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

بسم الله الرحمن الرحیم

دانشجو: محمدرضا جنیدی جعفری 9925253

درس مبانی سیستم های هوشمند

استاد: دکتر مهدی علیاری

مینی پروژه دوم

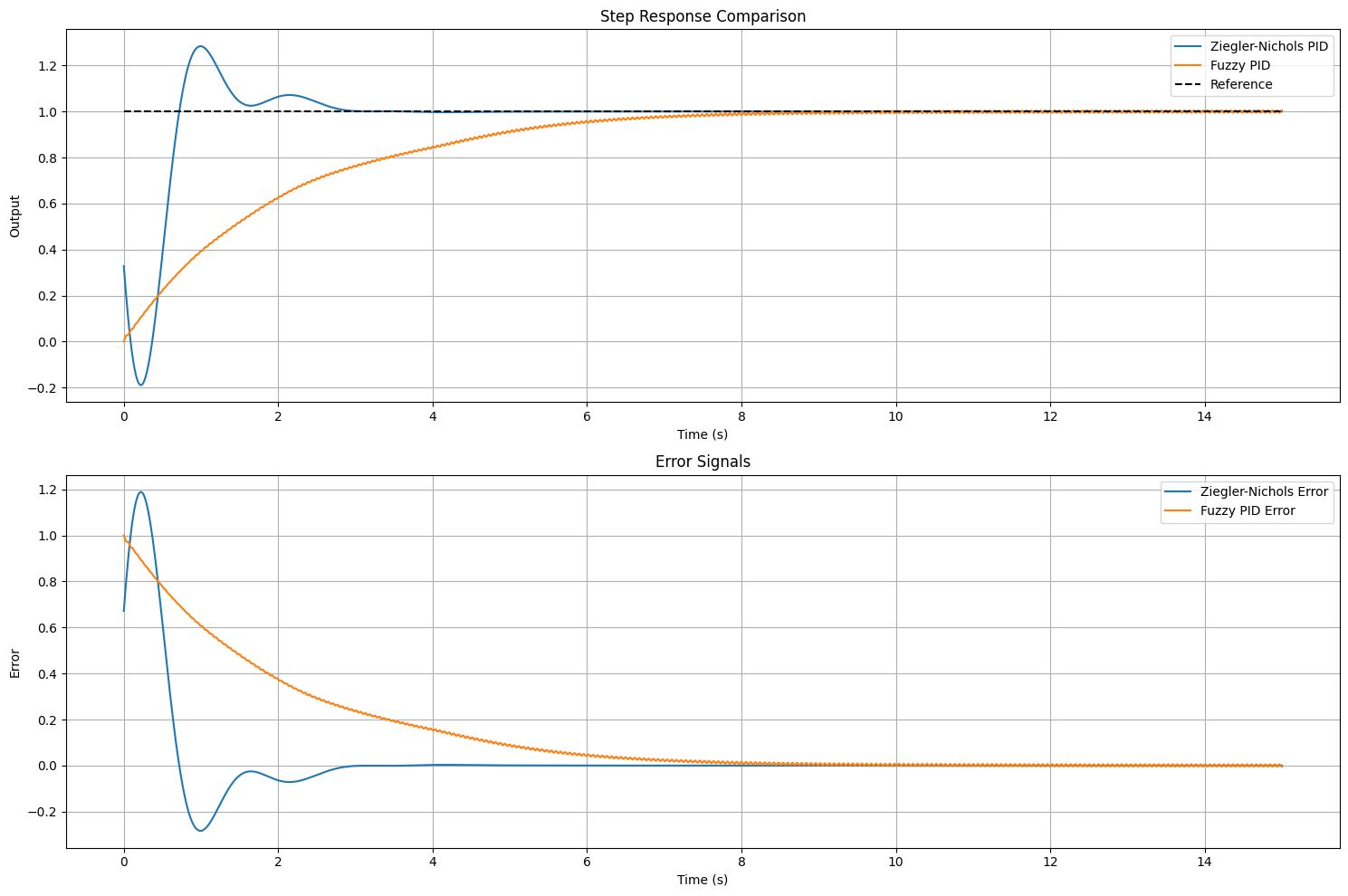
[لینک مخزن گیت هاب](https://github.com/mrjoneidi/FIS) - [گوگل کولب سوال 1](https://colab.research.google.com/drive/1WSuDmAqWgrxbcwBlRBHfhOEoYa4KBojj?usp=sharing) - [گوگل کولب سوال 3](https://colab.research.google.com/drive/1uT8j3ga3DxQX1g_oW31DaFLiqGRXMz6j?usp=sharing) – [گوگل کولب سوال 5](https://colab.research.google.com/drive/1KFNzz3Y6XHx8EcFsHIO4sl517j8lQB4C?usp=sharing)

1-

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 (0.18 ثانیه) دارد و پاسخ سریع‌تری به تغییرات ورودی نشان می‌دهد. همچنین زمان نشست آن نیز کم (0.721 ثانیه) است.
* معایب: پاسخ سیستم دارای فراجهش قابل توجهی (28.38%) 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 (0%) و پاسخ به طور کامل نرم و پایدار است. همچنین مقدار نهایی پاسخ کمی بالاتر از مقدار مرجع (1.005) است که نشان‌دهنده خطای پایدار بسیار کم است.
* معایب: زمان صعود (4.58 ثانیه) و زمان نشست (7.252 ثانیه) به‌مراتب بیشتر از Ziegler-Nichols است. این موضوع نشان می‌دهد که کنترل‌کننده فازی برای پاسخ‌های سریع و دینامیک مناسب نیست.

مقایسه کلی:

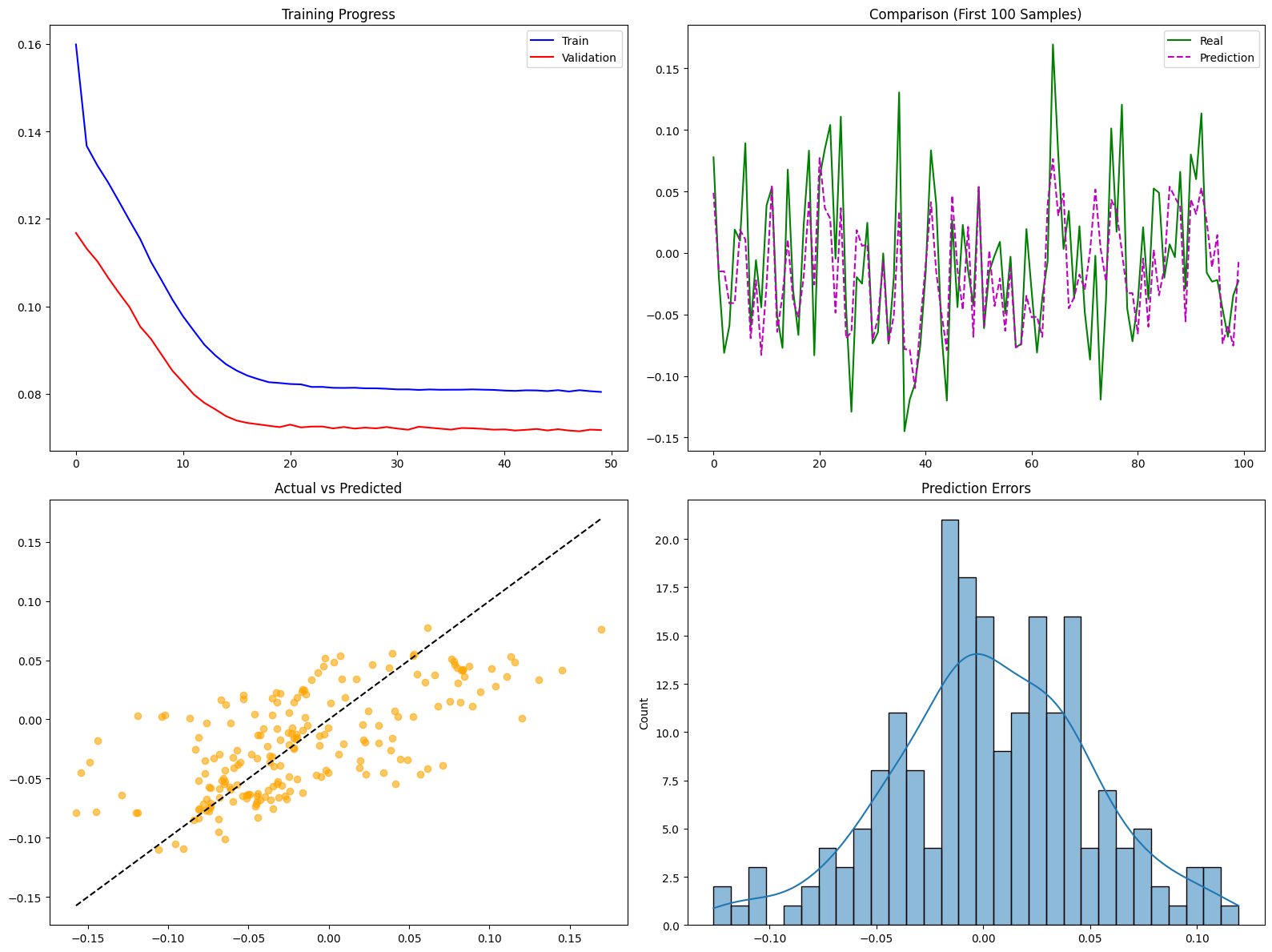
اگر سرعت پاسخ و زمان نشست برای سیستم اولویت داشته باشد، کنترل‌کننده Ziegler-Nichols مناسب‌تر است، اما باید فراجهش آن مدیریت شود. اگر پایداری و حذف فراجهش مهم‌تر باشد (به‌ویژه در سیستم‌های حساس)، کنترل‌کننده PID فازی عملکرد بهتری دارد، هرچند که سرعت پاسخ آن کندتر است.

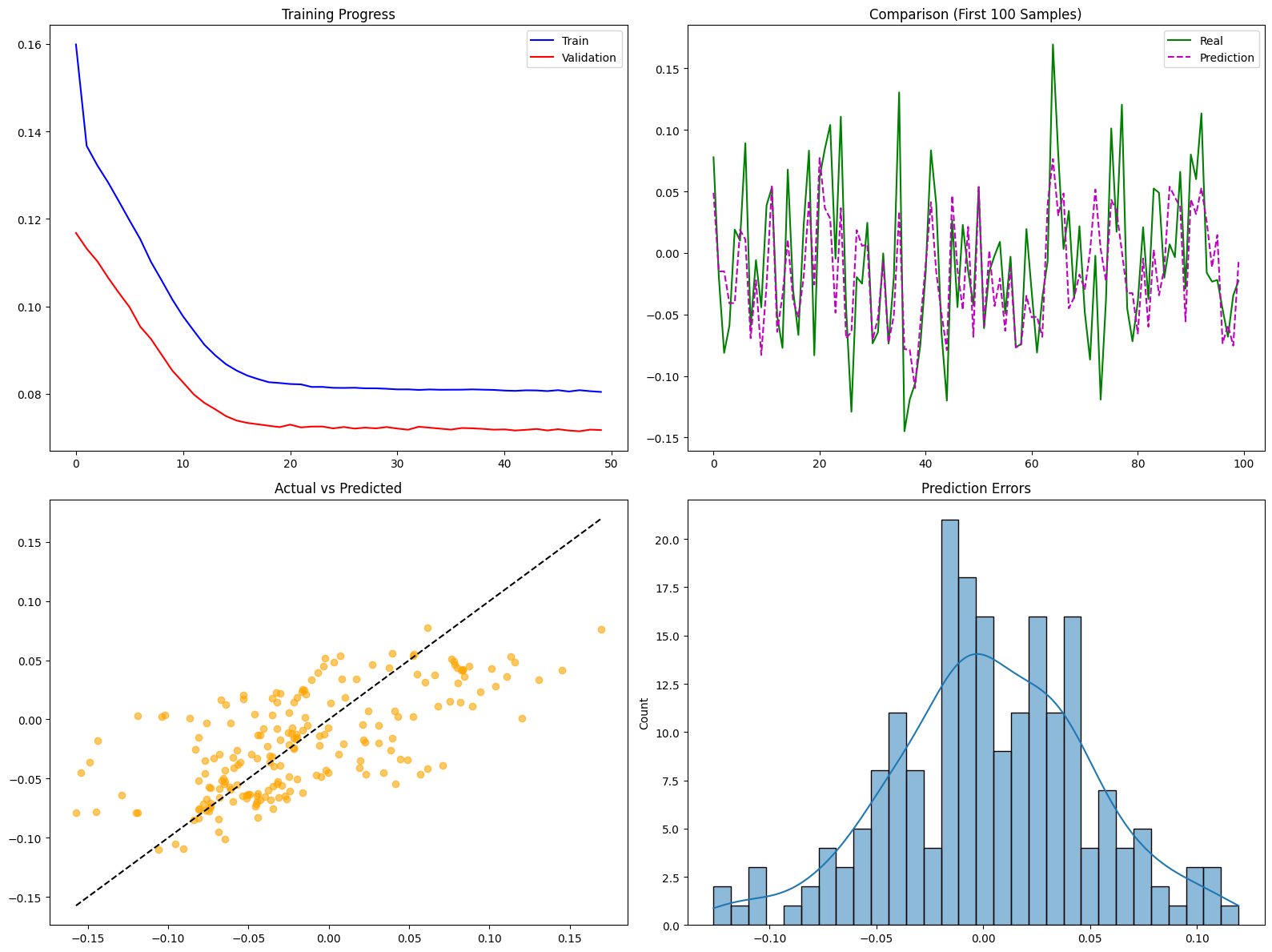
3-

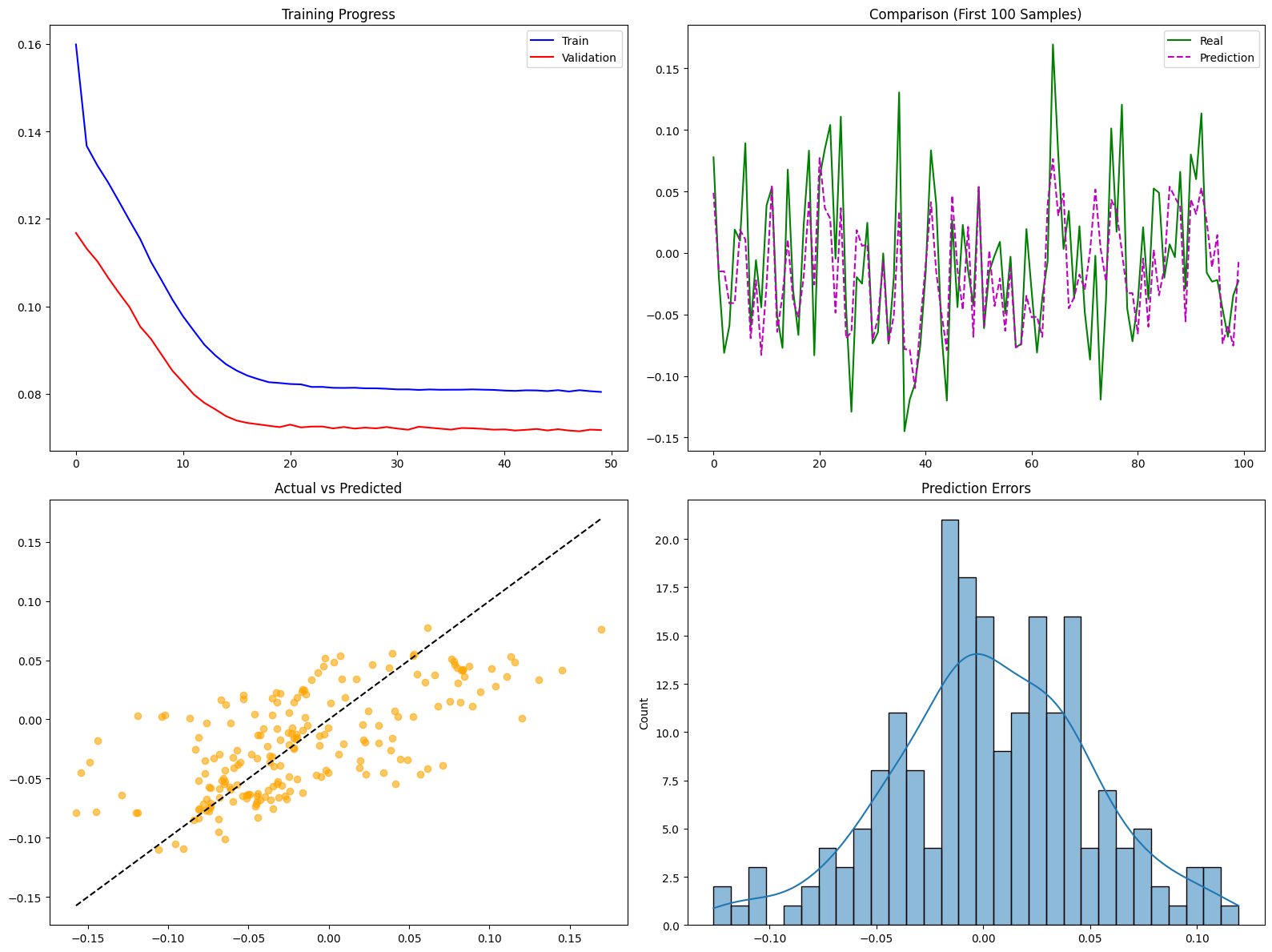
کد یک سیستم عصبی-فازی تطبیقی (ANFIS) را برای پیش بینی رفتار دینامیکی سیستم تیر-توپ پیاده سازی میکند. هدف اصلی ایجاد یک مدل پیش بینی کننده است که بتواند موقعیت توپ روی تیر را بر اساس داده های تاریخی پیش بینی کند. ساختار مدل حول یک لایه سفارشی به نام NeuroFuzzyUnit ساخته شده که هسته اصلی عملیات فازی را 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.

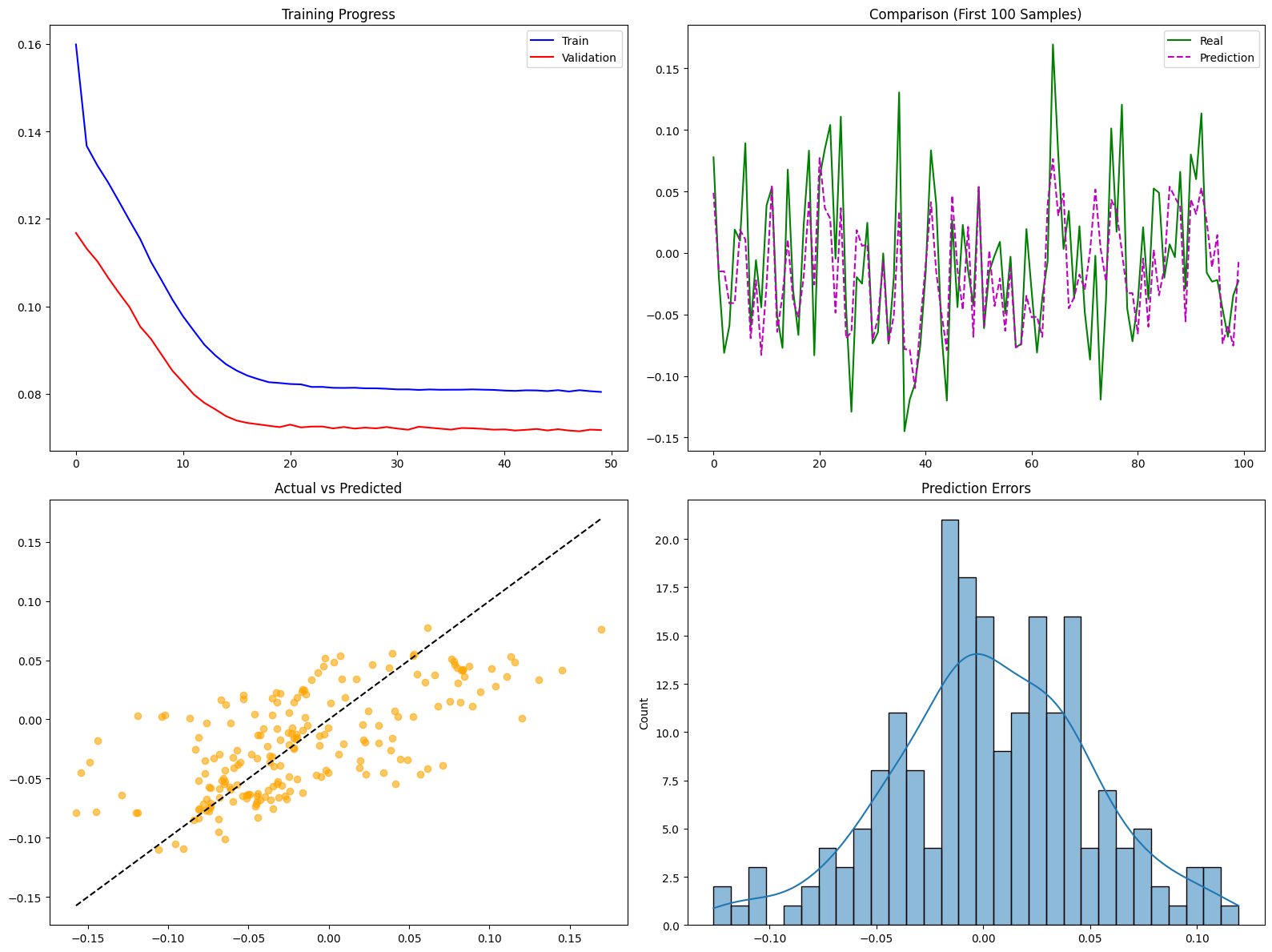
فرآیند آموزش از روش تقسیم داده به مجموعه های آموزش و تست استفاده کرده و با به کارگیری تکنیکهای جلوگیری از بیش برازش مانند توقف زودهنگام، مدل را آموزش میدهد. داده های نرمالیزه شده پس از پیش پردازش به صورت دنباله های سه تایی (با طول پنجره پیشفرض 3) تبدیل میشوند که هر دنباله شامل اطلاعات زاویه تیر و موقعیت 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 شامل MSE، RMSE و 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.









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.

4-

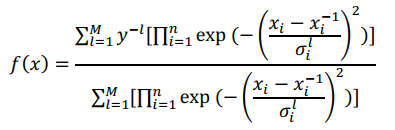
میخواهیم یک سیستم با معادله دیفرانسیل زیر را توسط یک شناساگرفازی شناسایی کنیم.



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



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



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:





تابع خطا:





5-

ابتدا دیتاست را در محیط کولب لود می کنیم، این دیتاست دارای 15 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

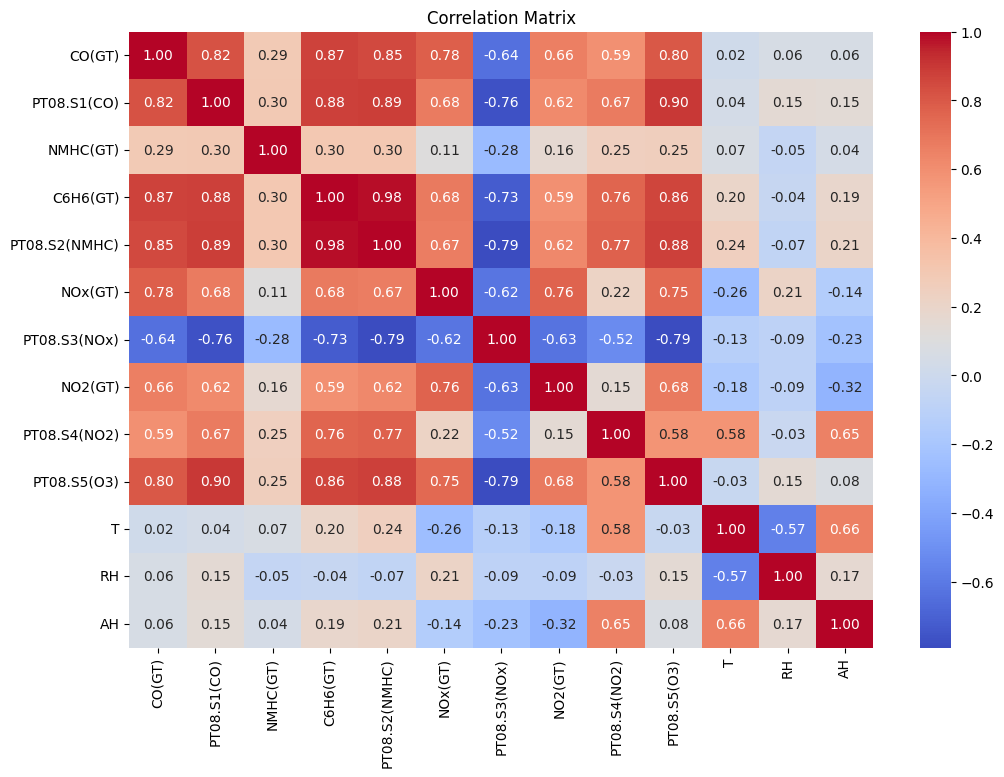
PT08.S4(NO2) 0.151681

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 4 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:

1. 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.weights ذخیره شده‌اند.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

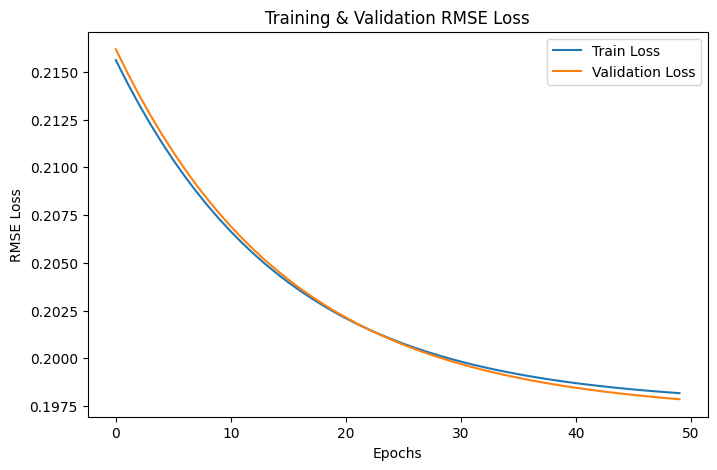
داده ها را به سه دسته آموزش، ارزیابی و تست تقسیم کردیم و با ویژگی های زیر و در 50 ایپاک خطا را محاسبه کردیم:

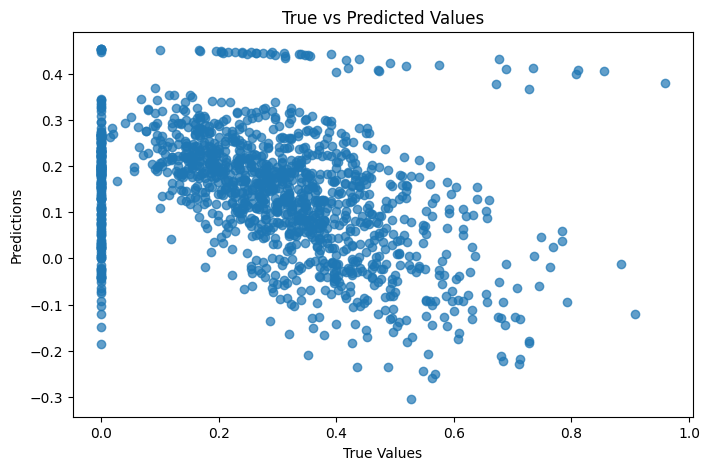
rbf\_model = RBFNetwork(input\_size=X\_train.shape[1], hidden\_size=3, output\_size=1, sigma=4, lr=0.01)

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 FIS اولیه (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 ANFIS با 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 15 دوره)، فرآیند آموزش را متوقف می‌کند.

در نهایت، مدل بهترین FIS که 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 RMSE یا MAE ارزیابی 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.
* RBFعملکرد خوبی در داده‌های آزمون نشان داده است، اما RMSE آزمون 0.183556 در مقایسه با حداقل خطای اعتبارسنجی ANFIS (0.114504) 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.