دانشکده مهندسی برق و الکترونیک

# گزارش کار پروژه درس کنترل فازی

## طراحی سیستم فازی با روش های گرادیان نزولی و حداقل مربعات خطا

**دانشجو:**

نیما جهان بازفرد (400113020)

**استاد درس:**

جناب آقای دکتر مختار شاصادقی

**اردیبهشت 1404**


بسم الله الرحمن الرحيم

# فهرست مطالب

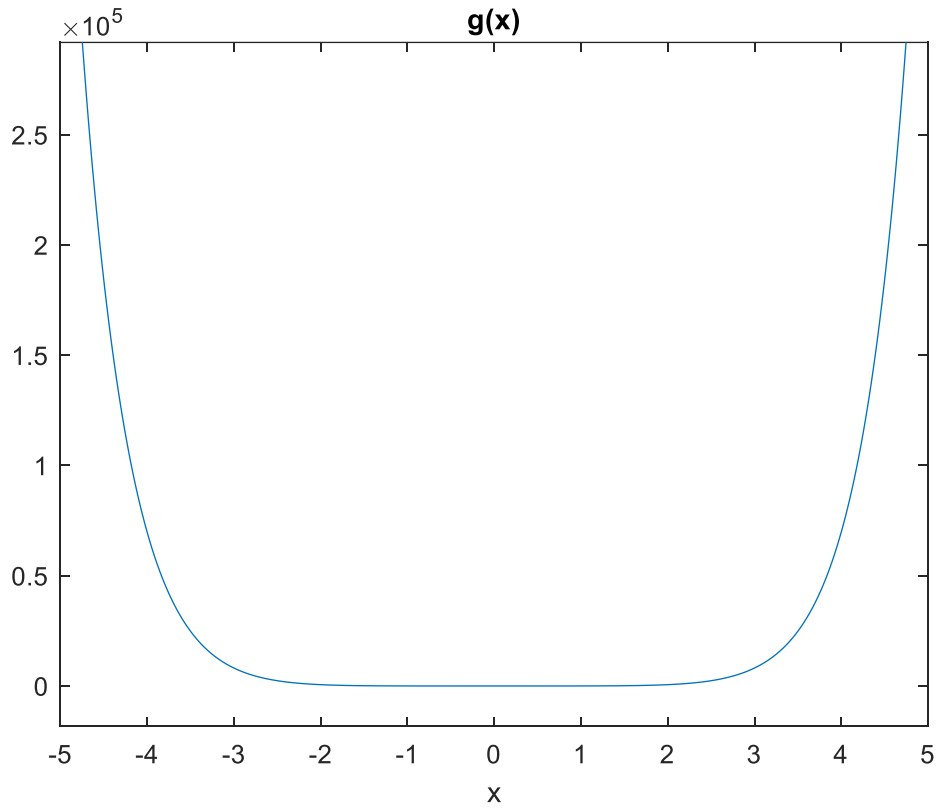
**عنوان** **صفحه**

# بخش 1

# تعربف تابع اصلی و بازه مورد نظر جهت مدل سازی فازی

## 1-1 عربف تابع اصلی و بازه مورد نظر جهت مدل سازی فازی

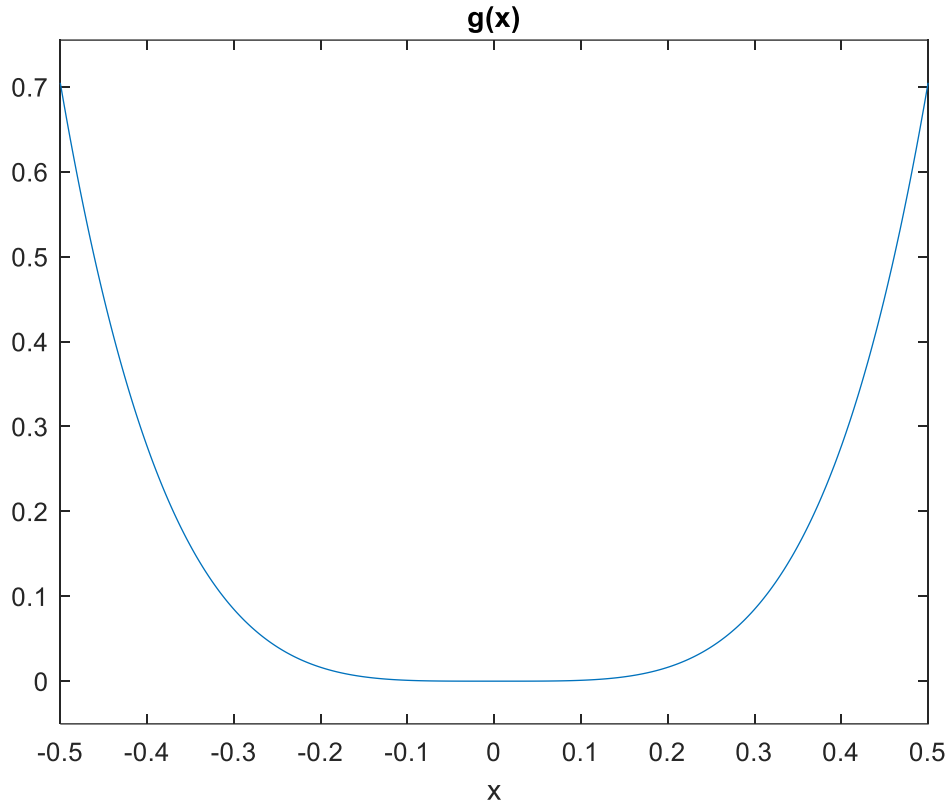
در این پروژه هدف مدلسازی فازی تابع $10x^4\frac{e^x-e^{-x}}{2}$ در بازه [5 -5] می باشد. اما با رسم این تابع در بازه گفته شده در متلب داریم:




شکل 1 : تابع $10x^4\frac{e^x-e^{-x}}{2}$ در بازه [5 -5]

کامل طبق شکل 1 قابل مشاهده می باشد که تابع $10x^4\frac{e^x-e^{-x}}{2}$ در دو حد 5- و 5 به سمت بی نهایت میل می کند و این باعث می شود که مدل سازی فازی با روش الگوریتم گرادیان نزولی (به دلیل استفاده از مشتق)غیرممکن و نیازمند سیستم بسیار قوی برای تحلیل داده ها می باشد.( سیستم بنده توانایی تحلیل داده ها برای رسیدن به دقت مطلوب برای بازه های بزرگتر را ندارد چون گام حرکت باید کوچک باشد چندین ساعت باید سیستم در حال پردازش دیتا باشد و سیستم توانایی لازم را جهت انجام اینکار ندارد البته الگوریتم های ELS و RLS مشکلی از جهت بازه بهم نمیکنند.

پس با تغییر بازه به [0.5  -0.5] داریم:




شکل 2: تابع $10x^4\frac{e^x-e^{-x}}{2}$ در بازه [0.5 -0.5]

چون هدف از پروژه مدلسازی یک سیستم غیرخطی می باشد پس تغییر بازه تا زمانی که غیرخطی بودن سیستم همچنان برقرار باشد موردی ندارد و همانگونه که مشخص می باشد سیستم در بازه [0.5 -0.5] کاملا شکل غیرخطی خود را حفظ کرده و هدف پروژه را ارضا می کند.

در ادامه بخش ها به طراحی سیستم فازی f(x) برای تابع $g(x)=10x^4\frac{e^x-e^{-x}}{2}$ در بازه [0.5 -0.5] می پردازیم.

# بخش 2

# طراحی سیستم فازی با روش گرادیان نزولی

## 2-1 روند طراحی

در روند طراحی سیستم فازی با روش گرادیان نزولی با تعیین خطای مطلوب در ادامه و تولید تعداد قواعد فازی به گونه ای که با انتخاب تابع گوسین به عنوان تابع عضویت شدت های آتش را بر اساس تعیین تعداد قواعد بدست آورده و با بدست آوردن مراکز ثقل بهینه توسط خروجی سیستم فازی را بدست می آوریم.هدف از پیاده سازی این اگوریتم به حداقل رساندن خطا با بروزرسانی پارامتر های متغیری که در ادامه روند طراحی گفته می شود.

هدف ما در کل رسیدن به خروجی مطلوب فازی به ازا زوج ورودی و خروجی می باشد. یعنی حداقل کردن خطا بین خروجی فازی و خروجی سیستم اصلی.

خروجی سیستم فازی به این شکل محاسبه می شود:

$$f(x)=\frac{\sum_{l=1}^{m}\bar{y}^{l}[\prod_{i=1}^{n}\exp(-(\frac{x_i-\bar{x}_i^{l}}{\sigma_i^{l}})^2)]}{\sum_{l=1}^{m}[\prod_{i=1}^{n}\exp(-(\frac{x_i-\bar{x}_i^{l}}{\sigma_i^{l}})^2)]}$$

در روند طراحی مقدار m ثابت در نظر گرفته می شود و پارامتر های $\sigma_i^{l}$ و $\bar{x}_i^{l}$ و $\bar{y}^{l}$ متغیر می باشند. هدف ما از این طراحی بکار گیری روش کرادیان نزولی برای بدست آوردن پارامتر های متغیر فوق می باشد.

سیستم فازی به این شکل عمل می کند:

ابتدای امر زاویه آتش تولید میشود:

$$z^{l}=\prod_{i=1}^{n}\exp(-(\frac{x_i-\bar{x}_i^{l}}{\sigma_i^{l}})^2)]$$

که این زاویه آتش وابسته به پارامتر های متغیر $\sigma_i^{l}$ و $\bar{x}_i^{l}$ که باید در روند الگوریتم مدام محاسبه شوند.

در مرحله بعد دو مقدار زیر محاسبه می شوند که خروجی سیستم فازی را باعث می شوند:

$$a = \sum_{l=1}^{m} \bar{y}^l z^l$$

$$b = \sum_{l=1}^{m} z^l$$

$$y = f(x) = \frac{a}{b}$$

هدف ما از این پیاده سازی این می باشد که مقدار $e^p = \frac{1}{2}[f(x_0{}^p) - y_0{}^p]^2$ کمینه شود.

روند بروزرسانی پارامتر های متغیر طبق فصل 13 کتاب وانگ در کد پیاده سازی شده است.

در ادامه :

با تغییر مقدار learning rate و m الگوریتم پیاده سازی می شود:

حالت های مختلف:

1-با خطای مطلوب $\varepsilon = 0.001$ و مقدار m=70 و learning rate=0.1 روند طراحی سیستم پیاده سازی می شود.

2- با خطای مطلوب $\varepsilon = 0.001$ و مقدار m=100 و learning rate=0.1 روند طراحی سیستم پیاده سازی می شود.

3- با خطای مطلوب $\varepsilon = 0.001$ و مقدار m=40 و learning rate=0.1 روند طراحی سیستم پیاده سازی می شود.

4- با خطای مطلوب $\varepsilon = 0.001$ و مقدار m=70 و learning rate=1 روند طراحی سیستم پیاده سازی می شود.

5- با خطای مطلوب $\varepsilon = 0.001$ و مقدار m=70 و learning rate=0.01 روند طراحی سیستم پیاده سازی می شود.

Learning rate گام حرکت در مسیر بروزرسانی پارامتر ها روی سیستم اصلی می باشد که بین 0 تا 1 تعیین می شود.

```
function fuzzy_system = fuzzy_modeling_3(input_data, output_data, M, alpha, max_epochs, epsilon)

    [num_samples, n] = size(input_data);


    % Centers of input membership functions
    x_bar = zeros(n, M);
    for i = 1:n
        x_bar(i,:) = linspace(min(input_data(:,i)), max(input_data(:,i)), M);
    end

    % Widths of input membership functions
    sigma = zeros(n, M);
    for i = 1:n
        sigma(i,:) = (max(input_data(:,i)) - min(input_data(:,i))) / M * 1.5; % Wider coverage
    end

    % Centers of output membership functions
    y_bar = linspace(min(output_data), max(output_data), M);

    % Training loop max_epoch=q according to wang book
    for epoch = 1:max_epochs
        total_error = 0;

        for p = 1:num_samples
            x0 = input_data(p,:)';
            y0 = output_data(p);

            % Calculate firing strengths (z^l)
            z = ones(1, M);
            for l = 1:M
                for i = 1:n
                    z(l) = z(l) * exp(-((x0(i) - x_bar(i,l)) / sigma(i,l))^2);
                end
            end

            % calculating output
            b = sum(z);
            a = sum(y_bar .* z);
            f = a / b;  % Final output

            % Error calculation
            e = 0.5 * (f - y0)^2;
            total_error = total_error + e;


            % Update y_bar
            for l = 1:M
                y_bar(l) = y_bar(l) - alpha * (f - y0) * (1 / b) * z(l);
            end

            % Update x_bar and sigma
            for l = 1:M
```

```
            for i = 1:n

                % Update x_bar
                x_bar(i,l) = x_bar(i,l) - alpha * ...
                ((f - y0) * (y_bar(l) - f) * z(l) / b) * ...
                        2*(x0(i) - x_bar(i,l)) / (sigma(i,l)^2);

                % Update sigma
                sigma(i,l) = sigma(i,l) - alpha *...
                ((f - y0) * (y_bar(l) - f) * z(l) / b) * ...
                        2*(x0(i) - x_bar(i,l))^2 / (sigma(i,l)^3);
            end
        end
    end

    fprintf('Epoch %d: Error = %.4f\n', epoch, total_error);

    % Early stopping if error is below threshold
    if total_error < epsilon
        break;
    end
end

% Store final parameters
fuzzy_system = struct();
fuzzy_system.M = M;
fuzzy_system.x_bar = x_bar;
fuzzy_system.sigma = sigma;
fuzzy_system.y_bar = y_bar;

% Evaluation function
fuzzy_system.evaluate = @(x) evaluate_fuzzy_system(x, x_bar, sigma, y_bar);
end

function y = evaluate_fuzzy_system(x, x_bar, sigma, y_bar)
% Evaluate fuzzy system for input x
[n, M] = size(x_bar);
z = ones(1, M);

% Calculate firing strengths
for l = 1:M
    for i = 1:n
        z(l) = z(l) * exp(-((x(i) - x_bar(i,l)) / sigma(i,l))^2);
    end
end

% Compute output
b = sum(z);
a = sum(y_bar .* z);
y = a / b;
end

clc
clear all
close all

%enter input threshold
xmin=input('enter down bound of input:');
xmax=input('enter up bound of input:');
```

```
% Generate sample data for y = 10x^4*cosh(x)
x = linspace(xmin, xmax, 100)';
y = (10.*(x.^4).*((exp(x)+exp(-1.*x))./2)) ;
% Training parameters
M = input('enter mont of fuzzy membership functions:');
alpha = input('enter learning rate:');
max_epochs = input('enter mont of epochs:');
epsilon = input('enter desirable precision:');


% Train fuzzy system
fuzzy_sys = fuzzy_modeling_3(x, y, M, alpha, max_epochs, epsilon);

% Test the system
y_pred = arrayfun(@(xi) fuzzy_sys.evaluate(xi), x);

% Plot results
figure,plot(x, y, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Actual Function');
grid on;
figure,plot(x, y_pred, 'r--', 'LineWidth', 2, 'DisplayName', 'Fuzzy Approximation');
grid on;
title('Fuzzy System Approximation of y = 10x^4 cosh(x)');
figure,plot(x, y_pred, 'r--',x, y, 'b-');
grid on;
xlabel('x');
ylabel('y');
legend();
figure,plot(x, y-y_pred, 'g', 'LineWidth', 2, 'DisplayName', 'error');
grid on;
title(‘error’);
```

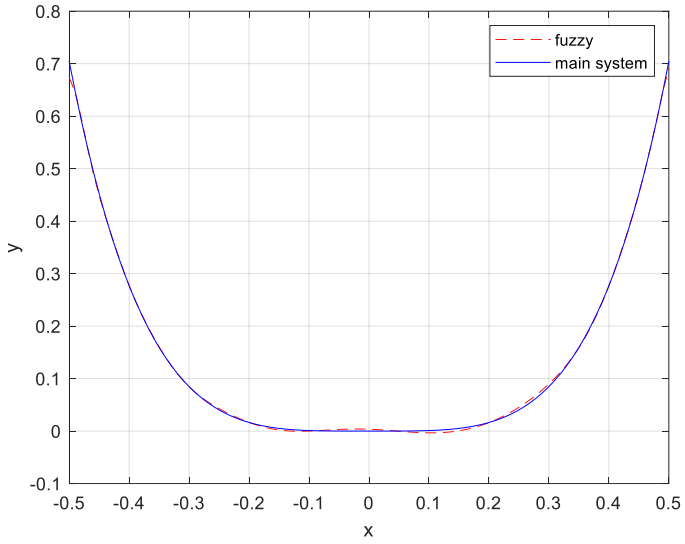
باید توجه کرد که max_epoch تعریف شده در کد مفهوم زمانی دارد و چون در شبیه سازی با مفهوم زمان ارتباطی گرفته نمی شود آن را با epoch مدل می کنند. در روند شبیه سازی مقدار max_epochs برابر با 20000 گرفته می شود.

در زیر بخش بعد به نمایش خروجی ها و تحلیل آنها می پردازیم.

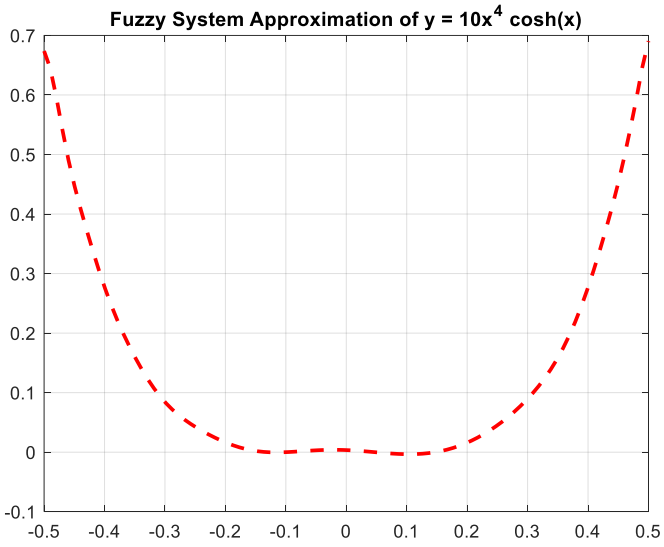
## 2-2 خروجی ها

**حالت 1: در epoch = 2676 توانسته تابع هزینه را به مقدار 0.001 برساند**

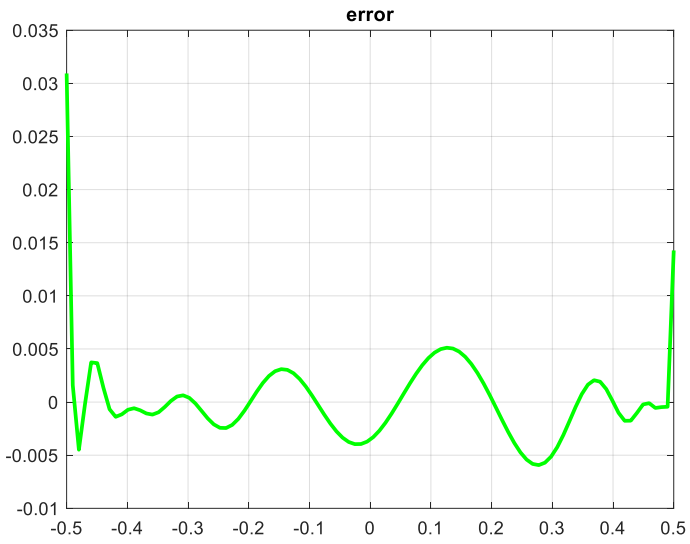
خروجی سیستم فازی و سیستم اصلی  برای حالت 1:




خروجی سیستم فازی:

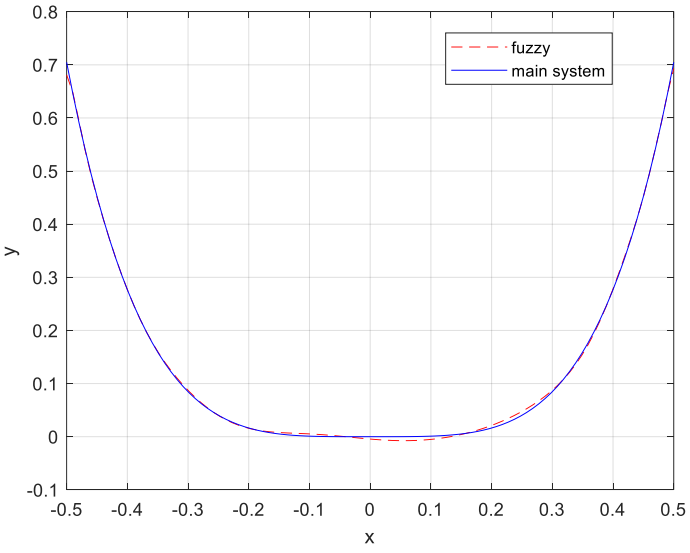



خطای بین سیستم اصلی و سیستم فازی حالت 1:

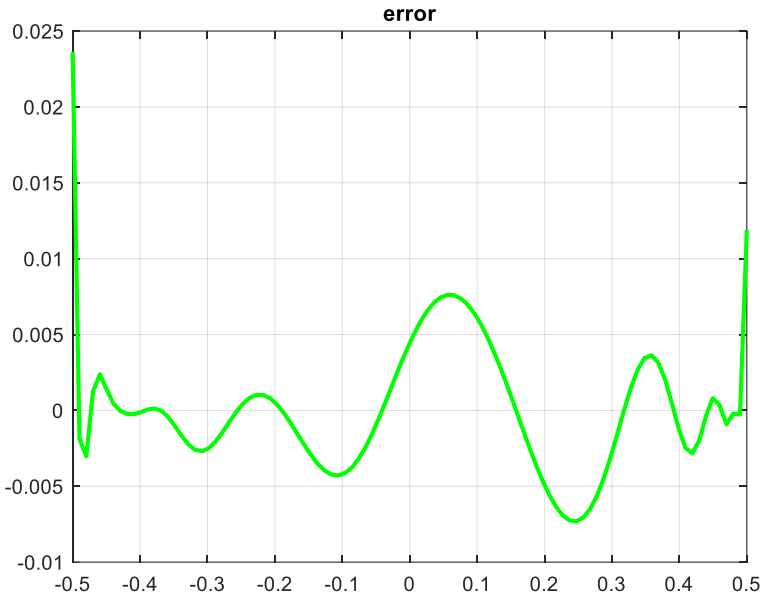



**حالت 2: در epoch = 826 توانسته تابع هزینه را به مقدار 0.001 برساند**

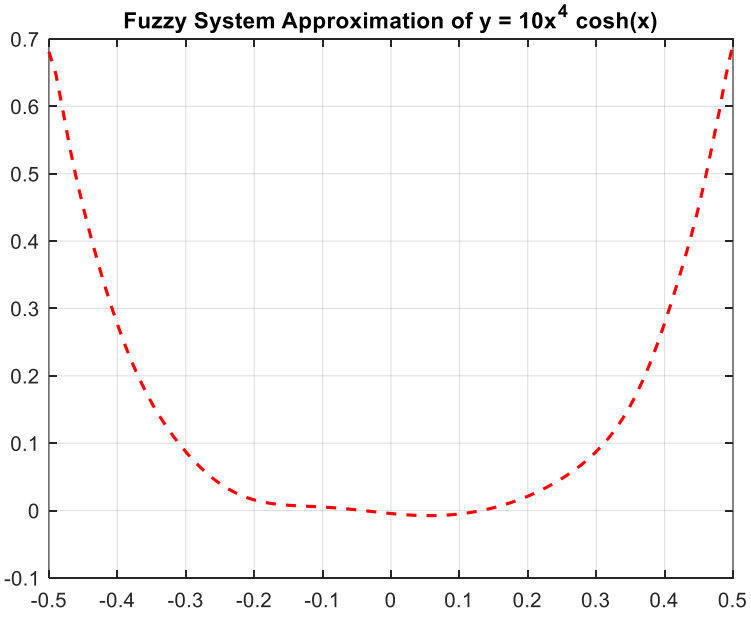
خروجی سیستم فازی و سیستم اصلی  برای حالت 2:




خطای بین سیستم اصلی و سیستم فازی حالت 2:

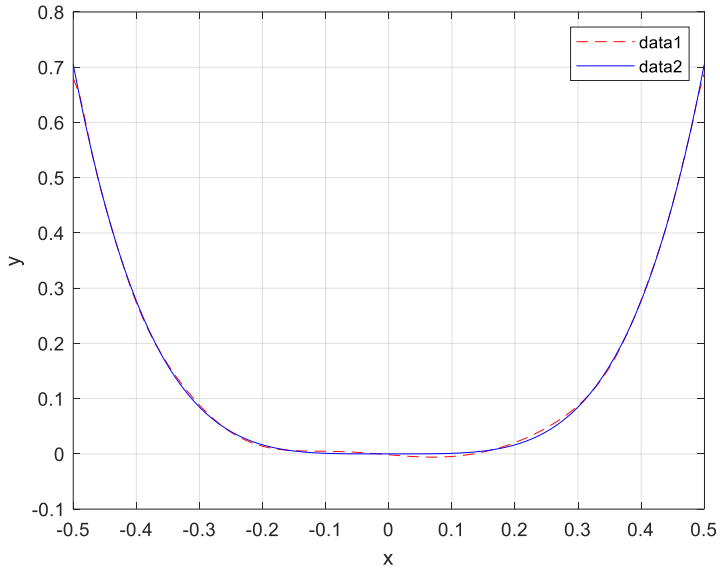



خروجی سیستم فازی حالت 2:

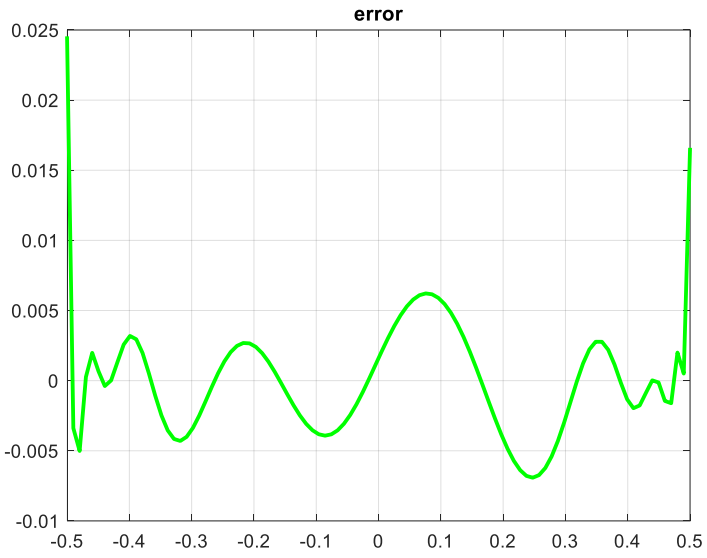



**حالت سوم: در epoch = 1840 توانسته تابع هزینه را به مقدار 0.001 برساند**

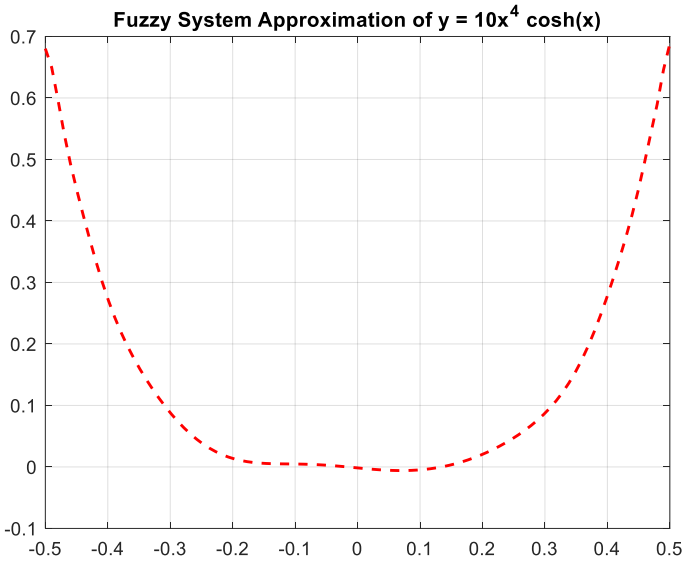
خروجی سیستم فازی و سیستم اصلی  برای حالت 3:




خطای بین سیستم اصلی و سیستم فازی حالت 3:

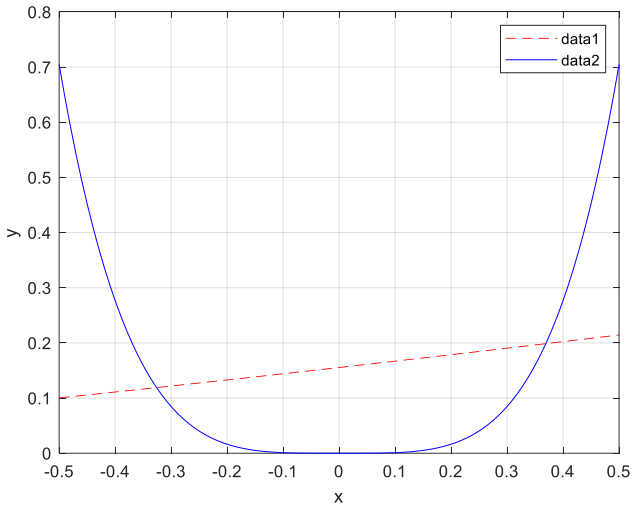



خروجی سیستم فازی حالت 3:

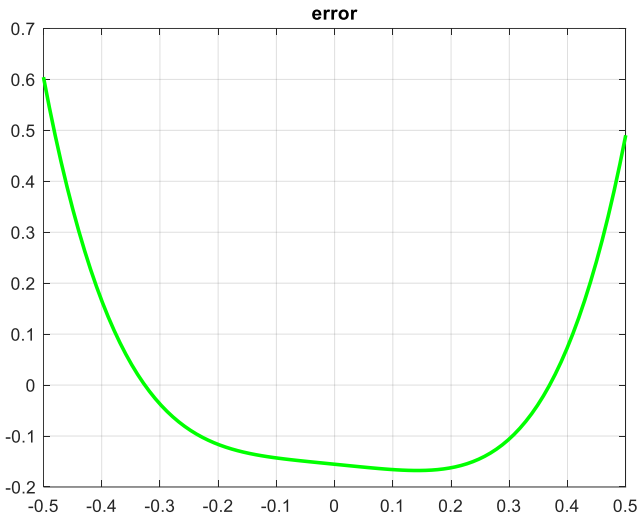



**حالت 4: در این حالت بدلیل بزرگ بودن learning rate=1 نتوانست سیستم فازی را به درستی طراحی کند.**

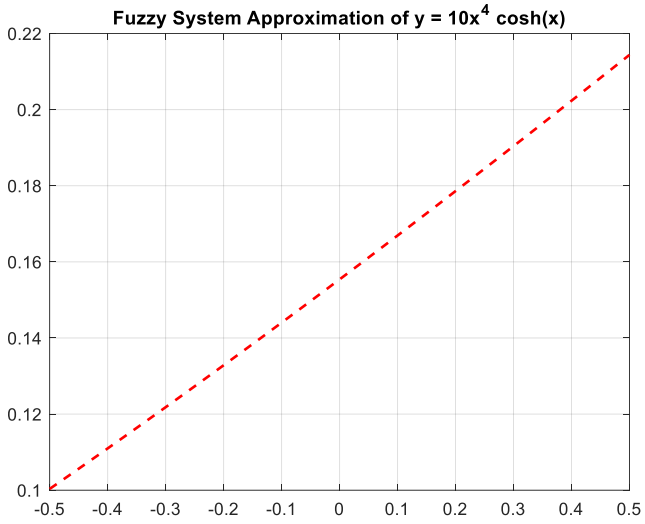
خروجی سیستم فازی و سیستم اصلی  برای حالت 4:




خطای بین سیستم اصلی و سیستم فازی حالت 4:

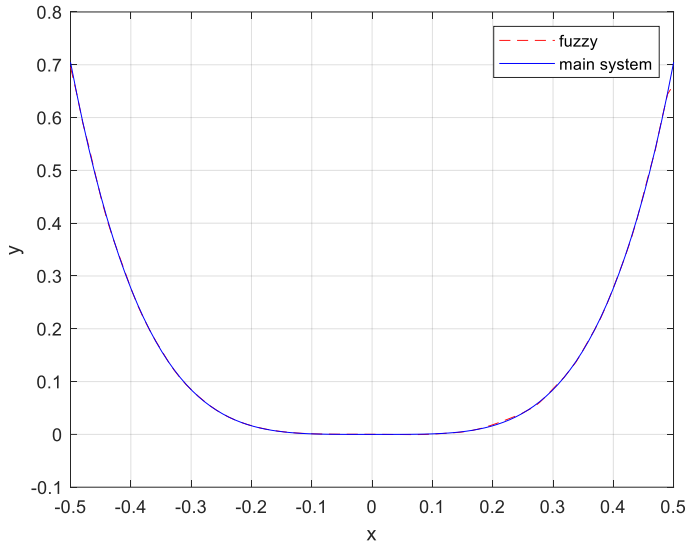



خروجی سیستم فازی حالت 4:

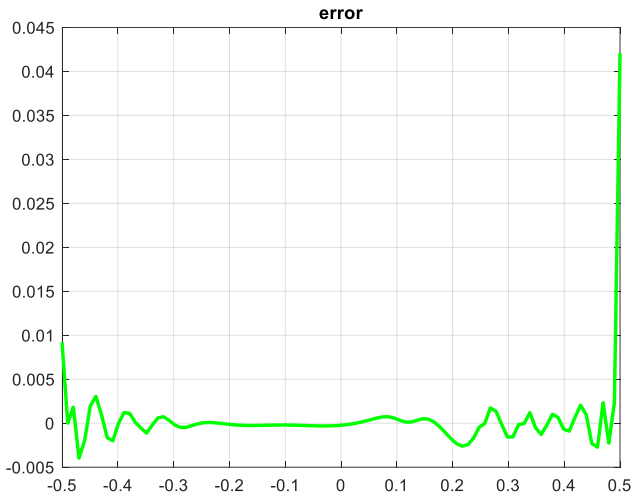



**حالت 5: در epoch = 1594 توانسته تابع هزینه را به مقدار 0.001 برساند**

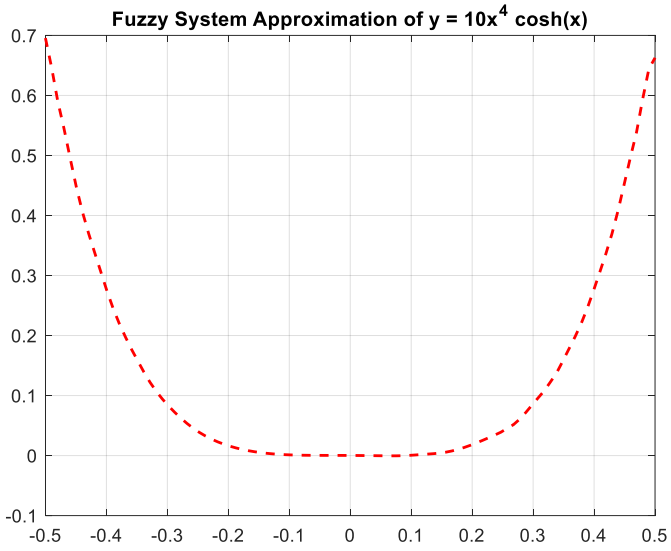
خروجی سیستم فازی و سیستم اصلی  برای حالت 5:




خطای بین سیستم اصلی و سیستم فازی حالت 5:




خروجی سیستم فازی حالت 5:




**تحلیل:**

در حالت های مختلف با مشخص می باشد مدل سازی که قرار است انجام بدهیم بر مقدار learning rate بسیار حساس می باشد و بزرگ تعیین کردن learning rate باعث مدل سازی اشتباه می شود اما قابل مشاهده است که در این سیستم غیر خطی هرچه مقدار learning rate کوچکتر بوده سیستم مدل شده فازی خطای بسیار کمتری داشته و تعداد توابع عضویت نیز در حد خود نیز تاثیر گذار بوده و هرچه مقدار آن بیشتر بوده در ابتدای امر خطای کمتری داشته ولی در گذر زمان خطا بیشتر می شود پس برای دست یابی به خطای کم در ابتدای مسیر و در گذر زمان باید trade off انجام داد و با آزمون و خطا مقدار آن را تعیین کرد .البته با الگوریتم های بهینه سازی نیز به این امر می توان دست یافت.(مانند الگوریتم ژنتیک)

# بخش 3

# طراحی سیستم فازی با روش حداقل مربعات خطا

## 1-3 روند طراحی برای داده های pure با روش RLS و تحلیل خروجی ها

در روند طراحی سیستم فازی با روش RLS به این گونه پیش می رویم که توابع عضویت ما گوسی می باشد . در روند طراحی باید ابتدای کار بردار رگرسور تشکیل داد که بردار رگرسور به شکل زیر می باشد:

$$b=\frac{\prod_{i=1}^{n}\exp(-(\frac{x_i-\bar{x}_i^{\,l}}{\sigma_i^{\,l}})^2)]}{\sum_{l=1}^{m}[\prod_{i=1}^{n}\exp(-(\frac{x_i-\bar{x}_i^{\,l}}{\sigma_i^{\,l}})^2)]}$$

در ادامه کار برای حل معادله زیر که خروجی مدل فازی می باشد باید مقدار اولیه ی $\theta(0)$ تعیین کرد که همان مراکزِ ثقل ما می باشند که با ضرب در بردار رگرسور ما خروجی فازی را برای ما تولید می کند. $\theta(0)$ به ازای هر ورودی بروزرسانی شده تا اینکه مراکز ثقل ما به مقدار نهایی خود همگرا شده.

همانطور که مشخص است سیستم ما سیستم استاتیکی می باشد و بردار رگرسور در ابتدای امر شامل مقادیر خود می باشد. چالش اصلی ما بروزرسانی پارامترها $\theta$ می باشد که به مقدار نهایی خود همگرا شوند.

تعیین خروجی:

$$f(x)=b^T\theta$$

مراحل بروزرسانی :

$$\theta(p)=\theta(p-1)+K(p)[y_0^p-b^T(x_0^p)\theta(p-1)]$$
$$K(p)=P(p-1)b(x_0^p)[b^T(x_0^p)P(p-1)b(x_0^p)+1]^{-1}$$
$$P(p)=P(p-1)-K(p)b^T(x_0^p)P(p-1)$$

کد برای سیستم در بازه [1 -1] پیاده سازی می شود و مقدار اولیه ماتریس P برابر با $P=1000\times I$ و مقدار اولیه مرکز ثقل ها پارامتر اول برابر 1 و بقیه پارامتر ها یزلبز صفر گرفته شده و تعداد داده های ورودی 10000 و تعداد قواعد هم 30 تا تعیین شده است.

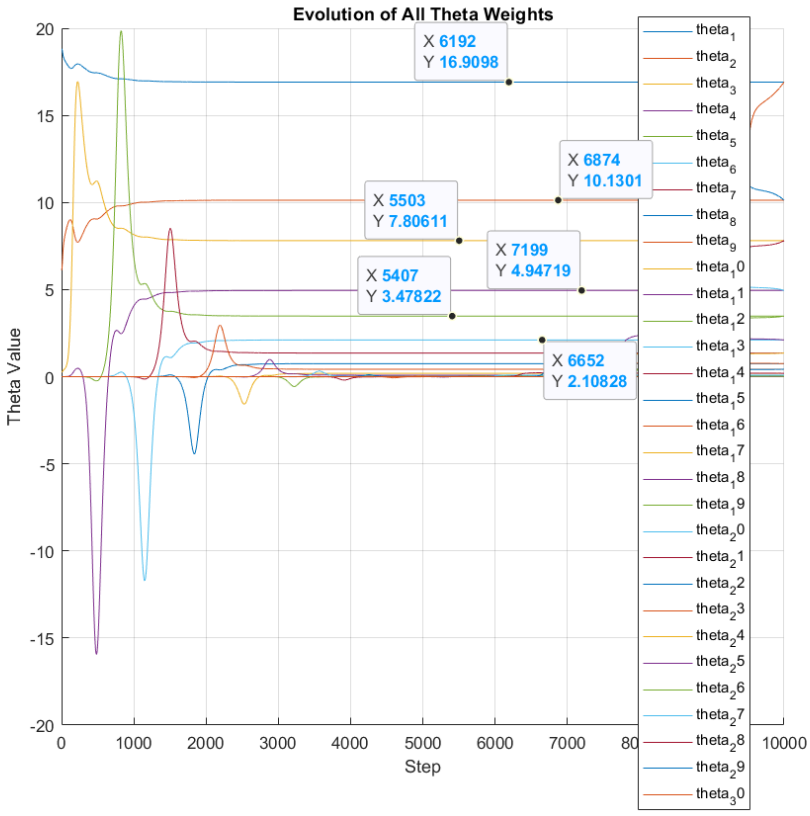
```
clc
clear all
```

```
close all

% User Inputs
min_input = input('Enter lower bound of input: ');
max_input = input('Enter upper bound of input: ');
num_rules = input('Enter number of rules: ');
learning_rate = input('Enter learning rate for P matrix: ');
num_samples = input('Enter number of input samples: ');

% Initialize Fuzzy System
centers = linspace(min_input, max_input, num_rules);
width = (max_input - min_input) / num_rules;
theta = eye(num_rules, 1);
P_matrix = learning_rate * eye(num_rules);

% Storage
predictions = [];
errors = [];
true_outputs = [];
inputs = linspace(min_input, max_input, num_samples)';
trace_p = [];
mean_errors = [];
theta_history = zeros(num_samples, num_rules);

for i = 1:num_samples

    x = inputs(i);
    y = 10 * (x^4) * cosh(x);
    true_outputs = [true_outputs; y];

    % making regressor vector with fuzzy rules
    mu = exp(-((x - centers)/width).^2)';
    b = mu / sum(mu);

    % Fuzzy system prediction
    y_pred = b' * theta;

    % Weight Update
    K = P_matrix * b / (1 + b' * P_matrix * b);
    theta = theta + K * (y - y_pred);
    P_matrix = P_matrix - K * b' * P_matrix;
    trace_p = [trace_p; trace(P_matrix)];

    % Store all theta values
    theta_history(i, :) = theta';

    % Store results
    predictions = [predictions; y_pred];
    errors = [errors; (y - y_pred)];
    current_mean_error = mean(errors(1:i));
    mean_errors = [mean_errors; current_mean_error];

    % Display progress
    fprintf('Step %d: x=%.4f, y_true=%.4f, y_pred=%.4f, Mean Error=%.4f\n', i,
x, y, y_pred, current_mean_error);
end

% plots
```

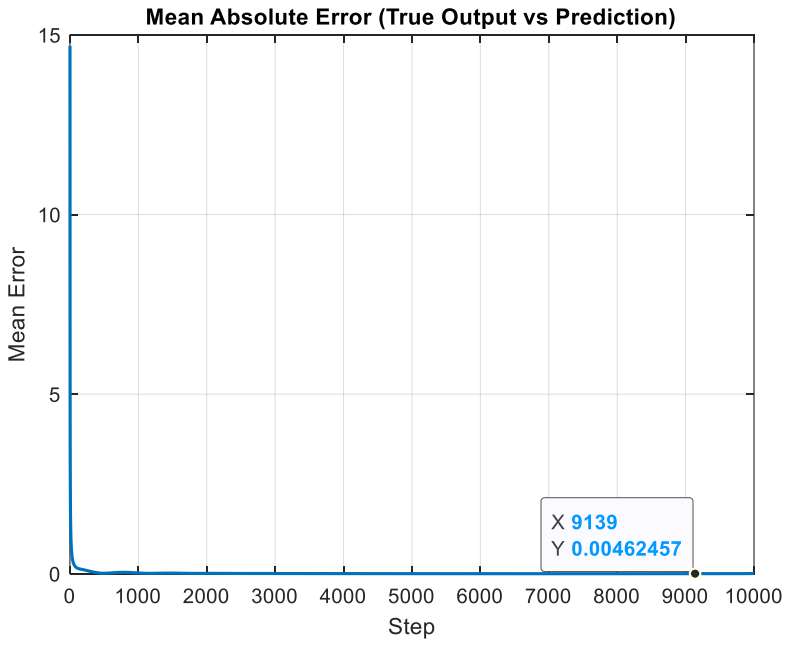
```
% main System and fuzzy modeling system
figure;
plot(inputs, true_outputs, inputs, predictions, 'r--');
legend('True','Fuzzy');
title('main System and fuzzy modeling system');
grid on;


% Error between true output and prediction
figure;
plot(true_outputs - predictions);
title('Error between True Output and Prediction');
grid on;

% Trace of P matrix
figure;
plot(trace_p);
title('Trace of P Matrix');
grid on;

% Plot all theta weights
figure;
hold on;
for k = 1:num_rules
    plot(theta_history(:, k), 'DisplayName', sprintf('theta_%d', k));
end
hold off;
title('Evolution of All Theta Weights');
xlabel('Step');
ylabel('Theta Value');
legend('show');
grid on;

% Mean Error plot
figure;
plot(mean_errors, 'LineWidth', 1.5);
title('Mean Absolute Error (True Output vs Prediction)');
xlabel('Step');
ylabel('Mean Error');
grid on;
```

**خروجی ها:**

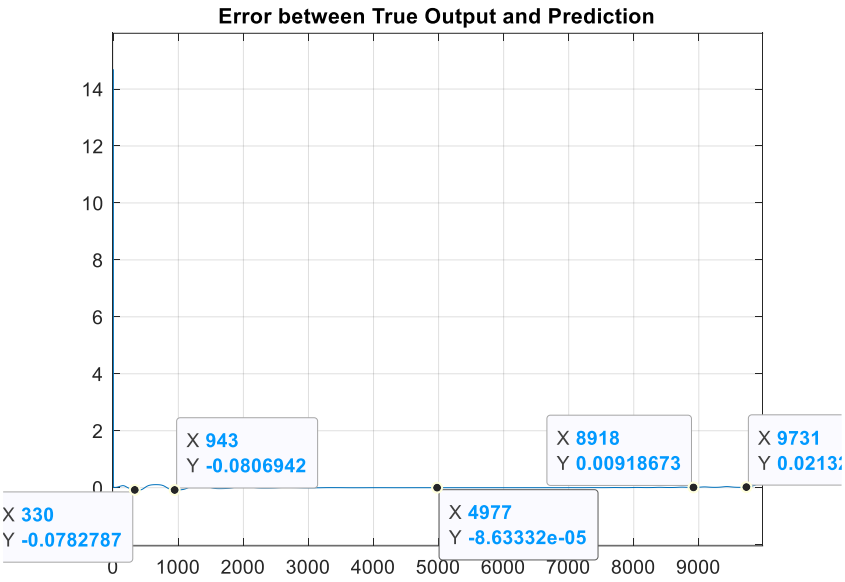
همگرایی پرامتر های بردار $\theta$:



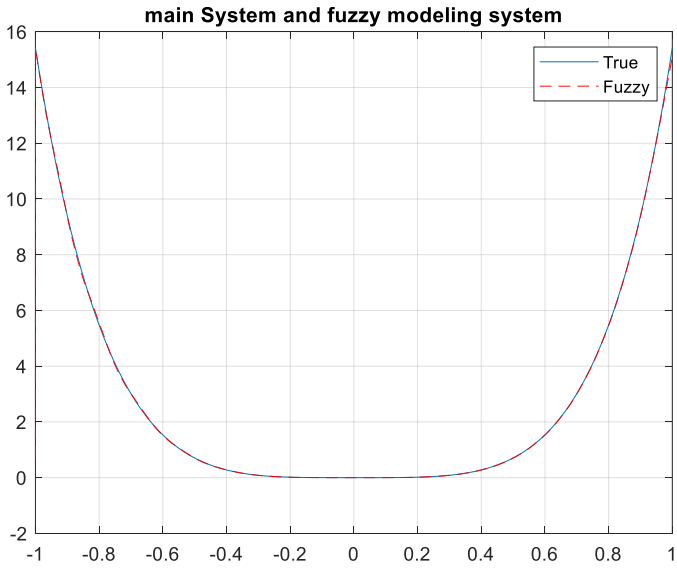

همگرایی میانگین خطا در طول فرآیند الگوریتم:



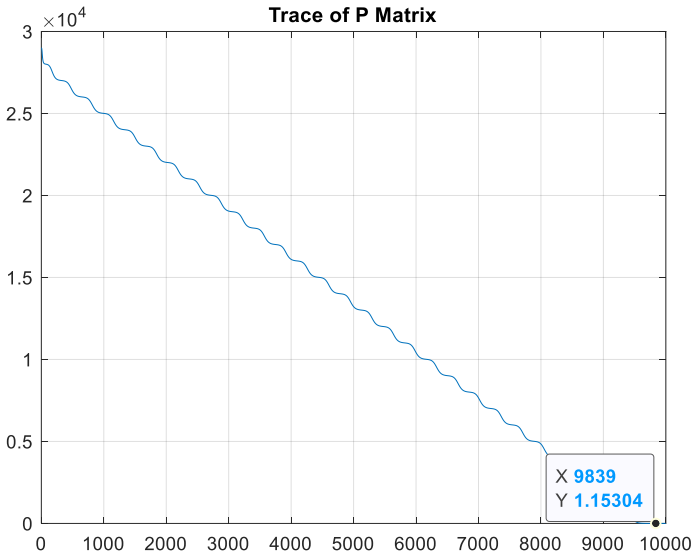
خطای بین سیستم اصلی و سیستم فازی:




سیستم اصلی و سیستم فازی:




همگرایی Trace ماتریس P :




**تحلیل:**

با توجه به trace ماتریس p کاملا مشخص است که به همگرایی لازم رسیدیم و مقدار آن تقریبا به صفر میل پیدا کرده پس می توان از روی پارامتر های تخمینی هم فهمید که به مقدار نهایی خور همگرا شده اند و این مورد ارتباط مستقیمی با همگرایی ماتریس P دارد. از روخی مقایسه ی خروجی اصلی و خروجی سیستم فازی کاملا قابل مشاهده می باشد به دلیل تعریف شرایط اولیه 1 برای تمام پارامتر های بردار $\theta$ خطای بزرگی داریم ولی رفته رفته با بروز رسانی پارامتر ها و همگرایی آنها خطای میان سیستم اصلی و سیستم فازی کاهش یافته و به نزدیک شدن به ورودی صفر خطا ی میان این دو تقریبا صفر می باشد ولی به دلیل تپولوژی سیستم زمانی که از 0 به سمت 1 حرکت میکنیم خطا کمی زیاد می شود ولی در حدود 0.001 و دلیل آن تغیر سریع و ناگهانی و حالت صعودی سیستم اصلی می باشد ولی همچنان الگوریتم به خوبی عمل کرده و خطای بسیار کوچکی داریم و همچنین از روی همگرایی میانگین خطا که در حدود 0.004 می باشد نشان دهنده عملکرد فوق العاده در تخمین پارامتر ها و unbiased بودت تخمین پارامترها می باشد.

## 2-3 روند طراحی برای خروجی های آغشته به نویز سفید با روش RLS و تحلیل خروجی ها

در این زیر بخش خروجی هارو آغشته به نویز سفید با standard deviation =0.1 و تعیین سایر مقادیر همانند زیر بخش 1 داریم:

کد مربوطه:

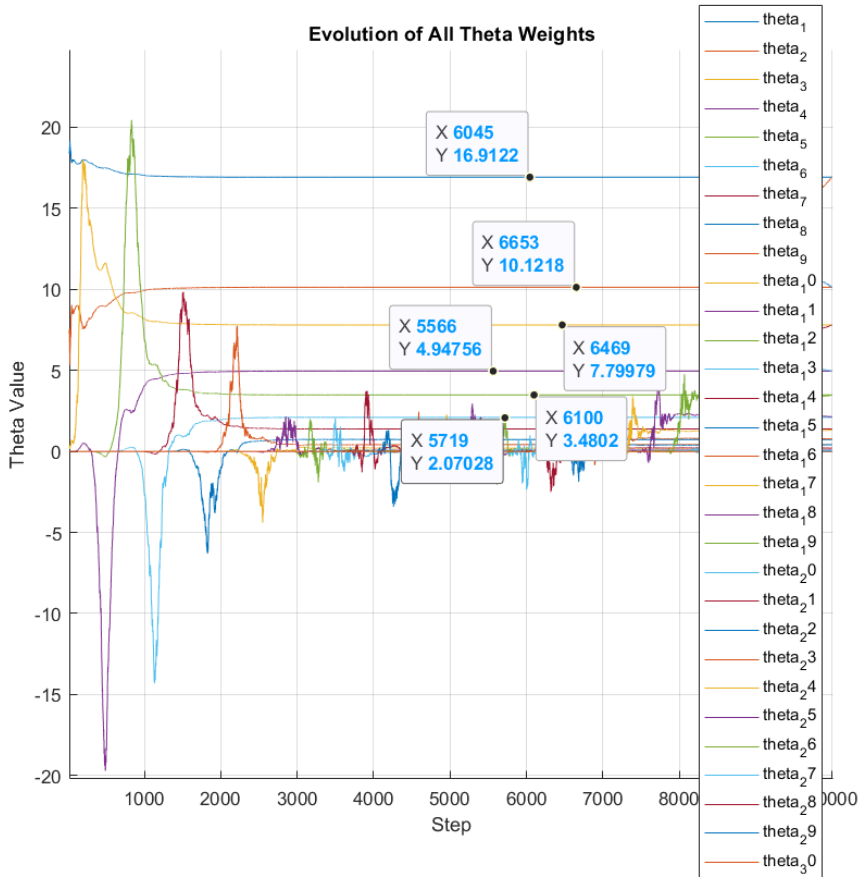
```
clc
clear all
close all

% Input the parameters
min_input = input('Enter lower bound of input: ');
max_input = input('Enter upper bound of input: ');
num_rules = input('Enter number of rules: ');
learning_rate = input('Enter learning rate for P matrix: ');
num_samples = input('Enter number of input samples: ');
noise_level = input('Enter noise level (standard deviation): ');

%  random number for fixing noise
rng(42);

% Initializing
centers = linspace(min_input, max_input, num_rules);
width = (max_input - min_input) / num_rules;
theta = eye(num_rules, 1);
P_matrix = learning_rate * eye(num_rules);

%Storage of Results
predictions = [];
true_outputs = [];
noisy_outputs = [];
inputs = linspace(min_input, max_input, num_samples)';
trace_p = [];
mean_errors = [];
theta_history = zeros(num_samples, num_rules);

% Generate noise
noise = noise_level * randn(num_samples, 1);
noise= noise(randperm(length(noise)));

% Training with Sequential Data
for i = 1:num_samples

    % making data
    x = inputs(i);
    y = 10 * (x^4) * cosh(x);
    y_noisy = y + noise(i);
    true_outputs = [true_outputs; y];
    noisy_outputs = [noisy_outputs; y_noisy];

    % making regressor vector with fuzzy rules
    mu = exp(-((x - centers)/width).^2)';
    b = mu/sum(mu);
```

```
    % Calculate fuzzy output
    y_pred = b' * theta;

    % Update weights
    K = P_matrix * b / (1 + b' * P_matrix * b);
    theta = theta + K * (y_noisy - y_pred);
    P_matrix = P_matrix - K * b' * P_matrix;
    trace_p = [trace_p; trace(P_matrix)];

    % Store all theta values
    theta_history(i, :) = theta';

    % Store results
    predictions = [predictions; y_pred];
    current_error = mean((true_outputs(1:i) - predictions(1:i)));
    mean_errors = [mean_errors; current_error];

    % Display progress
    fprintf('Step %d: x=%.4f, y_true=%.4f, y_noisy=%.4f, y_pred=%.4f\n , mean_errors=%.4f\n', i, x, y,
y_noisy, y_pred, current_error);
end

% plots

% main System and fuzzy modeling system
figure;
plot(inputs, true_outputs, inputs, predictions, 'r--');
legend('True','Fuzzy');
title('main System and fuzzy modeling system');
grid on;
% Error between true output and prediction
figure;
plot(true_outputs - predictions);
title('Error between True Output and Prediction');
grid on;
% Trace of P matrix
figure;
plot(trace_p);
title('Trace of P Matrix');
grid on;
% Plot all theta
figure;
hold on;
for k = 1:num_rules
    plot(theta_history(:, k), 'DisplayName', sprintf('theta_%d', k));
end
hold off;
title('Evolution of All Theta Weights');
xlabel('Step');
ylabel('Theta Value');
legend('show');
grid on;


% Mean Error plot
figure;
plot(mean_errors, 'LineWidth', 1.5);
title('Mean Error (True Output vs Prediction)');
```

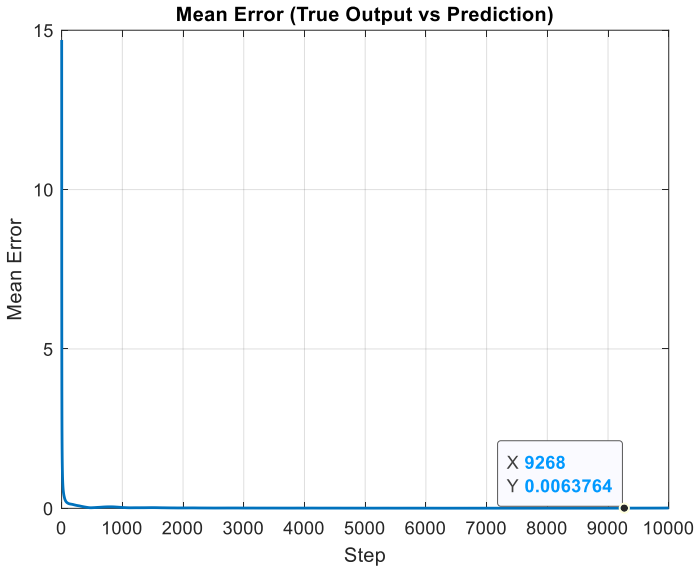
```
xlabel('Step');
ylabel('Mean Error');
grid on;
```

**خروجی ها:**

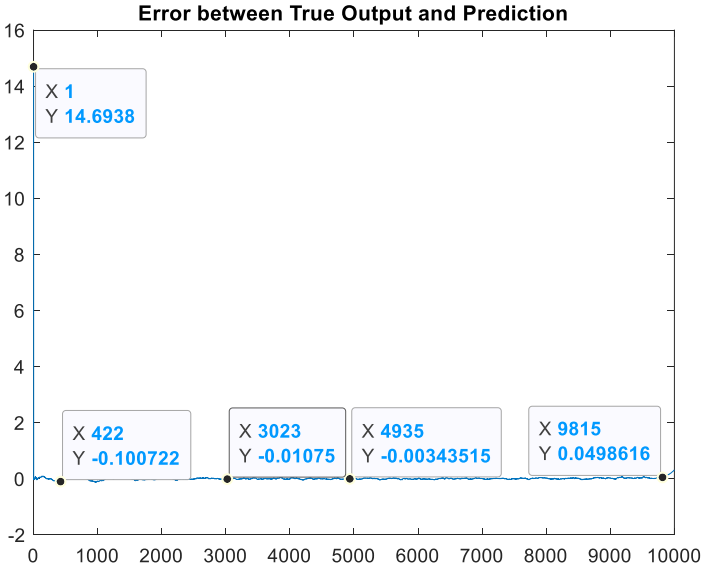
همگرایی پرامتر های بردار $\theta$:



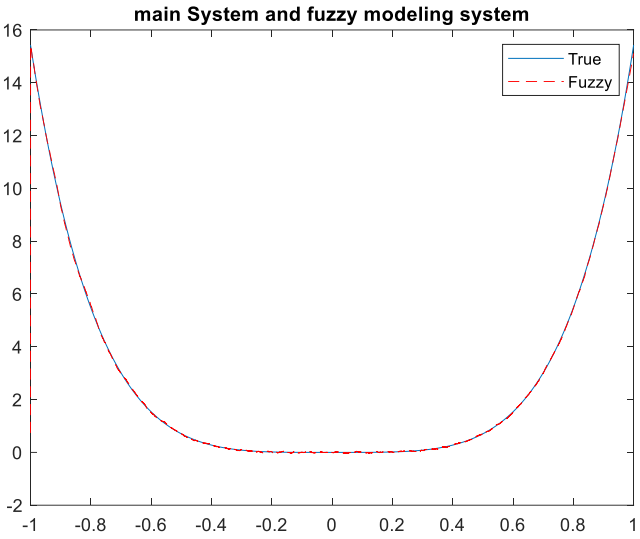

همگرایی میانگین خطا در طول فرآیند الگوریتم:



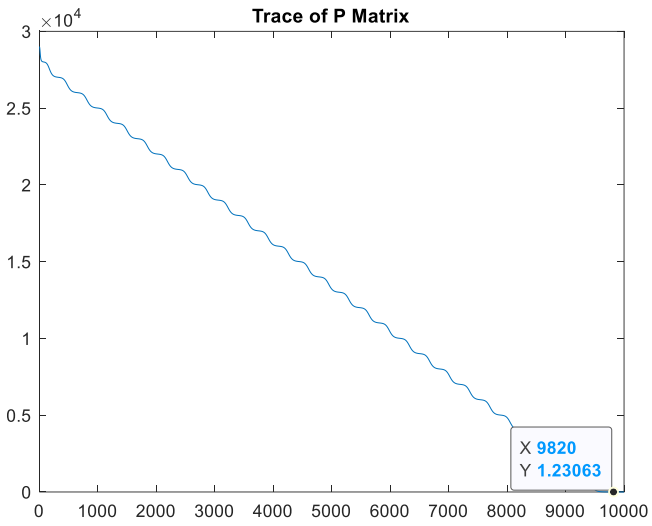

خطای بین سیستم اصلی و سیستم فازی:




سیستم اصلی و سیستم فازی:




همگرایی Trace ماتریس P :




**تحلیل:**

همانگونه که از روی trace ماتریس P مشخص می باشد بازهم همانند زیربخش 1 trace ماتریس P به صفر میل پیدا کرده و همگرا شده است و از اینرو تخمین پارامترها در ابتدای امر نویز روی آن تاثیر اندکی گذاشته است ولی در نهایت همه ی پارامترها ی $\theta$ با اختلاف بسیار اندکی در حد 0.01 یا حتی کمتر

نسبت به زیر بخش 1 به همگرایی رسیده اند و نشان دهنده unbiasedبودن پارامتر های تخمین زده شده می باشد و این نشان دهنده ی این است که الگوریتم RLS نسبت به داده هایی نویزی البته نویز سفید مقاوم بوده و توانایی خود برایی تخمین پارامتر هارا از دست نمیدهد و این مقاومت نیز از روی مینگین خطای خروجی نیز مشخص بوده که تقریبا برابر با میانگین خطای همگرا شده در زیر بخش 1 می باشد. و از روی خطای میان سیستم اصلی و سیستم مدل سشده فازی نیز می توان متجه این شد که نویز اعمالی باعث افزایش خطا میام این دو شده نسبت به زیربخش 1 ولی این زیاد شدن مقدار بزرگی ندارد و دلیل آن وجود نویز سفید می باشد و امری منطقی می باشد.

## 3-3 روند طراحی برای خروجی های آغشته به نویز رنگی (قهوه ای) با روش RLS و تحلیل خروجی ها

در ادامه در این زیر بخش نویز رنگی(قهوه ای) به خروجی سیستم اصلی افزوده شده با standard deviation=0.1 و سایر مقادیر هم مانند دو زیر بخش قبل می باشد.

در ادامه با کد مربوطه داریم:

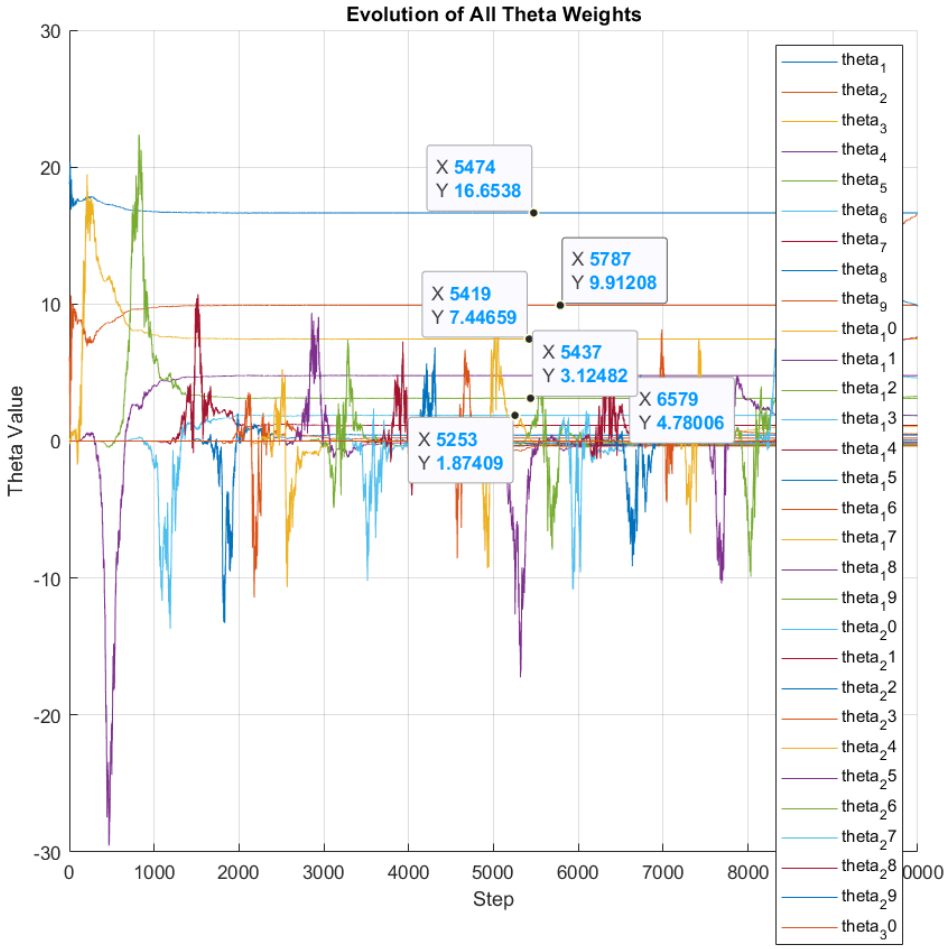
```
clc
clear all
close all

% Input the parameters
min_input = input('Enter lower bound of input: ');
max_input = input('Enter upper bound of input: ');
num_rules = input('Enter number of membership functions: ');
learning_rate = input('Enter learning rate for P matrix: ');
num_samples = input('Enter number of input samples: ');
noise_level = input('Enter noise level (standard deviation): ');



rng(42);

% Initializing
centers = linspace(min_input, max_input, num_rules);
width = (max_input - min_input) / num_rules;
theta = eye(num_rules, 1);
P_matrix = learning_rate * eye(num_rules);

%Storage of Results
predictions = [];
```

```
errors = [];
true_outputs = [];
noisy_outputs = [];
inputs = linspace(min_input, max_input, num_samples)';
trace_p = [];
mean_errors = [];
theta_history = zeros(num_samples, num_rules);

% Generate colored noise-brown
bnoise = zeros(num_samples, 1);
white_noise =noise_level* randn(num_samples,1);
bnoise=cumsum(white_noise);
bnoise=bnoise/max(abs(bnoise));
bnoise=bnoise(randperm(length(bnoise)));


for i = 1:num_samples

   % making data
   x = inputs(i);
   y = 10 * (x^4) * cosh(x);
   y_noisy = y + bnoise(i);
   true_outputs = [true_outputs; y];
   noisy_outputs = [noisy_outputs; y_noisy];

  % making regressor vector with fuzzy rules
   mu = exp(-((x - centers)/width).^2)';
   b = mu/sum(mu);

   % Calculate fuzzy output
   y_pred = b' * theta;

   % Update weights
   K = P_matrix * b / (1 + b' * P_matrix * b);
   theta = theta + K * (y_noisy - y_pred);
   P_matrix = P_matrix - K * b' * P_matrix;
   trace_p = [trace_p; trace(P_matrix)];

   % Store all theta values
   theta_history(i, :) = theta';

   % Store results
   predictions = [predictions; y_pred];
   current_error = mean((true_outputs(1:i) - predictions(1:i)));
   mean_errors = [mean_errors; current_error];

   % Display progress
   fprintf('Step %d: x=%.4f, y_true=%.4f, y_noisy=%.4f, y_pred=%.4f\n , mean_errors=%.4f\n', i, x, y,
y_noisy, y_pred,current_error);
end

% plots

% main System and fuzzy modeling system
figure;
plot(inputs, true_outputs, 'b', inputs, predictions, 'r--');
legend('True','Fuzzy');
title('main System and fuzzy modeling system');
grid on;
```

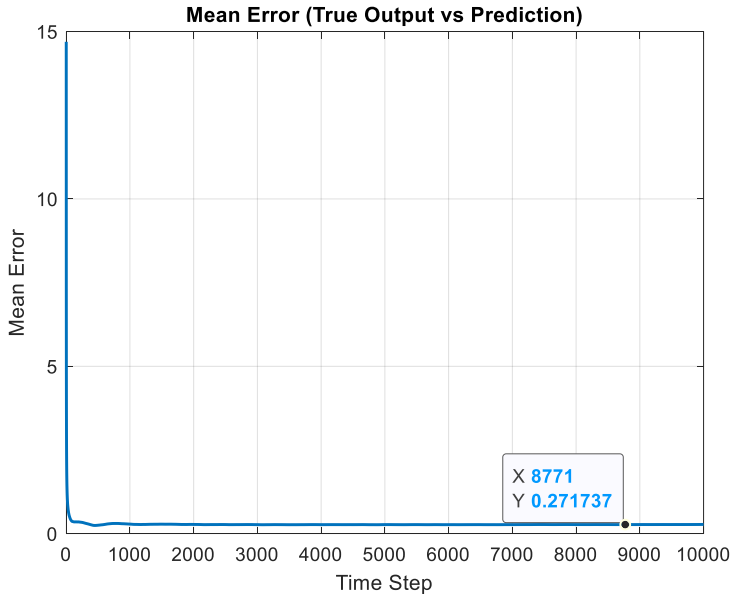
```
% Error between true output and prediction
figure;
plot(true_outputs - predictions);
title('Error between True Output and Prediction');
grid on;
% Trace of P matrix
figure;
plot(trace_p);
title('Trace of P Matrix');
grid on;
% Plot power spectral density of noise
figure;
pwelch(bnoise, [], [], [], 1);
title('Power Spectral Density of Colored Noise');
grid on;
% Plot all theta weights
figure;
hold on;
for k = 1:num_rules
    plot(theta_history(:, k), 'DisplayName', sprintf('theta_%d', k));
end
hold off;
title('Evolution of All Theta Weights');
xlabel('Step');
ylabel('Theta Value');
legend('show');
grid on;

% mean error plot
figure;
plot(mean_errors, 'LineWidth', 1.5);
title('Mean Error (True Output vs Prediction)');
xlabel('Time Step');
ylabel('Mean Error');
grid on;
```

**خروجی ها:**

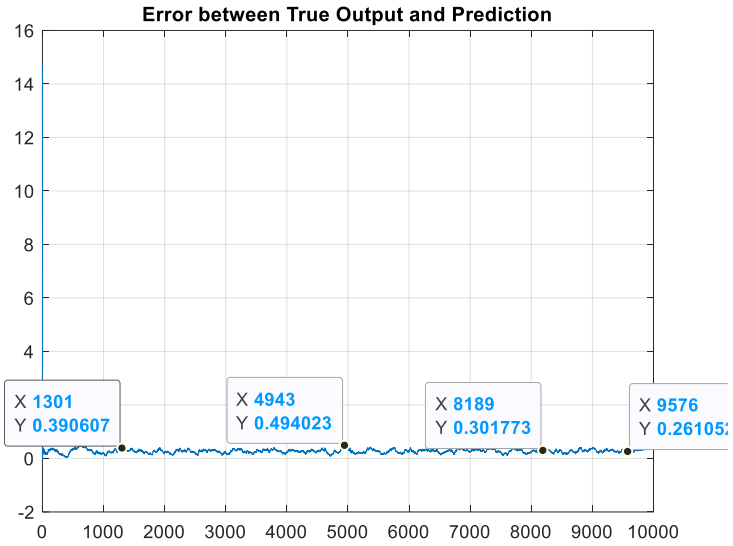
همگرایی پرامتر های بردار $\theta$:



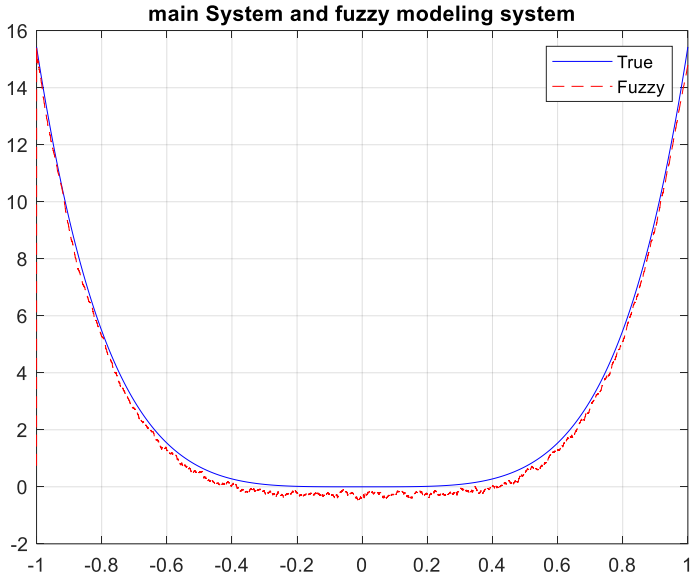

همگرایی میانگین خطا در طول فرآیند الگوریتم:



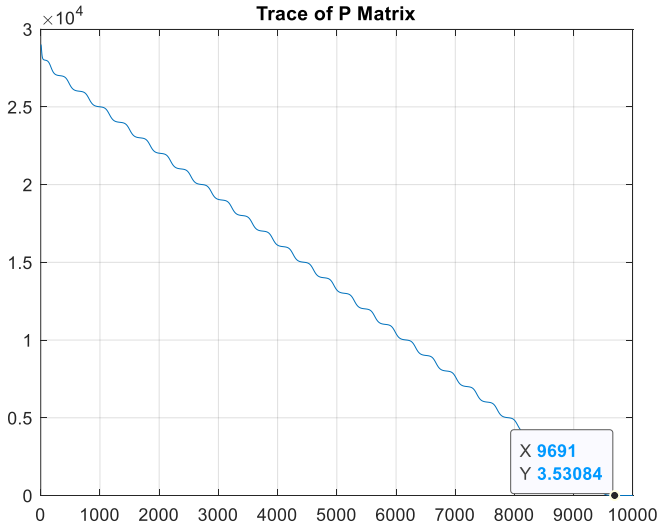
خطای بین سیستم اصلی و سیستم فازی:



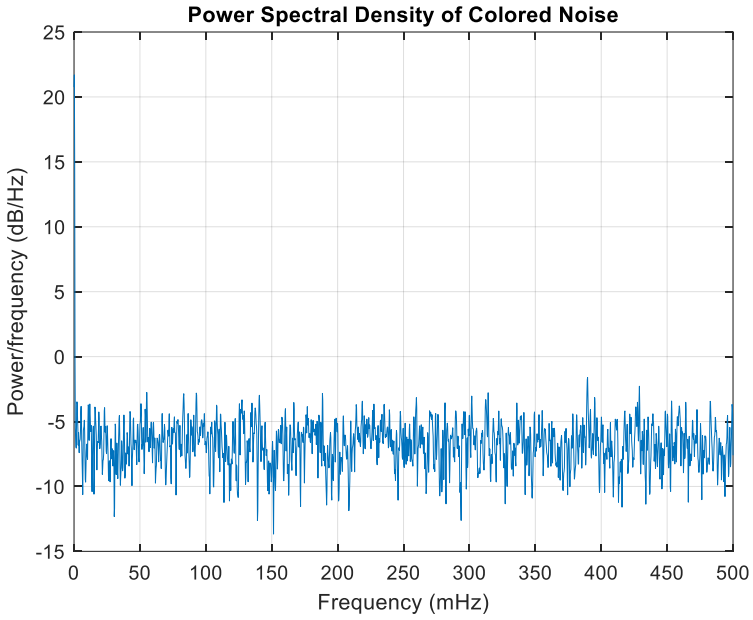

سیستم اصلی و سیستم فازی:




همگرایی Trace ماتریس P :




Power spectral نویز رنگی:




**تحلیل:**

با توجه به نمودار همگرایی پارامترهای بردار $\theta$ کاملا مشخص است که در ابتدای امر نویز رنگی تاثیر تقریبا زیادی روی تخمین پارامترها گذاشته است و در ادامه همگرا شده است اما این همگرایی با توجه به زیر

بخش 1 همگرایی با bias می باشد یا می توان گفت که تخمین پارامتر ها biased می باشد و همچنین از روی ارتباط مستقیم میان همگرایی پارامتر ها و همگرایی trace ماتریس p به صفر نیز پی برد و از روی نمودار آن کاملا مشهود است که trace ماتریس P تقریبا به صفر میل پیدا کرده است. این bias بوجود آمده در تخمین پارامتر ها موجب ایجاد bias در نمودار همگرایی میانگین خطا به صفر هم می شود و کاملا مشخص است که ئر نمودار میانگین خطا نیز bias وجود دارد و این bias حدود 0.27 که این عمر نشان دهنده وجود خطای مدل سازی فازی می باشد از روی نمودار خطای میان سیستم اصلی و همچنین نمایش دو سیستم روی هم قابل مشاهده می باشد که نشان دهنده عملکرد نسبتا نامقاوم الگوریتم RLS نسبت به وجود نویز رنگی می باشد. که در ادامه با الگوریتم ELS این امر را بهبود میبخشیم و الگوریتمی مقاوم تر نسبت به RLS طراحی میکنیم.

## 4-3 روند طراحی برای خروجی های آغشته به نویز رنگی (قهوه ای) با روش ELS و تحلیل خروجی ها

این الگوریتم تفاوت چندانی با الگوریتم RLS نداشته و عمل بروزرسانی آن همانند RLS می باشد . تنها تفاوت این الگوریتم بردار رگرسور آن می باشد که علی رغم وجود خود رگرسور b باید ترم خطا نیز به آن اضافه شود تا بتوان دینامیک نویز رنگی را در بردار $\theta$ تخمین زد و به توان به گونه ای با آن مقابله کرد.در ادامه با Extend یا گسترش بردار رگرسور داریم:

$$b^T = \left[ b^T \mid e \right]$$

و $\theta$ جدید هم به شکل زیر در می آید که ترم تخمین دینامیک نویز رنگی به آن افزوده شده است:

$$\theta = \left[ \frac{\theta}{C} \right]$$

خروجی سیستم فازی ما اینگونه تعیین میشود:

$$f(x) = \left[ b^T \mid e \right] \left[ \frac{\theta}{C} \right]$$

م همنطور که گفته شد بروزرسانی بردار تخمینی جدید هم مانند RLS صورت میگیرد:

$$\theta(p)=\theta(p-1)+K(p)[y_0^p-b^T(x_0^p)\theta(p-1)]$$
$$K(p)=P(p-1)b(x_0^p)[b^T(x_0^p)P(p-1)b(x_0^p)+1]^{-1}$$
$$P(p)=P(p-1)-K(p)b^T(x_0^p)P(p-1)$$

در ادامه روند پیاده سازی این الگوریتم روی این سیستم ابتدا برای 3 داده اول خطا ها تعیین شده و این 10 داده در بردار رگرسور جدید قرار میگیرد و برای سایر داده ها با استفاده از این بردار رگرسور جدید الگوریتم ELS پیاده سازی می شود و کار دیگه ای در پیاده سازی انجام شده این است که به دلیل ترم خطا در بردار رگرسور مقدار اولیه ماتریس P باید تقریبا خیلی کوچکتر نسبت به زیربخش های قبلی تعیین شود که در اینجا ماتریس P را به دوبخش تقسیم کرده که بخش اول مربوط به بردار رگرسور $b^T$ بوده و مقدار $P_{b^T}=5\times I$ و بخش دوم که مربوط به ترم خطا بوده را $P_e=2.5\times I$ تعیین کرده که این مقادیر با آزمون خطاهای متعدد بدست آمده و البته با الگوریتم های بهینه سلزی متنوعی نیز قابل پیاده سازی می باشد. و در ادامه ماتریس P اصلی که تلفیقی از این دو ماتریس در روند پیاده سازی می باشد بدست آورده می شود که با دستور $blkdig(p_{b^T},p_e)$ قابل پیاده سازی می باشد.در الگوریتم ElS تعیین مقدار اولیه برای بردار $\theta$ بسیار اهمیت دارد . در ادامه با تعیین شرایط اولیه 10 برای پارامترهای بخش اولیه که مربوط به $b^T$ و 0 مربوط به بخش خطا و سپس این دو بردار را تلفیق کرده و بردار $\theta$ جدید تشکیل می شود.

در ادامه با تعیین تعداد قواعد 30 و standard deviation=0.1و تعداد داده ها برابر با 10000 و همانطور که گفته شد تعداد خطای مورد استفاده در بردار رگرسور هم برابر با 3 می باشد.که در ادامه کد مربوطه آورده می شود و در انتها خروجی ها و تحلیل آنها آورده می شود:

کد مربوطه:

```
clc;
close all;
clear all;

% User-defined parameters
min_input = input('Enter lower bound of input: ');
max_input = input('Enter upper bound of input: ');
num_rules = input('Enter number of membership functions: ');
learning_rate_fuzzy = input('Enter learning rate for fuzzy part of P matrix: ');
learning_rate_error = input('Enter learning rate for error part of P matrix: ');
num_samples = input('Enter number of input samples: ');
```

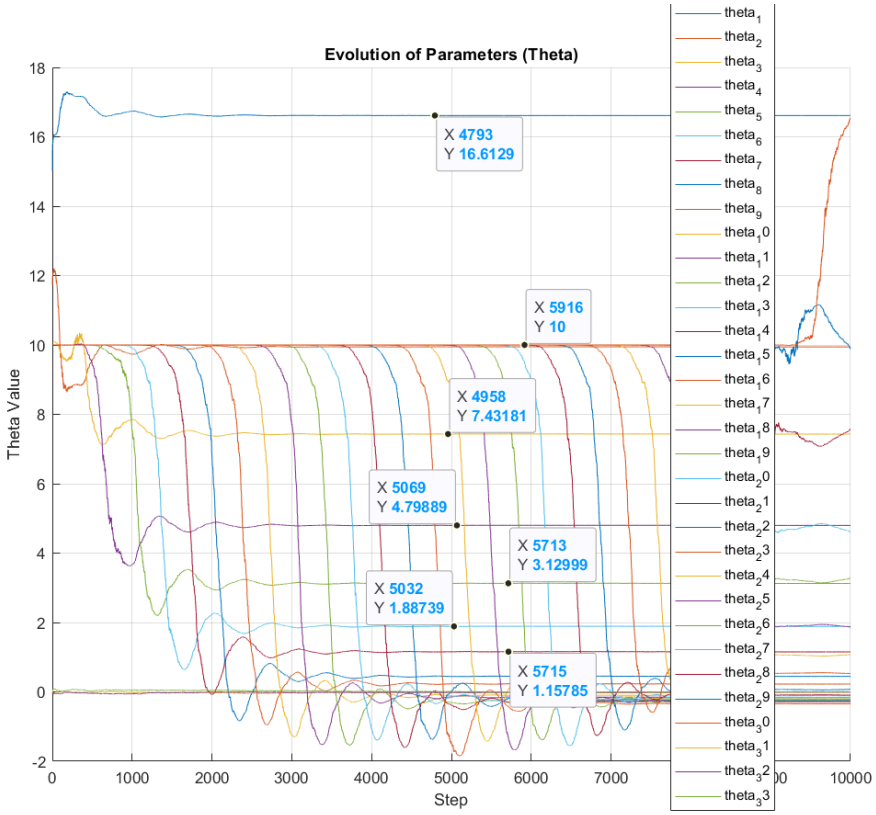
```
noise_level = input('Enter noise level (standard deviation): ');

rng(42);

% Fuzzy system parameters
centers = linspace(min_input, max_input, num_rules);
width = (max_input - min_input) / num_rules;
max_error_order = input('max_error_order: ');

% Initializing
theta =10*ones(num_rules, 1);
theta_e=zeros(max_error_order,1);
theta=[theta;theta_e];

% Create P_matrix with different learning rates for fuzzy and error parts
P_fuzzy = learning_rate_fuzzy * eye(num_rules);
P_error = learning_rate_error * eye(max_error_order);
P_matrix = blkdiag(P_fuzzy, P_error);

%storage
predictions = zeros(num_samples, 1);
true_outputs = zeros(num_samples, 1);
noisy_outputs = zeros(num_samples, 1);
inputs = linspace(min_input, max_input, num_samples)';
trace_p = zeros(num_samples, 1);
mean_errors = [];
theta_history = zeros(num_samples, num_rules + max_error_order);

% Generate colored noise-brown
white_noise = noise_level * randn(num_samples, 1);
bnoise = cumsum(white_noise);
bnoise = bnoise / max(abs(bnoise));
bnoise = bnoise(randperm(length(bnoise)));

% Error memory
error_memory = zeros(max_error_order, 1);

% making errors
for i = 1:max_error_order
    x = inputs(i);
    y = 10 * (x^4) * cosh(x);
    y_noisy = y + bnoise(i);

    true_outputs(i) = y;
    noisy_outputs(i) = y_noisy;

    % making regressor vector with fuzzy rules
    mu = exp(-((x - centers)/width).^2)';
    b = mu / sum(mu);

    % Regressor only contains membership vector
    b = [b; zeros(max_error_order, 1)];

    % Predict output
    y_pred = b' * theta;
    e = y_noisy - y_pred;

    % Update parameters using ELS
    K = (P_matrix * b) / (1 + b' * P_matrix * b);
```

```
    theta = theta + K * e;
    P_matrix = P_matrix - K * b' * P_matrix;

    % Save error into memory
    error_memory(i) = e;

    % Store results
    predictions(i) = y_pred;
    theta_history(i, :) = theta';
    trace_p(i) = trace(P_matrix);
    mean_errors = [mean_errors; mean(true_outputs(1:i) - predictions(1:i))];

    fprintf('Init Step %d: x=%.4f, y=%.4f, y_noisy=%.4f, y_pred=%.4f, mean_error=%.4f\n', ...
        i, x, y, y_noisy, y_pred, mean_errors(end));
end

for i = 1+max_error_order:num_samples
    x = inputs(i);
    y = 10 * (x^4) * cosh(x);
    y_noisy = y + bnoise(i);
    true_outputs(i) = y;
    noisy_outputs(i) = y_noisy;

    % Compute fuzzy rule activations
    mu = exp(-((x - centers)/width).^2)';
    b = mu / sum(mu);

    % Regressor
    b = [b; -1*error_memory];

    % fuzzy output
    y_pred = b' * theta;
    e = y_noisy - y_pred;

    % Update parameters
    K = (P_matrix * b) / (1 + b' * P_matrix * b);
    theta = theta + K * e;
    P_matrix = P_matrix - K * b' * P_matrix;

    % Update error memory (shift and add new error)
    error_memory = [e; error_memory(1:end-1)];

    % Store results
    predictions(i) = y_pred;
    theta_history(i, :) = theta';
    trace_p(i) = trace(P_matrix);
    mean_errors = [mean_errors; mean(true_outputs(1:i) - predictions(1:i))];

    fprintf('Train Step %d: x=%.4f, y=%.4f, y_noisy=%.4f, y_pred=%.4f, mean_error=%.4f\n', ...
        i, x, y, y_noisy, y_pred, mean_errors(end));
end

% Plot

% main System and fuzzy modeling system
figure;
plot(inputs, true_outputs, 'b', inputs, predictions, 'r--');
legend('True Output', 'Fuzzy Prediction');
title('main System and fuzzy modeling system');
```

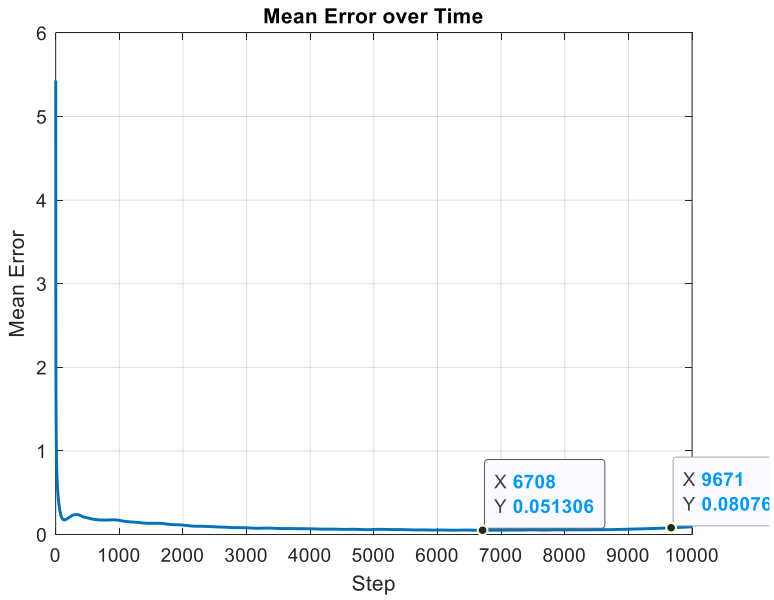
```
grid on;
% Error between true output and prediction
figure;
plot(true_outputs - predictions);
title('Prediction Error (True - Estimated)');
grid on;
%Trace p matrix
figure;
plot(trace_p);
title('Trace of P Matrix (Uncertainty)');
grid on;
% Plot power spectral density of noise
figure;
pwelch(bnoise, [], [], [], 1);
title('Power Spectral Density of Colored Noise');
grid on;
% Plot all theta weights
figure;
hold on;
for k = 1:(num_rules + max_error_order)
    plot(theta_history(:, k), 'DisplayName', sprintf('theta_%d', k));
end
hold off;
title('Evolution of Parameters (Theta)');
xlabel('Step');
ylabel('Theta Value');
legend('show');
grid on;

% mean error plot
figure;
plot(mean_errors, 'LineWidth', 1.5);
title('Mean Error over Time');
xlabel('Step');
ylabel('Mean Error');
grid on;
```

**خروجی ها:**

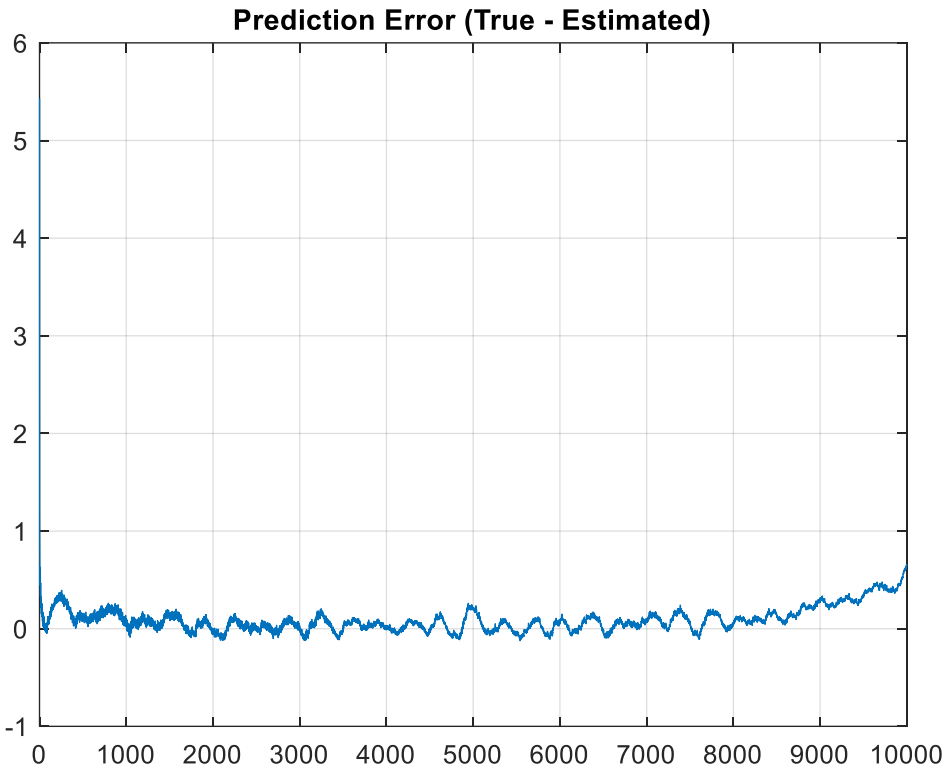
همگرایی پرامتر های بردار $\theta$:



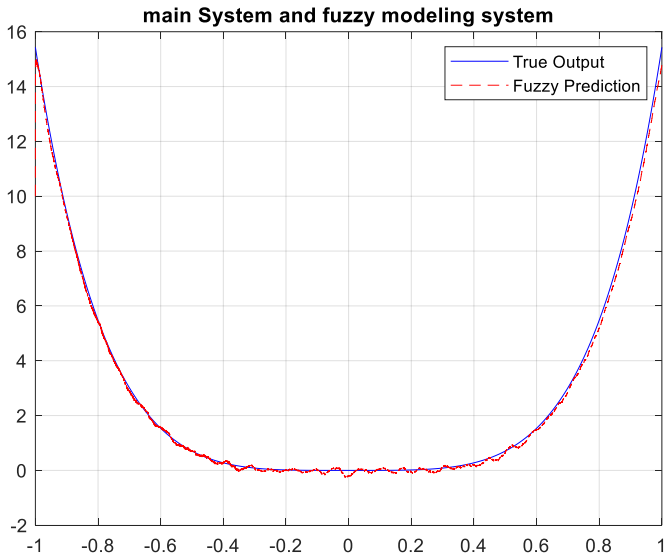

همگرایی میانگین خطا در طول فرآیند الگوریتم:



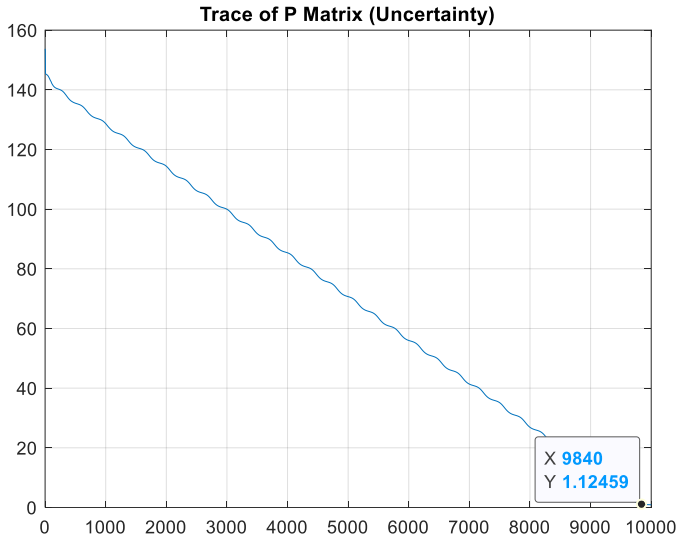
خطای بین سیستم اصلی و سیستم فازی:



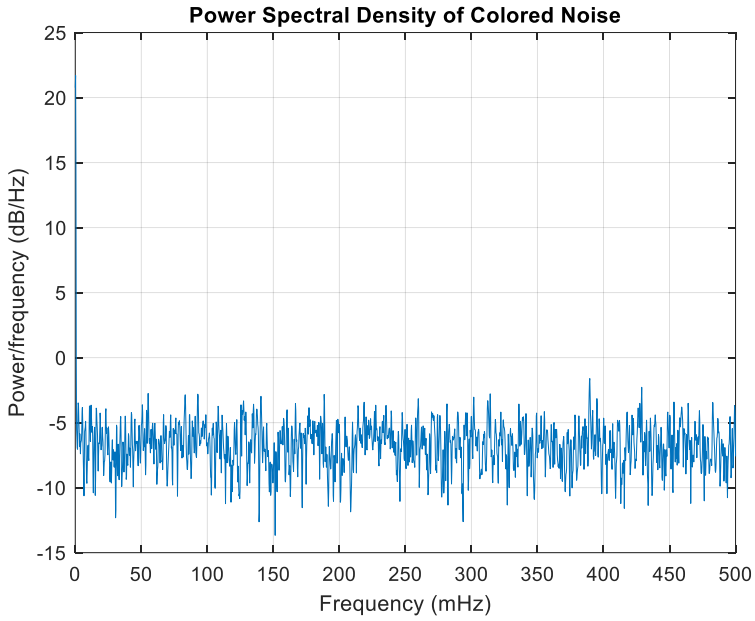

سیستم اصلی و سیستم فازی:




همگرایی Trace ماتریس P :




Power spectral نویز رنگی:




**تحلیل:**

همانطور که مشخص می باشد trace ماتریس p به همگرایی رسیده و تقریبا به صفر میل پیدا کرده است

همانطور که مشخص می باشد پرامتر ها هم متقابلا به همگرایی رسیده اند و همانطور که مشخص است

همچنان $\theta 1$ تا $\theta 30$ که مربوط به مرکز های ثقل می باشند نسبت به زیربخش 1 دارای biase می باشند

و این بایاس به دلیل وجود ترم نویز رنگی می باشد و همانطور که مشخص است $\theta 31$ تا $\theta 33$ که در تلاش برای تخمین دینامیک نویز رنگی و کم کردن اثر آن می باشند تقریبا به خوبی عمل کرده و از روی میانگین خطا که به $0.08$ میل کرده می توان به عملکرد بسیار خوب این الگوریتم پی برد زیرا در زیربخش قبل مشاهه شد که مینگین خطا به سمت $0.27$ همگرا شده است و همچنین از روی مقایسه ی سیستم اصلی و سیستم فازی مدل شده کاملا مشخص است که این الگوریتم عملکرد به شدت هتری نسبت به زیربخش قبل داشته است.