, lambda, number_of_Symbols, N)

     d = circshift(u, N);
     d(1:N)=0;

    %% Initial Conditions
    delta = 0.05;

    % Initializing the tap weights for the filter
    w = zeros(number_of_Symbols, tap_weights);
    w(tap_weights-1,(tap_weights + 1)/2)= 1;           % tap weights initialised

    % Initializing P matrix where P is a M*M matrix where M is the number of tap
    %weights in a filter

    P = eye(tap_weights)/delta;

    %% Implementation of RLS algorithm

    for i = tap_weights:number_of_Symbols

        range_V = (v(i:-1:i-tap_weights+1))';
        k = lambda^(-1)*P*range_V/((1+lambda^(-1))*range_V'*P*range_V);
        alpha(i) = d(i) - w(i-1,:)*range_V;
        w(i,:) = w(i-1,:)+(k'*conj(alpha(i)));
        y(i)= w(i,:)*range_V;
        e(i) = d(i) - y(i);
        P = lambda^(-1)*P-(lambda^(-1))*k*range_V'*P;

   end
end
```

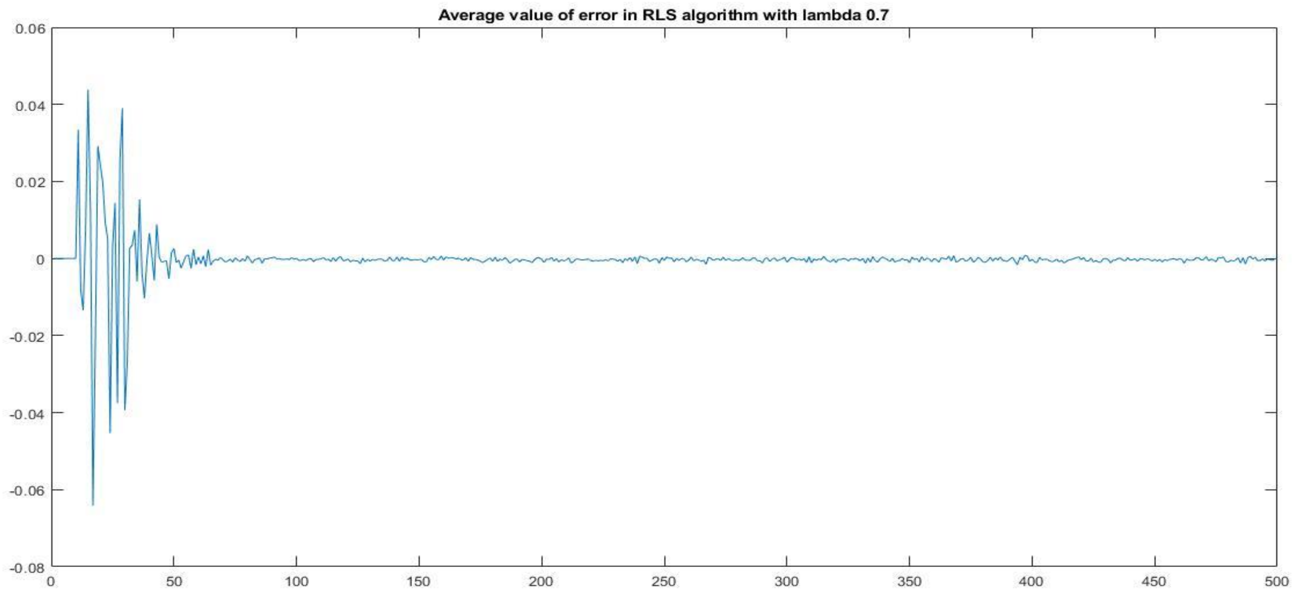
Plots:

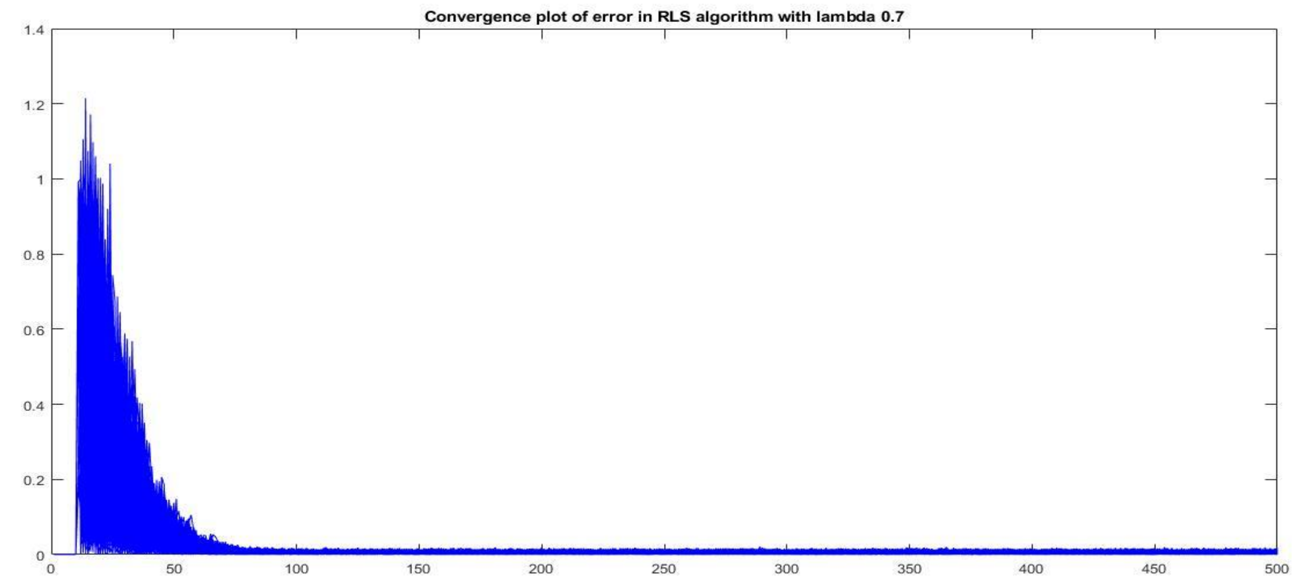

RLS Algorithm with lambda 0.5
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer

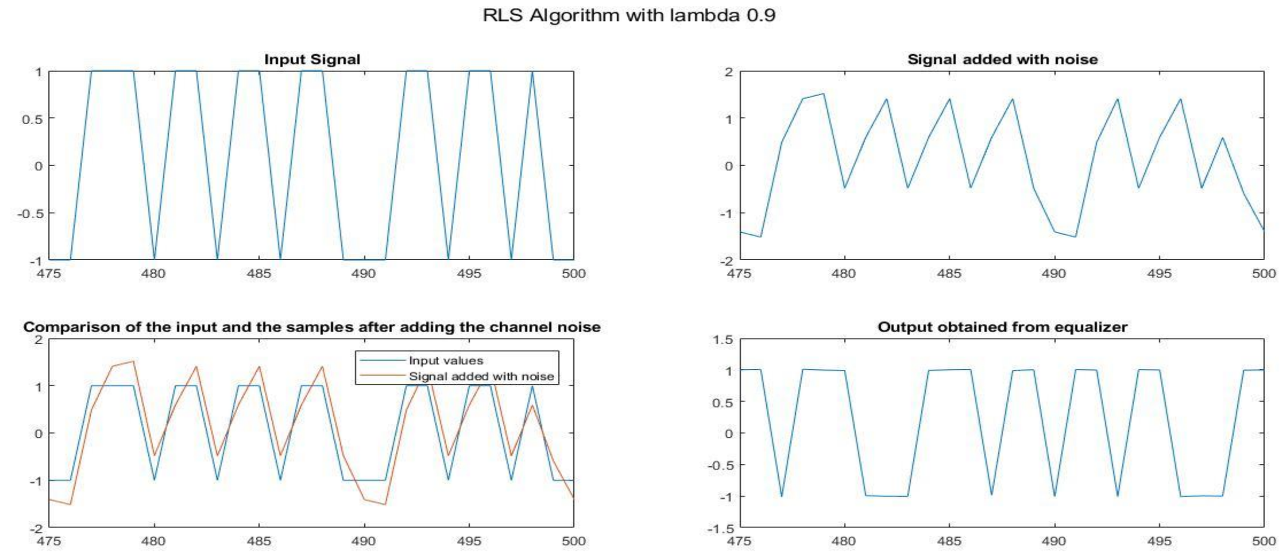

Average value of error in RLS algorithm with lambda 0.5

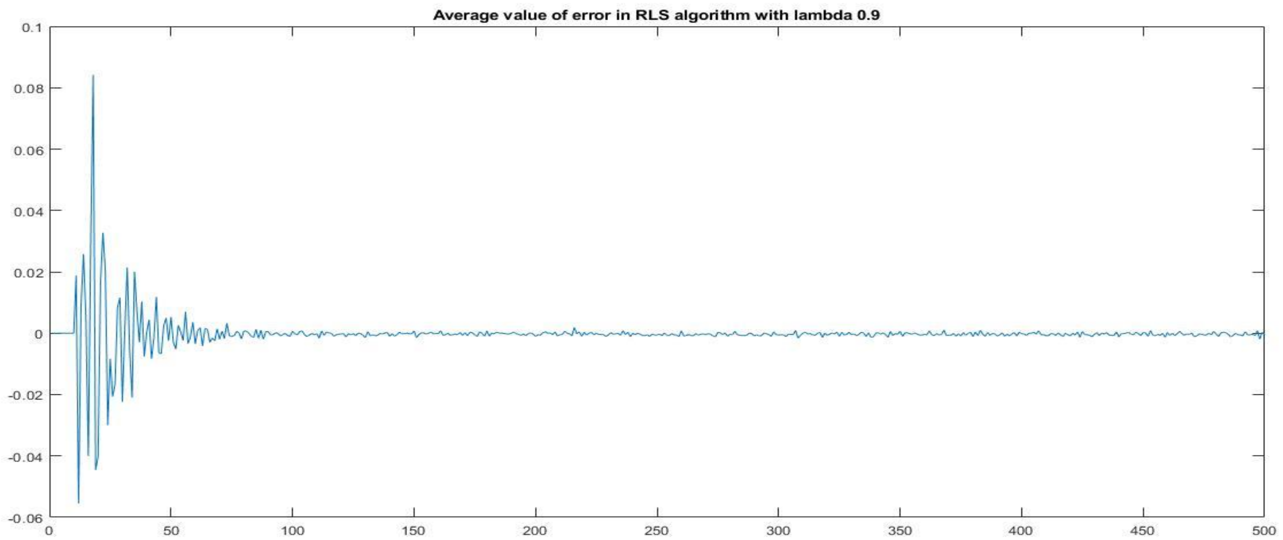

Convergence plot of error in RLS algorithm with lambda 0.5

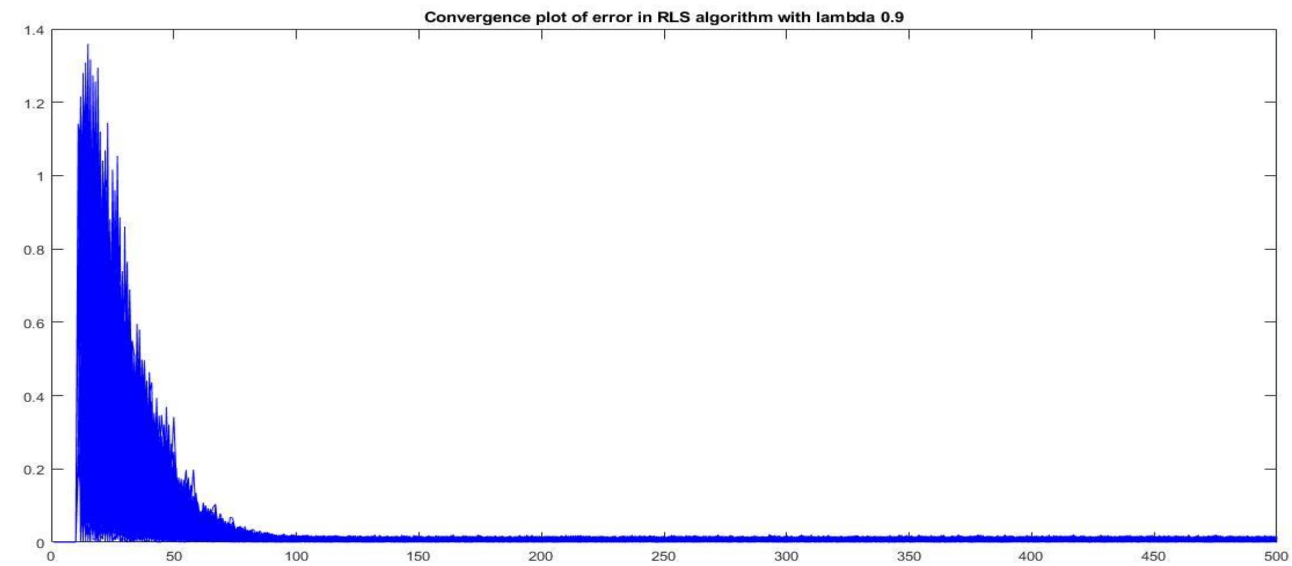

RLS Algorithm with lambda 0.7
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer


Average value of error in RLS algorithm with lambda 0.7

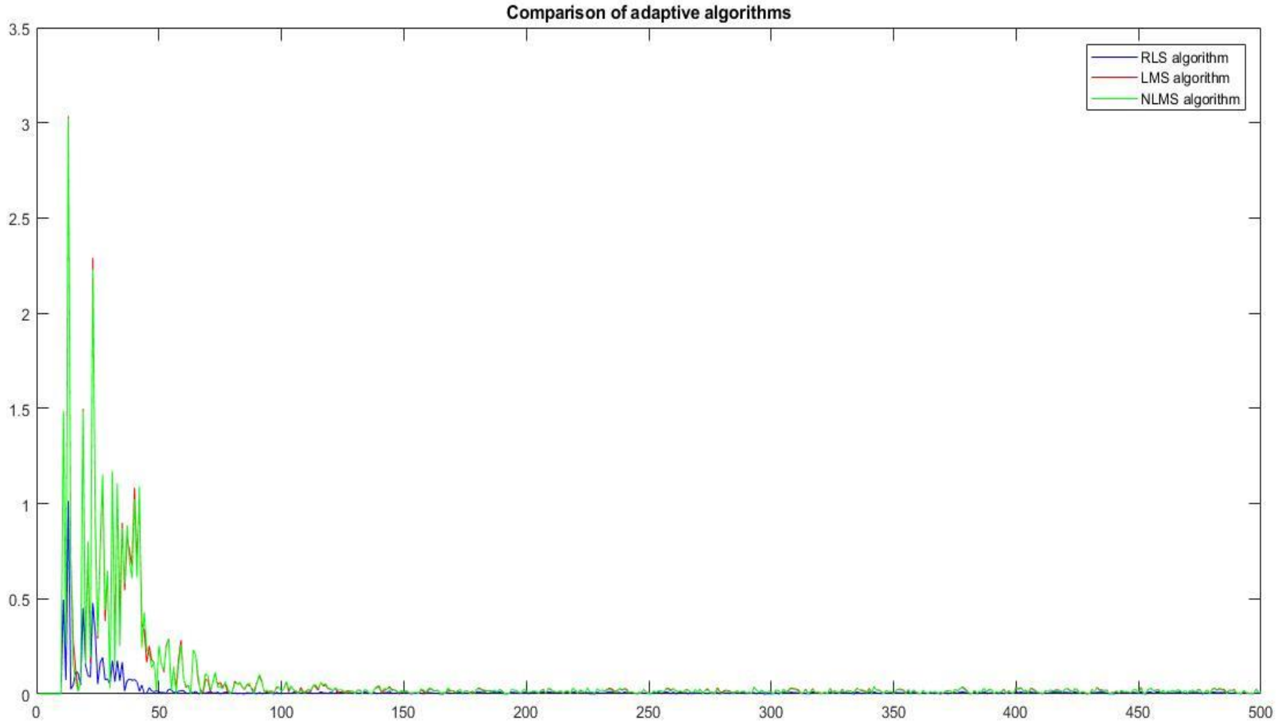

Convergence plot of error in RLS algorithm with lambda 0.7


RLS Algorithm with lambda 0.9
Input Signal
Signal added with noise
Comparison of the input and the samples after adding the channel noise
Input values
Signal added with noise
Output obtained from equalizer


Average value of error in RLS algorithm with lambda 0.9


Convergence plot of error in RLS algorithm with lambda 0.9

Comparison of Adaptive Filter Algorithms:



RLS algorithm converges faster, i.e. error between the input and output reduces to zero in a very less time as compared to other two algorithms. But the complexity of RLS algorithm is more as it involves more number of computations and involves larger set of variables therefore LMS algorithm is preferred over RLS algorithm.

References:

1. Adaptive Filter theory- Simon Haykin, Edition 4

2. Dhiman, J., Ahmad, S. and Gulia, K., 2013. Comparison between Adaptive filter Algorithms (LMS, NLMS and RLS). *International Journal of Science, Engineering and Technology Research (IJSETR)*, *2*(5), pp.1100-1103.