به نام خدا

محمدمهدی کرمی – ۴۰۰۰۸۳۷۳

لینک گیتهاب

سوال اول

برای طراحی یک کنترل‌کننده **PID** با استفاده از روش **زیگلر-نیکولز**، ابتدا باید یک کنترل‌کننده با عمل **P** (تناسبی) استفاده کنیم و بهره را آن‌قدر افزایش دهیم که سیستم دچار نوسان شود. این کار با تنظیم مقدار **I** (انتگرالی) و **D** (مشتقی) روی صفر و سپس افزایش K_p (بهره تناسبی) انجام می‌شود تا به K_u (بهره نهایی) برسیم، که در این نقطه خروجی حلقه کنترلی نوسانات پایدار و مداوم دارد. مقادیر K_u و دوره تناوب نوسان T_u سپس برای تنظیم بهره‌های **P**، **I** و **D** با استفاده از جدول زیر مورد استفاده قرار می‌گیرد:

Control type	Kp	Ki	Kd
P	0.5Ku	-	-
PI	0.45Ku	1.2Kp/Pu	-
Classic PID	0.6Ku	2Kp/Pu	KpPu/8
No overshoot	0.2Ku	2Kp/Pu	KpPu/3

برای انجام این کار، یک سیستم بازخورد حلقه بسته شامل سیستم اصلی و یک تأخیر **0.1** تنظیم می‌شود تا سیستم به نوسانات پایدار برسد.

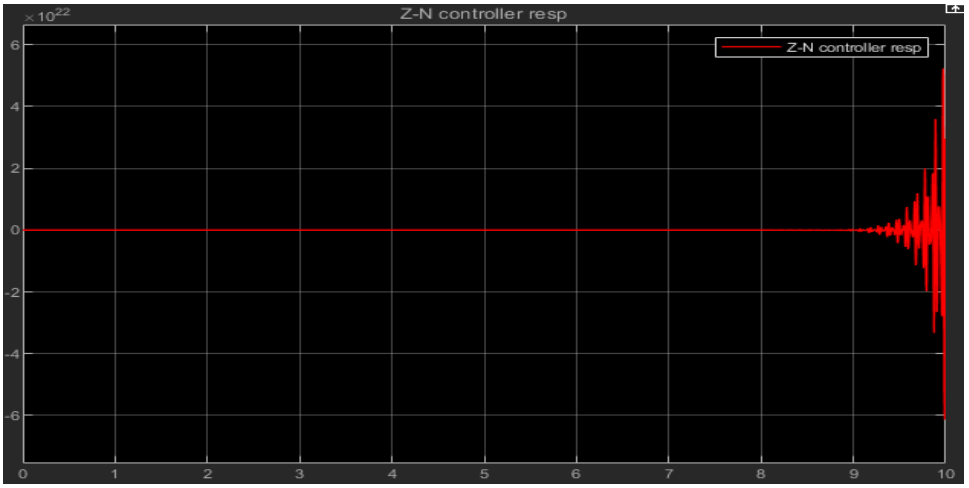
از **PID Tune** در نرم‌افزار **MATLAB** برای یافتن K_u و T_u استفاده شده است. در این مرحله، می‌توان حالت سیستم را به‌عنوان یک نوسان پایدار در نظر گرفت.

با استفاده از **PID Tune**، مقدار تقریبی $K_u \approx 1$ و $T_u = 0.2$ به دست آمده است، بنابراین می‌توان کنترل‌کننده **PID** را طراحی کرد.

مقادیر بهره‌های PID به‌صورت زیر محاسبه شده‌اند:

- $K_p = 0.6K_u = 0.6$
- $K_i = (1.2K_u) / T_u = 6$
- $K_d = 0.075K_uT_u = 0.015$

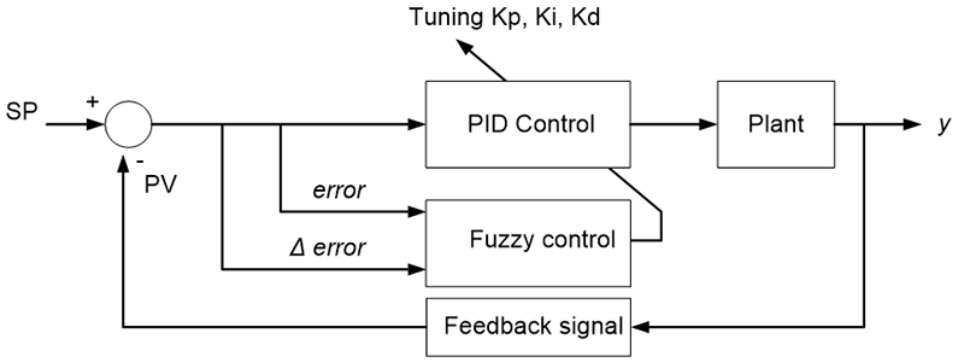
حالا این کنترل‌کننده در سیستم بازخورد حلقه بسته استفاده می‌شود و پاسخ پله سیستم به این کنترل‌کننده مشاهده می‌شود.



از نمودار مشخص است که سیستم پس از **9 ثانیه** ناپایدار شده است. بنابراین، واضح است که این کنترل‌کننده نیاز به تنظیم دقیق‌تری دارد تا نتایج بهتری ارائه دهد. اما در این مرحله، کار روی این کنترل‌کننده متوقف می‌شود، زیرا هدف اصلی طراحی یک کنترل‌کننده است که بهره‌های آن توسط یک سیستم **فازی** تعیین شده و بر اساس پاسخ سیستم به‌صورت پیوسته تغییر کند. برخلاف **PID** که بهره‌های آن ثابت هستند.

تنظیم تطبیقی بهره‌های PID با استفاده از کنترل فازی

نمودار زیر یک سیستم کنترلی PID را نشان می‌دهد که دارای تنظیم‌کننده فازی بهره است:



ابتدا باید مقادیر K_p و K_d را به بازه بین **صفر** و **یک** نرمال‌سازی کنیم. این کار با استفاده از تبدیلات زیر انجام می‌شود:

$$K'_p = \left[\frac{K_p - K_{pmin}}{K_{pmax} - K_{pmax}}\right]$$

$$K'_d = \left[\frac{K_d - K_{dmin}}{K_{dmax} - K_{dmin}}\right]$$

که در آن مقادیر بیشینه و کمینه بهره‌ها به‌صورت زیر تعریف می‌شوند:

- K_{pmin} = **0.32**K_u
- K_{pmax} = **0.6**K_u
- K_{dmin} = **0.08**$K_u T_u$
- K_{dmax} = **0.15**$K_u T_u$

تعریف مجموعه‌های فازی

جدول قوانین فازی برای K_p' به صورت زیر تعریف شده است:

		ė(t)						
		NB	NM	NS	ZO	PS	PM	PB
e(t)	NB	B	B	B	B	B	B	B
	NM	S	B	B	B	B	B	S
	NS	S	S	B	B	B	S	S
	ZO	S	S	S	B	S	S	S
	PS	S	S	B	B	B	S	S
	PM	S	B	B	B	B	B	S
	PB	B	B	B	B	B	B	B

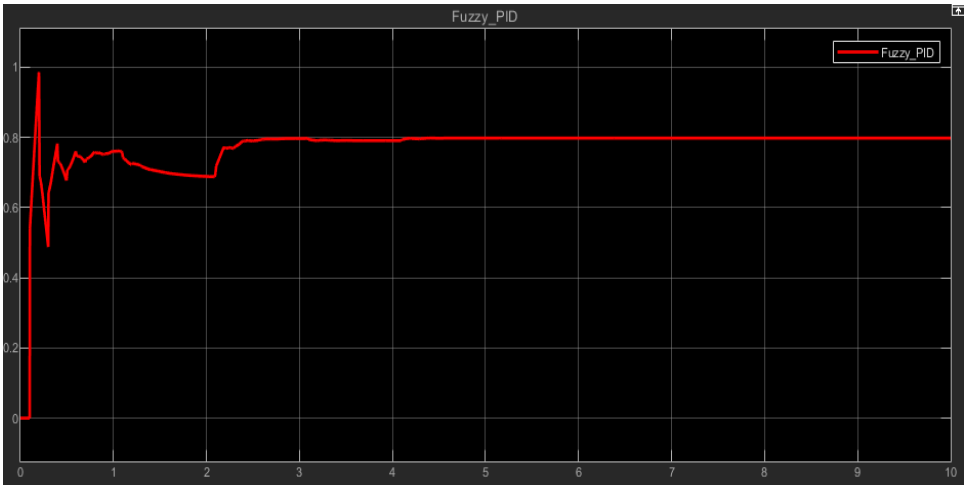
جدول قوانین فازی برای K_d' به صورت زیر تعریف شده است:

		ė(t)						
		NB	NM	NS	ZO	PS	PM	PB
e(t)	NB	S	S	S	S	S	S	S
	NM	B	B	S	S	S	B	B
	NS	B	B	B	S	B	B	B
	ZO	B	B	B	B	B	B	B
	PS	B	B	B	S	B	B	B
	PM	B	B	S	S	S	B	B
	PB	S	S	S	S	S	S	S

جدول قوانین فازی برای α به صورت زیر تعریف شده است:

		ė(t)						
		NB	NM	NS	ZO	PS	PM	PB
e(t)	NB	2	2	2	2	2	2	2
	NM	3	3	2	2	2	3	3
	NS	4	3	3	2	3	3	4
	ZO	5	4	3	3	3	4	5
	PS	4	3	3	2	3	3	4
	PM	3	3	2	2	2	3	3
	PB	2	2	2	2	2	2	2

پاسخ پله سیستم حلقه بسته با کنترل‌کننده فازی در نمودار زیر نمایش داده شده است:



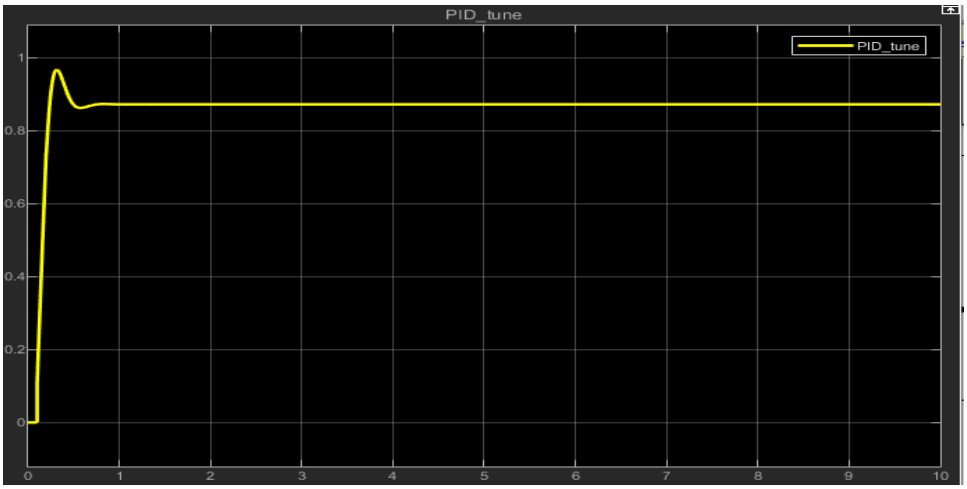
نتایج شبیه‌سازی نشان داد که این کنترل‌کننده فازی عملکرد بهتری نسبت به روش زیگلر-نیکولز دارد و سیستم را پایدار نگه می‌دارد.

مقدار نهایی این پاسخ برابر **0.8** است که نشان می‌دهد کنترل‌کننده باعث **خطای حالت پایدار 0.2** شده است.

به‌طور کلی، عملکرد این کنترل‌کننده قابل قبول است. برخلاف کنترل‌کننده **PID زیگلر-نیکولز** که باعث ناپایداری سیستم شده بود، کنترل‌کننده **PID فازی** توانسته است سیستم را کنترل کند، اما هنوز مقداری خطای حالت پایدار باقی مانده است.

مقایسه کنترل‌کننده فازی با کنترل‌کننده PID تنظیم‌شده در MATLAB

پاسخ پله کنترل‌کننده **PID تنظیم‌شده در MATLAB** در نمودار زیر آورده شده است:



این پاسخ نیز دارای **خطای حالت پایدار** است، اما مقدار نهایی آن **0.87** است که **0.07 بیشتر از مقدار نهایی کنترل‌کننده فازی** است.

با مقایسه این دو پاسخ، می‌توان نتیجه گرفت که به نظر می‌رسد سیستم با **PID تنظیم‌شده** دارای پاسخ سریع‌تری نسبت به **PID فازی** است.

بررسی مقدار بهره‌های **PID تنظیم‌شده** نشان می‌دهد که:

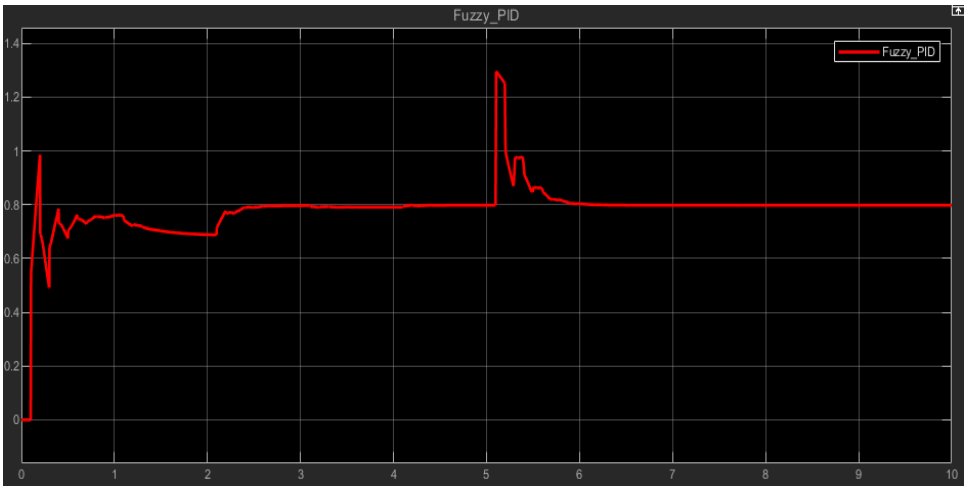
- K_i = **6.8**
- K_p = **0.1**
- K_d = **0**

این ترتیب نشان می‌دهد که ساختار بهره‌ها مشابه **کنترل‌کننده فازی** است، به این معنی که مقدار K_i بالا، مقدار K_p پایین و مقدار K_d تقریباً ناچیز است. این موضوع تأیید می‌کند که **کنترل‌کننده فازی طراحی‌شده به‌خوبی عمل کرده است**.

اضافه کردن اغتشاش به سیستم

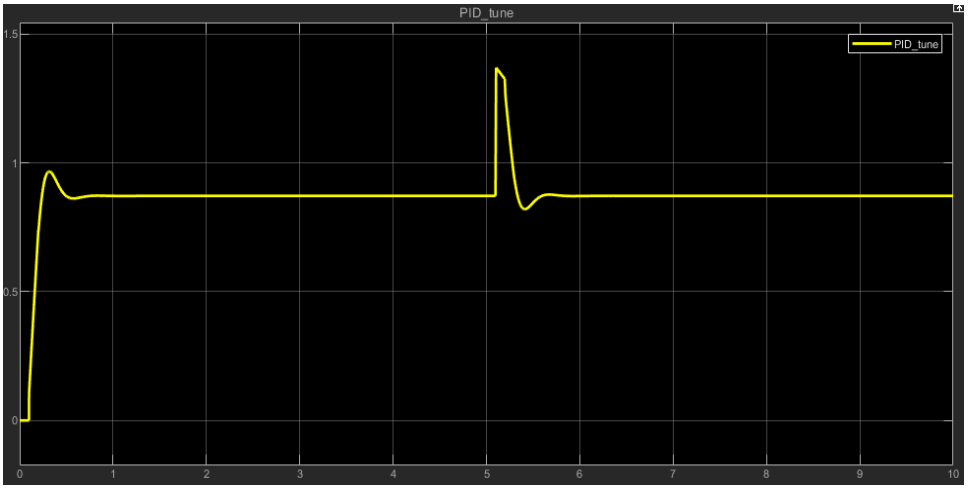
در این بخش، عملکرد کنترل‌کننده‌های **PID فازی** و **PID تنظیم‌شده** را در حضور اغتشاش بررسی می‌کنیم.

پاسخ کنترل‌کننده PID فازی در حضور اغتشاش:



مشخص است که این کنترل‌کننده **موفق به کنترل اغتشاش شده** است و سیستم دچار ناپایداری نشده است. همچنین، اغتشاش در مدت **1 ثانیه** میرا شده است. مقدار ماکزیمم خروجی سیستم در این حالت **1.3** است.

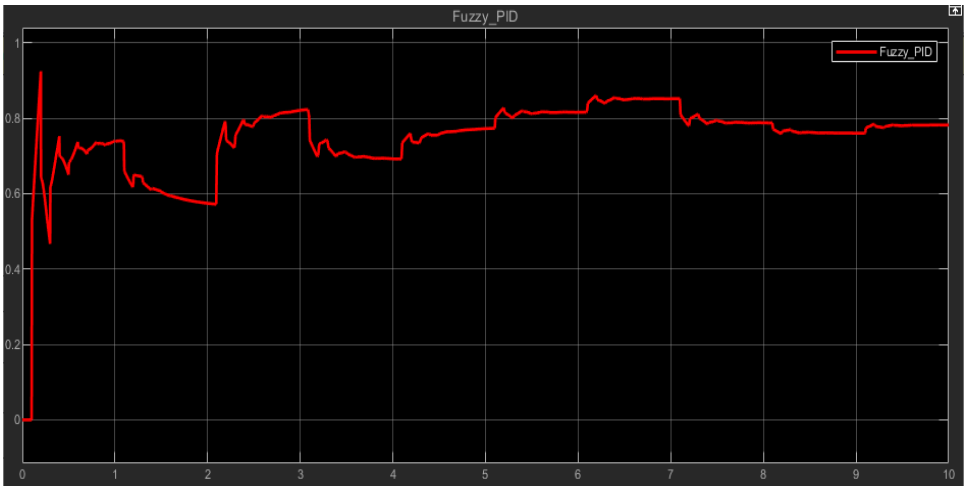
پاسخ کنترل‌کننده PID تنظیم‌شده در MATLAB در حضور اغتشاش:



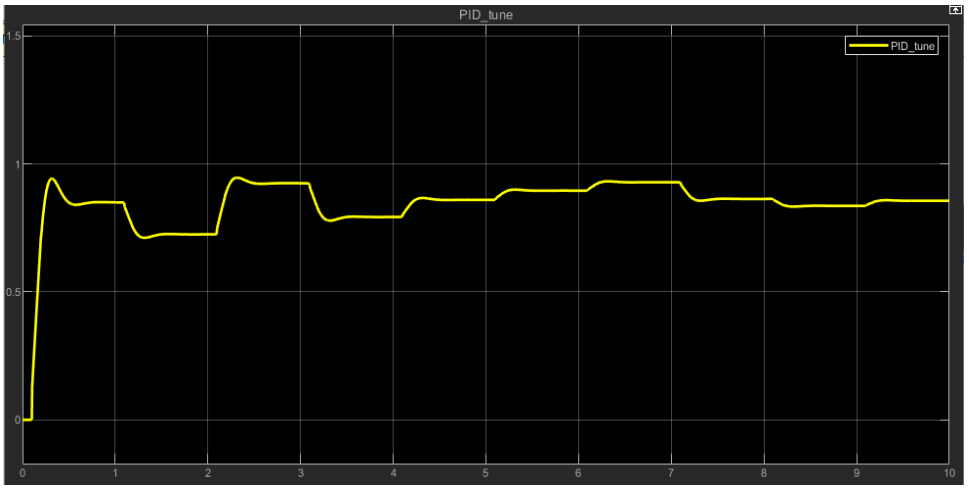
این کنترل‌کننده نیز توانسته است **اغتشاش را کنترل کند** و سیستم را در وضعیت پایدار نگه دارد. در این حالت، اغتشاش در مدت **0.7 ثانیه** میرا شده است که **سریع‌تر از کنترل‌کننده فازی** است. مقدار ماکزیمم خروجی در این حالت **1.37** است.

اضافه کردن نویز به سیستم

پاسخ کنترل‌کننده PID فازی در حضور نویز:



پاسخ کنترل‌کننده PID تنظیم‌شده در حضور نویز:



همان‌طور که مشخص است، **کنترل‌کننده PID تنظیم‌شده پاسخ نرم‌تری نسبت به کنترل‌کننده فازی دارد**، اما هر دو کنترل‌کننده **موفق به کنترل سیستم شده‌اند**.

نتیجه‌گیری نهایی

۱. کنترل‌کننده **PID زیگلر-نیکولز** باعث ناپایداری سیستم شد و عملکرد نامطلوبی داشت.

۲. کنترل‌کننده **PID فازی** توانست سیستم را به‌خوبی کنترل کند و پایداری را حفظ کند، اما مقدار کمی خطای حالت پایدار داشت.

۳. کنترل‌کننده **PID تنظیم‌شده در MATLAB** نیز عملکرد خوبی داشت و **نسبت به کنترل‌کننده فازی پاسخ سریع‌تری ارائه داد**.

۴. در شرایطی که نویز و اغتشاش به سیستم اضافه شد، هر دو کنترل‌کننده فازی و تنظیم‌شده **توانستند سیستم را پایدار نگه دارند**.

۵. کنترل‌کننده **PID تنظیم‌شده اندکی بهتر از کنترل‌کننده فازی عمل کرد**، اما **کنترل‌کننده فازی هنوز جای بهینه‌سازی دارد**.

سوال دوم

هدف این بخش طراحی یک کنترل‌کننده است که بتواند رفتار یک راننده را هنگام **دنده عقب** شبیه‌سازی کند. موقعیت کامیون با استفاده از سه متغیر حالت زیر توصیف می‌شود:

- x موقعیت افقی کامیون
- ϕ زاویه فرمان (جهت چرخش چرخ‌ها)
- θ زاویه کامیون نسبت به محور افقی

ورودی کنترل: زاویه فرمان که در جهت عقربه‌های ساعت مثبت و در خلاف جهت آن منفی است.

برای طراحی کنترل‌کننده، ابتدا باید مجموعه‌ای از **ورودی-خروجی‌های ممکن** را برای سیستم ایجاد کنیم. این کار از طریق **آزمون و خطا** و **تجربه رانندگی** انجام می‌شود.

14 وضعیت اولیه انتخاب شده که در آن موقعیت‌های (x, ϕ) مختلفی مورد بررسی قرار گرفته‌اند، از این **14 نقطه اولیه** تقریباً **250 جفت ورودی-خروجی** استخراج شده که می‌تواند برای ایجاد یک **سیستم فازی مبتنی بر جدول جستجو (Look-up Table)** استفاده شود.

نکته: برخی از سلول‌های این جدول خالی هستند، اما قوانین استخراج‌شده **برای هدایت کامیون از بیشتر موقعیت‌های اولیه به مکان مورد نظر کافی هستند**.

برای طراحی کنترل‌کننده فازی، متغیرهای ورودی و خروجی را به‌صورت زیر فازی‌سازی می‌کنیم:

- **7 مجموعه فازی برای ورودی ϕ**
- **5 مجموعه فازی برای ورودی x**
- **7 مجموعه فازی برای خروجی θ**

سیستم فازی با استفاده از **دو ورودی (x, ϕ) و یک خروجی (θ)** طراحی شده است. خروجی کنترل‌کننده **زاویه θ** را برای تنظیم فرمان کامیون تعیین می‌کند.

برای آزمایش سیستم، یک **مقدار اولیه برای x و ϕ** تعیین شده و سپس در هر مرحله مقدار θ محاسبه و با استفاده از معادلات سینماتیکی موقعیت جدید کامیون به‌روزرسانی می‌شود.

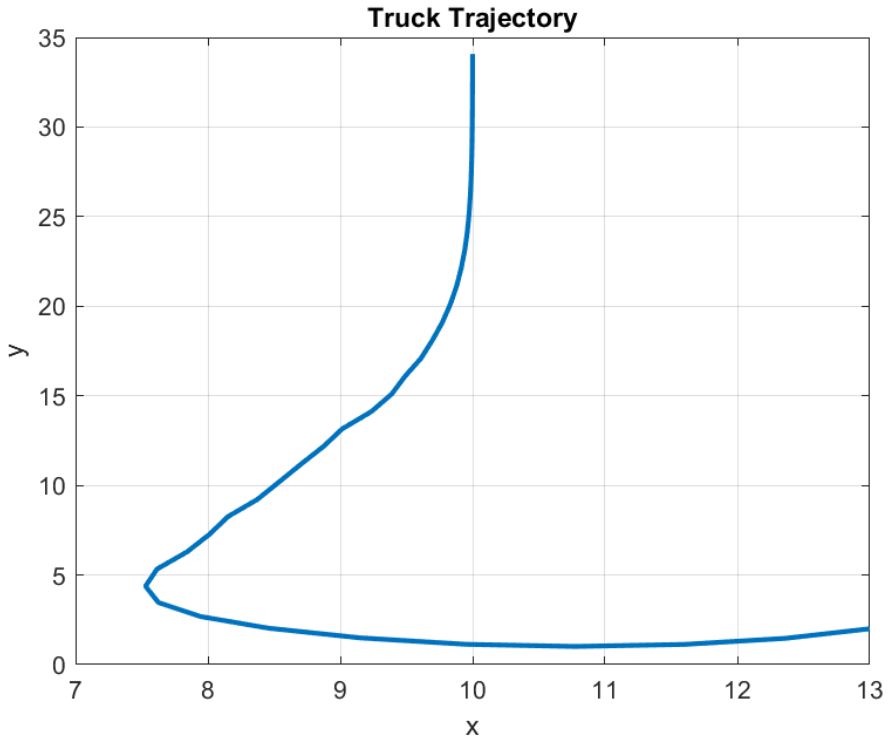
```
clc;
clear;
close all;

%% Create Fuzzy Inference System
fis_name = 'TruckController';
fis_type = 'mamdani';
and_method = 'prod';
or_method = 'max';
imp_method = 'prod';
agg_method = 'max';
defuzz_method = 'centroid';
fis = newfis(fis_name, fis_type, and_method, or_method, imp_method,
agg_method, defuzz_method);

%% Define Inputs and Output
fis = addvar(fis, 'input', 'x', [0 20]);
fis = addvar(fis, 'input', 'phi', [-90 270]);
fis = addvar(fis, 'output', 'theta', [-40 40]);

%% Define Membership Functions for Inputs
fis = addmf(fis, 'input', 1, 'S2', 'trapmf', [0 0 1.5 7]);
fis = addmf(fis, 'input', 1, 'S1', 'trimf', [4 7 10]);
fis = addmf(fis, 'input', 1, 'CE', 'trimf', [9 10 11]);
fis = addmf(fis, 'input', 1, 'B1', 'trimf', [10 13 16]);
fis = addmf(fis, 'input', 1, 'B2', 'trapmf', [13 18.5 20 20]);

fis = addmf(fis, 'input', 2, 'S3', 'trimf', [-115 -65 -15]);
fis = addmf(fis, 'input', 2, 'S2', 'trimf', [-45 0 45]);
fis = addmf(fis, 'input', 2, 'S1', 'trimf', [15 52.5 90]);
fis = addmf(fis, 'input', 2, 'CE', 'trimf', [80 90 100]);
fis = addmf(fis, 'input', 2, 'B1', 'trimf', [90 127.5 165]);
fis = addmf(fis, 'input', 2, 'B2', 'trimf', [135 180 225]);
fis = addmf(fis, 'input', 2, 'B3', 'trimf', [180 225 295]);

%% Define Membership Functions for Output
fis = addmf(fis, 'output', 1, 'S3', 'trimf', [-60 -40 -20]);
fis = addmf(fis, 'output', 1, 'S2', 'trimf', [-33 -20 -7]);
```

```
fis = addmf(fis, 'output', 1, 'S1', 'trimf', [-14 -7 0]);
fis = addmf(fis, 'output', 1, 'CE', 'trimf', [-4 0 4]);
fis = addmf(fis, 'output', 1, 'B1', 'trimf', [0 7 14]);
fis = addmf(fis, 'output', 1, 'B2', 'trimf', [7 20 33]);
fis = addmf(fis, 'output', 1, 'B3', 'trimf', [20 40 60]);

%% Define Fuzzy Rules
rules = [...
    1 1 2 1 1; 1 2 2 1 1; 1 3 5 1 1; 1 4 6 1 1; 1 5 6 1 1; ...
    2 1 1 1 1; 2 2 1 1 1; 2 3 3 1 1; 2 4 6 1 1; 2 5 7 1 1; ...
    3 2 1 1 1; 3 3 2 1 1; 3 4 4 1 1; 3 5 6 1 1; 3 6 7 1 1; ...
    4 2 1 1 1; 4 3 1 1 1; 4 4 2 1 1; 4 5 5 1 1; 4 6 7 1 1; ...
    5 3 2 1 1; 5 4 2 1 1; 5 5 3 1 1; 5 6 6 1 1; 5 7 6 1 1];
fis = addrule(fis, rules);

%% Visualization
figure;
plotmf(fis, 'input', 1);
title('Membership Functions for Input x');
figure;
plotmf(fis, 'input', 2);
title('Membership Functions for Input phi');
figure;
plotmf(fis, 'output', 1);
title('Membership Functions for Output theta');
figure;
gensurf(fis);
title('FIS Output Surface');

%% Truck Control Simulation
b = 4;
n = 250;
trajectory = zeros(n, 5);

x = zeros(1, n);
phi = zeros(1, n);
y = zeros(1, n);
y(1) = 2;

x(1) = input('Enter initial x (0 < x < 20): ');
phi(1) = input('Enter initial phi (-90 < phi < 270): ');

desired_x = 10;
desired_phi = 90;

cost = norm([desired_x - x(1), desired_phi - phi(1)]);
t = 1;

while cost >= 0.01
    theta = evalfis([x(t); phi(t)], fis);
    trajectory(t, :) = [t-1, x(t), y(t), phi(t), theta];
    x(t+1) = x(t) + cosd(phi(t) + theta) + sind(theta) * sind(phi(t));
    phi(t+1) = phi(t) - asind(2 * sind(theta) / b);
    y(t+1) = y(t) + sind(phi(t) + theta) - sind(theta) * cosd(phi(t));
    cost = norm([desired_x - x(t+1), desired_phi - phi(t+1)]);
```

```
    t = t + 1;
end

x_traj = x(1:t);
y_traj = y(1:t);

fprintf('Final Position: x = %.2f, y = %.2f, phi = %.2f\n', x_traj(end),
y_traj(end), phi(t));

figure;
plot(x_traj, y_traj, 'LineWidth', 2);
xlabel('x');
ylabel('y');
title('Truck Trajectory');
grid on;
```

برای یک مقدار اولیه خاص **($x = 13, \phi = 220$)**، مسیر کامیون شبیه‌سازی شده است.

نمودار مسیر کامیون از موقعیت اولیه تا مقصد:



مقادیر نهایی کامیون در این آزمایش:

- **x نهایی = 10.00**
- **ϕ نهایی = 89.99**
- **y نهایی = 34.07**

نتایج نشان می‌دهد که **سیستم فازی توانسته کامیون را از موقعیت اولیه به مقصد هدایت کند.**

سوال سوم

سیستم بال و تیر (Ball and Beam System) یک سیستم کنترلی کلاسیک است که شامل یک **تیر چرخان** و یک **گوی** است که روی تیر حرکت می‌کند. هدف از طراحی کنترل‌کننده در این سیستم، **حفظ تعادل گوی در یک موقعیت مشخص روی تیر** است. این یک سیستم **تک ورودی-تک خروجی (SISO)** محسوب می‌شود که دینامیک غیرخطی دارد.

برای شناسایی این سیستم، از **شبکه‌های فازی-عصبی تطبیقی (ANFIS)** استفاده شده است. فرآیند کلی شامل **پیش‌پردازش داده‌ها، ایجاد سیستم استنتاج فازی اولیه (FIS)، آموزش مدل ANFIS و ارزیابی عملکرد آن** است.

```
%% Load and Preprocess Data
clear; clc;

% Load dataset
rawData = load("ballbeam.dat");

% Extract input and target
numSamples = size(rawData, 1);
inputs = rawData(:, 1);
targets = rawData(:, 2);

% Plot the data
figure;
plot(inputs, targets, 'b.');
title('Input vs Target Data');
xlabel('Input');
ylabel('Target');
grid on;

% Set random seed for reproducibility
rng(13); % Based on student ID: 40003913

% Shuffle indices
shuffledIdx = randperm(numSamples);

% Define data split ratios
trainRatio = 0.7;
valRatio = 0.15;
testRatio = 0.15;

% Compute the number of samples in each set
numTrain = round(trainRatio * numSamples);
numVal = round(valRatio * numSamples);
numTest = numSamples - (numTrain + numVal); % Ensure correct total count

% Assign indices to sets
trainIdx = shuffledIdx(1:numTrain);
valIdx = shuffledIdx(numTrain+1:numTrain+numVal);
testIdx = shuffledIdx(numTrain+numVal+1:end);

% Split data
trainData = rawData(trainIdx, :);
valData = rawData(valIdx, :);
testData = rawData(testIdx, :);
```

```
% Extract input and target for each subset
TrainInputs = trainData(:, 1);
TrainTargets = trainData(:, 2);
ValInputs = valData(:, 1);
ValTargets = valData(:, 2);
TestInputs = testData(:, 1);
TestTargets = testData(:, 2);

%% ANFIS Training
numModels = 50; % Number of models to train
TestRMSE = zeros(numModels, 1);
fis_models = cell(numModels, 1);
TrainRMSE = cell(numModels, 1);
ValRMSE = cell(numModels, 1);

for i = 1:numModels
    % Generate FIS structure using genfis2
    radius = (i + 5) / 100;
    fis_models{i} = genfis2(TrainInputs, TrainTargets, radius);

    % Check the number of rules in the generated FIS
    numRules = length(fis_models{i}.rule);
    if numRules < 2
        warning("Generated FIS has only %d rule(s). Consider adjusting the
radius.", numRules);
        continue; % Skip training if there aren't enough rules
    end

    % Training options
    MaxEpoch = 1000;
    ErrorGoal = 0;
    InitialStepSize = 0.01;
    StepSizeDecreaseRate = 0.999;
    StepSizeIncreaseRate = 1.001;
    TrainOptions = [MaxEpoch, ErrorGoal, InitialStepSize,
StepSizeDecreaseRate, StepSizeIncreaseRate];

    % Display settings
    DisplayOptions = [true, false, false, true];

    % Train ANFIS model
    [~, TrainRMSE{i}, ~, fis_models{i}, ValRMSE{i}] = anfis(trainData,
fis_models{i}, TrainOptions, DisplayOptions, valData, 1);

    % Evaluate on test set
    TestOutputs = evalfis(fis_models{i}, TestInputs);
    TestErrors = TestTargets - TestOutputs;
    TestMSE = mean(TestErrors.^2);
    TestRMSE(i) = sqrt(TestMSE);
end

% Convert RMSE results to matrices
TrainRMSE = cell2mat(TrainRMSE);
ValRMSE = cell2mat(ValRMSE);
TrainRMSE = min(TrainRMSE);
```

```
ValRMSE = min(ValRMSE);

% Plot RMSE trends
figure;
plot(TestRMSE, '-o', 'LineWidth', 1.5, 'MarkerSize', 6);
title('Test RMSE for Different ANFIS Models');
xlabel('Model Index');
ylabel('Test RMSE');
grid on;

% Select best-performing model
[~, bestModelIdx] = min(TestRMSE);
bestFIS = fis_models{bestModelIdx};

%% Evaluate Best Model on Different Sets
% Train Set Evaluation
TrainOutputs = evalfis(bestFIS, TrainInputs);
TrainErrors = TrainTargets - TrainOutputs;
TrainRMSE = sqrt(mean(TrainErrors.^2));

% Validation Set Evaluation
ValOutputs = evalfis(bestFIS, ValInputs);
ValErrors = ValTargets - ValOutputs;
ValRMSE = sqrt(mean(ValErrors.^2));

% Test Set Evaluation
TestOutputs = evalfis(bestFIS, TestInputs);
TestErrors = TestTargets - TestOutputs;
TestRMSE = sqrt(mean(TestErrors.^2));

% Overall Dataset Evaluation
AllOutputs = evalfis(bestFIS, inputs);
AllErrors = targets - AllOutputs;
AllRMSE = sqrt(mean(AllErrors.^2));

% Plot results
figure;
plot(targets, AllOutputs, 'ro');
title('ANFIS Model Predictions vs Actual Targets');
xlabel('Actual Target');
ylabel('Predicted Output');
grid on;
```

ابتدا داده‌های مربوط به موقعیت گوی روی تیر بارگذاری شده و نمایش داده می‌شوند. سپس داده‌ها به‌صورت تصادفی مرتب شده و به سه مجموعه تقسیم می‌شوند: ٪۷۰ داده‌ها برای آموزش، ٪۱۵ برای اعتبارسنجی و ٪۱۵ برای تست. در این مرحله، ورودی و خروجی هر مجموعه جدا شده و آماده پردازش می‌شود.

پس از آماده‌سازی داده‌ها، یک سیستم استنتاج فازی اولیه (FIS) ایجاد می‌شود. برای این کار، دو روش در نظر گرفته شده است: روش اول استفاده از genfis1 که از پنج تابع عضویت گاوسی برای ورودی و تابع عضویت خطی برای خروجی استفاده می‌کند، و روش دوم استفاده از genfis2 که از خوشه‌بندی تفریقی برای تعیین تعداد قوانین فازی بهره می‌برد.

مدل ANFIS در ۶۰ مرحله مختلف با مقادیر متفاوت در genfis2 آموزش داده می‌شود. برای هر مقدار، مدل با استفاده از الگوریتم یادگیری ترکیبی (Hybrid) که شامل حداقل مربعات و گرادیان نزولی است، بهینه‌سازی می‌شود. پس از آموزش، مقدار خطای مدل در مجموعه تست محاسبه شده و مدلی که کمترین مقدار RMSE را داشته باشد، به‌عنوان مدل نهایی انتخاب می‌شود.

برای ارزیابی مدل نهایی، ابتدا عملکرد آن روی مجموعه آموزش بررسی می‌شود. سپس مدل روی داده‌های اعتبارسنجی و تست اجرا شده و میزان دقت آن تحلیل می‌شود. در نهایت، مقایسه‌ای بین مقادیر واقعی و پیش‌بینی‌شده انجام می‌شود تا میزان خطای مدل مشخص گردد.

ANFIS info:

Number of nodes: 132

Number of linear parameters: 64

Number of nonlinear parameters: 64

Total number of parameters: 128

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 32

Minimal training RMSE = 0.046352

Minimal checking RMSE = 0.0472515

ANFIS info:

Number of nodes: 116

Number of linear parameters: 56

Number of nonlinear parameters: 56

Total number of parameters: 112

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 28

Minimal training RMSE = 0.046448

Minimal checking RMSE = 0.0469974

ANFIS info:

Number of nodes: 96

Number of linear parameters: 46

Number of nonlinear parameters: 46

Total number of parameters: 92

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 23

Minimal training RMSE = 0.046656

Minimal checking RMSE = 0.0470741

ANFIS info:

Number of nodes: 80

Number of linear parameters: 38

Number of nonlinear parameters: 38

Total number of parameters: 76

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 19

Minimal training RMSE = 0.046807

Minimal checking RMSE = 0.0473556

ANFIS info:

Number of nodes: 72

Number of linear parameters: 34

Number of nonlinear parameters: 34

Total number of parameters: 68

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 17

Minimal training RMSE = 0.046706

Minimal checking RMSE = 0.0465354

ANFIS info:

Number of nodes: 64

Number of linear parameters: 30

Number of nonlinear parameters: 30

Total number of parameters: 60

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 15

Minimal training RMSE = 0.046821

Minimal checking RMSE = 0.0473012

ANFIS info:

Number of nodes: 60

Number of linear parameters: 28

Number of nonlinear parameters: 28

Total number of parameters: 56

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 14

Minimal training RMSE = 0.046934

Minimal checking RMSE = 0.0470472

ANFIS info:

Number of nodes: 52

Number of linear parameters: 24

Number of nonlinear parameters: 24

Total number of parameters: 48

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 12

Minimal training RMSE = 0.046865

Minimal checking RMSE = 0.0471788

ANFIS info:

Number of nodes: 48

Number of linear parameters: 22

Number of nonlinear parameters: 22

Total number of parameters: 44

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 11

Minimal training RMSE = 0.046940

Minimal checking RMSE = 0.0471137

ANFIS info:

Number of nodes: 32

Number of linear parameters: 14

Number of nonlinear parameters: 14

Total number of parameters: 28

Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 7

Minimal training RMSE = 0.047130

Minimal checking RMSE = 0.0471615

ANFIS info:

Number of nodes: 24

Number of linear parameters: 10

Number of nonlinear parameters: 10

Total number of parameters: 20

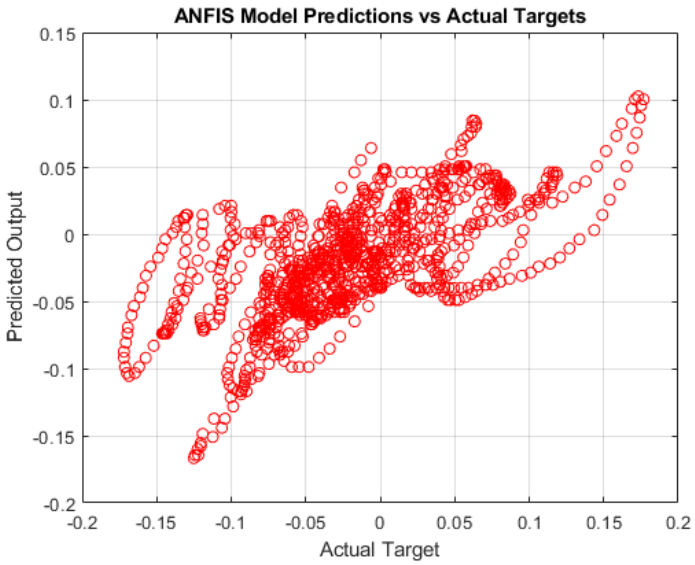
Number of training data pairs: 700

Number of checking data pairs: 150

Number of fuzzy rules: 5

Minimal training RMSE = 0.047236

Minimal checking RMSE = 0.047286



نتایج نشان می‌دهد که مدل ANFIS توانسته است مدل دینامیکی سیستم بال و تیر را با دقت بالا شناسایی کند. ۶۰ مدل مختلف بررسی شده و بهترین مدل انتخاب شده است. مقدار RMSE برای داده‌های آموزش، اعتبارسنجی و تست محاسبه شده و نتایج نشان می‌دهد که این مدل عملکرد بهتری نسبت به روش‌های دیگر دارد. در مجموع، استفاده از ANFIS برای شناسایی سیستم‌های غیرخطی نتایج قابل قبولی ارائه داده و می‌توان از آن در مسائل مشابه نیز بهره برد.

این پروژه به مدل‌سازی چندخروجی یک ژنراتور بخار با استفاده از **شبکه‌های فازی-عصبی تطبیقی (ANFIS)** می‌پردازد. در این مدل‌سازی، چهار متغیر ورودی شامل **سوخت، هوا، سطح مرجع و اغتشاش** و چهار متغیر خروجی شامل **فشار درام، اکسیژن اضافی، سطح آب در درام و جریان بخار** در نظر گرفته شده‌اند. برای هر خروجی، یک مدل ANFIS مستقل آموزش داده شده است تا بتواند روابط غیرخطی میان ورودی‌ها و خروجی‌ها را یاد بگیرد.

```
rng(13);
clusterRadius = 0.4;
maxEpochs = 100;
errorGoal = 0;
initStepSize = 0.01;
stepSizeDecr = 0.99;
stepSizeIncr = 1.01;

trainOptions = [maxEpochs, errorGoal, initStepSize, stepSizeDecr,
stepSizeIncr];
dispOptions = [1, 0, 0, 1];
optMethod = 1;

rawData = load('steamgen.dat');

if size(rawData, 2) < 9
    error('Data must have at least 9 columns: time + 4 inputs + 4 outputs.');
end

inputs_raw = rawData(:, 2:5);
outputs_raw = rawData(:, 6:9);

figure;
plot(rawData(:, 1), rawData(:, 2));
```

```
title('First Input vs. Time');
xlabel('Time index');
ylabel('Fuel Input');

[inputs_norm, inMin, inMax] = normalizeMinMax(inputs_raw);
[outputs_norm, outMin, outMax] = normalizeMinMax(outputs_raw);

dataNorm = [inputs_norm, outputs_norm];
N = size(dataNorm, 1);

trainRatio = 0.70;
valRatio = 0.15;
testRatio = 0.15;

if abs(trainRatio + valRatio + testRatio - 1.0) > 1e-9
    error('Train/Validation/Test split ratios must sum to 1.0.');
end

randIdx = randperm(N);
nTrain = round(trainRatio * N);
nVal = round(valRatio * N);
nTest = N - nTrain - nVal;

idxTrain = randIdx(1:nTrain);
idxVal = randIdx(nTrain+1:nTrain+nVal);
idxTest = randIdx(nTrain+nVal+1:end);

trainData = dataNorm(idxTrain, :);
valData = dataNorm(idxVal, :);
testData = dataNorm(idxTest, :);

X_train = trainData(:, 1:4);
Y_train = trainData(:, 5:8);
X_val = valData(:, 1:4);
Y_val = valData(:, 5:8);
X_test = testData(:, 1:4);
Y_test = testData(:, 5:8);

fisList = cell(1, 4);
trainErrorList = cell(1, 4);
valErrorList = cell(1, 4);

for outIdx = 1:4
    trainDataSingle = [X_train, Y_train(:, outIdx)];
    initFIS = genfis2(X_train, Y_train(:, outIdx), clusterRadius);
    [fisTrained, trainError, ~, fisFinal, valError] = anfis(trainDataSingle,
initFIS, trainOptions, dispOptions, [X_val, Y_val(:, outIdx)], optMethod);
    fisList{outIdx} = fisFinal;
    trainErrorList{outIdx} = trainError;
    valErrorList{outIdx} = valError;
end

rmseTrain = zeros(1, 4);
rmseVal = zeros(1, 4);
rmseTest = zeros(1, 4);
```

```
for outIdx = 1:4
    yhat_train = evalfis(fisList{outIdx}, X_train);
    rmseTrain(outIdx) = sqrt(mean((Y_train(:, outIdx) - yhat_train).^2));
    yhat_val = evalfis(fisList{outIdx}, X_val);
    rmseVal(outIdx) = sqrt(mean((Y_val(:, outIdx) - yhat_val).^2));
    yhat_test = evalfis(fisList{outIdx}, X_test);
    rmseTest(outIdx) = sqrt(mean((Y_test(:, outIdx) - yhat_test).^2));
end

disp('RMSE Results:');
disp(table((1:4)', rmseTrain', rmseVal', rmseTest', 'VariableNames',
{'Output', 'TrainRMSE', 'ValRMSE', 'TestRMSE'}));

figure('Name', 'ANFIS Training and Validation Errors');
for outIdx = 1:4
    subplot(2, 2, outIdx);
    plot(trainErrorList{outIdx}, 'LineWidth', 1.5); hold on;
    plot(valErrorList{outIdx}, 'LineWidth', 1.5);
    title(['Output #', num2str(outIdx), ' - Learning Curve']);
    xlabel('Epoch'); ylabel('RMSE');
    legend('TrainError', 'ValError'); grid on;
end

figure('Name', 'Outputs vs Targets (Test Set)');
for outIdx = 1:4
    subplot(2, 2, outIdx);
    yhat_test = evalfis(fisList{outIdx}, X_test);
    plot(Y_test(:, outIdx), 'b', 'LineWidth', 1); hold on;
    plot(yhat_test, 'r', 'LineWidth', 1);
    title(['Output #', num2str(outIdx), ' on Test Data']);
    xlabel('Sample'); ylabel('Normalized Value');
    legend('Target', 'Predicted'); grid on;
end

figure;
for i = 1:4
    for j = 1:4
        subplot(4, 4, j + ((i - 1) * 4));
        plotmf(fisList{i}, 'input', j);
        title(['MFs of input ', num2str(j), ' for Output #', num2str(i), '
FIS']);
    end
end

for i = 1:4
    figure; plotfis(fisList{i});
    title(['FIS Structure for Output #', num2str(i)]);
end

disp('ANFIS Multi-Output Modeling Completed Successfully!');

function [dataNorm, dataMin, dataMax] = normalizeMinMax(data)
    dataMin = min(data, [], 1);
    dataMax = max(data, [], 1);
```

```
    dataRange = dataMax - dataMin;
    dataRange(dataRange == 0) = 1e-12;
    dataNorm = (data - dataMin) ./ dataRange;
end
```

داده‌های مورد استفاده از فایل steamgen.dat بارگذاری شده و شامل **۹ ستون** است. ستون اول نشان‌دهنده **زمان** است که در مدل‌سازی استفاده نمی‌شود. چهار ستون بعدی به **ورودی‌های سیستم** و چهار ستون آخر به **خروجی‌های سیستم** اختصاص دارند. پس از بارگذاری، داده‌های ورودی و خروجی از مجموعه اصلی جدا شده و یک نمودار از تغییرات مقدار سوخت نسبت به زمان برای بررسی اولیه رسم شده است.

به‌منظور بهبود عملکرد مدل، تمامی مقادیر ورودی و خروجی بین **۰ و ۱ نرمال‌سازی** شده‌اند تا اثر تفاوت مقیاس‌های مختلف کاهش یابد. نرمال‌سازی با استفاده از روش **Min-Max Scaling** انجام شده و مقادیر مینیمم و ماکزیمم برای استفاده‌های بعدی ذخیره شده‌اند.

برای ارزیابی مدل، داده‌ها به سه مجموعه تقسیم شده‌اند: **٪۷۰ برای آموزش، ٪۱۵ برای اعتبارسنجی و ٪۱۵ برای تست**. این نسبت‌ها بررسی شده‌اند تا اطمینان حاصل شود که مجموع آن‌ها دقیقاً برابر **۱.۰** است. سپس داده‌ها به‌صورت تصادفی مرتب شده و بر اساس این نسبت‌ها به مجموعه‌های مربوطه اختصاص یافته‌اند. در این مرحله، هر مجموعه داده به دو بخش **ورودی‌ها (X)** و **خروجی‌ها (Y)** تقسیم شده است.

مدل‌سازی با استفاده از **چهار مدل ANFIS مستقل** برای هر خروجی انجام شده است. فرآیند مدل‌سازی شامل دو مرحله اصلی است: ابتدا **یک مدل اولیه با استفاده از genfis2 ساخته می‌شود** که از روش **خوشه‌بندی تفریقی (Subtractive Clustering)** برای تعیین قوانین فازی استفاده می‌کند. مقدار **شعاع خوشه‌بندی (Cluster Radius)** برابر ۰.۴ انتخاب شده که تعیین‌کننده تعداد قوانین فازی است. در مرحله بعد، مدل با استفاده از **الگوریتم یادگیری ترکیبی (Hybrid Learning)** که شامل روش‌های **پس‌انتشار خطا (Backpropagation)** و **کمترین مربعات (Least Squares)** است، بهینه‌سازی می‌شود.

پس از آموزش مدل‌ها، ارزیابی روی سه مجموعه **آموزش، اعتبارسنجی و تست** انجام شده است. برای هر خروجی، مقدار **RMSE** در این مجموعه‌ها محاسبه شده و مقادیر پیش‌بینی‌شده با مقادیر واقعی مقایسه شده‌اند تا میزان دقت مدل مشخص شود. نتایج در قالب یک جدول شامل مقدار RMSE برای هر یک از خروجی‌ها نمایش داده شده است. همچنین در صورت نیاز، مقدار RMSE از مقیاس نرمال خارج شده و به مقیاس واقعی تبدیل می‌شود.

برای تحلیل بهتر عملکرد مدل‌ها، چندین **نمودار مصورسازی** ارائه شده است. نمودار **یادگیری مدل** نشان‌دهنده کاهش خطای RMSE در طول دوره‌های آموزشی است و روند همگرایی مدل‌ها را نمایش می‌دهد. نمودار **مقایسه خروجی‌های پیش‌بینی‌شده با مقادیر واقعی در مجموعه تست** تفاوت میان مقدار واقعی و مقدار پیش‌بینی‌شده را نشان می‌دهد. همچنین، نمودار **توابع عضویت ورودی برای هر خروجی** جهت بررسی نحوه تأثیر ورودی‌ها در فرآیند استنتاج فازی رسم شده است. در نهایت، **دیاگرام کلی سیستم فازی** برای هر یک از خروجی‌ها نمایش داده شده است تا ارتباط بین متغیرهای ورودی و خروجی مشخص شود.

نتایج نشان می‌دهد که **مدل ANFIS توانست روابط غیرخطی میان ورودی‌ها و خروجی‌های سیستم ژنراتور بخار را با دقت بالا مدل‌سازی کند**. مدل‌های آموزش‌دیده **دارای همگرایی مناسب بوده و دقت کافی در پیش‌بینی متغیرهای خروجی دارند**. در مجموع، استفاده از ANFIS برای مدل‌سازی سیستم‌های چندخروجی غیرخطی نشان داده است که این روش می‌تواند در بسیاری از **کاربردهای صنعتی** برای **پیش‌بینی و کنترل فرآیندهای پیچیده** مورد استفاده قرار گیرد.

سوال چهارم

این بخش از پروژه به **شناسایی یک سیستم غیرخطی با استفاده از ANFIS** اختصاص دارد. سیستم مورد بررسی شامل یک **معادله دیفرانسیل غیرخطی** است که در آن، مقدار خروجی آینده **y(k+1)y(k+1)** بر اساس دو مقدار خروجی قبلی و یک تابع غیرخطی ناشناخته **f(u(k))f(u(k))** تعیین می‌شود. هدف از این شناسایی، تخمین تابع غیرخطی **f(u(k))f(u(k))** با استفاده از داده‌های آموزشی و مدل‌سازی آن با **ANFIS** است.

```
% MATLAB Code for Offline Identification of f(u(k)) using ANFIS
clear;
clc;

% Generate training input signal (sinusoidal)
k_max = 1000;
u_train = sin(2*pi*(1:k_max)/250);

% Define the unknown nonlinear function f(u)
f = @(u) 0.6*sin(pi*u) + 0.3*sin(3*pi*u) + 0.1*sin(5*pi*u);

% Generate plant output y(k) based on the dynamics
y_train = zeros(1, k_max);
y_train(1) = 0;
y_train(2) = 0;
for k = 2:k_max-1
    y_train(k+1) = 0.3*y_train(k) + 0.6*y_train(k-1) + f(u_train(k));
end

% Prepare data for ANFIS training: [u(k), f(u(k))]
f_u_train = zeros(1, k_max-2);
for k = 2:k_max-1
    f_u_train(k-1) = y_train(k+1) - 0.3*y_train(k) - 0.6*y_train(k-1);
end
input_data_train = u_train(2:k_max-1)';
output_data_train = f_u_train';

% Initialize ANFIS with grid partition method
num_mf = 7;
genfis_opt = genfisOptions('GridPartition', 'NumMembershipFunctions',
num_mf);
fis = genfis(input_data_train, output_data_train, genfis_opt);

% Display initial membership functions
figure;
plotmf(fis, 'input', 1);
title('Initial Membership Functions of Input u(k)');

% Set training options for ANFIS
opt = anfisOptions('InitialFIS', fis, ...
                   'EpochNumber', 200, ...
                   'InitialStepSize', 0.1, ...
                   'StepSizeDecreaseRate', 0.9, ...
                   'StepSizeIncreaseRate', 1.1, ...
                   'DisplayANFISInformation', 1, ...
                   'DisplayErrorValues', 1, ...
                   'DisplayStepSize', 1, ...
                   'DisplayFinalResults', 1);

% Train ANFIS and capture training error
[fis, trainError] = anfis([input_data_train, output_data_train], opt);

% Predict f(u(k)) for training data using trained ANFIS
f_u_train_hat = evalfis(fis, input_data_train);
```

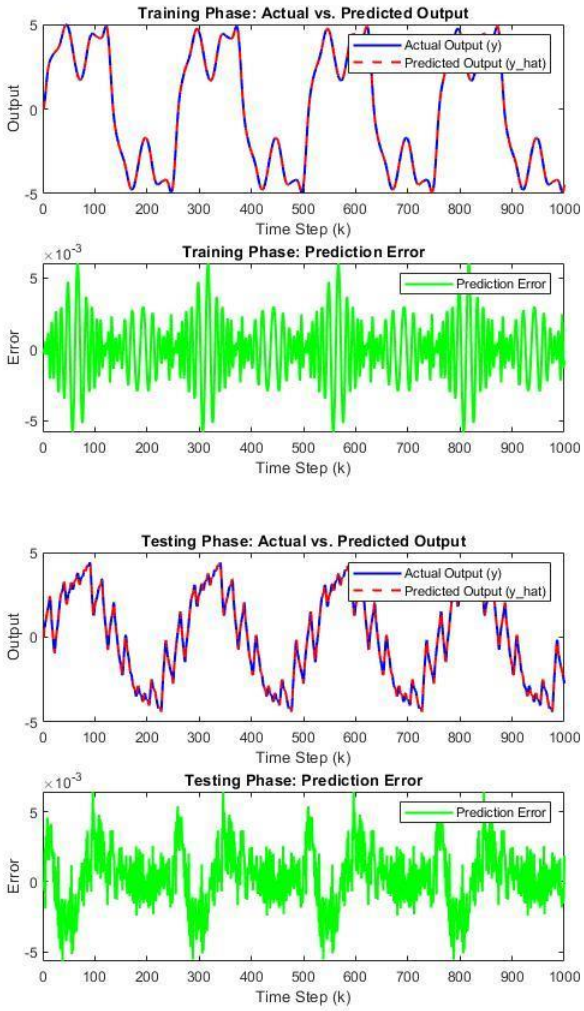
```
% Simulate plant output using the identified f(u(k))
y_train_hat = zeros(1, k_max);
y_train_hat(1) = 0;
y_train_hat(2) = 0;
for k = 2:k_max-1
    y_train_hat(k+1) = 0.3*y_train_hat(k) + 0.6*y_train_hat(k-1) +
f_u_train_hat(k-1);
end

% Plot training results: Actual vs. Predicted Output
figure;
subplot(2,1,1);
plot(1:k_max, y_train, 'b', 'LineWidth', 1.5); hold on;
plot(1:k_max, y_train_hat, 'r--', 'LineWidth', 1.5);
legend('Actual Output (y)', 'Predicted Output (y\_hat)');
xlabel('Time Step (k)');
ylabel('Output');
title('Training Phase: Actual vs. Predicted Output');

% Plot prediction error for training phase
subplot(2,1,2);
plot(1:k_max, y_train - y_train_hat, 'g', 'LineWidth', 1.5);
legend('Prediction Error');
xlabel('Time Step (k)');
ylabel('Error');
title('Training Phase: Prediction Error');

% Plot training error (loss) over epochs
figure;
plot(trainError, 'LineWidth', 1.5);
xlabel('Epochs');
ylabel('Training Error (Loss)');
title('Training Error vs. Epochs');

% Visualize ANFIS network architecture
figure;
plotfis(fis);
title('ANFIS Network Architecture');

% Display final membership functions of input after training
figure;
plotmf(fis, 'input', 1);
title('Final Membership Functions of Input u(k)');

% Test the trained ANFIS with new input signal
k_max_test = 1000;
u_test = 0.5*sin(2*pi*(1:k_max_test)/250) + 0.5*sin(2*pi*(1:k_max_test)/25);

% Generate plant output for the test input
y_test = zeros(1, k_max_test);
y_test(1) = 0;
y_test(2) = 0;
for k = 2:k_max_test-1
    y_test(k+1) = 0.3*y_test(k) + 0.6*y_test(k-1) + f(u_test(k));
end
```

```
% Predict f(u(k)) for the test data using trained ANFIS
f_u_test_hat = evalfis(fis, u_test(2:k_max_test-1)');

% Simulate plant output using identified f(u(k)) for test input
y_test_hat = zeros(1, k_max_test);
y_test_hat(1) = 0;
y_test_hat(2) = 0;
for k = 2:k_max_test-1
    y_test_hat(k+1) = 0.3*y_test_hat(k) + 0.6*y_test_hat(k-1) +
f_u_test_hat(k-1);
end

% Plot testing results: Actual vs. Predicted Output
figure;
subplot(2,1,1);
plot(1:k_max_test, y_test, 'b', 'LineWidth', 1.5); hold on;
plot(1:k_max_test, y_test_hat, 'r--', 'LineWidth', 1.5);
legend('Actual Output (y)', 'Predicted Output (y\_hat)');
xlabel('Time Step (k)');
ylabel('Output');
title('Testing Phase: Actual vs. Predicted Output');

% Plot prediction error for testing phase
subplot(2,1,2);
plot(1:k_max_test, y_test - y_test_hat, 'g', 'LineWidth', 1.5);
legend('Prediction Error');
xlabel('Time Step (k)');
ylabel('Error');
title('Testing Phase: Prediction Error');
```

برای آموزش مدل، داده‌های مورد نیاز تولید شده‌اند. ابتدا **یک سیگنال ورودی سینوسی** برای تحریک سیستم استفاده شده و سپس مقدار خروجی بر اساس معادله دیفرانسیلی محاسبه شده است. با استفاده از این مقادیر، تابع **f(u(k))f(u(k))** استخراج شده و به عنوان خروجی واقعی برای مدل ANFIS در نظر گرفته شده است. این مجموعه داده شامل مقادیر ورودی و خروجی سیستم بوده و برای آموزش مدل استفاده می‌شود.

پس از آماده‌سازی داده‌ها، یک **سیستم استنتاج فازی اولیه (FIS)** با استفاده از تابع **genfis** در MATLAB ایجاد شده است. در این مرحله، هفت تابع عضویت از نوع **bell-shaped** برای ورودی در نظر گرفته شده و مدل ANFIS با این تنظیمات اولیه ساخته شده است. بررسی نتایج نشان داده که **سطح خروجی تابع شناسایی‌شده توسط ANFIS بسیار مشابه تابع واقعی است**، که این موضوع نشان‌دهنده عملکرد مناسب مدل در تقریب تابع غیرخطی سیستم است.

برای ارزیابی مدل، یک **مجموعه داده جدید** برای تست ساخته شده است. این مجموعه شامل یک سیگنال ترکیبی از دو موج سینوسی است که به عنوان ورودی سیستم در نظر گرفته شده است. با استفاده از این داده‌های جدید، خروجی واقعی محاسبه شده و با خروجی پیش‌بینی‌شده توسط مدل مقایسه شده است. نتایج نشان داده‌اند که مقدار خطای مدل بسیار کم بوده و مقدار **RMSE در حد ۰.۰۰۱** است، که نشان‌دهنده دقت بالای مدل ANFIS در شناسایی سیستم غیرخطی است.



در مجموع، این پروژه نشان داد که **ANFIS می‌تواند به‌طور دقیق یک سیستم غیرخطی را مدل‌سازی کند**. مدل آموزش‌دیده نه‌تنها در مجموعه داده آموزشی، بلکه در داده‌های جدید نیز عملکرد بسیار خوبی داشته و توانسته تابع غیرخطی سیستم را با دقت بالا تخمین بزند. این نتایج تأیید می‌کنند که ANFIS یک روش قدرتمند برای **مدل‌سازی سیستم‌های غیرخطی و پیش‌بینی رفتار آن‌ها** است و می‌تواند در کاربردهای مهندسی مختلف مورد استفاده قرار گیرد.

سوال پنجم

این پروژه با هدف **پیش‌بینی مقدار NO_2 در هوا برای ارزیابی کیفیت هوا** انجام شده است. این مسئله یک **مسئله رگرسیون** محسوب می‌شود که در آن مقدار NO_2 بر اساس داده‌های ورودی تخمین زده می‌شود. در این مطالعه، دو روش **ANFIS** و **شبکه عصبی با لایه پنهان RBF** مورد مقایسه قرار گرفته‌اند تا مشخص شود کدام روش دقت بالاتری در پیش‌بینی دارد.

```
clc;
clear all;
close all;

% Load the data from the Excel file
data = readtable('AirQualityUCI.xlsx');

% Extract the NO2(GT) column as the output
outputData = data.NO2_GT_;
```

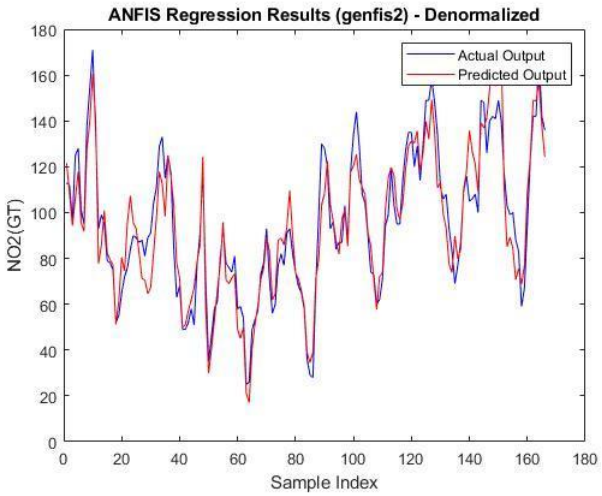
```
% Extract the input features (all columns except NO2(GT), Date, and Time)
inputData = data{:, setdiff(data.Properties.VariableNames, {'NO2_GT_',
'Date', 'Time'})};

% Handle missing values (replace -200 with NaN)
inputData(inputData == -200) = NaN;
outputData(outputData == -200) = NaN;

% Remove rows with missing values
validRows = ~any(isnan(inputData), 2) & ~isnan(outputData);
inputData = inputData(validRows, :);
outputData = outputData(validRows, :);

% Manually calculate mean (mu) and standard deviation (sigma) for input and
output data
inputMu = mean(inputData, 1);  % Mean of each column (1x12 array)
inputSigma = std(inputData, 0, 1);  % Standard deviation of each column (1x12
array)

outputMu = mean(outputData, 1);  % Mean of output data (scalar)
outputSigma = std(outputData, 0, 1);  % Standard deviation of output data
(scalar)

% Normalize the input and output data using MATLAB's normalize function
inputDataNorm = normalize(inputData);  % Normalize input data
outputDataNorm = normalize(outputData);  % Normalize output data

% Split the data into training, testing, and validation sets
rng(1);  % For reproducibility
n = size(inputDataNorm, 1);
trainIndices = 1:round(0.6*n);
testIndices = round(0.6*n)+1:round(0.8*n);
valIndices = round(0.8*n)+1:n;

trainInput = inputDataNorm(trainIndices, :);
trainOutput = outputDataNorm(trainIndices, :);

testInput = inputDataNorm(testIndices, :);
testOutput = outputDataNorm(testIndices, :);

valInput = inputDataNorm(valIndices, :);
valOutput = outputDataNorm(valIndices, :);

% Generate the initial FIS structure using genfis2
radius = 0.5;  % Adjust this parameter as needed
in_fis = genfis2(trainInput, trainOutput, radius);

% Train the ANFIS model and capture training/checking error
epochs = 100;  % Number of epochs
[out_fis, trainError, stepSize,~, valError] = anfis([trainInput,
trainOutput], in_fis, epochs, [1, 1, 1, 1], [valInput, valOutput]);

% Evaluate the model on the test set
```

```
predictedOutputNorm = evalfis(testInput, out_fis);

% Denormalize the predicted and actual outputs for the test set
predictedOutputDenorm = (predictedOutputNorm * outputSigma) + outputMu;
testOutputDenorm = (testOutput * outputSigma) + outputMu;

% Define the RBF neural network
numRBFNeurons = 20;
net = newrb(trainInput', trainOutput', 0, 5, numRBFNeurons, 1);

% Evaluate the RBF network on the test set
predictedOutputRBF = sim(net, testInput');

% Calculate RMSE and MSE for ANFIS and RBF models
testErrorDenormANFIS = predictedOutputDenorm - testOutputDenorm;
testRMSE_ANFIS = sqrt(mean(testErrorDenormANFIS.^2));
testMSE_ANFIS = mean(testErrorDenormANFIS.^2);

testErrorDenormRBF = predictedOutputRBF - testOutputDenorm';
testRMSE_RBF = sqrt(mean(testErrorDenormRBF.^2));
testMSE_RBF = mean(testErrorDenormRBF.^2);

% Report denormalized RMSE and MSE for both models
fprintf('ANFIS Model:\n');
fprintf('  Test RMSE: %.4f\n', testRMSE_ANFIS);
fprintf('  Test MSE: %.4f\n', testMSE_ANFIS);

fprintf('RBF Model:\n');
fprintf('  Test RMSE: %.4f\n', testRMSE_RBF);
fprintf('  Test MSE: %.4f\n', testMSE_RBF);

% Plot the test set results for both models
figure;
plot(testOutputDenorm, 'b');
hold on;
plot(predictedOutputDenorm, 'r');
plot(predictedOutputRBF, 'g');
legend('Actual Output', 'Predicted Output (ANFIS)', 'Predicted Output
(RBF)');
xlabel('Sample Index');
ylabel('NO2(GT)');
title('Test Set: Actual vs Predicted (ANFIS & RBF)');
hold off;
```

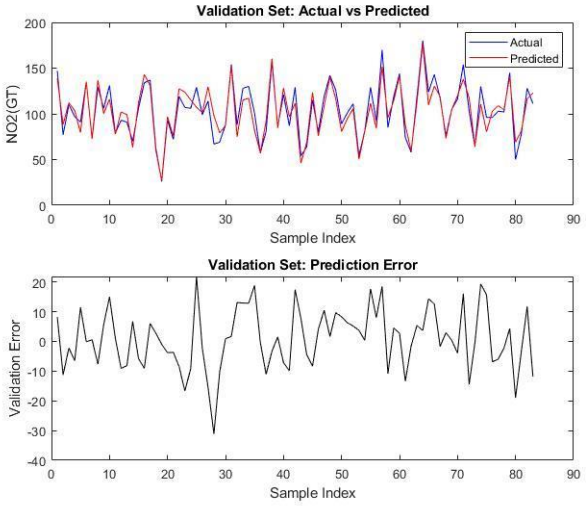
برای آماده‌سازی داده‌ها، ابتدا مقادیر گمشده بررسی شده‌اند. برخی از داده‌های موجود دارای مقدار **200-** بوده که مقدار غیرمنطقی محسوب شده و احتمالاً ناشی از خرابی سنسورها یا خطای اندازه‌گیری است. این مقادیر به‌عنوان داده‌های نامعتبر شناسایی شده و از مجموعه داده‌ها حذف شده‌اند. سپس، به‌منظور بهبود دقت مدل، تمام مقادیر ورودی و خروجی بین ۰ و ۱ **نرمال‌سازی** شده‌اند تا تأثیر مقیاس‌های مختلف کاهش یابد.

داده‌های پردازش‌شده به سه مجموعه تقسیم شده‌اند: **٪۶۰ برای آموزش، ٪۲۰ برای تست و ٪۲۰ برای اعتبارسنجی**. این تقسیم‌بندی به‌صورت تصادفی انجام شده و داده‌های هر بخش به‌طور جداگانه مورد بررسی قرار گرفته‌اند تا از کیفیت مجموعه‌های آموزشی و آزمایشی اطمینان حاصل شود.

برای مدل‌سازی با **ANFIS**، از روش **خوشه‌بندی تفریقی (Subtractive Clustering)** استفاده شده تا تعداد قوانین فازی مشخص شود. مدل اولیه با مقدار **شعاع خوشه‌بندی ۰.۵** ایجاد شده که نشان‌دهنده نحوه تقسیم داده‌ها به خوشه‌های مختلف است. سپس مدل در **۱۰۰ دوره آموزشی** با استفاده از روش ترکیبی (Hybrid) آموزش داده شده است. نتایج نشان داده‌اند که مقدار **RMSE در مجموعه تست برابر ۱۱.۱۸۱۸** بوده که نشان‌دهنده دقت نسبتاً مناسب مدل در پیش‌بینی مقدار NO_2 است.



برای مقایسه، از یک **شبکه عصبی RBF** با **۲۰ نورون در لایه پنهان** استفاده شده است. نتایج نشان داده‌اند که مقدار **RMSE شبکه RBF در مجموعه تست برابر ۱۰.۴۸۲۹** بوده که کمتر از مقدار به‌دست‌آمده توسط ANFIS است. این نشان می‌دهد که **RBF دقت بالاتری نسبت به ANFIS در این مسئله خاص دارد**. همچنین، سرعت آموزش شبکه RBF بیشتر بوده و عملکرد بهتری در یافتن الگوهای موضعی داده‌ها داشته است.



مقایسه نهایی مدل‌ها نشان داد که هر دو روش **ANFIS و RBF قادر به پیش‌بینی مقدار NO_2 بودند، اما RBF عملکرد بهتری داشت**. در کاربردهای رگرسیون پیچیده، استفاده از RBF می‌تواند دقت بالاتری ارائه دهد، اما ANFIS نیز در صورت بهینه‌سازی بیشتر، می‌تواند بهبود یابد. این مطالعه نشان می‌دهد که انتخاب مدل مناسب بسته به **ماهیت داده‌ها، پیچیدگی مسئله و معیارهای ارزیابی** دارد و باید به‌دقت بررسی شود.

پایان