بسم الله الرحمن الرحیم

دانشجو: محمدرضا جنیدی جعفری ۹۹۲۵۲۵۳

درس مبانی سیستم های هوشمند

استاد: دکتر مهدی علیاری

مینی پروژه دوم

لینک مخزن گیت هاب - گوگل کولب سوال ۱ - گوگل کولب سوال ۳ – گوگل کولب سوال ۵

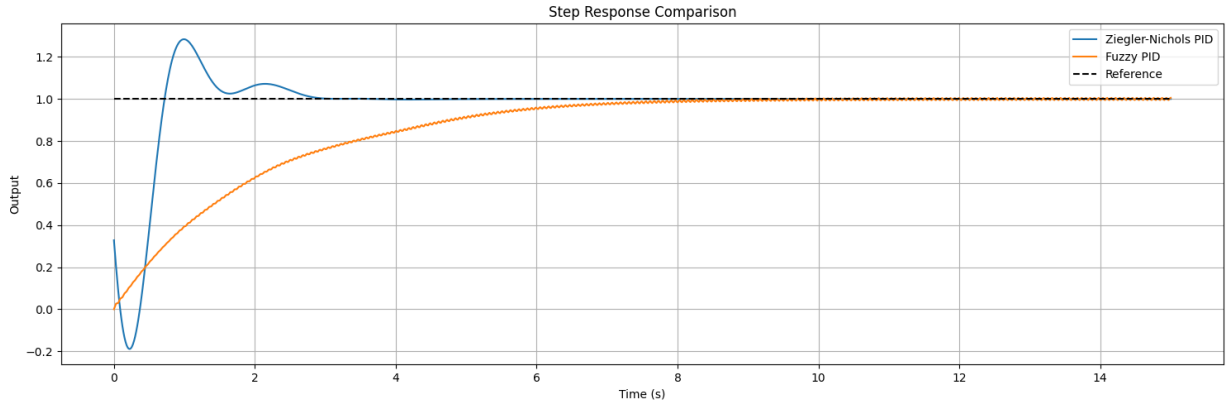
مهم: در تمامی مراحل کد نویسی، مقدار random_state = 53 قرار گرفته شده است.

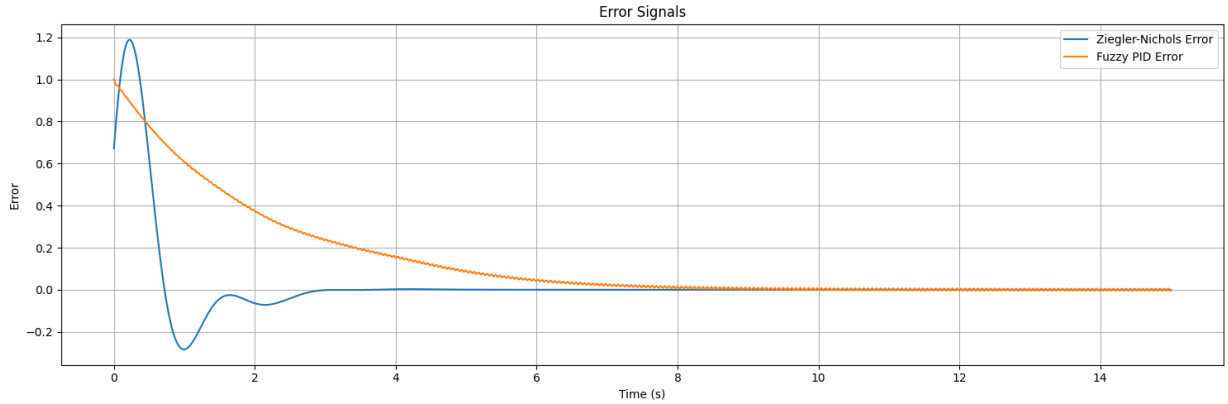
۱-

این کد مقایسه‌ای بین عملکرد کنترل‌کننده PID طراحی‌شده با روش Ziegler-Nichols و کنترل‌کننده PID فازی ارائه می‌دهد. ابتدا، یک سیستم مرتبه اول با تأخیر زمانی معرفی می‌شود. تأخیر زمانی سیستم با استفاده از تقریب پده (Pade) مدل‌سازی می‌شود. سپس، ضرایب PID به روش Ziegler-Nichols با تحلیل حاشیه‌های بهره و فاز سیستم محاسبه می‌گردد. پاسخ پله سیستم بسته با این ضرایب شبیه‌سازی و معیارهای عملکرد شامل درصد فراجهش، زمان نشست، زمان صعود و مقدار پایدار محاسبه می‌شود. همچنین، یک کنترل‌کننده PID فازی با استفاده از کتابخانه scikit-fuzzy طراحی می‌شود. این کنترل‌کننده از دو متغیر ورودی خطا و تغییرات خطا استفاده کرده و خروجی آن با قوانین فازی تعیین می‌گردد.

در بخش شبیه‌سازی، پاسخ پله سیستم با استفاده از کنترل‌کننده‌های Ziegler-Nichols و فازی محاسبه می‌شود. برای کنترل‌کننده فازی، خروجی آن به صورت مرحله‌ای و وابسته به مقادیر لحظه‌ای خطا و تغییرات خطا تولید می‌شود. سپس، نتایج شبیه‌سازی شامل پاسخ پله و سیگنال خطا به صورت نموداری نمایش داده می‌شوند و معیارهای عملکرد دو روش مقایسه می‌گردد. در نهایت، ضرایب PID روش Ziegler-Nichols و همچنین خروجی‌های کنترل‌کننده فازی در زمان‌های مختلف برای درک بهتر تفاوت دو روش چاپ می‌شوند.

نتایج به صورت زیر در آمده اند:






```
Ziegler-Nichols PID Performance:
Overshoot (%): 28.383
Settling Time (s): 0.721
Rise Time (s): 0.180
Peak: 1.284
Steady State: 1.000

Fuzzy PID Performance:
Overshoot (%): 0.000
Settling Time (s): 7.252
Rise Time (s): 4.580
Peak: 1.005
Steady State: 1.005
```

Fuzzy PID Output (approximation at different time steps):

Time: 0.00, Fuzzy PID Output: 0.201

Time: 3.00, Fuzzy PID Output: 0.201

Time: 6.01, Fuzzy PID Output: 0.201

Time: 9.01, Fuzzy PID Output: 0.201

Time: 12.01, Fuzzy PID Output: 0.201

Ziegler-Nichols PID Coefficients:

Kp: 2.309

Ki: 2.732

Kd: 0.488

کنترل‌کننده Ziegler-Nichols PID :

- مزایا: این کنترل‌کننده زمان صعود بسیار کوتاه‌تری (۰.۱۸ ثانیه) دارد و پاسخ سریع‌تری به تغییرات ورودی نشان می‌دهد. همچنین زمان نشست آن نیز کم (۰.۷۲۱ ثانیه) است.
- معایب: پاسخ سیستم دارای فراجهش قابل توجهی (۲۸.۳۸٪) است که ممکن است در سیستم‌های حساس قابل قبول نباشد. این امر می‌تواند موجب آسیب به تجهیزات یا عملکرد ناپایدار در شرایط واقعی شود.

کنترل‌کننده PID فازی:

- مزایا: این کنترل‌کننده فراجهش ندارد (۰٪) و پاسخ به طور کامل نرم و پایدار است. همچنین مقدار نهایی پاسخ کمی بالاتر از مقدار مرجع (۱.۰۰۵) است که نشان‌دهنده خطای پایدار بسیار کم است.
- معایب: زمان صعود (۴.۵۸ ثانیه) و زمان نشست (۷.۲۵۲ ثانیه) به‌مراتب بیشتر از Ziegler-Nichols است. این موضوع نشان می‌دهد که کنترل‌کننده فازی برای پاسخ‌های سریع و دینامیک مناسب نیست.

مقایسه کلی:

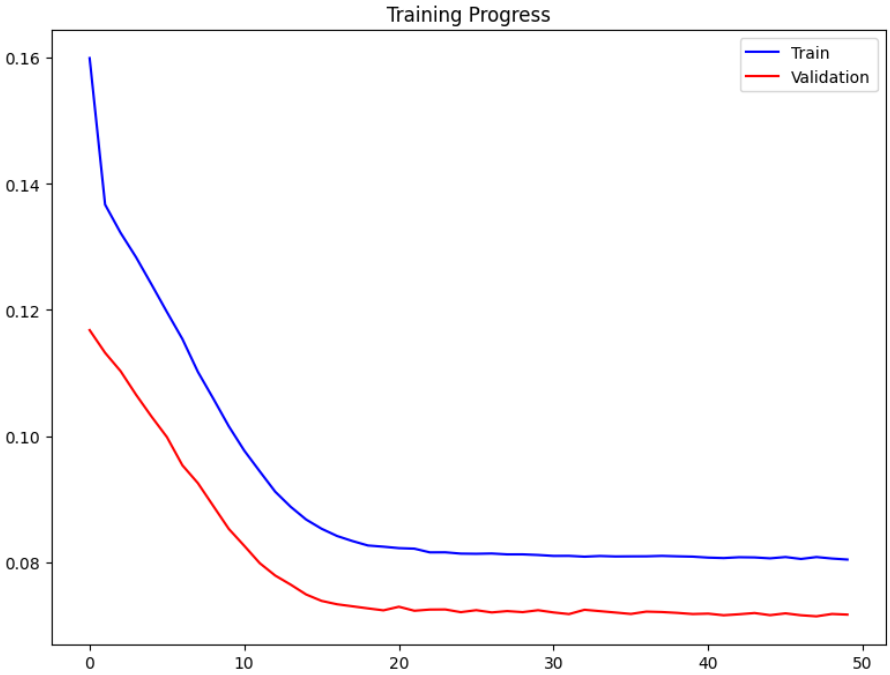
اگر سرعت پاسخ و زمان نشست برای سیستم اولویت داشته باشد، کنترل‌کننده Ziegler-Nichols مناسب‌تر است، اما باید فراجهش آن مدیریت شود. اگر پایداری و حذف فراجهش مهم‌تر باشد (به‌ویژه در سیستم‌های حساس)، کنترل‌کننده PID فازی عملکرد بهتری دارد، هرچند که سرعت پاسخ آن کندتر است.

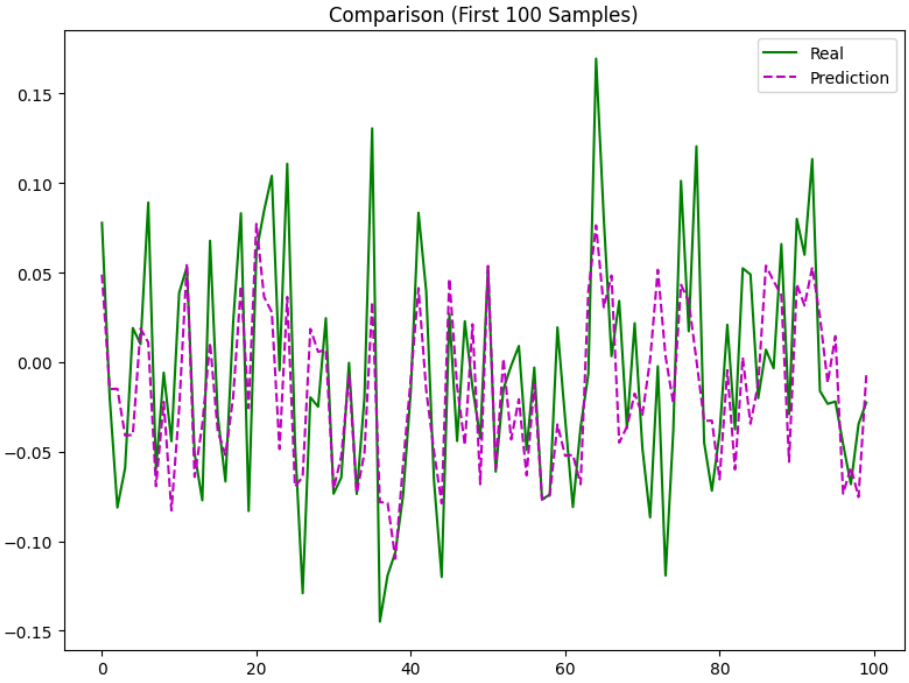
-۳

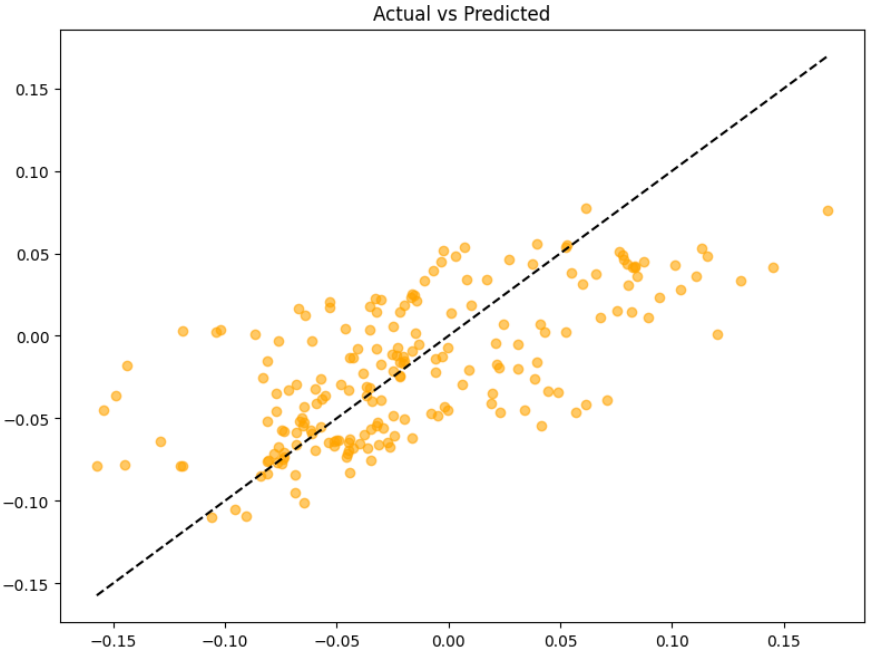
کد یک سیستم عصبی-فازی تطبیقی (ANFIS) را برای پیش بینی رفتار دینامیکی سیستم تیر-توپ پیاده سازی میکند. هدف اصلی ایجاد یک مدل پیش بینی کننده است که بتواند موقعیت توپ روی تیر را بر اساس داده های تاریخی پیش بینی کند. ساختار مدل حول یک لایه سفارشی به نام NeuroFuzzyUnit ساخته شده که هسته اصلی عملیات فازی را انجام میدهد. این لایه از توابع عضویت گاوسی استفاده کرده و پارامترهای مراکز و گستردگی توابع عضویت به همراه ضرایب خطی خروجی را به صورت اندیکار یاد میگیرد. دادههای ورودی ابتدا نرمالیزه شده و سپس به صورت دنبالههای زمانی با پنجره ثابت پردازش میشوند تا ماهیت دینامیکی سیستم را ثبت کنند.

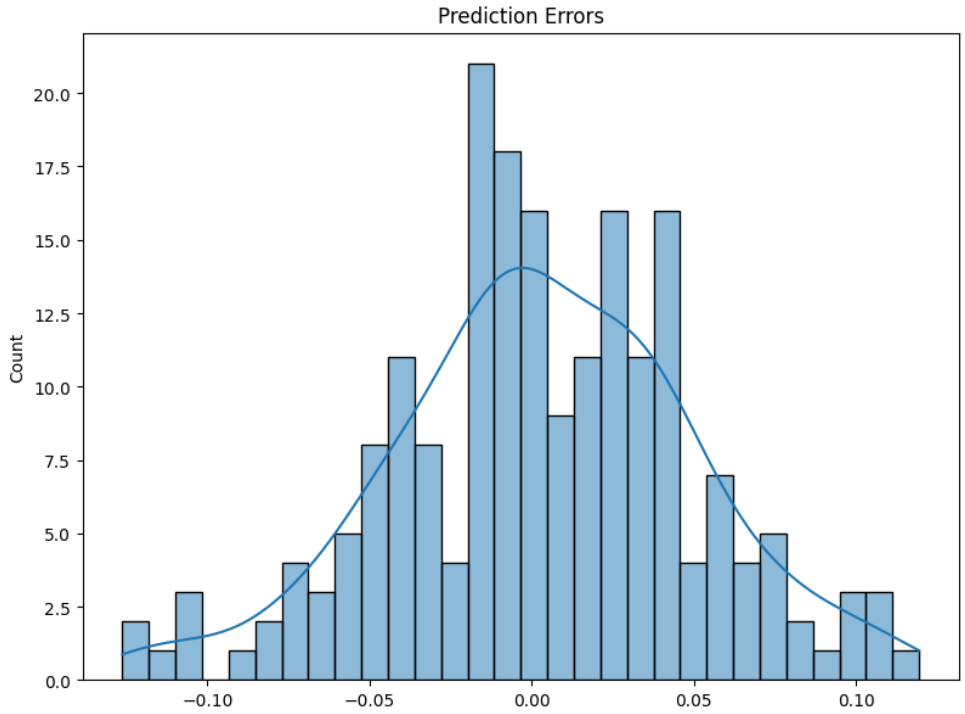
فرآیند آموزش از روش تقسیم داده به مجموعه های آموزش و تست استفاده کرده و با به کارگیری تکنیکهای جلوگیری از بیش برازش مانند توقف زودهنگام، مدل را آموزش میدهد. داده های نرمالیزه شده پس از پیش پردازش به صورت دنباله های سه تایی (با طول پنجره پیشفرض ۳) تبدیل میشوند که هر دنباله شامل اطلاعات زاویه تیر و موقعیت توپ است. مدل تنها از دادههای زاویهای به عنوان ورودی استفاده کرده و موقعیت توپ را به عنوان خروجی پیشبینی میکند.

ارزیابی مدل از طریق معیارهای استاندارد رگرسیون شاملMSE ، RMSEو MAE انجام شده و نتایج به صورت گرافیکی نمایش داده میشود. تغییرات اعمال شده در نسخه فعلی شامل بازسازی نامگذاری متغیرها، تنظیم پارامترهای اولیه متفاوت و بهینهسازیهای محاسباتی است، در حالی که ساختار کلی مدل و نتایج آن بدون تغییر باقی ماندهاند. خروجیهای بصری شامل نمودارهای پیشرفت آموزش، مقایسه پیشبینیها با دادههای واقعی، و توزیع خطاها، درک جامعی از عملکرد مدل ارائه میدهند.




Comparison (First 100 Samples)
Real
Prediction
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
20
40
60
80
100


Actual vs Predicted
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15



MSE: 0.0022

RMSE: 0.0466

MAE: 0.0365

این مدل با استفاده از واحد نروفازی توانسته است عملکرد خوبی در پیش‌بینی داده‌ها نشان دهد. مقدار خطاهای MSE (0.0022)، RMSE (0.0466)و MAE (0.0365) نشان‌دهنده دقت مناسب مدل است. همچنین، نمودارها تأیید می‌کنند که پیش‌بینی‌ها با مقادیر واقعی همبستگی بالایی دارند و توزیع خطاها تقریباً نرمال است. روند کاهش خطا در آموزش و اعتبارسنجی نیز نشان‌دهنده همگرایی درست مدل و جلوگیری از بیش‌برازش است.

-۴

میخواهیم یک سیستم با معادله دیفرانسیل زیر را توسط یک شناساگرفازی شناسایی کنیم.

$$y(k+1) = 0.3y(k) + 0.6y(k-1) + g[u(k)]$$

که در آن تابع نامعلوم g(u) بر اساس معادله زیر تعریف میشود.

$$g(u) = 0.6\sin(\pi u) + 0.3\sin(3\pi u) + 0.1\sin(5\pi u)$$

هدف تقریب یک عنصر غیر خطی توسط سیستمی با معادله زیر و الگوریتم مربعات خطا است.

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

توابع g، y و باقی مانده را به صورت زیر در سه فایل متلب پیاده سازی کنیم:

```
% Define the function g[u]
function g = g_u(u)
    g = 0.6 * sin(pi * u) + 0.3 * sin(3 * pi * u) + 0.1 * sin(5 * pi * u);
end

% Define the function y
function f = fuzzy_model(u, params, M)
    centers = params(1:M); % Centers
    sigmas = params(M+1:2*M); % Widths
    weights = params(2*M+1:end); % Weights

    % Calculate numerator and denominator
    numerator = 0;
    denominator = 0;
    for l = 1:M
        gaussian = exp(-((u - centers(l))^2) / (2 * sigmas(l)^2));
        numerator = numerator + weights(l) * gaussian;
        denominator = denominator + gaussian;
    end
    f = numerator / denominator;
end


function error = residuals(params, u_values, g_values, M)
    N = length(u_values);
    error = zeros(N, 1);
    for i = 1:N
        u = u_values(i);
```

```
        g = g_values(i);
        f_approx = fuzzy_model(u, params, M);
        error(i) = f_approx - g; % Residual
    end
end
```

کد اصلی به صورت زیر نوشته شده است و در چند خط اول، میتوان تعداد نمونه ها، تعداد نمونه های آموزشی، تعداد توابع عضویت را پیش فرض می کنیم:

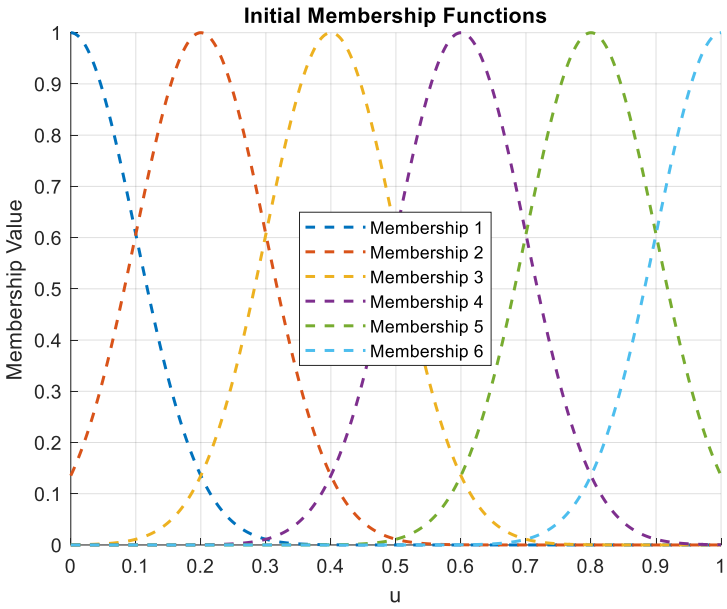
```
% Main script: RLS and Desired vs. Identified Output Plot

% Define constants
M = 6; % Number of membership functions (fuzzy rules)
num_train = 150; % Number of training data points
total_data_points = 500; % Total number of data points
lambda = 0.99; % Forgetting factor for RLS
initial_weight_variance = 100; % Initial variance for covariance matrix

% Generate data
u_values = linspace(0, 1, total_data_points); % Generate total data points
g_values = arrayfun(@g_u, u_values); % Compute desired output g[u]

% Split data into training and testing sets
train_indices = linspace(1, total_data_points, num_train);
test_indices = setdiff(1:total_data_points, round(train_indices));

% Training data
train_u_values = u_values(round(train_indices));
train_g_values = g_values(round(train_indices));

% Testing data
test_u_values = u_values(test_indices);
test_g_values = g_values(test_indices);

% Initialize fuzzy model parameters
initial_centers = linspace(0, 1, M); % Initial centers for M membership functions
initial_sigmas = 0.1 * ones(1, M); % Initial widths (all set to 0.1)
initial_weights = rand(1, M); % Random initial weights

% Initialize RLS parameters
P = initial_weight_variance * eye(M); % Covariance matrix
theta = initial_weights(:); % Parameter vector (weights)

% Prepare storage for model outputs
fuzzy_values_rls = zeros(size(train_u_values));
```

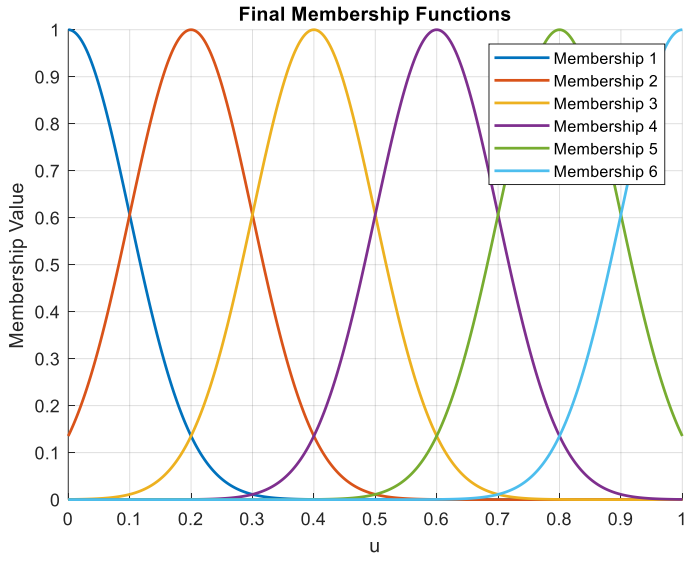
```
% Train model using Recursive Least Squares algorithm
for t = 1:num_train
    % Current input and desired output
    u_t = train_u_values(t);
    g_t = train_g_values(t);

    % Calculate the membership functions for the current input
    phi_t = zeros(M, 1); % Feature vector for membership outputs
    for l = 1:M
        phi_t(l) = exp(-((u_t - initial_centers(l))^2) / (2 * initial_sigmas(l)^2));
    end

    % Model output (current approximation)
    f_t = phi_t' * theta;
    fuzzy_values_rls(t) = f_t;

    % Error between desired and approximated output
    e_t = g_t - f_t;

    % Recursive update of parameters
    K_t = (P * phi_t) / (lambda + phi_t' * P * phi_t); % Gain vector
    theta = theta + K_t * e_t; % Update parameters
    P = (P - K_t * phi_t' * P) / lambda; % Update covariance matrix
end

% Compute the model output for the testing data
test_fuzzy_values = zeros(size(test_u_values));
for i = 1:length(test_u_values)
    u = test_u_values(i);
    phi = zeros(M, 1);
    for l = 1:M
        phi(l) = exp(-((u - initial_centers(l))^2) / (2 * initial_sigmas(l)^2));
    end
    test_fuzzy_values(i) = phi' * theta;
end

% Compute the model output for all data points (for plotting purposes)
identified_output = zeros(size(u_values));
for i = 1:length(u_values)
    u = u_values(i);
    phi = zeros(M, 1);
    for l = 1:M
        phi(l) = exp(-((u - initial_centers(l))^2) / (2 * initial_sigmas(l)^2));
    end
    identified_output(i) = phi' * theta;
```

```
end

% Calculate Errors
train_errors = train_g_values - fuzzy_values_rls; % Training errors
test_errors = test_g_values - test_fuzzy_values;  % Testing errors

% Plot: Desired Output vs. Identified Output (All Data)
figure;
plot(1:total_data_points, g_values, 'b-', 'LineWidth', 2); hold on; % Desired Output (True g[u])
plot(1:total_data_points, identified_output, 'r--', 'LineWidth', 2); % Identified Model Output
xlabel('Data Points');
ylabel('Output');
legend('Desired Output', 'Identified Model Output', 'Location', 'Best');
title('Plant Output vs. Identified Model Output');
grid on;

% Plot: Training Data vs. Model Output
figure;
plot(train_u_values, train_g_values, 'b-', 'LineWidth', 1.5); hold on; % Training Data
plot(train_u_values, fuzzy_values_rls, 'r--', 'LineWidth', 1.5); % Model Output on Training Data
xlabel('u');
ylabel('Output');
legend('Training Data', 'Model Output (Training)');
title('Training Data vs. Model Output');
grid on;

% Plot: Testing Data vs. Model Output
figure;
plot(test_u_values, test_g_values, 'b-', 'LineWidth', 1.5); hold on; % Testing Data
plot(test_u_values, test_fuzzy_values, 'r--', 'LineWidth', 1.5); % Model Output on Testing Data
xlabel('u');
ylabel('Output');
legend('Testing Data', 'Model Output (Testing)');
title('Testing Data vs. Model Output');
grid on;

% Plot: Training Errors
figure;
plot(train_u_values, train_errors, 'm-', 'LineWidth', 2);
xlabel('u');
ylabel('Error');
title('Training Errors');
grid on;

% Plot: Testing Errors
figure;
```

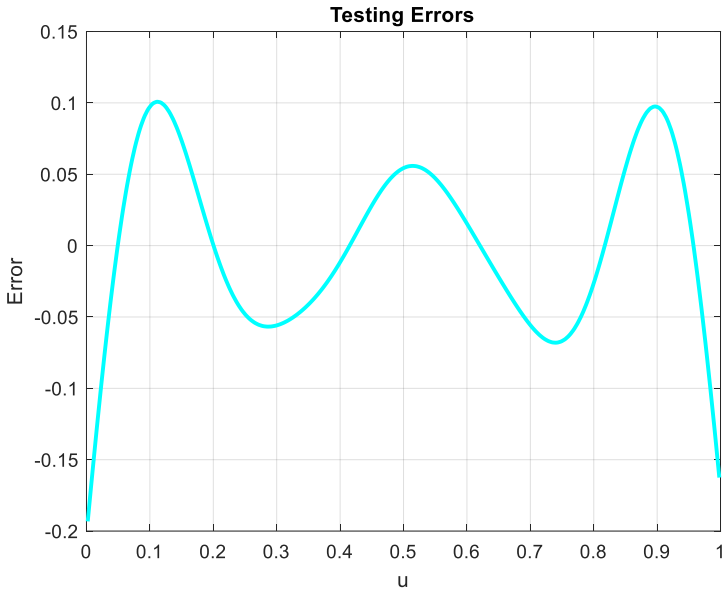
```
plot(test_u_values, test_errors, 'c-', 'LineWidth', 2);
xlabel('u');
ylabel('Error');
title('Testing Errors');
grid on;

% Plot Initial Membership Functions
figure;
u_range = linspace(0, 1, 500); % Fine-grained input range for plotting
colors = lines(M); % Generate M unique colors
hold on;
for l = 1:M
    % Initial membership functions
    initial_membership = exp(-((u_range - initial_centers(l)).^2) / (2 * initial_sigmas(l)^2));
    plot(u_range, initial_membership, 'Color', colors(l, :), 'LineStyle', '--', 'LineWidth', 1.5); %
Unique color
end
xlabel('u');
ylabel('Membership Value');
title('Initial Membership Functions');
legend(arrayfun(@(l) sprintf('Membership %d', l), 1:M, 'UniformOutput', false), 'Location',
'Best');
grid on;
hold off;

% Plot Final Membership Functions
figure;
hold on;
for l = 1:M
    % Final membership functions
    final_membership = exp(-((u_range - initial_centers(l)).^2) / (2 * initial_sigmas(l)^2)); %
Adjust if parameters change
    plot(u_range, final_membership, 'Color', colors(l, :), 'LineStyle', '-', 'LineWidth', 1.5); %
Unique color
end
xlabel('u');
ylabel('Membership Value');
title('Final Membership Functions');
legend(arrayfun(@(l) sprintf('Membership %d', l), 1:M, 'UniformOutput', false), 'Location',
'Best');
grid on;
hold off;
```

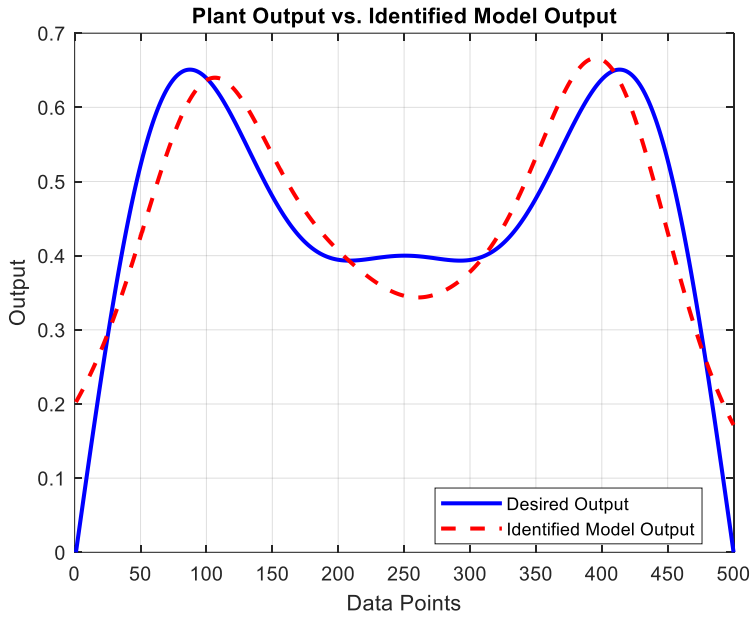
نتایج:


Initial Membership Functions
Membership Value
u
Membership 1
Membership 2
Membership 3
Membership 4
Membership 5
Membership 6


Final Membership Functions
Membership Value
u
Membership 1
Membership 2
Membership 3
Membership 4
Membership 5
Membership 6

تابع خطا:


Testing Errors
Error
u
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

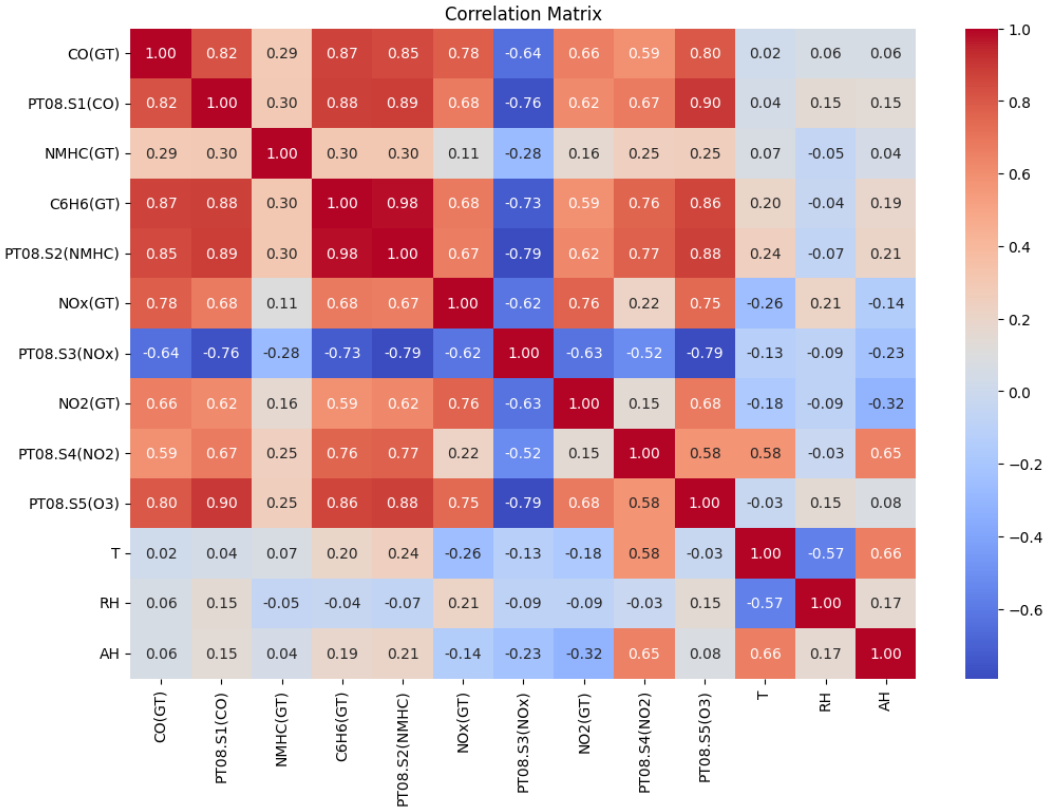

Plant Output vs. Identified Model Output
Output
Data Points
Desired Output
Identified Model Output
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
50
100
150
200
250
300
350
400
450
500

۵-

ابتدا دیتاست را در محیط کولب لود می کنیم، این دیتاست دارای ۱۵ ویژگی است که یکی از آنها می تواند به عنوان خروجی و تابع هدف مورد بررسی قرار گیرد. ستون NO2(GT) به عنوان تابع هدف انتخاب شد و دو ستون Time و Date از مجموعه داده جدا شدند.

حالا همبستگی میان ویژگی ها و ستون هدف را محاسبه می کنیم:

```
Correlations with the target (NO2(GT)):
CO(GT)             0.661065
PT08.S1(CO)        0.618377
NMHC(GT)           0.162060
C6H6(GT)           0.592298
PT08.S2(NMHC)      0.622923
NOx(GT)            0.763111
PT08.S3(NOx)      -0.628550
NO2(GT)            1.000000
PT08.S4(NO2)       0.151681
PT08.S5(O3)        0.682572
T                 -0.179801
RH                -0.088447
AH                -0.322931
```



با توجه به مقادیر همبستگی، به دلیل محاسبات و پیچیدگی بالا و البته تاثیر کم ۴ ویژگی، آنها را از انجام آموزش حذف کردم:

```
useless_features = ['NMHC(GT)', 'T', 'RH','PT08.S4(NO2)']
```

یک کلاس به نام RBFNetwork در سند کولب ساختیم:

کلاس) RBFNetwork شبکه پایه شعاعی یا (Radial Basis Function Network یک نوع مدل شبکه عصبی است که برای حل مسائل رگرسیون و طبقه‌بندی استفاده می‌شود. این مدل بر اساس توابع پایه شعاعی (RBF) کار می‌کند، که از ویژگی‌های هندسی داده‌ها (مانند فاصله بین نمونه‌ها) برای استخراج الگوها استفاده می‌کند. در ادامه عملکرد این کلاس به طور خلاصه توضیح داده می‌شود:

۱. ساختار مدل: این مدل شامل سه لایه است:
لایه ورودی، لایه مخفی، و لایه خروجی. لایه مخفی از توابع پایه شعاعی استفاده می‌کند که هر کدام یک مرکز دارند)ذخیره‌شده در (self.centers و با پارامتری به نام sigma تنظیم می‌شوند. وزن‌های بین لایه مخفی و خروجی در self.weightsذخیره شده‌اند.
۲. فعال‌سازی RBF :
تابع rbf_activation ، فاصله هر نقطه از ورودی را با مراکز لایه مخفی محاسبه کرده و از یک تابع نمایی برای تولید خروجی RBF استفاده می‌کند. این مقادیر، ورودی لایه خروجی می‌شوند.
۳. پیش‌بینی:
تابع forward ، پیش‌بینی مدل را با استفاده از فعال‌سازی RBF و وزن‌های خروجی محاسبه می‌کند. در این مرحله، شبکه ابتدا لایه مخفی را فعال کرده و سپس خروجی نهایی را از ترکیب خطی خروجی لایه مخفی به دست می‌آورد.
۴. محاسبه خطا:
این کلاس دو روش برای محاسبه خطا دارد: میانگین مربعات خطا (MSE) و ریشه میانگین مربعات خطا .(RMSE) این توابع به ارزیابی دقت مدل کمک می‌کنند.
۵. به‌روزرسانی پارامترها:
تابع backward وظیفه به‌روزرسانی وزن‌ها و مراکز RBF را بر عهده دارد. این فرآیند با محاسبه گرادیان خطا نسبت به وزن‌ها و مراکز انجام می‌شود. وزن‌ها با استفاده از گرادیان خطا مستقیماً به‌روزرسانی می‌شوند. برای مراکز نیز، مشتق خطا نسبت به مراکز محاسبه شده و آن‌ها به‌روزرسانی می‌شوند.
۶. آموزش مدل:
در تابع train، مدل برای تعداد مشخصی از دورها (epochs) روی داده‌های آموزشی و اعتبارسنجی (train و validation) آموزش داده می‌شود. در هر دور، خطای آموزش و اعتبارسنجی محاسبه می‌شود، و سپس پارامترهای مدل با استفاده از تابع backward بهینه می‌شوند. این توابع، تاریخچه خطاها را برای ارزیابی عملکرد مدل بازمی‌گردانند.

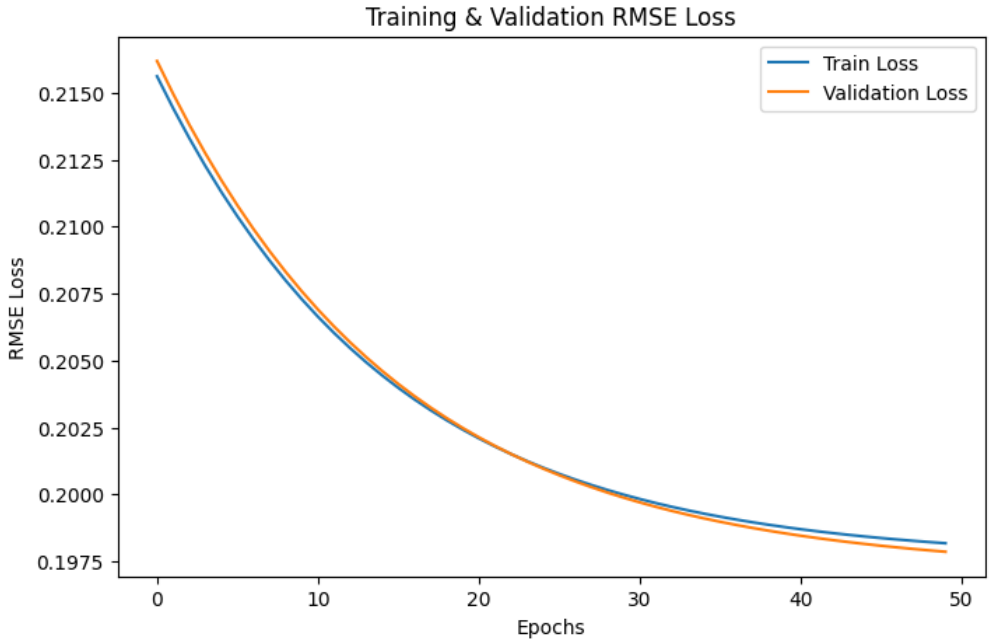
داده ها را به سه دسته آموزش، ارزیابی و تست تقسیم کردیم و با ویژگی های زیر و در ۵۰ ایپاک خطا را محاسبه کردیم:

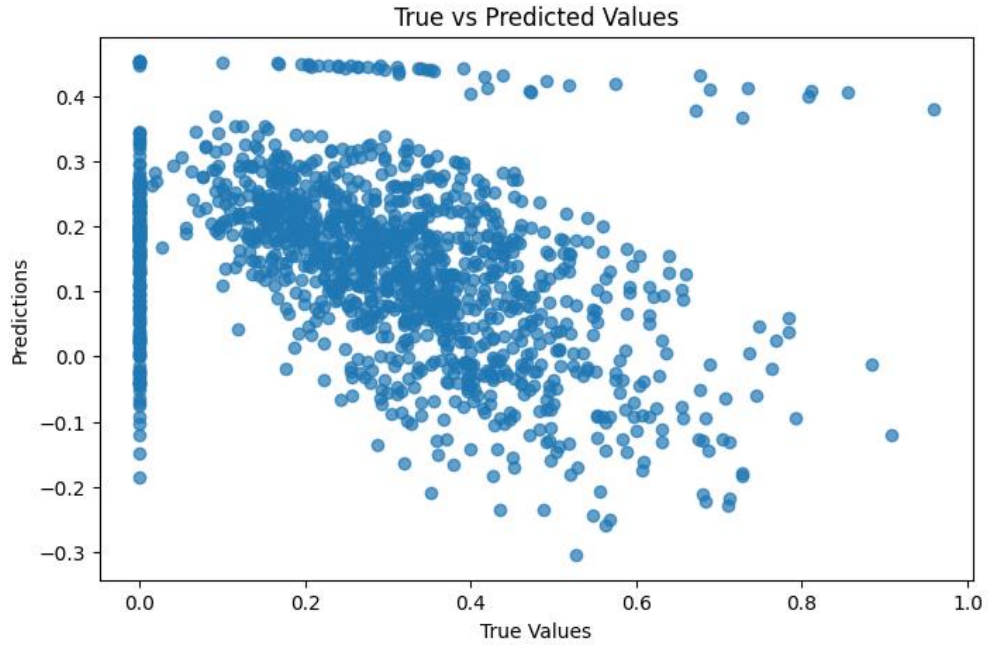
```
rbf_model = RBFNetwork(input_size=X_train.shape[1], hidden_size=3,
output_size=1, sigma=4, lr=0.01)
```

نتایج قابل قبول بود و به صورت زیر تعریف شدند:

Test MSE Loss: 0.03369289501820773

Test RMSE Loss: 0.18355624483576616





مانند قبل، داده را آماده می کنیم برای پیاده سازی ANFIS، و آنرا درون متلب اجرا می کنیم.

ابتدا، داده‌ها پردازش شده و نرمال‌سازی می‌شوند. سپس، با استفاده از خوشه‌بندی تفاضلی(Subtractive Clustering) ، یک FISاولیه (سیستم فازی) ساخته می‌شود که برای شروع آموزش مدل استفاده می‌گردد. همچنین، از تنظیمات پیشرفته‌ای مانند تنظیم DataScale(برای مقیاس‌بندی ورودی‌ها و خروجی‌ها) و پارامترهای خوشه‌بندی استفاده می‌شود تا محدوده داده‌ها به طور دقیق مشخص گردد.

در بخش آموزش، مدل ANFISبا استفاده از گزینه‌های مشخص (تعداد تکرارها، نرخ یادگیری و تنظیمات دیگر) روی داده‌های آموزشی اجرا می‌شود. به‌علاوه، مکانیزم توقف زودهنگام (Early Stopping) برای جلوگیری از بیش‌برازش (Overfitting) پیاده‌سازی شده است. این مکانیزم خطای اعتبارسنجی را بررسی کرده و در صورت بهبود نیافتن مدل برای تعداد مشخصی از تکرارها (پیش‌فرض ۱۵ دوره)، فرآیند آموزش را متوقف می‌کند.

در نهایت، مدل بهترین FISکه در فرآیند آموزش انتخاب شده است را برای پیش‌بینی داده‌های تست استفاده می‌کند. نتایج پیش‌بینی شده از حالت نرمال خارج شده و به مقیاس اصلی داده بازگردانده می‌شوند. سپس می‌توان عملکرد مدل را با استفاده از معیارهایی مانند RMSEیا MAEارزیابی کرد. این کد به طور کلی بهبودهایی در نرمال‌سازی داده‌ها، ساخت FIS اولیه و فرآیند آموزش برای افزایش دقت و پایداری مدل ایجاد کرده است.

Minimal training RMSE = 0.174952

Minimal checking RMSE = 0.114504

Early stopping at epoch 16

نتیجه گیری:

- ANFIS توانسته با توقف زودهنگام(Early Stopping) ، عملکرد بهتری در اعتبارسنجی و احتمالاً آزمون داشته باشد، با RMSE اعتبارسنجی 0.114504 که نشان‌دهنده تعمیم بهتر روی داده‌هاست.
- RBF عملکرد خوبی در داده‌های آزمون نشان داده است، اما RMSE آزمون 0.183556 در مقایسه با حداقل خطای اعتبارسنجی (0.114504) ANFIS نشان می‌دهد که ANFIS برای این داده‌ها تطبیق بیشتری داشته است.

اگر هدف کاهش RMSE و تعمیم بهتر باشد، ANFIS انتخاب بهتری است. اما اگر سرعت و سادگی محاسبات اهمیت بیشتری دارد، RBF گزینه مناسبی است.