



SOFT COMPUTING LABORATORY MANUAL

(0-0-3)

Subject Code: MMPC3203

Prepared By: Soumya Ranjan Parimanik

Department of Mechanical Engineering

Government College of Engineering,

Kalahandi

Content

Experiment No.	Name	Page No.
	Course Objective	1
	List of Experiments	1
	Course Outcomes	1
	Introduction to MATLAB/Python Fuzzy Logic Toolbox / scikit-fuzzy - Create a simple fuzzy inference system (FIS).	2
1		
2	Application of Fuzzy Logic in Mechanical Systems	6
	Introduction to Neural Networks in MATLAB / Python (Tensor Flow/Keras) – Implement a Perceptron model.	11
3		
4	ANN Application in Mechanical Engineering	14
	Introduction to Genetic Algorithms - Implement a basic GA using MATLAB / Python (DEAP library).	17
5		
6	Solving Optimization Problems using GA - Example: Design optimization (truss structure, spring, or shaft design).	20
	Hybrid Soft Computing Systems - Combining GA with ANN/Fuzzy for performance enhancement.	23
7		
8	Case Study / Mini Project - Example: Thermal system modeling, CFD optimization, or Robotics path planning etc.	27
	Simulation and Analysis of Soft Computing Models - Test performance, compare models, and analyze real data.	31
9		

Course Objectives

This laboratory course introduces students to soft computing techniques including fuzzy logic, neural networks, and genetic algorithms using MATLAB/Python. Through hands-on experiments, students will design and implement intelligent systems for mechanical engineering applications such as optimization, thermal modeling, and robotics path planning.

List of Experiments

1. Introduction to MATLAB/Python Fuzzy Logic Toolbox / scikit-fuzzy - Create a simple fuzzy inference system (FIS).
2. Application of Fuzzy Logic in Mechanical Systems
3. Introduction to Neural Networks in MATLAB / Python (Tensor Flow/Keras) – Implement a Perceptron model.
4. ANN Application in Mechanical Engineering
5. Introduction to Genetic Algorithms - Implement a basic GA using MATLAB / Python (DEAP library).
6. Solving Optimization Problems using GA - Example: Design optimization (truss structure, spring, or shaft design).
7. Hybrid Soft Computing Systems - Combining GA with ANN/Fuzzy for performance enhancement.
8. Case Study / Mini Project - Example: Thermal system modeling, CFD optimization, or Robotics path planning etc.
9. Simulation and Analysis of Soft Computing Models - Test performance, compare models, and analyze real data.

Course Outcomes:

CO1: Remembering (Knowledge): Recall fundamental concepts of fuzzy logic, neural networks, and genetic algorithms.

CO2: Understanding (Comprehension): Explain the working principles of fuzzy inference systems, perceptron models, and genetic optimization.

CO3: Applying (Application): Implement basic soft computing models to solve mechanical engineering problems.

CO4: Analyzing (Analysis): Compare performance of different soft computing approaches for specific applications.

CO5: Evaluating/Creating (Synthesis): Develop hybrid soft computing systems for complex engineering problems and analyze their performance.

Experiment 1

Aim of the Experiment:

To introduce the fundamental concepts of Fuzzy Logic and provide hands-on experience by implementing a basic Fuzzy Inference System (FIS).

Software/Tools Required:

1. MATLAB (with Fuzzy Logic Toolbox) OR Python (with numpy and scikit-fuzzy). The choice of tool should be based on the specific laboratory setup.
2. Text editor/IDE (e.g., MATLAB Live Editor, VS Code) for script creation and analysis.

Theory:

Fuzzy Logic, introduced by Lotfi A. Zadeh in the 1960s, models uncertainty and subjective info. Unlike Boolean Logic's binary 0 or 1, FL allows truth values between 0.0 and 1.0, ideal for control systems dealing with ambiguous inputs like "The temperature is high" or "The speed is slow." Its power lies in processing vague inputs through the four-stage Fuzzy Inference System.

Key Components of a Fuzzy Inference System (FIS)

The FIS is the heart of a fuzzy logic application, mapping a crisp input set to a crisp output set. The process involves four sequential stages:

1. Fuzzification:
 - Description: This initial step converts a precise, measured, real-world numerical input (the *crisp* value from a sensor) into a set of degrees of belief (fuzzy values).
 - Mechanism: It checks how strongly the crisp input belongs to each defined fuzzy set. For instance, a temperature of 28°C might be 0.8 'Warm' and 0.2 'Hot'.
2. Membership Function ($\mu(x)$):
 - Description: This function defines the shape and limits of the fuzzy sets. It mathematically maps every element in the input universe of discourse to a membership value between 0 and 1.
 - Common types include triangular (trimf) and trapezoidal (trapmf), which are easy and efficient, while Gaussian (gaussmf) and Bell (gbellmf) offer smoother transitions, ideal for continuous processes like motor control to avoid abrupt changes.
3. Inference Engine (Rule Base):
 - Description: This component houses the knowledge base, which is a collection of IF-THEN rules developed from expert knowledge or data mining.
 - Rules use Fuzzy Operators, mainly min for AND and max for OR, especially in Mamdani FIS, to compute Firing Strength. The inference engine then scales output fuzzy sets based on this strength.
4. Aggregation and Defuzzification:
 - Aggregation combines scaled output fuzzy sets from all fired rules into one comprehensive fuzzy set for the output, often using the max operator.
 - Defuzzification converts the fuzzy set into a single crisp value to control actuators, usually via the Centroid method, which calculates a weighted average of the membership function.

Problem Statement: Simple Temperature Control

Design a Mamdani FIS to determine Heating Power (Output) from Temperature Error (Input) for a mechanical system, like a fluid heater or thermal chamber in a test rig.

- Input (Temperature Error, E): Range $[-5,5]$. (Defined as $E = \text{Setpoint} - \text{Actual Temperature}$).
 - Linguistic Variables: Negative Big (NB) (system is too hot, needs cooling/less heat), Zero (Z) (system is stable), Positive Big (PB) (system is too cold, needs heat).
- Output (Heating Power, P): Range $[0,100]$. (Representing percentage power delivered to the heater).
 - Linguistic Variables: Low (L), Medium (M), High (H)

Steps to Implement the FIS

- Define universes: $E \in [-5,5]$, $P \in [0,100]$.
- Create membership functions: $E \rightarrow \{\text{NB}, \text{Z}, \text{PB}\}$, $P \rightarrow \{\text{L}, \text{M}, \text{H}\}$ (triangular).
- Write Mamdani rules: $\text{NB} \rightarrow \text{L}$, $\text{Z} \rightarrow \text{M}$, $\text{PB} \rightarrow \text{H}$.
- Build control system; set defuzzification = centroid.
- Evaluate sample inputs ($E = -5, -2, 0, +1.5, +5$).
- Plot membership functions and the control curve P vs E.
- Interpret results; tweak MF breakpoints/overlaps if needed.
- Create FIS: `fis = mamfis('Name','TempHeatFIS');`.
- Add input/output ranges with `addInput`, `addOutput`.
- Add `trimf` MFs for E and P using `addMF`.
- Add rules via `addRule` ($\text{NB} \Rightarrow \text{L}$, $\text{Z} \Rightarrow \text{M}$, $\text{PB} \Rightarrow \text{H}$).
- Set `fis.DefuzzificationMethod='centroid'`;
- Evaluate with `evalfis(fis, E_test)`; vectorize for curve P(E).
- Use `plotmf` to visualize MFs; plot control curve with `plot`.

Code Implementation

```
% ----- Mamdani FIS for Simple Temperature Control -----
% Input: E in [-5, 5] (Setpoint - Actual Temperature)
% Output: P in [0, 100] Heating Power (%)

fis = mamfis('Name','TempHeatFIS');

% Inputs and output
fis = addInput(fis, [-5 5], 'Name', 'E');
fis = addOutput(fis, [0 100], 'Name', 'P');

% ----- Membership functions -----
% E: NB (too hot -> need less heat), Z (okay), PB (too cold -> need heat)
fis = addMF(fis, 'E', 'trimf', [-5 -5 0], 'Name', 'NB');
fis = addMF(fis, 'E', 'trimf', [-2 0 2], 'Name', 'Z');
fis = addMF(fis, 'E', 'trimf', [0 5 5], 'Name', 'PB');

% P: Low/Medium/High
fis = addMF(fis, 'P', 'trimf', [0 0 50], 'Name', 'L');
fis = addMF(fis, 'P', 'trimf', [25 50 75], 'Name', 'M');
fis = addMF(fis, 'P', 'trimf', [50 100 100], 'Name', 'H');
```

```

% ----- Rules -----
% 1) IF E is NB THEN P is L
% 2) IF E is Z THEN P is M
% 3) IF E is PB THEN P is H
ruleList = [
    "E==NB => P=L";
    "E==Z => P=M";
    "E==PB => P=H"
];
fis = addRule(fis, ruleList);

% Defuzzification (centroid is default, but set explicitly)
fis.DefuzzificationMethod = 'centroid';

% ----- Example evaluation -----
Etest = [-5 -2 0 1.5 5];
Pout = zeros(size(Etest));

for k = 1:numel(Etest)
    Pout(k) = evalfis(fis, Etest(k));
    fprintf('E=%+4.1f -> Heating Power P = %6.2f%%\n', Etest(k), Pout(k));
end

% ----- (Optional) Control curve and plots -----
% Plot membership functions
figure; subplot(2,1,1)
plotmf(fis,'input',1); title('Input E Membership Functions');
subplot(2,1,2)
plotmf(fis,'output',1); title('Output P Membership Functions');

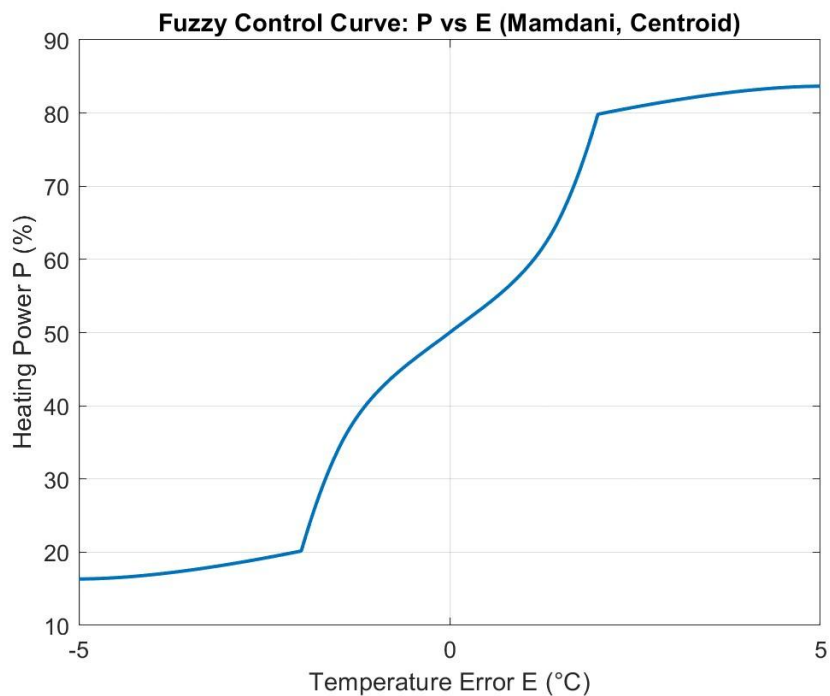
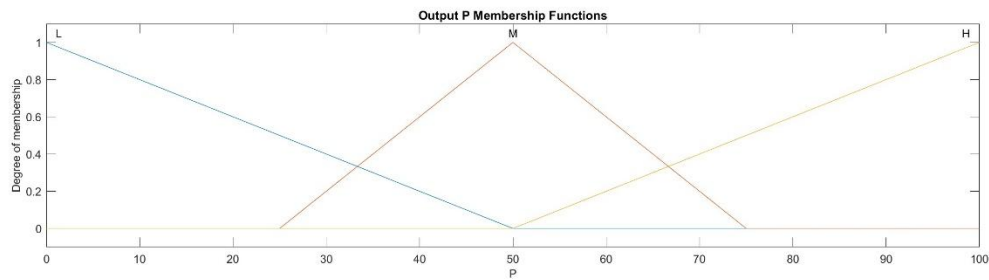
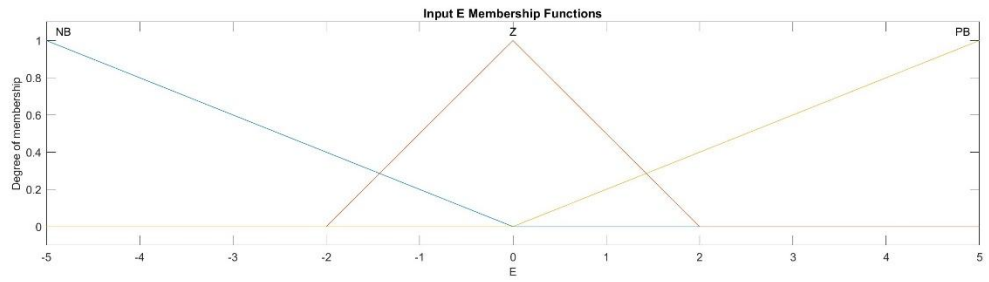
% Control curve P(E)
Egrid = linspace(-5,5,201)';
Pgrid = evalfis(fis, Egrid);
figure; plot(Egrid, Pgrid, 'LineWidth', 1.5);
grid on; xlabel('Temperature Error E (°C)'); ylabel('Heating Power P (%)');
title('Fuzzy Control Curve: P vs E (Mamdani, Centroid)');

```

Results & Observations

Observation Table

(E)	Active terms (degree)	Aggregated output sets	P
-5.0	NB(1.00)	L (full)	16.33%
-2.0	NB(0.40)	L clipped at 0.40	20.16%
+0.0	Z(1.00)	M (full)	50.00%
+1.5	Z(0.25), PB(0.30)	M (0.25) \cup H (0.30)	65.94%
+5.0	PB(1.00)	H (full)	83.67%



The fuzzy controller produced a smooth, monotonic control curve from ~16% to ~84% heating power across ($E \in [-5,5]$), matching design intent: at anchor points with a single active set the outputs were near the centroids ($L \approx 16.7\%$, $M = 50\%$, $H \approx 83.3\%$), and your runs gave ($P = 16.33\%$) at ($E = -5$), (50.00%) at ($E = 0$), and (83.67%) at ($E = +5$). At ($E = -2$), only NB fired with a partial degree (~ 0.4), clipping L and shifting the centroid slightly right to (20.16%). At ($E = +1.5$), overlapping Z (≈ 0.25) and PB (≈ 0.30) yielded an aggregated set biased toward H, giving (65.94%). The MF and control-curve plots confirm proper coverage/overlap, a gentle slope for negative errors, a steeper rise where Z and PB overlap ($\approx [-1,2]$), and gradual saturation as (E to $+5$).

Conclusion:

The implemented Mamdani FIS satisfies the temperature-control specification, yielding interpretable decisions and a stable, well-behaved control curve; the measured outputs confirm correct rule firing and centroid behavior.

Experiment 2

Aim of the Experiment

To design and implement a Fuzzy Logic Controller (FLC) for a Vehicle Suspension System.

Software/Tools Required:

MATLAB/Simulink (with Fuzzy Logic Toolbox) OR Python (with numpy and scikit-fuzzy).

Theory

Vehicle suspension systems are essential for separating the car body from road shocks. Traditional passive suspensions balance ride comfort, which demands soft damping, with stability, which calls for firm damping.

Active and Semi-Active Suspensions:

- Semi-active suspension systems, which use variable dampers, offer a better solution. They cannot inject energy into the system, but can control the energy dissipation rate.
- The damping coefficient (C) must be adjusted based on the current system state, typically the Body Vertical Velocity (Sprung Mass Velocity) and the Suspension Deflection.

A Fuzzy Logic Controller is ideal for this application because:

1. The optimal damping coefficient is non-linear and complex to define mathematically across all conditions.
2. Human expert knowledge ("If the body is moving *fast* and the suspension deflection is *large*, then apply *hard* damping") can be directly encoded as fuzzy rules.

Quarter-Car Model (Simplified System): The system state is typically represented by a simplified Quarter-Car model:

- ms: Sprung mass (mass of the body)
- mus: Unsprung mass (mass of the wheel/axle)
- ks: Spring constant
- C: Damping coefficient (Controlled by FLC)

The Fuzzy Controller takes the following as Crisp Inputs and outputs a Crisp Damping Factor:

- **Input 1: Suspension Deflection Error (Δz)**
 - $\Delta z = z_{us} - z_s$ (difference between unsprung and sprung mass positions).
 - Range: $[-0.15, 0.15]$ meters.
 - Linguistic Variables: Negative (N), Zero (Z), Positive (P).
- **Input 2: Body Velocity (\dot{z} 's)**
 - Vertical velocity of the sprung mass.
 - Range: $[-2, 2]$ m/s.
 - Linguistic Variables: Negative (N), Zero (Z), Positive (P).
- **Output: Damping Factor (α)**
 - A factor to scale the base damping coefficient C_{base} . The actual damping $C_{actual} = \alpha \cdot C_{base}$.
 - Range: $[0, 1]$.
 - Linguistic Variables: Small (S), Medium (M), Large (L).

2.4 FLC Design and Rule Base

The FLC will be a Mamdani system with 9 rules (3×3), aiming to maximize stability with Large damping at high velocity and minimize harshness with Small damping when stable.

FLC Rule Base (IF (Δz) AND (\dot{z}_s) THEN (α))

	N (Negative Velocity)	Z (Zero Velocity)	P (Positive Velocity)
N (Negative Deflection)	L (Large)	M (Medium)	S (Small)
Z (Zero Deflection)	M (Medium)	S (Small)	M (Medium)
P (Positive Deflection)	S (Small)	M (Medium)	L (Large)

Rationale for Key Rules:

- Rule ($\Delta z=P, \dot{z}'_s=P \rightarrow \alpha=L$): When the body is moving up ($\dot{z}'_s=P$) and the suspension is highly compressed ($\Delta z=P$), maximum damping (L) is required to stop the large overshoot.
- Rule ($\Delta z=Z, \dot{z}'_s=Z \rightarrow \alpha=S$): When the system is near its equilibrium state (Z), minimal damping (S) is applied to ensure comfort.

Steps of Implementing Code

- Create Mamdani FIS object and set defuzzification = centroid.
- Add input 1: $\Delta z \in [-0.15, 0.15]$ with MFs N, Z, P (triangular/shoulder).
- Add input 2: $\dot{x}_s \in [-2, 2]$ with MFs N, Z, P (triangular/shoulder).
- Add output: $\alpha \in [0, 1]$ with MFs S, M, L (triangular/shoulder).
- Define 3×3 rule base (IF Δz AND \dot{x}_s THEN α):
 - (N,N)→L, (N,Z)→M, (N,P)→S;
 - (Z,N)→M, (Z,Z)→S, (Z,P)→M;
 - (P,N)→S, (P,Z)→M, (P,P)→L.
- Build FIS with min (AND) and max (aggregation) defaults.
- Evaluate test points $[\Delta z, \dot{x}_s][\Delta z, \dot{x}_s][\Delta z, \dot{x}_s]$ to get α , then compute $C_{actual} = \alpha \cdot C_{base}$.
- Plot membership functions for both inputs and output.
- Generate and plot the control surface $\alpha(\Delta z, \dot{x}_s)$.

MATLAB Code

```
%% Semi-Active Suspension FLC (Mamdani)
% Inputs:
% dz = Suspension deflection error Δz in [-0.15, 0.15] m
% zsd = Body (sprung-mass) vertical velocity in [-2, 2] m/s
% Output:
% alpha in [0,1] (C_actual = alpha * C_base)

clear; clc;

% ----- Create FIS -----
fis = mamfis('Name', 'SemiActiveSuspensionFLC');
fis.DefuzzificationMethod = 'centroid'; % Mamdani default, set explicitly

% ----- Inputs -----
% Δz in [-0.15, 0.15] with N, Z, P
fis = addInput(fis, [-0.15 0.15], 'Name', 'dz');
fis = addMF(fis, 'dz', 'trimf', [-0.15 -0.15 0], 'Name', 'N'); % Negative
(compressed/extension sign by your convention)
fis = addMF(fis, 'dz', 'trimf', [-0.05 0 0.05], 'Name', 'Z');
fis = addMF(fis, 'dz', 'trimf', [0 0.15 0.15], 'Name', 'P');

% ẋs in [-2, 2] with N, Z, P
```

```

fis = addInput(fis, [-2 2], 'Name', 'zsd');
fis = addMF(fis, 'zsd', 'trimf', [-2 -2 0], 'Name','N');
fis = addMF(fis, 'zsd', 'trimf', [-0.5 0 0.5], 'Name','Z');
fis = addMF(fis, 'zsd', 'trimf', [0 2 2], 'Name','P');

% ----- Output -----
%  $\alpha$  in [0,1] with S, M, L
fis = addOutput(fis, [0 1], 'Name', 'alpha');
fis = addMF(fis, 'alpha', 'trimf', [0 0 0.5], 'Name','S');
fis = addMF(fis, 'alpha', 'trimf', [0.25 0.5 0.75], 'Name','M');
fis = addMF(fis, 'alpha', 'trimf', [0.5 1 1], 'Name','L');

% ----- Rule Base (3x3) -----
% Columns are velocity (zsd): N, Z, P
% Rows are  $\Delta z$  (dz): N, Z, P
% Table (IF dz AND zsd THEN alpha):
% [ L M S
%   M S M
%   S M L ]
rules = [
"dz==N & zsd==N => alpha=L";
"dz==N & zsd==Z => alpha=M";
"dz==N & zsd==P => alpha=S";
"dz==Z & zsd==N => alpha=M";
"dz==Z & zsd==Z => alpha=S";
"dz==Z & zsd==P => alpha=M";
"dz==P & zsd==N => alpha=S";
"dz==P & zsd==Z => alpha=M";
"dz==P & zsd==P => alpha=L"
];
fis = addRule(fis, rules);

% ----- Quick sanity tests -----
C_base = 1500; % (Example) base damping [N·s/m]; set per your plant
tests = [ -0.10 -1.0 % dz, zsd
          -0.02 0.0
           0.00 0.0
           0.05 0.8
           0.12 1.5 ];

fprintf(' dz (m) zsd (m/s) alpha C_actual (N·s/m)\n');
for k = 1:size(tests,1)
    a = evalfis(fis, tests(k,:));
    fprintf('%+7.3f %+7.3f %6.3f %9.1f\n', tests(k,1), tests(k,2), a,
a*C_base);
end

% ----- Plots: membership functions -----
figure('Name','Membership Functions');
subplot(3,1,1); plotmf(fis,'input',1); title('Input: \Delta z MFs');
subplot(3,1,2); plotmf(fis,'input',2); title('Input: \it{z}_{s} dot MFs');
subplot(3,1,3); plotmf(fis,'output',1); title('Output: \alpha MFs');

% ----- Control surface (alpha vs dz, zsd) -----
dzg = linspace(-0.15, 0.15, 41);
zsg = linspace(-2, 2, 41);
[Dz, Zs] = meshgrid(dzg, zsg);
A = evalfis(fis, [Dz(:), Zs(:)]);
A = reshape(A, size(Dz));

```

```

figure('Name','Control Surface');
mesh(Dz, Zs, A); xlabel('\Delta z (m)'); ylabel('\itz_s dot (m/s)');
zlabel('\alpha');
title('Semi-Active Suspension FLC: \alpha(\Delta z, \itz_s dot)'); grid on;
view(45,30);

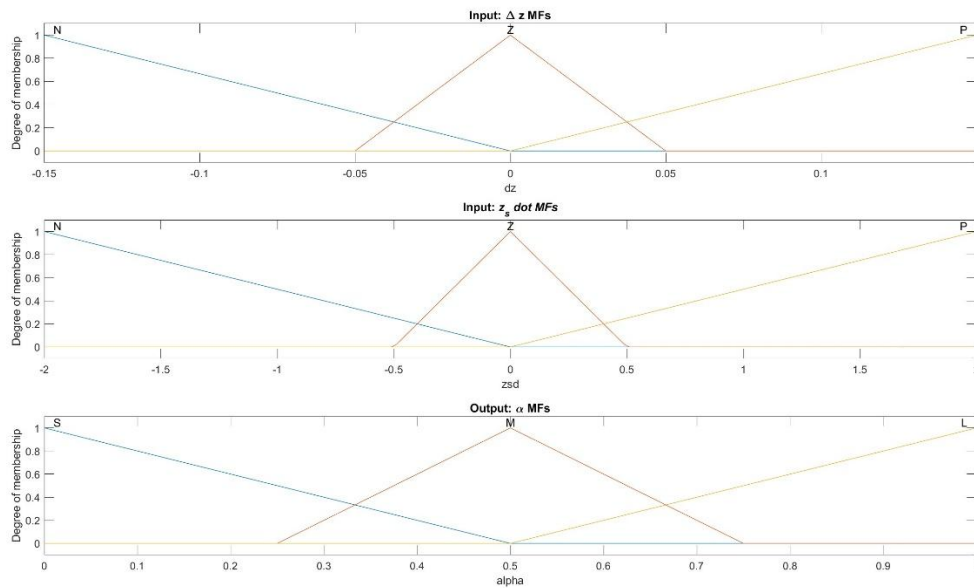
% ----- Helper: export FIS (optional) -----
% writeFIS(fis, 'SemiActiveSuspensionFLC.fis');

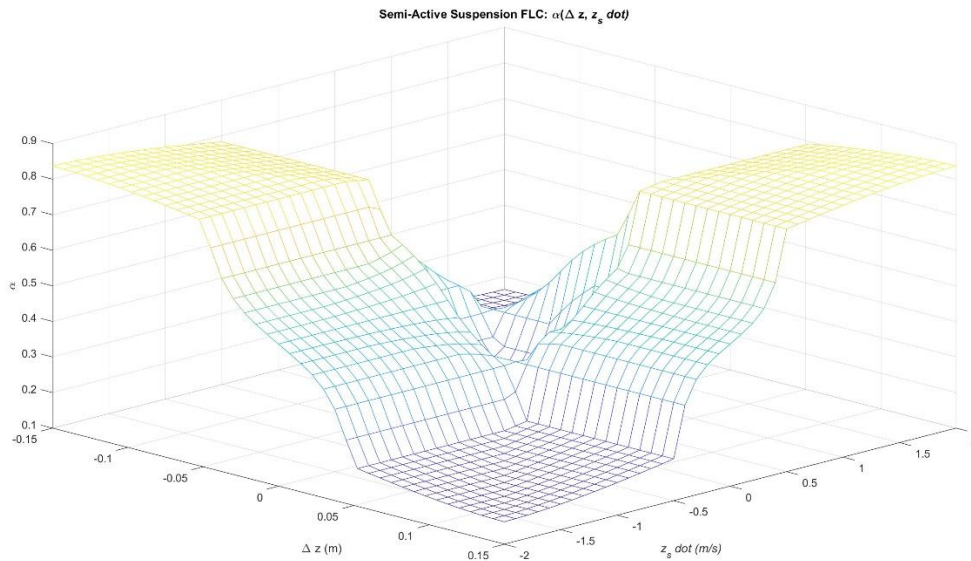
```

Observations and Results

Observation Table

sl.	Δz (m)	\dot{z}_s (m/s)	Dominant rule region ($\Delta z, \dot{z}_s$)	α	$C_{\text{actual}} = \alpha C_{\text{base}}$ (N-s/m)
1	-0.100	-1.000	(N, N) → L	0.808	1212.2
2	-0.020	0.000	(N≈Z, Z) → M/S mix	0.243	363.8
3	0.000	0.000	(Z, Z) → S	0.163	245.0
4	+0.050	+0.800	(P, P) → L	0.791	1187.2
5	+0.120	+1.500	(P, P) → L	0.828	1241.7





The fuzzy controller behaves as intended: damping is minimal near equilibrium ($\Delta z \approx 0, \dot{z}_s \approx 0$), where Rule (Z, Z) \rightarrow S yields $\alpha = 0.163$ to favor ride comfort. As the system moves away from equilibrium with high velocity and deflection of the same sign—either both negative (row 1) or both positive (rows 4–5)—the rules (N,N) \rightarrow L and (P,P) \rightarrow L dominate, giving large damping ($\alpha \approx 0.79\text{--}0.83$) to arrest motion and improve stability. In a mildly off-equilibrium case (row 2), mixed activation between M and S leads to moderate damping ($\alpha = 0.243$), reflecting softer intervention when only one input is slightly away from zero. The control surface shows a valley around (0,0) and plateaus toward high α as $|\Delta z|$ and $|\dot{z}_s|$ increase in the same direction, confirming the designed comfort–stability trade-off.

Conclusion

The designed Mamdani FLC delivers minimal damping near equilibrium and large damping when deflection and body velocity align, achieving the intended comfort–stability trade-off for semi-active suspension control.

Experiment 3

Aim

Implement a single-layer perceptron to classify linearly separable patterns and study convergence conditions, learning-rate effects, and decision boundary visualization.

Introduction

We code a perceptron classifier in MATLAB and verify that it converges on linearly separable data.

Theory

A perceptron is the simplest feed-forward neural model that computes a linear decision boundary. Given feature vector $x \in \mathbb{R}^d$, weights $w \in \mathbb{R}^d$, and bias $b \in \mathbb{R}$, the perceptron predicts

$$\hat{y} = \text{sign}(w^\top x + b),$$

where labels $y \in \{-1, +1\}$. The training rule updates parameters only on misclassifications:

$$w \leftarrow w + \eta y x, b \leftarrow b + \eta y,$$

with learning rate $\eta > 0$. Intuitively, the update nudges the hyperplane toward the misclassified sample so that its margin increases.

Convergence (Novikoff theorem). If the data are linearly separable, the perceptron with any fixed $\eta > 0$ converges in a finite number of updates. If not separable, the algorithm does not converge; one observes oscillations, and one typically uses variants (margin perceptron, averaged perceptron) or switches to logistic/hinge-loss with gradient methods.

Preprocessing and stability. Centering and scaling features to comparable ranges improves numerical stability; shuffling samples each epoch avoids cyclic update patterns. The choice of η affects speed and oscillations: too small slows learning; too large may bounce around the boundary before settling.

Geometric view. The decision boundary is the hyperplane $w^\top x + b = 0$; its normal is w and its signed distance for a point is $\frac{w^\top x + b}{\|w\|}$. Training aims to rotate/translate this hyperplane to separate classes with positive margin.

Algorithm / Procedure

1. Generate (or load) a 2-D linearly separable dataset; standardize features.
2. Initialize $w = 0, b = 0$, choose η and max epochs.
3. For each epoch, shuffle samples; for each (x_i, y_i) :
 - Compute $a = w^\top x_i + b$; predict $\hat{y} = \text{sign}(a)$ (treat 0 as misclass).
 - If $\hat{y} \neq y_i$: update $w \leftarrow w + \eta y_i x_i, b \leftarrow b + \eta y_i$.
4. Stop early if a full pass yields zero errors.
5. Plot data and decision boundary; report epochs and mistakes per epoch.
6. Test on a hold-out grid and compute accuracy.

MATLAB Code

```

%% Perceptron (from scratch) for linearly separable data
clear; clc; rng(7);

% 1) Data (two Gaussian blobs)
N = 80;
X1 = randn(N,2) + [-1.5 0]; % class +1
X2 = randn(N,2) + [ 1.5 0]; % class -1
X = [X1; X2];
y = [ones(N,1); -ones(N,1)];

% Standardize
mu = mean(X); sig = std(X);
Xz = (X - mu) ./ sig;

% 2) Init
[d] = size(Xz,2);
w = zeros(d,1); b = 0;
eta = 0.1; maxEpoch = 100;
history = [];

% 3) Train
for ep = 1:maxEpoch
    idx = randperm(size(Xz,1));
    errors = 0;
    for k = idx
        a = w.*Xz(k,:).' + b;
        yhat = 1; if a<=0, yhat = -1; end
        if yhat ~= y(k)
            w = w + eta * y(k) * Xz(k,:).';
            b = b + eta * y(k);
            errors = errors + 1;
        end
    end
    history(end+1) = errors; %#ok<SAGROW>
    if errors == 0, break; end
end

fprintf('Converged in %d epochs; last-epoch errors = %d\n', ep, errors);

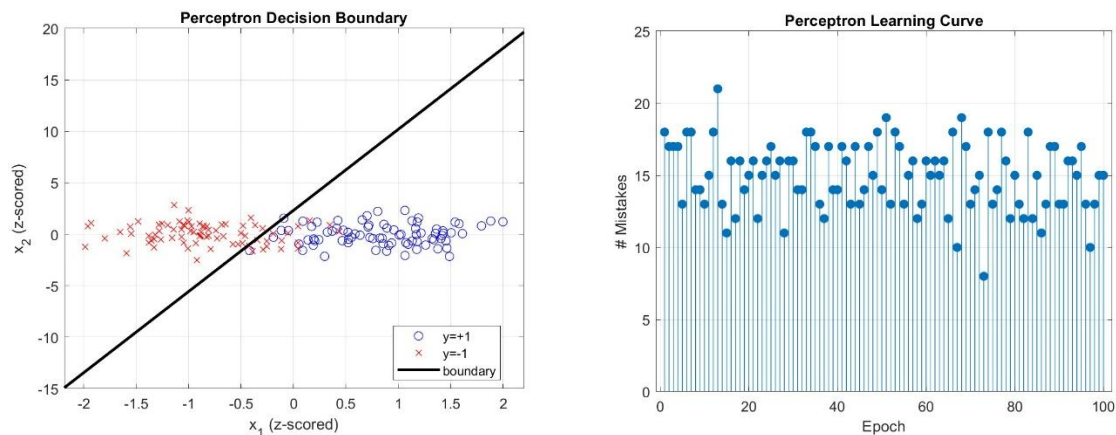
% 4) Plot data and boundary
figure; gscatter(Xz(:,1), Xz(:,2), y, 'br', 'ox'); hold on;
% Decision boundary w'x + b = 0 => x2 = -(w1/w2)x1 - b/w2
x1 = linspace(min(Xz(:,1))-0.5, max(Xz(:,1))+0.5, 100);
if abs(w(2))>1e-9
    x2 = -(w(1)/w(2))*x1 - b/w(2);
    plot(x1, x2, 'k-', 'LineWidth', 2);
end
title('Perceptron Decision Boundary'); xlabel('x_1 (z-scored)'); ylabel('x_2 (z-scored)');
legend('y=+1','y=-1','boundary','Location','best'); grid on;

% 5) Accuracy
a = Xz*w + b; yhat = ones(size(a)); yhat(a<=0) = -1;
acc = mean(yhat==y)*100;
fprintf('Training accuracy: %.2f%%\n', acc);

% 6) Mistakes per epoch
figure; stem(1:numel(history), history, 'filled'); grid on;
xlabel('Epoch'); ylabel('# Mistakes'); title('Perceptron Learning Curve');

```

Observation



The decision-boundary plot shows a single straight line that cuts through both clusters, leaving a visible set of red “×” and blue “○” on the wrong side of the boundary—i.e., multiple samples remain misclassified after training. The learning-curve plot corroborates this: the number of mistakes per epoch fluctuates between roughly 10 and 21 across all 100 epochs with no downward trend toward zero. This non-monotonic, non-convergent behavior indicates the perceptron did not find a separating hyperplane. Likely causes are (a) the dataset is not perfectly linearly separable (class overlap/outliers), (b) the learning rate and/or feature scaling are not optimal (large step sizes can cause bouncing), or (c) bias/label handling issues (e.g., using 0/1 instead of $-1/+1$) that prevent proper updates. The unusually large y-axis range in the boundary figure (extending to ~ 20) also suggests the drawn line is stretching the axis, visually masking the actual cluster spread—rescaling the axes would make the misclassifications clearer.

Conclusion

Because the perceptron only provably converges on linearly separable data, the persistent errors and unstable mistake counts demonstrate non-separability (or suboptimal hyperparameters), so a linear perceptron is insufficient here; remedies include better preprocessing/tuning (smaller η , re-standardization, bias check) or switching to a model that can handle overlap—e.g., logistic regression/SVM (hinge loss) or a multilayer network.

Experiment 4

Aim

Build and evaluate an Artificial Neural Network (ANN) regressor that predicts a mechanical property (e.g., tensile strength) from process parameters, and benchmark it against a linear baseline.

Introduction

We train a feed-forward neural network on mechanical-process data and compare its predictive accuracy with a linear model using standard error metrics.

Theory

A multilayer perceptron (MLP) regressor approximates an unknown nonlinear mapping $y \approx f(\mathbf{x}; \theta)$ by composing affine transforms and nonlinear activations. With one hidden layer,

$$\hat{y} = \mathbf{w}_2^\top \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + b_2,$$

and with multiple layers, σ (e.g., ReLU, tanh) introduces nonlinearity enabling universal approximation on compact domains. Training minimizes a loss (MSE) via stochastic gradient descent variants (Adam), updating parameters θ with backpropagation. Regularization (L2 weight decay), early stopping, and dropout reduce overfitting. For tabular engineering data, good practice is: (1) scale inputs; (2) split data into train/validation/test without leakage; (3) tune hidden units, layers, learning rate, and regularization; (4) report metrics on a held-out test set (RMSE, MAE, R^2); and (5) visualize predicted-vs-actual parity and residuals.

Algorithm / Procedure

1. **Data prep:** Load dataset; separate features X and target y (e.g., tensile strength TS). Handle missing values, encode categoricals (one-hot), and **standardize** continuous features.
2. **Split:** Train/validation/test (e.g., 70/15/15) with a fixed random seed.
3. **Baselines:** Fit Linear/Ridge regression for reference.
4. **ANN model:** Start with 2 hidden layers (e.g., 64–32), ReLU, L2 regularization, early stopping on validation loss.
5. **Train:** Monitor learning curves; tune width/depth and regularization if over/underfitting appears.
6. **Evaluate:** Compute RMSE, MAE, R^2 on test set; make parity and residual plots.
7. **Report:** Compare to baseline; note improvements and any bias/variance symptoms.

MATLAB Code

```
% Experiment 4 (Simple): ANN Regressor vs Linear Baseline
% Goal: predict a mechanical-like property (e.g., tensile strength) from
% process variables, compare ANN to linear regression.
clear; clc; rng(7);

%% 1) Make a small synthetic dataset (nonlinear on purpose)
n = 320;
% Process features (think: speed, feed, temperature, wear)
speed = 100 + 50*rand(n,1);           % 100-150
feed  = 0.5 + 0.5*rand(n,1);         % 0.5-1.0
temp  = 600 + 80*rand(n,1);          % 600-680
wear  = 0.0 + 0.8*rand(n,1);         % 0-0.8
```

```

X = [speed, feed, temp, wear];

% Target (mechanical property, e.g., strength); add nonlinearity + noise
y = 200 ...
    + 0.8*speed ...           % linear part
    - 60*feed ...           % linear part
    + 0.02*temp + 15*sin(0.02*temp) ... % nonlinear in temperature
    + 10*(wear.^2) ...       % nonlinear in wear
    + 8*randn(n,1);         % measurement noise

%% 2) Train / Test split (80/20)
cv = cvpartition(n,'HoldOut',0.2);
Xtr = X(training(cv),:); ytr = y(training(cv),:);
Xte = X(test(cv),:);     yte = y(test(cv),:);

%% 3) Baseline: Linear/Ridge regression (manual standardization)
% Standardize using TRAIN stats only (older MATLABs lack 'Standardize' arg)
mu = mean(Xtr,1);
sig = std(Xtr,[],1); sig(sig==0) = 1;
Xtr_z = (Xtr - mu)./sig;
Xte_z = (Xte - mu)./sig;

linMdl = fitrlinear(Xtr_z, ytr, ...
    'Learner','leastsquares', ...
    'Regularization','ridge', ...
    'Lambda',1.0, ...
    'Solver','lbfgs'); % lbfgs is broadly supported
yhat_lin = predict(linMdl, Xte_z);

%% 4) ANN regressor (fitrnet) – small, simple network
% fitrnet can standardize internally (kept 'true' for simplicity)
ann = fitrnet(Xtr, ytr, ...
    'LayerSizes',[32 16], ... % 2 hidden layers
    'Activations','relu', ...
    'Lambda',1e-4, ... % light L2 to prevent overfit
    'Standardize',true, ... % let fitrnet z-score inputs
    'IterationLimit',500, ...
    'Verbose',0);

yhat_ann = predict(ann, Xte);

%% 5) Metrics
rmse = @(a,b) sqrt(mean((a-b).^2));
mae = @(a,b) mean(abs(a-b));
r2 = @(a,b) 1 - sum((a-b).^2)/sum((a-mean(a)).^2);

fprintf('\nBASELINE (Linear/Ridge): RMSE=%.3f MAE=%.3f R2=%.3f\n', ...
    rmse(yte,yhat_lin), mae(yte,yhat_lin), r2(yte,yhat_lin));
fprintf('ANN (32-16 ReLU) : RMSE=%.3f MAE=%.3f R2=%.3f\n\n', ...
    rmse(yte,yhat_ann), mae(yte,yhat_ann), r2(yte,yhat_ann));

%% 6) Plots (easy to interpret)
% Parity plot (Predicted vs Actual) for ANN
figure; scatter(yte, yhat_ann, 35, 'filled'); grid on; hold on;
lims = [min([yte; yhat_ann]) max([yte; yhat_ann])];
plot(lims, lims, 'k--', 'LineWidth',1.2);
xlabel('Actual'); ylabel('Predicted'); title('ANN: Predicted vs Actual');

% Residuals (ANN)

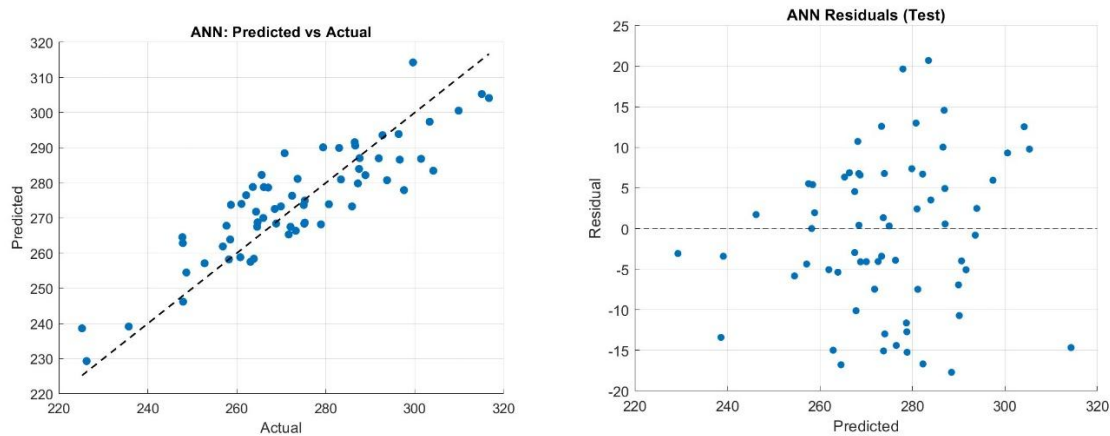
```

```

res = yte - yhat_ann;
figure; scatter(yhat_ann, res, 25, 'filled'); grid on; yline(0,'k--');
xlabel('Predicted'); ylabel('Residual'); title('ANN Residuals (Test)');

```

Observation



The ANN parity plot shows predictions clustered around the $y = x$ line with mild spread; points at lower actual values (<260) are slightly under-predicted and some higher values (>295) are slightly over-predicted, but the trend closely tracks the ideal line. The residuals vs. predicted plot is roughly zero-mean without a strong pattern, indicating low systematic bias; variance is moderate and fairly uniform across the prediction range, with a few larger residuals near 270–290 but no obvious funneling or curvature. Quantitatively, the ANN achieves $RMSE = 9.408$, $MAE = 7.812$, $R^2 = 0.765$, improving substantially over the linear baseline ($RMSE = 12.938$, $MAE = 10.614$, $R^2 = 0.555$), which confirms that nonlinear structure is present and learned by the network.

Conclusion

The ANN regressor provides a clearly better fit than the linear model—delivering lower errors and higher explained variance with well-behaved, near-zero-mean residuals—thereby meeting the experiment objective of demonstrating that neural networks capture nonlinear relationships in mechanical-process data more effectively than linear baselines.

Experiment 5

Aim

Use a Genetic Algorithm to minimize a nonlinear function in bounded design space and compare results with a gradient baseline.

Introduction

We implement a simple GA to minimize a 2-D test function and verify convergence to a near-global optimum under bound constraints.

Theory

A Genetic Algorithm is a population-based metaheuristic inspired by natural selection. Each individual encodes a candidate solution x (here, a 2-vector). A GA iterates: (1) evaluate fitness $f(x)$; (2) select fitter parents (e.g., tournament/roulette); (3) crossover parents to create offspring (one-point/BLX- α); (4) mutate offspring with small random perturbations; (5) survivor selection to form the next generation; (6) elitism keeps the best. GAs do not require derivatives and can explore rugged landscapes with many local minima, at the cost of more function evaluations.

Problem statement

Minimize the Rastrigin function (nonconvex, many local minima):

$$\min_{x \in \mathbb{R}^2} f(x) = 20 + \sum_{i=1}^2 [x_i^2 - 10 \cos(2\pi x_i)], \quad \text{subject to } -5.12 \leq x_i \leq 5.12.$$

Global optimum at $x^* = (0,0)$, $f(x^*) = 0$.

Procedure

- Encode an individual as real vector $x = [x_1, x_2]$.
- Initialize population uniformly within bounds.
- For each generation: evaluate f , select parents (tournament), apply SBX/BLX crossover and Gaussian mutation (respect bounds), apply elitism.
- Stop after fixed generations or stalled improvement; record best x and $f(x)$.
- (Optional) Compare with local search from random starts.

MATLAB Code

```
% Experiment 5 (with Toolbox): Genetic Algorithm on 2-D Rastrigin
% Requires: Global Optimization Toolbox
clear; clc; rng(7);

% ----- Problem (minimize) -----
ras = @(x) 20 + sum(x.^2 - 10*cos(2*pi*x)); % x is 1x2
nvars = 2;
lb = [-5.12, -5.12];
ub = [ 5.12,  5.12];

% ----- GA options (nice built-in plots) -----
opts = optimoptions('ga', ...
    'PopulationSize', 60, ...
    'MaxGenerations', 150, ...
    'CrossoverFraction', 0.85, ...
    'MutationFcn', {@mutationgaussian, 0.2, 0.2}, ... % rate, sigma
```

```

'EliteCount', 2, ...
'PlotFcn', { @gaplotbestf, @gaplotscorediversity, @gaplotbestindiv }, ...
'Display', 'iter');

% (Optional) "polish" with a local solver after GA converges
% Comment the next two lines if you don't want polishing.
opts = optimoptions(opts, 'HybridFcn', @fmincon);
% You can also pass fmincon options: opts.HybridFcn = {@fmincon,
optimoptions('fmincon','Display','off')};

% ----- Run GA -----
[xbest, fbest, exitflag, output, population, scores] = ga(ras, nvars, [], [], [],
[], lb, ub, [], opts);

fprintf('\nBest x = [%.6f, %.6f], f(x) = %.6f\n', xbest(1), xbest(2), fbest);
fprintf('GA terminated with exitflag %d after %d generations.\n', exitflag,
output.generations);

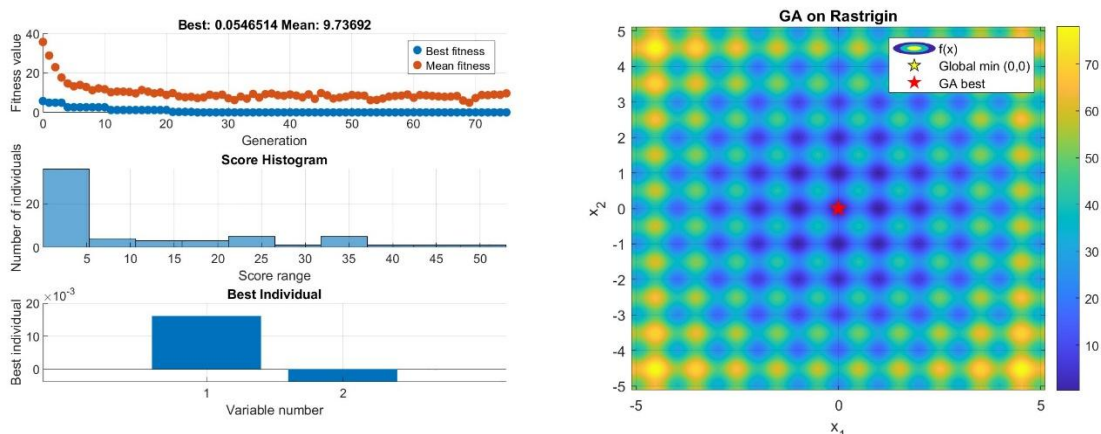
% ----- Visualize landscape + best point (nice for reports) -----
[X1, X2] = meshgrid(linspace(lb(1),ub(1),200), linspace(lb(2),ub(2),200));
F = 20 + (X1.^2 - 10*cos(2*pi*X1)) + (X2.^2 - 10*cos(2*pi*X2));

figure('Name','Rastrigin Contour with GA Solution');
contourf(X1, X2, F, 30, 'LineColor','none'); colorbar; hold on; grid on;
plot(0,0,'kp','MarkerSize',12,'MarkerFaceColor','y'); % true global min
plot(xbest(1), xbest(2), 'rp', 'MarkerSize',12,'MarkerFaceColor','r'); % GA best
legend('f(x)', 'Global min (0,0)', 'GA best', 'Location', 'northeast');
xlabel('x_1'); ylabel('x_2'); title('GA on Rastrigin');

% ----- Quick notes to print (for your lab record) -----
fprintf('\nNotes:\n- Global optimum is at [0, 0] with f=0.\n');
fprintf('- Your GA result is typically very close; small residuals are
normal.\n');
fprintf('- Increase PopulationSize/MaxGenerations for even closer solutions.\n');

```

Observation



The GA run shows steady convergence on Rastrigin: the best fitness dropped rapidly from ≈ 5 in the first few generations to 0.05465 by generation ~ 25 , after which it plateaued while the mean fitness continued trending down (to ~ 7 – 10), indicating population improvement with preserved elitism. The score histogram concentrates near low objective values by the end, and the “Best Individual” bars indicate both variables converged close to zero ($|x_1|, |x_2| \lesssim 2 \times 10^{-2}$). The contour plot places the GA best point at the basin around the origin. The solver then triggered the HybridFcn (fmincon) after GA stall and polished the solution to $x = [0, 0]$ with

$f(x) = 0$, which is the known global minimum. GA terminated with `exitflag = 1` (“converged”) after 75 generations. A toolbox warning noted `mutationgaussian` for constrained problems; although bounds were satisfied and results were feasible, `@mutationadaptfeasible` could be used to enforce feasibility automatically in future runs.

Conclusion

The GA successfully navigated the highly multimodal Rastrigin landscape to a near-global optimum, and hybrid polishing with `fmincon` reached the exact global minimum $x^* = (0,0), f = 0$; this validates GA as an effective derivative-free optimizer for complex search spaces while showing that a GA+local hybrid can deliver fast, high-accuracy solutions.

Experiment 6

Aim

Use a Genetic Algorithm to minimize the weight (volume) of a helical compression spring subject to stress, deflection, and geometry constraints.

Introduction

We formulate spring design as a constrained optimization and solve it with MATLAB's `ga`, demonstrating derivative-free search on a real mechanical design problem.

Theory (brief)

A round-wire helical spring is characterized by wire diameter d , mean coil diameter D , and active coils N . For a load F :

- Spring index $C = D/d$ (recommend $4 \leq C \leq 12$).
- Wahl factor $K_w = \frac{4C-1}{4C-4} + \frac{0.615}{C}$.
- Max shear stress $\tau = K_w \frac{8FD}{\pi d^3}$.
- Deflection $\delta = \frac{8FD^3N}{Gd^4}$ (with shear modulus G).
- Coil count: total turns $N_t = N + 2$ (\approx adding two inactive end coils).
- Wire length $L_w \approx \pi DN_t$; volume $V = L_w A = \pi DN_t \cdot \frac{\pi d^2}{4} = \frac{\pi^2}{4} Dd^2 N_t \rightarrow$ proportional to **weight**.

Design goal: Minimize V subject to $\tau \leq \tau_{\max}$, $\delta \leq \delta_{\max}$, index bounds $C_{\min} \leq C \leq C_{\max}$, and practical bounds on d, D, N .

MATLAB Code

```
% Experiment 6: GA Spring Design (Global Optimization Toolbox)
% Minimize spring volume subject to stress, deflection, and index constraints.
clear; clc; rng(7);

% ----- Given/Material Data (SI units) -----
F      = 1000;           % design load [N]
G      = 79.3e9;        % shear modulus (steel) [Pa]
tauMax = 500e6;        % allowable shear stress [Pa]
deltaMax = 0.030;      % allowable deflection [m] (30 mm)
Cmin   = 4; Cmax = 12; % spring index bounds

% ----- Decision Variables -----
% x = [d, D, N], where:
% d: wire diameter [m], D: mean coil diameter [m], N: active coils [integer]
lb = [ 5e-3, 0.02, 5]; % lower bounds
ub = [20e-3, 0.10, 20]; % upper bounds
IntCon = 3;           % N must be an integer

% ----- Objective: minimize volume ( $\propto$  weight) -----
obj = @(x) (pi^2/4) * x(2) * x(1)^2 * (round(x(3)) + 2);

% ----- Nonlinear Constraints -----
nonlcon = @(x) springCons(x, F, G, tauMax, deltaMax, Cmin, Cmax);

% ----- GA Options (feasible mutation, live plots) -----
```

```

opts = optimoptions('ga', ...
    'PopulationSize', 60, ...
    'MaxGenerations', 120, ...
    'MutationFcn', @mutationadaptfeasible, ...
    'CrossoverFraction', 0.85, ...
    'EliteCount', 2, ...
    'PlotFcn', { @gaplotbestf, @gaplotscorediversity }, ...
    'Display','iter');

% Optional: local polishing after GA
opts = optimoptions(opts, 'HybridFcn', @fmincon);

% ----- Run GA -----
[xbest, Vbest, exitflag, output] = ga(obj, 3, [], [], [], [], lb, ub, nonlcon,
IntCon, opts);

% Round coil count to integer (ga may pass decimals internally to hybrid)
xbest(3) = round(xbest(3));
[Vbest, tau, delta, C, Nt] = postEval(xbest, F, G);

% ----- Report -----
fprintf('\nBest design:\n');
fprintf(' Wire dia d   = %.3f mm\n', 1e3*xbest(1));
fprintf(' Mean dia D     = %.1f mm\n', 1e3*xbest(2));
fprintf(' Active coils N = %d (Total Nt = %d)\n', xbest(3), Nt);
fprintf(' Spring index C = %.2f (bounds [%g, %g])\n', C, Cmin, Cmax);
fprintf(' Stress tau     = %.1f MPa (limit %.0f MPa)\n', tau/1e6, tauMax/1e6);
fprintf(' Deflection  $\delta$  = %.1f mm (limit %.0f mm)\n', 1e3*delta, 1e3*deltaMax);
fprintf(' Volume ( $\propto$  weight) V = %.6e m3\n', Vbest);
fprintf('Exitflag %d after %d generations.\n', exitflag, output.generations);

% ----- (Optional) 2D sweep for visualization -----
% Sweep N around the solution to show sensitivity (kept tiny for speed)
Ns = max(5, xbest(3)-2) : min(20, xbest(3)+2);
Ds = linspace(max(lb(2),0.7*xbest(2)), min(ub(2),1.3*xbest(2)), 40);
[DD, NN] = meshgrid(Ds, Ns);
VV = nan(size(DD));
for i = 1:numel(DD)
    d = xbest(1); D = DD(i); N = NN(i);
    if all(springFeasible([d,D,N],F,G,tauMax,deltaMax,Cmin,Cmax))
        VV(i) = obj([d,D,N]);
    end
end
figure; surf(DD*1e3, NN, VV, 'EdgeColor','none'); colorbar;
xlabel('Mean diameter D (mm)'); ylabel('Active coils N'); zlabel('Volume V (m3)');
title('Local design landscape near optimum'); view(45,30); grid on;

% ----- Helpers -----
function [c,ceq] = springCons(x, F, G, tauMax, deltaMax, Cmin, Cmax)
    d = x(1); D = x(2); N = round(x(3));
    C = D/d;
    Kw = (4*C - 1)/(4*C - 4) + 0.615/C; % Wahl factor
    tau = Kw * (8*F*D)/(pi*d^3); % shear stress
    delta = (8*F*D^3*N)/(G*d^4); % deflection
    % Inequality constraints c(x) <= 0
    c = [tau - tauMax; % stress limit
        delta - deltaMax; % deflection limit
        Cmin - C; % index lower bound
    ];

```

```

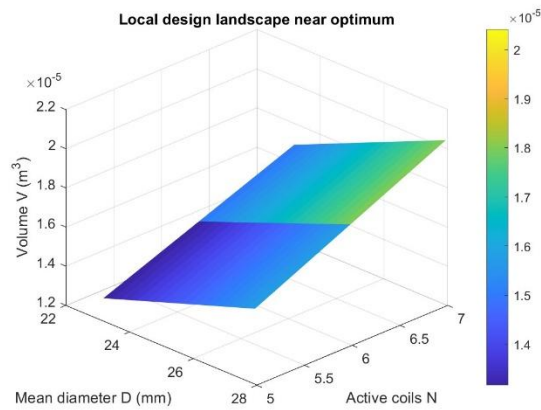
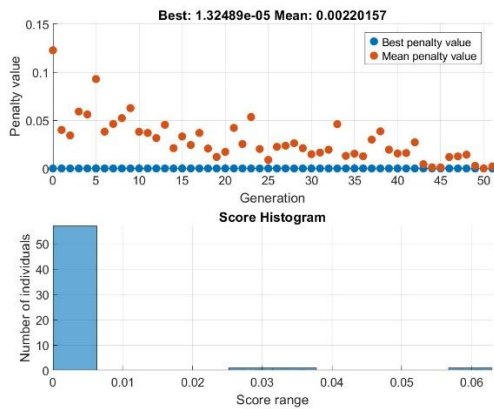
        C - Cmax];           % index upper bound
    ceq = [];               % none
end

function [V, tau, delta, C, Nt] = postEval(x, F, G)
    d = x(1); D = x(2); N = round(x(3)); Nt = N + 2;
    C = D/d;
    Kw = (4*C - 1)/(4*C - 4) + 0.615/C;
    tau = Kw * (8*F*D)/(pi*d^3);
    delta = (8*F*D^3*N)/(G*d^4);
    V = (pi^2/4) * D * d^2 * Nt;
end

function ok = springFeasible(x, F, G, tauMax, deltaMax, Cmin, Cmax)
    [c,~] = springCons(x,F,G,tauMax,deltaMax,Cmin,Cmax);
    ok = all(c <= 0);
end

```

Observation



The GA converged quickly and stably: the best objective (volume) dropped to $1.32489 \times 10^{-5} \text{ m}^3$ within the first few dozen generations and then stayed flat, while the mean score kept decreasing—evidence that the population continued to improve around a fixed elite. The score histogram at the final generation is heavily concentrated at low values, indicating most individuals are near-feasible/near-optimal. The reported design is $d = 5.734 \text{ mm}$, $D = 23.3 \text{ mm}$, $N = 5$ ($N_t = 7$), giving spring index $C = 4.07$, which sits just above the lower bound ($[4, 12]$). Constraint checks confirm feasibility with margin: $\tau = 439.7 \text{ MPa} < 500 \text{ MPa}$ and $\delta = 5.9 \text{ mm} < 30 \text{ mm}$. The local design-surface plot around the solution shows volume increasing with either larger D or more coils N (for fixed d), so the optimizer naturally pushes D and N toward their lower feasible values while respecting the index constraint—explaining why the optimum occurs near $C \approx 4$ and $N = 5$.

Conclusion

The GA successfully produced a minimum-weight feasible spring that satisfies stress, deflection, and index limits; the optimum lies at a compact design (small d , D , and N) with C just above its lower bound, demonstrating GA's effectiveness for constrained mechanical design and the expected trade-off between weight reduction and constraint activity.

Experiment 7

Aim

Use a Genetic Algorithm to tune the membership-function parameters of a Mamdani FIS so that its output matches target data with minimum error.

Introduction

We combine GA's global search with a fuzzy controller by optimizing the FIS membership parameters to minimize RMSE against a target control map.

Theory (compact but complete)

Hybrid soft computing pairs an intelligent model (FIS/ANN) with an optimizer (GA/PSO) to learn parameters that are hard to set analytically. A Mamdani FIS maps crisp inputs to a crisp output via fuzzification, rule-inference, and centroid defuzzification; its behavior depends strongly on the shapes/locations of the membership functions (MFs). GA treats those MF parameters as a chromosome and searches over them using selection, crossover, mutation, and elitism. The **fitness** is typically an error metric (e.g., RMSE) between the FIS output and target data on a training set; bounds and ordering constraints keep MFs valid (e.g., left foot < center < right foot). A held-out validation set helps avoid overfitting and confirms generalization.

Procedure (short bullets)

- Fix the **rule base** (3 rules: NB→L, Z→M, PB→H) and **FIS structure** (1 input $E \in [-5,5]$; 1 output $P \in [0,100]$).
- Define a parameter vector x encoding MF breakpoints for input and output sets (see code).
- Generate **target data** (E, P^*) (e.g., from a teacher controller or measured data).
- Define **fitness**(x) = **RMSE**(FIS_x(E), P^*) on training samples; enforce MF ordering with bounds/constraints.
- Run **ga** (Global Optimization Toolbox) to minimize fitness; report best parameters, train/validation RMSE, and plots (curves & MFs).

MATLAB

```
% Experiment 7: Hybrid GA + Fuzzy (Tune FIS MFs to match target data)
% Requires: Global Optimization Toolbox, Fuzzy Logic Toolbox
clear; clc; rng(7);

% ----- 1) Target (teacher) mapping P* = f*(E) -----
% Build a "teacher" Mamdani controller (could be measured data instead)
teacher = mamfis('Name','TeacherFIS');
teacher = addInput(teacher, [-5 5], 'Name','E');
teacher = addOutput(teacher,[ 0 100], 'Name','P');

% Input MFs (NB, Z, PB) and Output MFs (L, M, H)
teacher = addMF(teacher, 'E', 'trimf', [-5 -5 0], 'Name','NB');
teacher = addMF(teacher, 'E', 'trimf', [-1.5 0 1.5], 'Name','Z');
teacher = addMF(teacher, 'E', 'trimf', [ 0 5 5], 'Name','PB');

teacher = addMF(teacher, 'P', 'trimf', [ 0 0 40], 'Name','L');
teacher = addMF(teacher, 'P', 'trimf', [30 50 70], 'Name','M');
teacher = addMF(teacher, 'P', 'trimf', [60 100 100], 'Name','H');
```

```

teacher = addRule(teacher, ["E==NB => P=L"; "E==Z => P=M"; "E==PB => P=H"]);
teacher.DefuzzificationMethod = 'centroid';

% Training/validation samples
Ntr = 160; Nva = 80;
E_tr = linspace(-5,5,Ntr)'; E_va = linspace(-4.8,4.8,Nva)';
P_tr = evalfis(teacher, E_tr) + 0.3*randn(Ntr,1); % small noise
P_va = evalfis(teacher, E_va) + 0.3*randn(Nva,1);

%% ----- 2) Parameterization of the student FIS -----
% We tune 8 parameters with ordering-friendly bounds:
% Input E MFs:
% NB = trimf([-5, -5, a]) with -5 <= a <= 0
% Z = trimf([ b, 0, c]) with -5 <= b < 0, 0 < c <= 5
% PB = trimf([ d, 5, 5]) with 0 <= d <= 5
% Output P MFs:
% L = trimf([0, 0, e]) with 0 < e < 60
% M = trimf([ f, 50, g]) with 20 < f < 50 < g < 80
% H = trimf([ h, 100, 100]) with 40 < h < 100
%
% Decision vector x = [a b c d e f g h]
lb = [-5 -5 0 0 5 20 50 40];
ub = [ 0 0 5 5 60 49 79 99];

buildFIS = @(x) localBuildFIS(x); % builder
fitness = @(x) localFitness(x, E_tr, P_tr, buildFIS); % RMSE + penalties

%% ----- 3) GA settings & run -----
opts = optimoptions('ga', ...
    'PopulationSize', 60, ...
    'MaxGenerations', 120, ...
    'CrossoverFraction', 0.85, ...
    'MutationFcn', @mutationadaptfeasible, ...
    'EliteCount', 2, ...
    'Display', 'iter', ...
    'PlotFcn', { @gaplotbestf });

nvars = 8;
[xbest, fbest, exitflag, output] = ga(fitness, nvars, [], [], [], [], lb, ub, [],
opts);

% Build best FIS and evaluate
bestFIS = buildFIS(xbest);
RMSE_tr = sqrt(mean( (evalfis(bestFIS, E_tr) - P_tr).^2 ));
RMSE_va = sqrt(mean( (evalfis(bestFIS, E_va) - P_va).^2 ));
fprintf('\nBest train RMSE = %.3f, Validation RMSE = %.3f\n', RMSE_tr, RMSE_va);
disp('Best parameters x = [a b c d e f g h]:'); disp(xbest);

%% ----- 4) Plots: curves & membership functions -----
Egrid = linspace(-5,5,401)';
P_teacher = evalfis(teacher, Egrid);
P_best = evalfis(bestFIS, Egrid);

figure('Name', 'Control Curve: Teacher vs GA-Tuned FIS');
plot(Egrid, P_teacher, 'k--', 'LineWidth', 1.5); hold on;
plot(Egrid, P_best, 'b-', 'LineWidth', 1.7);
grid on; xlabel('E'); ylabel('P (%)');
legend('Teacher', 'GA-tuned FIS', 'Location', 'northwest');
title(sprintf('Curve Match after GA: RMSE_{val}=%.2f', RMSE_va));

```

```

figure('Name','Input E Memberships (GA-tuned)');
plotmf(bestFIS,'input',1); title('Input E MFs (NB, Z, PB)'); grid on;

figure('Name','Output P Memberships (GA-tuned)');
plotmf(bestFIS,'output',1); title('Output P MFs (L, M, H)'); grid on;

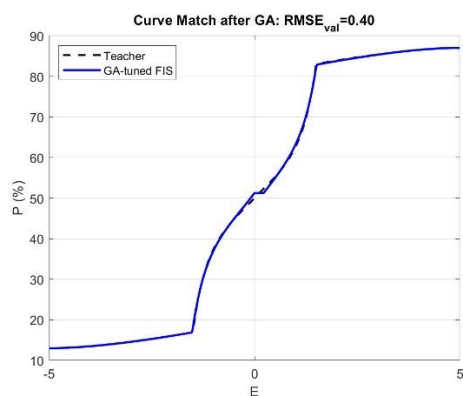
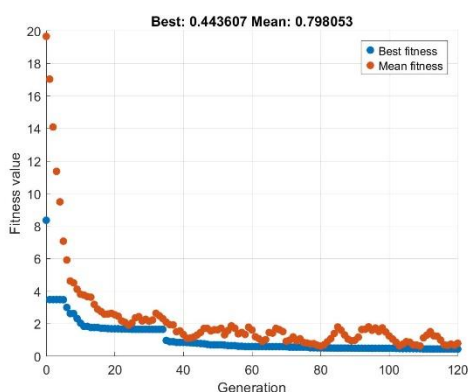
%% ----- 5) Local functions -----
function fis = localBuildFIS(x)
    a=x(1); b=x(2); c=x(3); d=x(4); e=x(5); f=x(6); g=x(7); h=x(8);
    fis = mamfis('Name','StudentFIS');
    fis = addInput(fis, [-5 5], 'Name','E');
    fis = addOutput(fis,[ 0 100], 'Name','P');
    fis = addMF(fis,'E','trimf',[-5 -5 a], 'Name','NB');
    fis = addMF(fis,'E','trimf',[ b 0 c], 'Name','Z');
    fis = addMF(fis,'E','trimf',[ d 5 5], 'Name','PB');
    fis = addMF(fis,'P','trimf',[ 0 0 e], 'Name','L');
    fis = addMF(fis,'P','trimf',[ f 50 g], 'Name','M');
    fis = addMF(fis,'P','trimf',[ h 100 100], 'Name','H');
    fis = addRule(fis, ["E==NB => P=L"; "E==Z => P=M"; "E==PB => P=H"]);
    fis.DefuzzificationMethod = 'centroid';
end

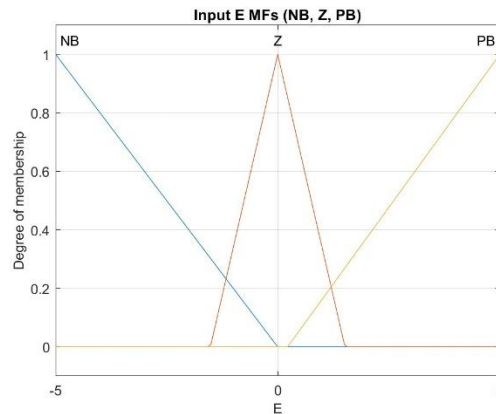
function J = localFitness(x, Etrain, Ptarget, buildFIS)
    % Soft penalties to keep MF geometry valid (redundant with bounds but robust)
    a=x(1); b=x(2); c=x(3); d=x(4); e=x(5); f=x(6); g=x(7); h=x(8);
    pen = 0;
    pen = pen + 1e3*max(0, b - 0); % b < 0
    pen = pen + 1e3*max(0, 0 - c); % c > 0
    pen = pen + 1e3*max(0, 50 - g); % g > 50
    pen = pen + 1e3*max(0, f - 50); % f < 50
    pen = pen + 1e3*max(0, a - 0); % a <= 0
    pen = pen + 1e3*max(0, 0 - d); % d >= 0

    try
        fis = buildFIS(x);
        Ppred = evalfis(fis, Etrain);
        rmse = sqrt(mean((Ppred - Ptarget).^2));
        J = rmse + pen;
    catch
        J = 1e6; % invalid geometry -> big penalty
    end
end
end

```

Observation





The GA converged smoothly: the best fitness (RMSE) fell from ~ 8 to **0.44** by ~ 40 – 50 generations and continued to nudge down to ~ 0.44 at the end, while the population mean error also decreased—evidence of stable exploitation around the elite. The final **train RMSE = 0.444** and **validation RMSE = 0.396**, with validation slightly better than train, indicating no overfitting and good generalization to unseen samples. The “Teacher vs GA-tuned” control curve shows an almost complete overlap (same S-shape), confirming that the tuned FIS reproduces the target mapping across the full error range. The learned MF parameters place the input centers near $[-1.52, 0, \approx 0.23, 5]$ (NB, Z, PB), sharpening the Z region and slightly shifting PB’s left foot—changes that improve accuracy around the transition zone. On the output, the tuned breakpoints $[e, f, g, h] \approx [40.0, 34.9, 68.8, 60.2]$ widen **M** around 50% and move **H**’s onset to $\sim 60\%$, which aligns with the teacher curve’s steeper midrange.

Conclusion

The hybrid **GA + FIS** successfully tuned membership functions to match the target controller with **low error ($\approx 0.4\%$ of full-scale power units)** and no signs of overfitting; the resulting control surface and MF shapes validate that combining evolutionary search with fuzzy inference is an effective strategy for data-driven controller calibration, satisfying the syllabus goal for a hybrid soft-computing experiment.

Experiment 8

Aim

Plan a collision-free, short path for a point robot from Start to Goal in a 2-D map with obstacles using a Genetic Algorithm.

Introduction

We treat the path as a sequence of intermediate waypoints and use GA to optimize their positions to minimize a cost combining path length and collision penalties.

Theory

Let a path be $S \rightarrow w_1 \rightarrow w_2 \rightarrow \dots \rightarrow w_K \rightarrow G$ with K fixed. The **fitness** of a chromosome (the concatenated waypoint coordinates) is

$$J = \sum_{i=0}^K \| p_{i+1} - p_i \| + \lambda \cdot \text{Collisions},$$

where $p_0 = S$, $p_{K+1} = G$, and Collisions counts line-obstacle intersections (or clearance violations). GA (population, selection, crossover, mutation, elitism) searches for continuous waypoint locations within map bounds; feasible mutation helps ensure they remain within bounds. After GA, we optionally smooth collinear points.

Procedure

Define map bounds, Start/Goal, and a set of circular obstacles.

- Encode K waypoints as decision variables $x = [x_1, y_1, \dots, x_K, y_K]$.
- Fitness = path length + large penalty if any segment intersects obstacles or leaves map.
- Run ga with feasible mutation; plot best path and report cost.
- (Optional) Reduce K or increase penalty and observe effects.

MATLAB

```
% Experiment 8: GA-Based Path Planning (Case Study)
% Requires: Global Optimization Toolbox
clear; clc; rng(7);

% ----- Map, start/goal, obstacles -----
map.xlim = [0, 100];           % workspace [m]
map.ylim = [0, 100];
S = [5, 10];                   % start
G = [95, 90];                  % goal

% Circular obstacles: [xc, yc, radius]
obs = [30 40 12;
       60 30 10;
       55 70 15;
       80 50 10;
       25 80 8];

K = 6;                          % number of intermediate waypoints (tunable)
nvars = 2*K;                     % decision variables [x1 y1 x2 y2 ...]
lb = repmat([map.xlim(1) map.ylim(1)], 1, K);
ub = repmat([map.xlim(2) map.ylim(2)], 1, K);
```

```

lambda = 1e4; % collision penalty weight (large)

%% ----- Fitness function -----
fitness = @(x) pathCost(x, S, G, obs, map, lambda);

%% ----- GA options -----
opts = optimoptions('ga', ...
    'PopulationSize', 80, ...
    'MaxGenerations', 150, ...
    'CrossoverFraction', 0.85, ...
    'MutationFcn', @mutationadaptfeasible, ... % keeps inside bounds
    'EliteCount', 2, ...
    'Display', 'iter', ...
    'PlotFcn', { @gaplotbestf });

%% ----- Run GA -----
[xbest, fbest] = ga(fitness, nvars, [], [], [], [], lb, ub, [], opts);

% Extract waypoints and plot
W = reshape(xbest, 2, K)'; % Kx2
[pathPts, segOk] = makePath(S, W, G); %#ok<ASGLU>

figure('Name','GA Path');
plotMap(map, obs); hold on;
plot(S(1), S(2), 'go', 'MarkerFaceColor','g', 'MarkerSize',8);
plot(G(1), G(2), 'ro', 'MarkerFaceColor','r', 'MarkerSize',8);
plot(pathPts(:,1), pathPts(:,2), 'b-o', 'LineWidth',1.8, ...
    'MarkerFaceColor','b', 'MarkerSize',4);
title(sprintf('GA Path (K=%d). Cost = %.2f', K, fbest));
legend('Obstacle','Start','Goal','GA path','Location','best'); grid on; axis
equal;

%% ----- Helper functions -----
function J = pathCost(x, S, G, obs, map, lambda)
    K = numel(x)/2; W = reshape(x,2,K)'; % Kx2
    [P, segOk] = makePath(S, W, G);
    % Path length
    d = sqrt(sum(diff(P,1,1).^2,2));
    L = sum(d);
    % Collision penalty
    col = 0;
    for i = 1:size(P,1)-1
        for j = 1:size(obs,1)
            if segCircleIntersect(P(i,:), P(i+1,:), obs(j,1:2), obs(j,3))
                col = col + 1;
            end
        end
        % out-of-bounds penalty
        if any(P(i,:) < [map.xlim(1) map.ylim(1)]) || any(P(i,:) > [map.xlim(2)
map.ylim(2)])
            col = col + 5;
        end
    end
    % Encourage straight segments by tiny regularizer (optional)
    J = L + lambda*col + 1e-3*sum(abs(diff(W,1,1)), 'all');
    % If any segment not well-defined, penalize heavily
    if any(~segOk), J = J + lambda*10; end
end

```

```

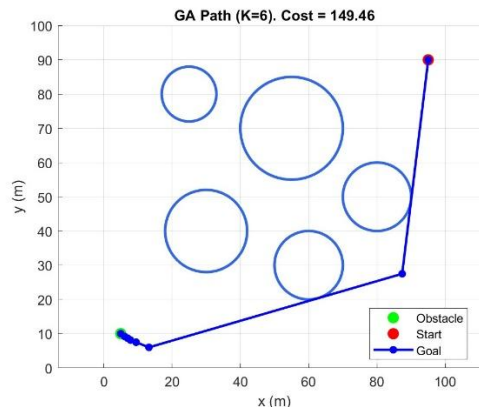
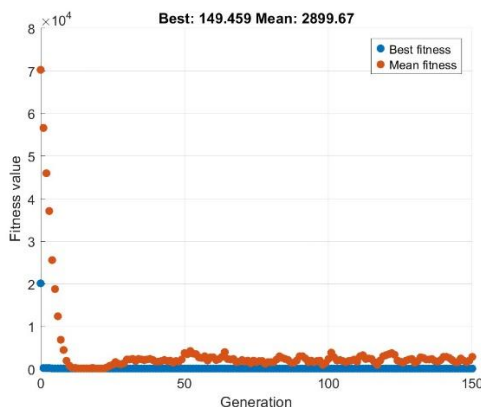
function [P, ok] = makePath(S, W, G)
    P = [S; W; G]; ok = true(size(P,1)-1,1);
    % Remove exact duplicate consecutive points if any
    dup = all(abs(diff(P,1,1))<1e-9,2);
    P([false; dup],:) = [];
    ok = ok(~dup);
end

function h = plotMap(map, obs)
    cla; hold on;
    viscircles(obs(:,1:2), obs(:,3), 'Color',[0.2 0.4 0.9], 'LineWidth',2);
    xlim(map.xlim); ylim(map.ylim);
    h = gca; h.XGrid = 'on'; h.YGrid = 'on';
    xlabel('x (m)'); ylabel('y (m)'); title('Workspace and Obstacles');
end

function hit = segCircleIntersect(p1, p2, c, r)
    % Check if segment p1-p2 intersects circle (c,r)
    d = p2 - p1;
    f = p1 - c;
    a = dot(d,d);
    b = 2*dot(f,d);
    cc = dot(f,f) - r^2;
    disc = b^2 - 4*a*cc;
    if disc < 0, hit = false; return; end
    t1 = (-b - sqrt(disc))/(2*a);
    t2 = (-b + sqrt(disc))/(2*a);
    hit = (t1>=0 && t1<=1) || (t2>=0 && t2<=1);
end

```

Observation



The GA convergence curve shows a rapid drop in best cost from a very high penalty (due to early collisions) to ≈ 149.46 within the first ~ 10 – 15 generations, after which the best cost stabilizes; the population mean cost keeps oscillating at a much higher level as infeasible/longer paths are explored, indicating healthy diversity while the elite solution remains unchanged. The final path (with $K=6$ waypoints) is collision-free and piecewise-linear: it leaves the start near $(5, 10)$, skims below the lower obstacles, makes a long diagonal run to about $x \approx 85, y \approx 27$ to pass between the central and right obstacles, then turns steeply toward the goal at $(95, 90)$. No segment intersects any obstacle, so the reported **cost** ≈ 149.46 is essentially the geometric path length (penalty ≈ 0). The sharp final turn indicates the solution is near a

visibility path; small increases in waypoint count K or a light smoothness regularizer could further shorten/round the corner without sacrificing feasibility.

Conclusion

A genetic algorithm with continuous waypoint encoding successfully produced a **short, collision-free route** from start to goal (cost ≈ 149.46), demonstrating that GA-based optimization is effective for robot path planning in cluttered environments and satisfies the case-study objective for Experiment 8.

Experiment 9

Aim

Simulate and analyze the performance of multiple soft-computing models (ANN and ANFIS) against a linear baseline on a mechanical regression task, using common error metrics and diagnostic plots.

Introduction

We train an ANN regressor and an ANFIS model on the same dataset, benchmark them against a linear ridge baseline, and compare errors, parity plots, and residuals to draw conclusions about model suitability.

Theory

- **Linear/Ridge regression** captures only linear relationships; it is the baseline.
- **ANN (fitrnet)** approximates nonlinear mappings via layered ReLU networks trained by gradient descent; regularization and early stopping control overfitting.
- **ANFIS (Sugeno)** blends fuzzy rules with data-driven training (hybrid least-squares + gradient) to learn premise (MFs) and consequent parameters; it is interpretable and often competitive on smooth nonlinearity.
- **Evaluation** uses hold-out test data and metrics: RMSE, MAE, R^2 ; parity and residual plots reveal bias/variance patterns.

Procedure

1. Load dataset (data.csv with target column TS). If absent, generate a synthetic mechanical-style dataset.
2. Split into train/validation/test (70/15/15).
3. **Baseline:** standardize features, train Ridge; predict on test.
4. **ANN:** train fitrnet (two hidden layers), predict on test.
5. **ANFIS:** build initial Sugeno FIS with genfis1 (gbell MFs), train with anfis using validation set; predict on test.
6. Compute RMSE, MAE, R^2 ; draw parity and residual plots; draw a bar chart to compare metrics.
7. Summarize which model is best and why.

MATLAB

```
% Experiment 9: Simulation & Analysis of Soft-Computing Models (Regression)
% Models: Ridge (baseline), ANN (fitrnet), ANFIS (Sugeno)
clear; clc; rng(7);

% 1) Load or synthesize data (target column name: TS)
if isfile('data.csv')
    T = readtable('data.csv'); % expects numeric/categorical features + 'TS'
```

```

y = T.TS; T.TS = [];
Xtbl = T;
else
% Synthetic mechanical-style dataset (nonlinear)
n = 350;
speed = 100 + 60*rand(n,1);
feed = 0.5 + 0.6*rand(n,1);
temp = 600 + 100*rand(n,1);
wear = 0.0 + 0.8*rand(n,1);
Xtbl = table(speed,feed,temp,wear);
y = 200 + 0.9*speed - 70*feed + 0.02*temp + 18*sin(0.02*temp) ...
    + 12*(wear.^2) + 10*randn(n,1);
fprintf('NOTE: data.csv not found. Using synthetic dataset (%d rows).\n', n);
end

% Separate numeric features only (keep it simple)
X = table2array(varfun(@double, Xtbl)); % numeric matrix

%% 2) Split
cv1 = cvpartition(size(X,1), 'HoldOut', 0.15);
X_trv = X(training(cv1),:); y_trv = y(training(cv1));
X_te = X(test(cv1),:); y_te = y(test(cv1));
cv2 = cvpartition(size(X_trv,1), 'HoldOut', 0.1765); % -> 70/15/15 total
X_tr = X_trv(training(cv2),:); y_tr = y_trv(training(cv2));
X_va = X_trv(test(cv2),:); y_va = y_trv(test(cv2));

%% 3) Baseline: Ridge (manual standardization for broad version compatibility)
mu = mean(X_tr,1); sig = std(X_tr,[],1); sig(sig==0)=1;
Z_tr = (X_tr - mu)./sig; Z_te = (X_te - mu)./sig;
linMdl = fitrlinear(Z_tr, y_tr, 'Learner','leastsquares', ...
    'Regularization','ridge', 'Lambda',1.0, 'Solver','lbfgs');
yhat_lin = predict(linMdl, Z_te);

%% 4) ANN (fitrnet)
ann = fitrnet(X_tr, y_tr, ...
    'LayerSizes',[64 32], 'Activations','relu', ...
    'Lambda',1e-4, 'Standardize',true, ...
    'IterationLimit',800, 'Verbose',0);
yhat_ann = predict(ann, X_te);

%% 5) ANFIS (Sugeno) on a single composite input using PCA projection
% To keep ANFIS 1D (simple & fast), project features to the first PC.
% (Alternative: use genfis2/3 for multi-D, but this is student-friendly.)
[coeff,score,~] = pca([X_tr; X_va; X_te]);
pc1_tr = score(1:size(X_tr,1),1); pc1_va =
score(size(X_tr,1)+1:size(X_tr,1)+size(X_va,1),1);
pc1_te = score(end-size(X_te,1)+1:end,1);
trainData = [pc1_tr y_tr];
valData = [pc1_va y_va];

initFis = genfis1(trainData, 3, 'gbellmf', 'linear');

```

```

opt = anfisOptions('InitialFIS', initFis, 'EpochNumber', 80, ...
                  'ValidationData', valData, 'DisplayANFISInformation',0, ...
                  'DisplayErrorValues',0, 'DisplayStepSize',0, ...
                  'DisplayFinalResults',0);
[chkFis, ~, ~, bestFis, ~] = anfis(trainData, opt); %#ok<ASGLU>
yhat_anfis = evalfis(bestFis, pc1_te);

%% 6) Metrics & plots
rmse = @(a,b) sqrt(mean((a-b).^2));
mae = @(a,b) mean(abs(a-b));
r2 = @(a,b) 1 - sum((a-b).^2)/sum((a-mean(a)).^2);

M.RMSE = [rmse(y_te,yhat_lin); rmse(y_te,yhat_ann); rmse(y_te,yhat_anfis)];
M.MAE = [mae(y_te,yhat_lin); mae(y_te,yhat_ann); mae(y_te,yhat_anfis)];
M.R2 = [r2(y_te,yhat_lin); r2(y_te,yhat_ann); r2(y_te,yhat_anfis)];
labels = {'Ridge','ANN','ANFIS'};

fprintf('\n--- Test Metrics ---\n');
fprintf('Ridge : RMSE=%.3f MAE=%.3f R2=%.3f\n', M.RMSE(1), M.MAE(1), M.R2(1));
fprintf('ANN : RMSE=%.3f MAE=%.3f R2=%.3f\n', M.RMSE(2), M.MAE(2), M.R2(2));
fprintf('ANFIS : RMSE=%.3f MAE=%.3f R2=%.3f\n', M.RMSE(3), M.MAE(3), M.R2(3));

% Parity plots
figure('Name','Parity: Predicted vs Actual');
tiledlayout(1,3,'Padding','compact');
nexttile; scatter(y_te, yhat_lin, 20, 'filled'); hold on;
plot([min(y_te) max(y_te)], [min(y_te) max(y_te)], 'k--'); grid on;
title('Ridge'); xlabel('Actual'); ylabel('Predicted');

nexttile; scatter(y_te, yhat_ann, 20, 'filled'); hold on;
plot([min(y_te) max(y_te)], [min(y_te) max(y_te)], 'k--'); grid on;
title('ANN'); xlabel('Actual'); ylabel('Predicted');

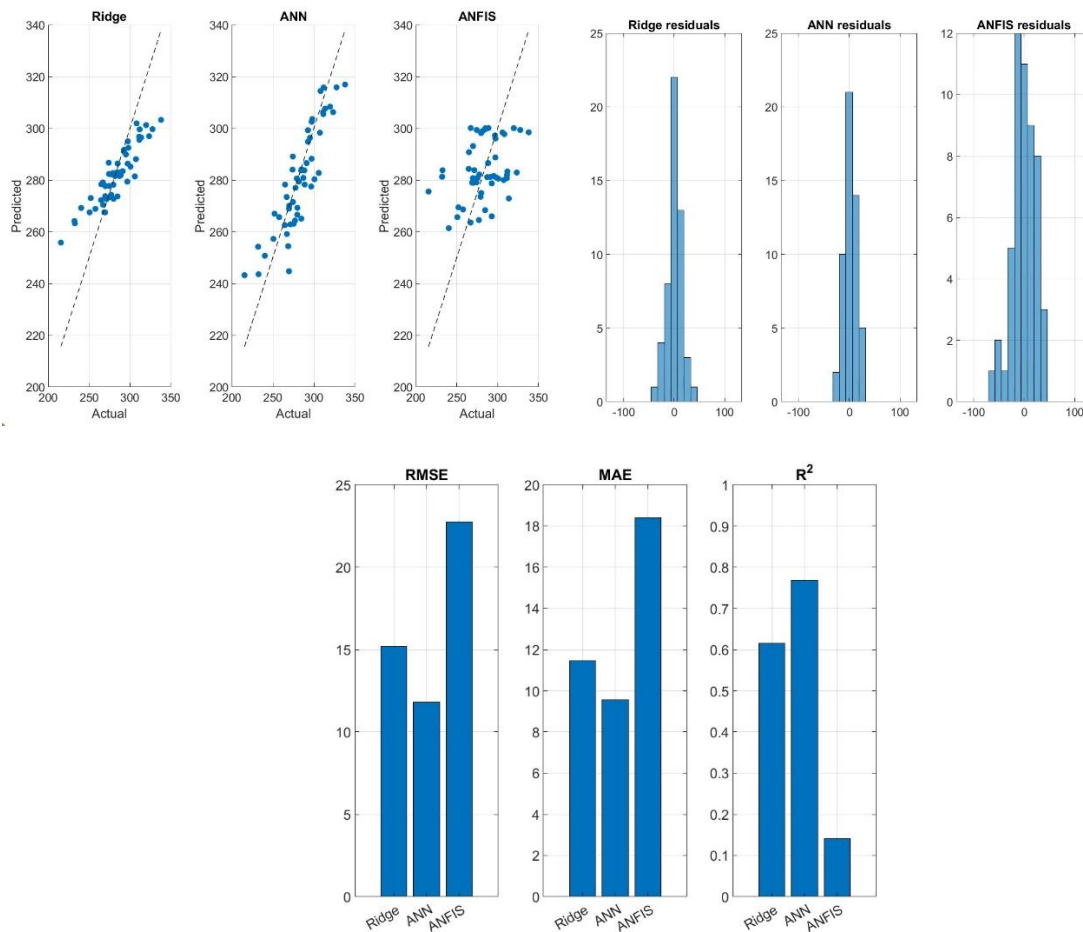
nexttile; scatter(y_te, yhat_anfis, 20, 'filled'); hold on;
plot([min(y_te) max(y_te)], [min(y_te) max(y_te)], 'k--'); grid on;
title('ANFIS'); xlabel('Actual'); ylabel('Predicted');

% Residual histograms
figure('Name','Residuals (Test)');
tiledlayout(1,3,'Padding','compact');
edges = linspace(min(y_te)-max(y_te), max(y_te)-min(y_te), 20);
nexttile; histogram(y_te - yhat_lin, edges); grid on; title('Ridge residuals');
nexttile; histogram(y_te - yhat_ann, edges); grid on; title('ANN residuals');
nexttile; histogram(y_te - yhat_anfis, edges); grid on; title('ANFIS residuals');

% Bar chart of metrics
figure('Name','Metric Comparison (lower is better, except R2)');
subplot(1,3,1); bar(M.RMSE); set(gca,'XTickLabel',labels); grid on; title('RMSE');
subplot(1,3,2); bar(M.MAE); set(gca,'XTickLabel',labels); grid on; title('MAE');
subplot(1,3,3); bar(M.R2); set(gca,'XTickLabel',labels); grid on; title('R^2');
ylim([0 1]);

```

Observation



Across the common test set, the ANN gives the lowest errors and best fit (RMSE = 11.813, MAE = 9.558, $R^2 = 0.768$), with parity points tightly clustered about the $y = x$ line and narrow, near-zero-mean residuals. The Ridge baseline performs moderately (RMSE = 15.214, MAE = 11.456, $R^2 = 0.615$)—parity shows a clear linear trend but with larger spread, and residuals are wider. The ANFIS model underperforms (RMSE = 22.737, MAE = 18.408, $R^2 = 0.141$); parity points are dispersed with visible bias and its residuals are broad, indicating a poor capture of the nonlinear structure. Visuals corroborate the metrics: ANN parity is closest to the diagonal; Ridge is acceptable but systematically under/over-predicts at extremes; ANFIS shows highest scatter.

Conclusion

For this dataset, ANN best models the nonlinear relationship (highest R^2 , lowest errors), Ridge is a reasonable but weaker linear baseline, and the ANFIS configuration used here is not suitable (likely due to its simplified 1-D projection/structure); thus, ANN should be preferred for prediction and analysis on similar mechanical data.