Soft Computing and Optimization Laboratory Manual

3rd Sem, B.Tech. (Electrical Engineering)

Department of Electrical Engineering

Veer Surendra Sai University of Technology Burla

Contents

VISION

To be recognized as a centre of excellence in education and research in the field of Electrical Engineering by producing innovative, creative and ethical Electrical Engineering professionals for socio-economic development of society in order to meet the global challenges.

MISSION

Electrical Engineering Department of VSSUT Burla strives to impart quality education to the students with enhancement of their skills to make them globally competitive through:

M1. Maintaining state of the art research facilities to provide enabling environment to create, analyze, apply and disseminate knowledge.

M2. Fortifying collaboration with world class R&D organizations, educational institutions, industry and alumni for excellence in teaching, research and consultancy practices to fulfil 'Make in India' policy of the Government.

M3. Providing the students with academic environment of excellence, leadership, ethical guidelines and lifelong learning needed for a long productive career.

PROGRAM EDUCATIONAL OBJECTIVES of B.Tech. (EE)

The program educational objectives of B.Tech. in Electrical Engineering program of VSSUT Burla are to prepare its graduates:

1. To have basic and advanced knowledge in Electrical Engineering with specialized knowledge in design and commissioning of electrical systems/renewable energy systems comprising of generation, transmission and distribution to become eminent, excellent and skillful engineers.
2. To succeed in getting engineering position with electrical design, manufacturing industries or in software and hardware industries, in private or government sectors, at Indian and in Multinational organizations.
3. To have a well-rounded education that includes excellent communication skills, working effectively on team-based projects, ethical and social responsibility.
4. To have the ability to pursue study in specific area of interest and be able to become successful entrepreneur.
5. To have broad knowledge serving as foundation for lifelong learning in multidisciplinary areas to enable career and professional growth in top academic, industrial and government/corporate organizations.

List of Experiments

1. Solution of single objective optimization problem using MATLAB Optimization Toolbox (linprog, quadprog, fmincon).
2. Solution of single objective optimization using OCTAVE sqp and GAMS solvers.
3. Implementation of fuzzy tool box to solve optimization problem.
4. Design of Fuzzy rule base and Fuzzy Inference System to solve an optimization problem.
5. Implementation of Genetic Algorithms to solve an optimization problem.
6. Implementation of Artificial Neural Networks to solve optimization problems.
7. Implementation of Particle Swarm Optimization to solve optimization problems.

Course Outcomes:

Upon completion of the course, the students will be able to:

CO1	Demonstrate the use of MATLAB, OCTAVE and GAMS solvers.
CO2	Demonstrate the use of fuzzy logic to solve optimization problems
CO3	Demonstrate the use of genetic algorithm to solve optimization problems
CO4	Demonstrate the use of artificial neural networks to solve optimization problems
CO5	Demonstrate the use of swarm optimization algorithms to solve optimization problems

Experiment 1

Solving Single Objective Optimization Problem using MATLAB Optimization Toolbox

1.1 Aim of the Experiment

Solution of single objective optimization problem using MATLAB Optimization Toolbox (linprog, quadprog, fmincon).

1.2 Software/tools required:

MATLAB/ optimization tool

1.3 Theory

Optimization is the act of obtaining the best result under given circumstances. In design, construction, and maintenance of any engineering system, engineers have to make technological and managerial decisions at several stages. The ultimate goal of all such decisions is either to minimize the effort required or to maximize the desired benefit. Since the effort required or the benefit desired in any practical situation can be expressed as a function of certain decision variables, optimization can be defined as the process of finding the conditions that give the maximum or minimum value of a function.

Optimization, in its broadest sense, can be applied to solve any engineering problem from different engineering disciplines.

1.3.1 Elements of Problem Formulation

Design Vector Any engineering system or component is defined by a set of quantities some of which are viewed as variables during the design process. In general, certain quantities are usually fixed at the outset and these are called pre-assigned parameters. All the other quantities are treated as variables in the design process and are called design or decision variables xi, where i = 1, 2,...n. The design variables are collectively represented as a design vector $X = \{x1, x2, \ldots, xn\}$ T.

Design Constraints In many practical problems, the design variables cannot be chosen arbitrarily; rather, they have to satisfy certain specified functional and other requirements. The restrictions that must be satisfied to produce an acceptable design are collectively called design constraints. Constraints that represent limitations on the behavior or performance of the system are termed behavior or functional constraints. Constraints that represent physical limitations on design variables, such as availability, fabricability, and transportability, are known as geometric or side constraints.

Constraint Surface To visualize a constrained surface, consider an optimization problem with only inequality constraints $g_j(X) \leq 0$. The set of values of X that satisfy the equation $g_j(X) = 0$ forms a hypersurface in the design space and is called a constraint surface. Note that this is an $(n-1)$ dimensional subspace, where n is the number of design variables. The constraint surface divides the design space into two regions: one in which $g_j(X) < 0$ and the other in which $g_j(X) > 0$. Thus the points lying on the hypersurface will satisfy theconstraint $g_j(X)$ critically, whereas the points lying in the region where $g_j(X) > 0$ are infeasible or unacceptable, and the points lying in the region where $g_j(X) < 0$ are feasible or acceptable. The collection of all the constraint surfaces $g_j(X) = 0$, $j = 1, 2, \ldots m$,which separates the acceptable region is called the composite constraint surface.

A design point that lies on one or more than one constraint surface is called a bound point, and the associated constraint is called an active constraint. Design points that donot lie on any constraint surface are known as free points. Depending on whether a particular design point belongs to the acceptable or unacceptable region, it can be categorized as free and acceptable point; Free and unacceptable point; Bound and acceptable point. Bound and unacceptable point.

Objective Function The conventional design procedures aim at finding an acceptable or adequate design that merely satisfies the functional and other requirements of the problem. In general, there will be more than one acceptable design, and the purpose of optimization is to choose the best one of the many acceptable designs available. Thus a criterion has to be chosen for comparing the different alternative acceptable designs and for selecting the best one. The criterion, with respect to which the design is optimized, when expressed as a function of the design variables, is known as the criterion or merit orobjective function.

The locus of all points satisfying $f(X) = C$ = constant, forms a hyper surface in the design space, and each value of C corresponds to a different member of a family of surfaces. These surfaces, called objective function surfaces. The main problem is that as the number of design variables exceeds two or three, the constraint and objective function surfaces become complex even for visualization and the problem has to be solved purely as a mathematical problem.

1.4 Objective:

To solve a single objective optimization problem using the MATLAB Optimization Toolbox with linear programming.

1.5 Problem Statement:

Maximize $3x_1+2x_2$

Subject to

$x_1 + x_2 \leq 4$

$2x_1 + x_2 \leq 5$

$x_1, x_2 \geq 0$

1.6 Steps to Solve the Problem:

Define the Objective Function Coefficients:

The coefficients of the objective function are represented as a vector c.

F = [-3; - 2];

Define the Inequality Constraints:

The inequality constraints are represented by matrix A and vector b.

A = [1, 1;2,1];

b = [4; 5];

Define the Lower Bounds:

The lower bounds ensure that the variables x1 and x2 are non-negative.

lb = [0; 0];

Solve the Linear Programming Problem:

Use the linprog function to solve the problem.

x = linprog(f, A, b, Aeq, beq, lb, ub);

Display the Solution:

The optimal solution and the objective function value are displayed.

fprintf('Optimal solution:\n');
fprintf('x1 = %.2f\n', x(1));
fprintf('x2 = %.2f\n', x(2));

1.7 MATLAB Code:

% Define the coefficients of the objective function

f = [-3; -2]; % since we are maximizing, use -3 and -2

% Define the inequality constraints
A = [1 1; 2 1];
b = [4; 5];

```
% Define the equality constraints (none in this example)
Aeq = [ ];
beq = [ ];

% Define the bounds on the variables
lb = [0; 0];
ub = [ ];

% Solve the linear programming problem
x = linprog(f, A, b, Aeq, beq, lb, ub);
% Display the results
fprintf('Optimal solution:\n');
fprintf('x1 = %.2f\n', x(1));
fprintf('x2 = %.2f\n', x(2));
```

Explanation:

Objective Function Coefficients:
The vector f specifies the coefficients of x1 and x2 in the objective function.
Inequality Constraints:
The matrix A and vector b specify the coefficients and constants in the inequalityconstraints, respectively.
Lower Bounds:
The vector lb ensures that both x1 and x2 are greater than or equal to 0.
linprog Function:
The linprog function solves the linear programming problem by finding the optimalvalues of the decision variables x and the objective function value fval.
Displaying the Solution:
The optimal values of the decision variables and the objective function value are displayed using the disp function.

1.8 Results:
After executing the MATLAB code, the following results are obtained:
Optimal solution:
$x_1 = 1$

$x_2 = 3$

1.9 Conclusion:
In this practical, we successfully solved a single objective optimization problem using

MATLAB's Optimization Toolbox. The optimal solution was found to be x1 = 2 and x2 = 2, with an objective function value of 10. This demonstrates the effectiveness of the linprog function in solving linear programming problem.

Experiment No-2

Solving single objective optimization using OCTAVE sqp and GAMS solvers

2.1 Aim of the Experiment:

Solution of single objective optimization using OCTAVE sqp and GAMS solvers.

2.2 Theory

2.2.1 Part-A: OCTAVE solvers

Description:
GNU Octave is a scientific programming language for scientific computing and numerical computation. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB. It may also be used as a batch-oriented language. As part of the GNU Project, it is free software under the terms of the GNU General Public License. Basic features of OCTAVE are

- Powerful mathematics-oriented syntax with built-in 2D/3D plotting and visualization tools
- Free software, runs on GNU/Linux, macOS, BSD, and Microsoft Windows
- Drop-in compatible with many Matlab scripts

Installation:
Install OCTAVE from the following:

https://octave.org/

Examples:
Linear Programming using OCTAVE:

https://docs.octave.org/latest/Linear-Programming.html

Quadratic Programming using OCTAVE:

https://docs.octave.org/latest/Quadratic-Programming.html

Non-linear Programming using OCTAVE:

https://docs.octave.org/latest/Nonlinear-Programming.html

2.2.2 Part-B: GAMS

Description:
GAMS is a high level modelling system for mathematical programming and optimization. It consists of a language compiler and a range of associated solvers.

The GAMS modelling language allows modellers to quickly translate real world optimization problems into computer code. The gams language compiler then translates this code into a format the solvers can understand and solve. This architecture provides great flexibility, by allowing changing the solvers used without changing the model formulation.

Community licenses are for non-commercial, non-production use, and are a popular option for students at degree granting institutions, or for non-profit organizations.

- Generate and solve linear models (LP, MIP, and RMIP) that do not exceed 5000 variables and 5000 constraints
- For all other model types the model cannot be larger than 2500 variables and 2500 constraints
- Additional limits enforced by some solvers
- Limited to 12 months
- Cannot be combined with a professional license

Installation:
Download and install from the following location. Apply for community license using institution mail id.

https://www.gams.com/

Overview:
A GAMS model follows the basic format as given below

1. **Sets**: Announcement and Allocation of members;
2. **Data (parameters, tables, scalars)**: Announcement and Allocation of values;
3. **Variables:** Announcement and Allocation of type, upper and lower limits and/or initial values (optional);
4. **Equations:** Announcement and Definition;
5. Model and solve statements;
6. Display statements (optional)

Each GAMS model consists of the following main elements:

Sets: In models' algebraic representations, indices are defined by sets. For instance, a set of generating units, a set of slack buses, a set of network buses, a set of timeframes, etc.

Data: Each GAMS model receives its input data in the configuration of Parameters, Tables, or **Scalars.** The sets are used to define the parameters and tables. All of the scalars are single-valued variables.

Variables: Prior to solving the model, the variables, which are decision sets, are unspecified.

Equations: The relationships between the variables and data are described by the equations.

Model and Solve Statements: An objective function is contained in the collec- tion of equations that make up the model. The solve command instructs GAMS to resolve the model.

Output: The results of the solved model can be seen in a variety of ways, including by displaying them or saving them in an XLS file.

Example

The economic emission dispatch problem aims to minimize the total cost of operation and total emission of generating units to meet certain MW load demand. The objective functions and constraints are as follows.

Minimize

$$F_i = \sum_{i=1}^{N_t} a_i + b_i P_i + c_i P_i^2 + |d_i sin\left(P_i^{min} - P_i\right)^2| \quad (1)$$

$$E_i = \sum_{i=1}^{N_t} \alpha_i + \beta_i P_i + \gamma_i P_i^2 + \eta_i exp\left(\delta_i P_i\right) \quad (2)$$

Subject to

$$P_i^{min} \leq P_i \leq P_i^{max} \quad (3)$$

$$\sum_{i=1}^{N} P_i = P_D + P_L \quad (4)$$

The coefficients of a ten-generator system are provided below.

Gen#	P_i^{max} (MW)	P_i^{min} (MW)	a_i $(\$/h)$	b_i $(\$/MWh)$	$c_i(\times 0.01)$ $(\$/(MW^2h)$	d_i $\$/h$	e_i rad/MW	α_i (lb/h)	β_i (lb/MWh)	$\gamma_i(\times 0.01)$ (lb/MW^2h)	$\eta_i(\times 0.01)$ (lb/h)	$\delta_i(\times 0.01)$ $(1/MW)$
1	10	55	1000.403	40.5407	0.12951	33	0.0174	360.0012	-3.9864	0.04702	0.25475	0.01234
2	20	80	950.606	39.5804	0.10908	25	0.0178	350.0056	-3.9524	0.04652	0.25475	0.01234
3	47	120	900.705	36.5104	0.12511	32	0.0162	330.0056	-3.9023	0.04652	0.25163	0.01215
4	20	130	800.705	39.5104	0.12111	30	0.0168	330.0056	-3.9023	0.04652	0.25163	0.01215
5	50	160	756.799	38.539	0.15247	30	0.0148	13.8593	0.3277	0.0042	0.2497	0.012
6	70	240	451.325	46.1592	0.10587	20	0.0163	13.8593	0.3277	0.0042	0.2497	0.012
7	60	300	1243.531	38.3055	0.03546	20	0.0152	40.2669	-0.5455	0.0068	0.248	0.0129
8	70	340	1049.998	40.3965	0.02803	30	0.0128	40.2669	-0.5455	0.0068	0.2499	0.01203
9	135	470	1658.569	36.3278	0.02111	60	0.0136	42.8955	-0.5112	0.0046	0.2547	0.01234
10	150	470	1356.659	38.2704	0.01799	40	0.0141	42.8955	-0.5112	0.0046	0.2547	0.01234

GAMS code to solve the problem is **EED_10G.gms**.

The Solution is obtained from **EED_10G.lst** which is obtained after running the GAMS code.

$title Economic Emission Dispatch 10 Generator.

2.3 MATLAB Code for EED_10G.gms

Set i Generating units / 1*10 /;

alias (i,j);

Parameter report(*,*);

Scalar load /2000/;

Table data(i,*)

a b c d1 e1 d e f g h Pmin Pmax

1	1000.403	40.5407	0.12951	33	0.0174	360.0012	-3.9864	0.04702	0.25475	0.01234	10	55
2	950.606	39.5804	0.10908	25	0.0178	350.0056	-3.9524	0.04652	0.25475	0.01234	20	80
3	900.705	36.5104	0.12511	32	0.0162	330.0056	-3.9023	0.04652	0.25163	0.01215	47	120
4	800.705	39.5104	0.12111	30	0.0168	330.0056	-3.9023	0.04652	0.25163	0.01215	20	130
5	756.799	38.539	0.15247	30	0.0148	13.8593	0.3277	0.0042	0.2497	0.012	50	160
6	451.325	46.1592	0.10587	20	0.0163	13.8593	0.3277	0.0042	0.2497	0.012	70	240
7	1243.531	38.3055	0.03546	20	0.0152	40.2669	-0.5455	0.0068	0.248	0.0129	60	300
8	1049.998	40.3965	0.02803	30	0.0128	40.2669	-0.5455	0.0068	0.2499	0.01203	70	340
9	1658.569	36.3278	0.02111	60	0.0136	42.8955	-0.5112	0.0046	0.2547	0.01234	135	470
10	1356.659	38.2704	0.01799	40	0.0141	42.8955	-0.5112	0.0046	0.2547	0.01234	150	470;

Table Bcoef(i,j)

	1	2	3	4	5	6	7	8	9	10
1	0.000049	0.000014	0.000015	0.000015	0.000016	0.000017	0.000017	0.000018	0.000019	0.000020
2	0.000014	0.000045	0.000016	0.000016	0.000017	0.000015	0.000015	0.000016	0.000018	0.000018
3	0.000015	0.000016	0.000039	0.000010	0.000012	0.000012	0.000014	0.000014	0.000016	0.000016
4	0.000015	0.000016	0.000010	0.000040	0.000014	0.000010	0.000011	0.000012	0.000014	0.000015
5	0.000016	0.000017	0.000012	0.000014	0.000035	0.000011	0.000013	0.000013	0.000015	0.000016
6	0.000017	0.000015	0.000012	0.000010	0.000011	0.000036	0.000012	0.000012	0.000014	0.000015
7	0.000017	0.000015	0.000014	0.000011	0.000013	0.000012	0.000038	0.000016	0.000016	0.000018
8	0.000018	0.000016	0.000014	0.000012	0.000013	0.000012	0.000016	0.000040	0.000015	0.000016
9	0.000019	0.000018	0.000016	0.000014	0.000015	0.000014	0.000016	0.000015	0.000042	0.000019
10	0.000020	0.000018	0.000016	0.000015	0.000016	0.000015	0.000018	0.000016	0.000019	0.000044;

```
Variable P(i),Cost,Emission,Ploss,TR;
P.lo(i) = data(i,'Pmin');
P.up(i) = data(i,'Pmax');
Equation eq1, eq2, eq3, eq4;
eq1.. Cost =e= sum(i,data(i,'a')+ data(i,'b')*P(i)+ data(i,'c')*P(i)*P(i)
        + abs(data(i,'d1')*sin(data(i,'e1')*((data(i,'Pmin')-P(i))))));
eq2.. Emission =e= sum(i, data(i,'d') + data(i,'e')*P(i) + data(i,'f')*P(i)*P(i)
          + (data(i,'g')*exp(data(i,'h')*P(i))));
```

```
eq3.. sum(i,P(i))-sum((i,j),P(i)*Bcoef(i,j)*P(j)) =g= load;
eq4.. Ploss =e= sum((i,j),P(i)*Bcoef(i,j)*P(j));
Model EED_10G / eq1, eq2, eq3, eq4/;
option dnlp=CONOPT;
solve EED_10G using dnlp minimizing Cost;
report('MC','maxTE') = Emission.l;
report('MC','minTC') = Cost.l;
TR.l=EED_10G.etSolver;
report('MC','minTC_time')=TR.l;
display 'FOR MINIMUM COST';
display Cost.l, Emission.l, Ploss.l, P.l;
solve EED_10G using dnlp minimizing Emission;
report('ME','maxTC') = Cost.l;
report('ME','minTE') = Emission.l;
TR.l=EED_10G.etSolver;
report('ME','minTE_time')=TR.l;
display 'FOR MINIMUM EMISSION';
display Cost.l, Emission.l, Ploss.l, P.l;
display report;
```

2.4 Conclusion

Write what conclusion you have drawn from the experiments performed.

Experiment 3 & 4

Design a Fuzzy Logic Controller (FLC) for a DC motor control using MATLAB/SIMULINK Toolbox.

3.1 Aim of the Experiment

To design a Fuzzy Logic Controller (FLC) for a DC motor control using MATLAB/SIMULINK Toolbox.

3.2 Software/tools required:

MATLAB/ Simulink

3.3 Theory

Fuzzy logic was developed by Lotfi A. Zadeh in the 1960s in order to provide mathematical rules and functions which permitted natural language queries. Fuzzy logic provides a means of calculating intermediate values between absolute true and absolute false with resulting values ranging between 0.0 and 1.0. With fuzzy logic, it is possible to calculate the degree to which an item is a member. For example, if a person is .83 of tallness, they are "rather tall. " Fuzzy logic calculates the shades of gray between black/white and true/false.

Fuzzy logic is a super set of conventional (or Boolean) logic and contains similarities and differences with Boolean logic. Fuzzy logic is similar to Boolean logic, in that Boolean logic results are returned by fuzzy logic operations when all fuzzy memberships are restricted to 0 and 1. Fuzzy logic differs from Boolean logic in that it is permissive of natural language queries and is more like human thinking; it is based on degrees of truth.
The graphical representation of fuzzy and boolean sets are different as well.

Fuzzy Sets

A fuzzy set is a pair (A,m) where A is a set and m : A -->[0,1].
For each x ε A, m(x) is called the grade of membership of x in (A,m). For a finite set A = {x1,...,xn}, the fuzzy set (A,m) is often denoted by {m(x1) / x1,...,m(xn) / xn}.
Let x ε A. Then x is called not included in the fuzzy set (A,m) if m(x) = 0, x is called fully included if m(x) = 1, and x is called a fuzzy member if 0 < m(x) < 1.The set { x ε A | m(x) > 0 } is called the support of (A,m) and the set { x ε A | m(x) =1 } is called its kernel.

Fuzzy Set Operations:

Fuzzy Addition

Let us consider A_1 = [a,b] and A_2 = [c,d]
The addition of A_1 and A_2 is: [a,b] + [c,d] = [a+c, b+d]

Fuzzy Substraction

Let us consider A_1 = [a,b] and A_2 = [c,d]
The substraction of A_1 and A_2 is: [a,b] - [c,d] = [a-d, b-c]

Fuzzy Complement

The degree to which you believe something is not in the set is 1.0 minus the degree to which you believe it is in the set.

Fuzzy Intersection

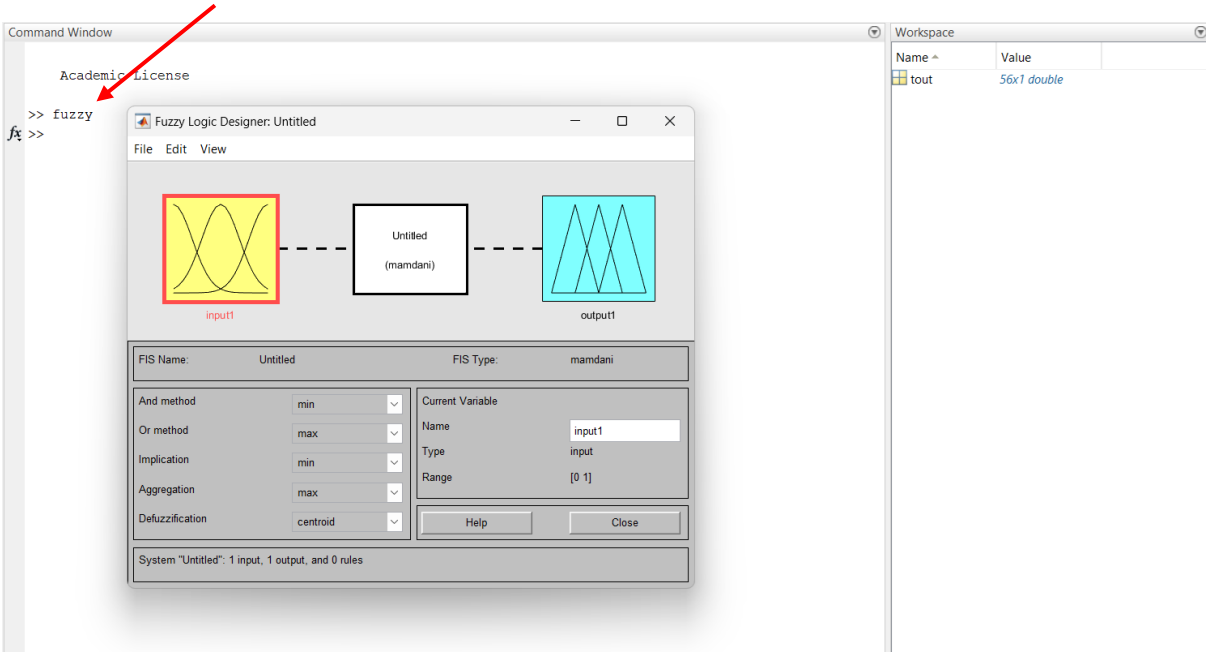
If you have x degree of faith in statement A, and y degree of faith in statement B, how much faith do you have in the statement A and B?
Eg: How much faith in "that person is about 6' high and tall"

Fuzzy Union

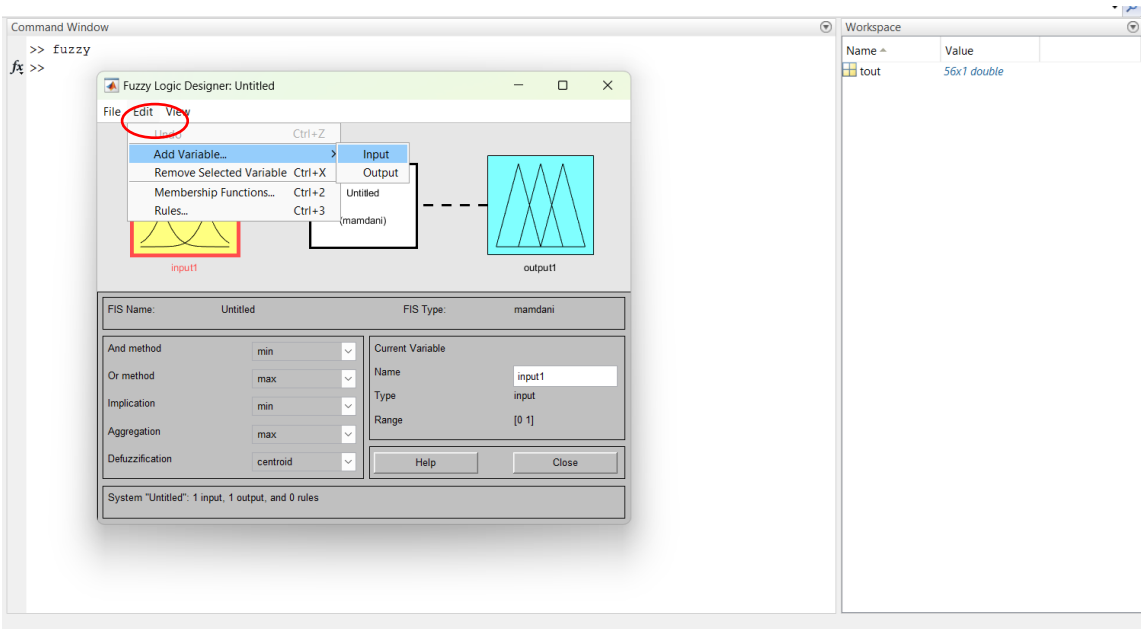
If you have x degree of faith in statement A, and y degree of faith in statement B, how much faith do you have in the statement A or B?
Eg: How much faith in "that person is about 6' high or tall"

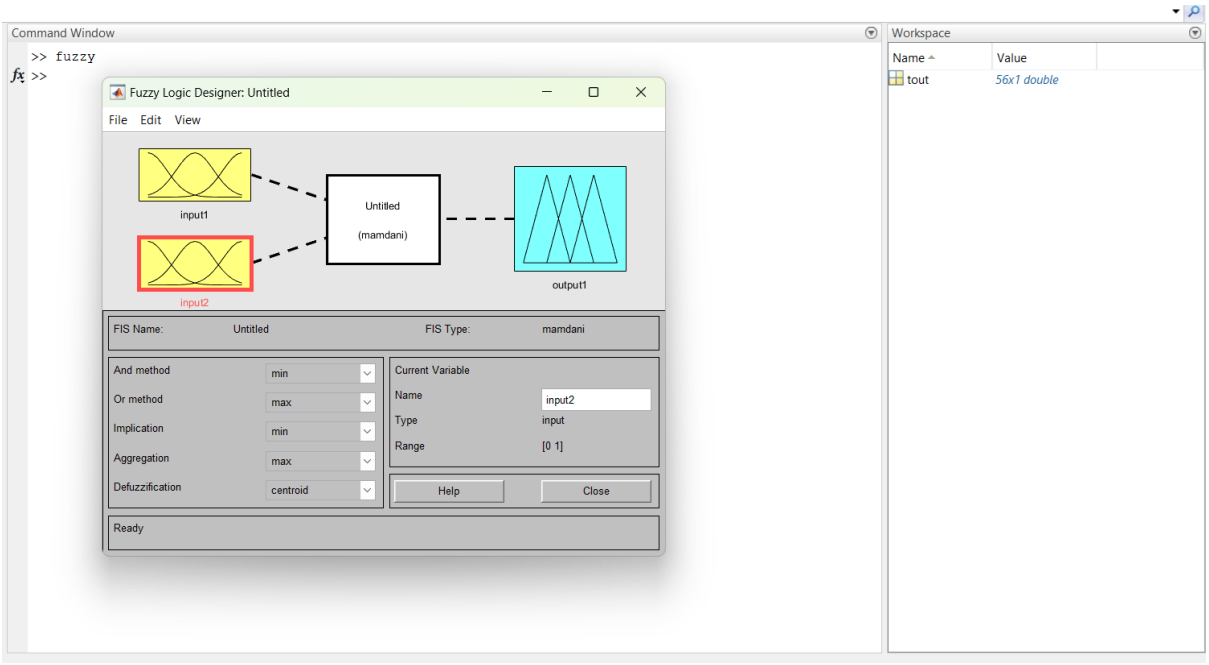
3.4 Steps:

1. Write fuzzy in Command Window.

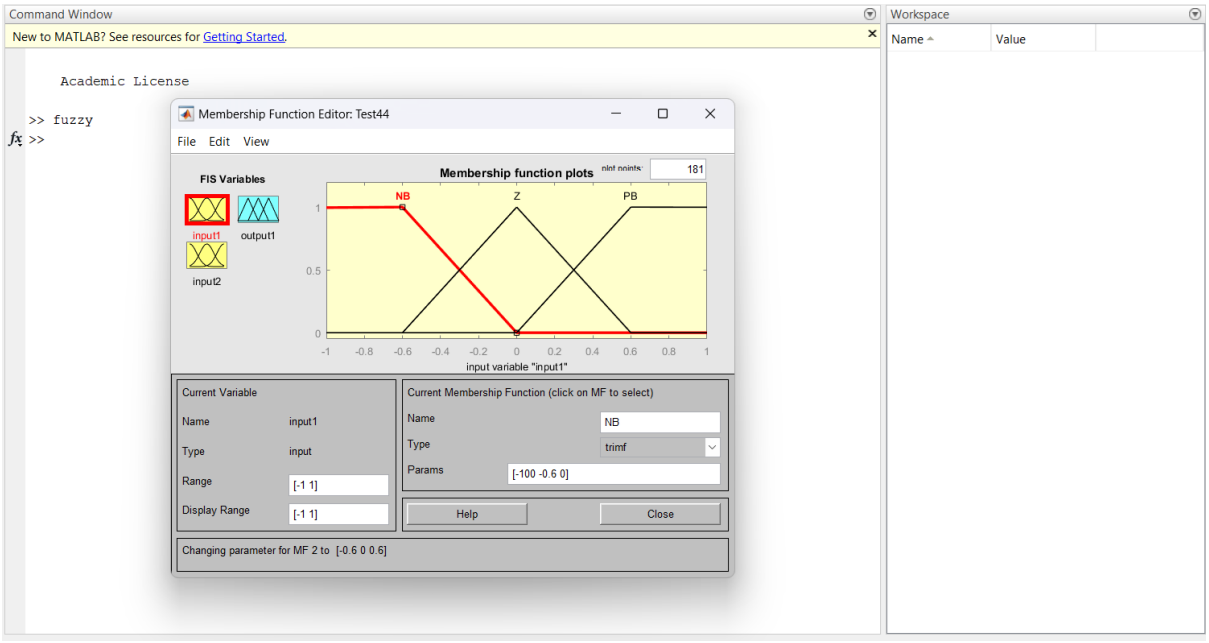


2. Go to Edit.
3. Add Variables/Input.





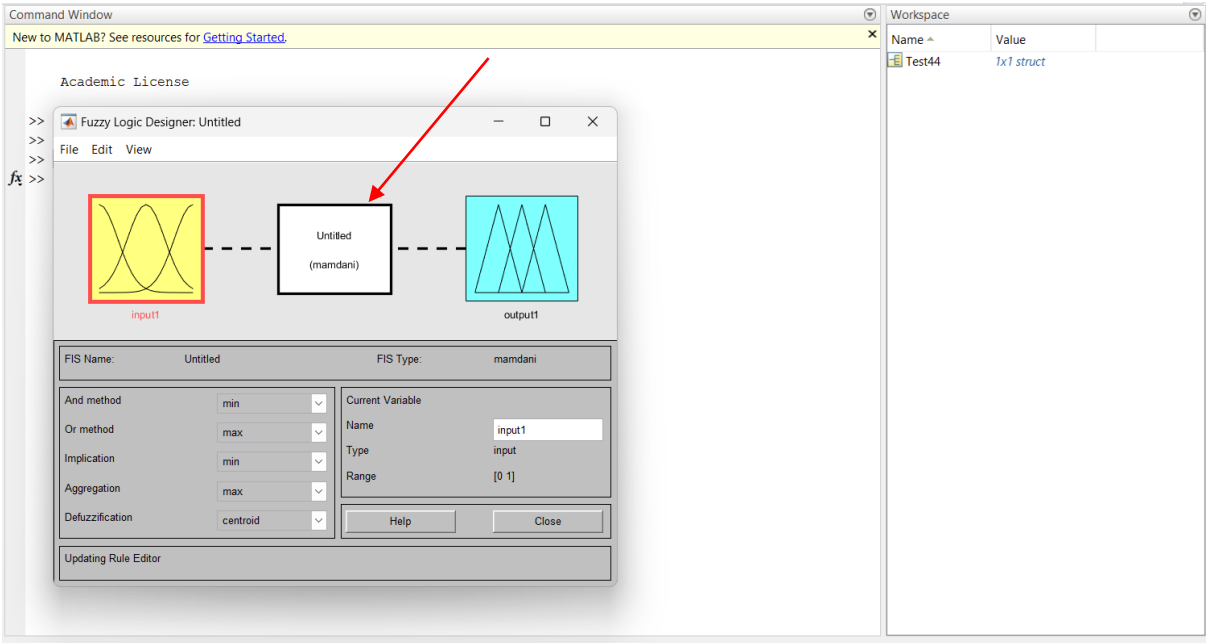
4. Double click on the input 1.
5. Set the range and Display range [-1 1] respectively.
6. Select mf1.



7. Give its name and Params.
8. Select mf2 and mf3.
9. Give Theirs name and Params.

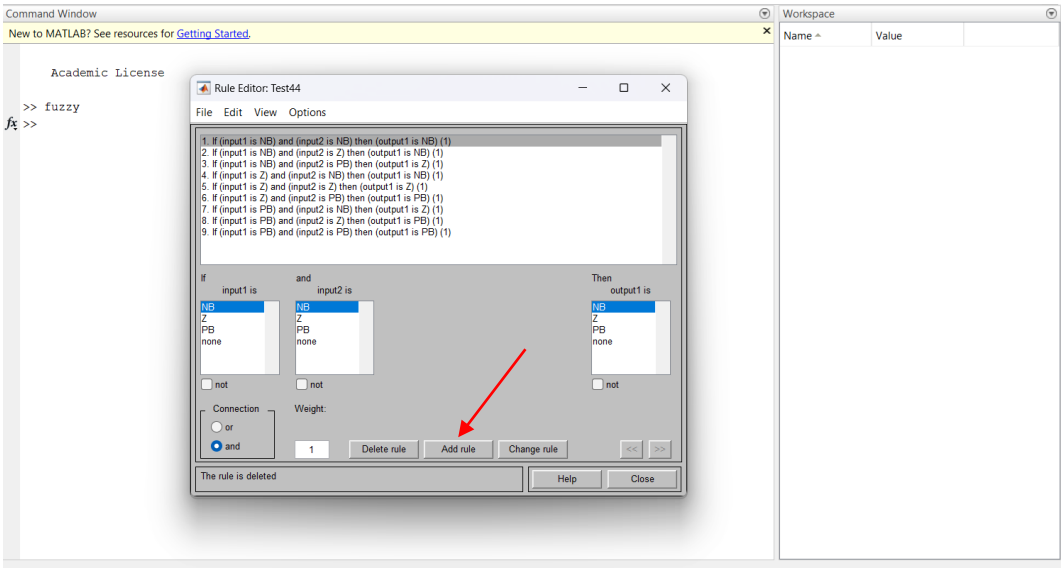
Membership function	**Name**	**Params**
mf1	NB	[-100 -0.6 0]
mf2	Z	[-0.6 0 0.6]
mf3	PB	[0 0.6 500]

10. Do the same for input2 and output1.
11. Double click on Untitled (mamdani) to add rule base.

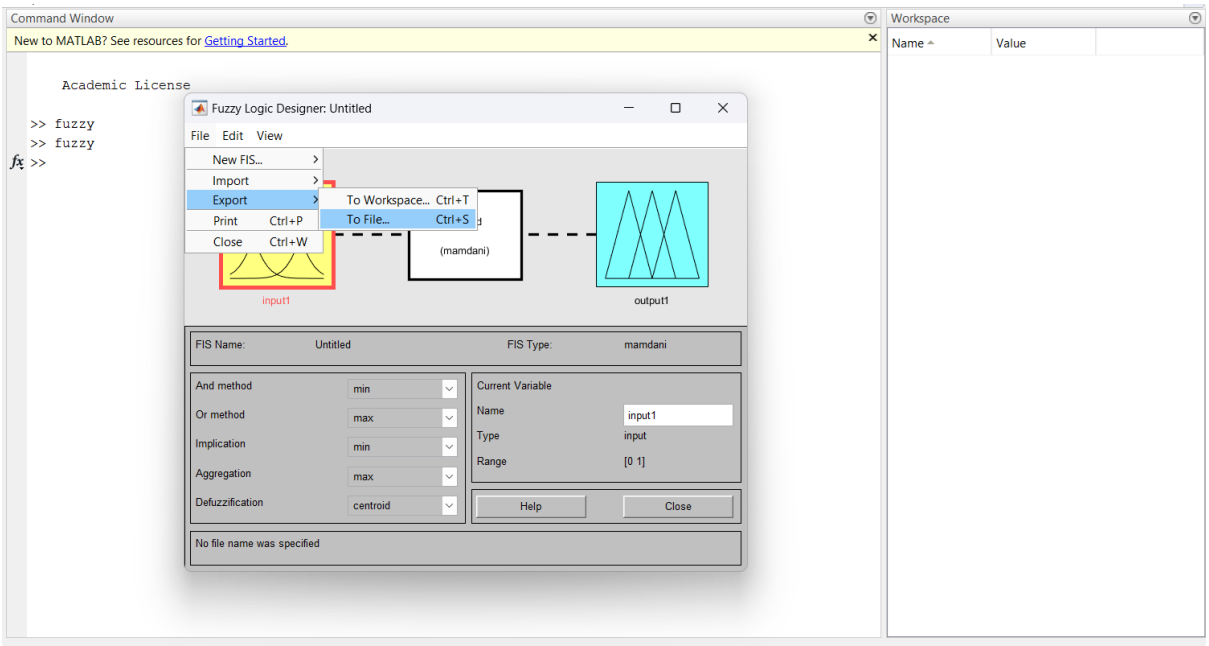


12. Add the rule base following the mentioned table.

e / Δe	NB	Z	PB
NB	NB	NB	Z
Z	NB	Z	PB
PB	Z	PB	PB

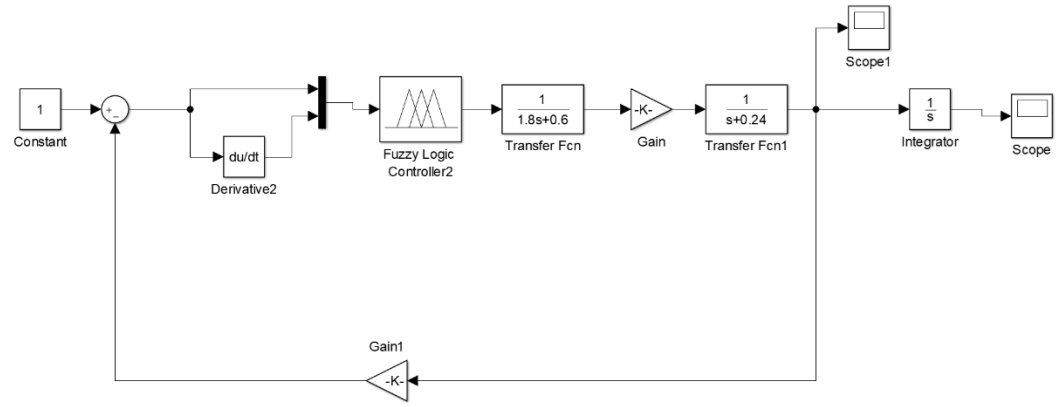


13. File/Export/To file/ Give a file name.



14. Go to Simulink Library and type Fuzzy logic Controller (FLC)

15. Drag the Fuzzy logic Controller to the Simulink model.



16. Double click on the Fuzzy logic Controller and Give the file name same as the .fis file name.
17. Put the Simulink file and FLC file (.fis) in one folder.
18. Go to Command window and write the command Test44=readfis('Test44');
19. This will run the .fis file and load the rule base for the FLC.
20. Run the Simulink file

3.5 Conclusion

Write what conclusion you have drawn from the experiments performed.

Experiment 5

Implementation of Genetic Algorithms to solve an optimization problem.

5.1 Aim of the Experiment

Solution of single objective optimization problem using Genetic algorithm in MATLAB.

5.2 Software/tools required:

MATLAB/ optimization tool

5.3 Theory

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution.

5.4 Problem Statement:

Objective: Minimize the function

$$f(x) = \left(4 - 2{x_1}^2 + {x_1}^{\frac{4}{3}}\right){x_1}^2 + x_1x_2 + (-4 + 4{x_2}^2){x_2}^2$$

Constraints:

1. $x_1x_2 + x_1 - x_2 + 1.5 \leq 0$ (Nonlinear constraint)
2. $10 - x_1x_2 \leq 0$ (Nonlinear constraint)
3. $0 \leq x_1 \leq 1,\ 0 \leq x_2 \leq 13$ (Boundary)

5.5 MATLAB Code:

%Create a MATLAB file named **simple_objective.m** containing the following code:

```
function y = simple_objective(x)
   y = (4 - 2.1*x(1) ^2 + x(1) ^4/3)*x(1) ^2 + x(1)*x(2) + (-4 + 4*x(2) ^2)*x(2) ^2;
end
```

%Create a MATLAB file named **simple_constraint.m** containing the following code:

function [c, ceq] = simple_constraint(x) % SIMPLE_CONSTRAINT nonlinear inequality constraints.

```
    c = [1.5 + x(1)*x(2) + x(1) - x(2);
          -x(1)*x(2) + 10];
    ceq = [ ]; % No nonlinear equality constraints:
    end
% Create a new MATLAB file and specify the objective function
ObjectiveFunction = @simple_objective;

% Set bounds for the variables
lb = [0 0];
ub = [1 13];
%Specify the nonlinear constraint function
ConstraintFunction = @simple_constraint;

%Specify number of problem variable
nvars = 2;
rng default   % for reproducibility

% Solve the genetic algorithm problem
 [x, fval] = ga(ObjectiveFunction, nvars, [ ], [ ], [ ], [ ], lb, ub, ConstraintFunction)

%for visualization
options = optimoptions ('ga', 'PlotFcn', {@gaplotbestf, @gaplotmaxconstr},…
    'Display','iter');
[x, fval] = ga(ObjectiveFunction, nvars, [ ], [ ], [ ], [ ], lb, ub, …
    ConstraintFunction, options)
```

5.6 Results:

Upon running the MATLAB code, the optimal solution was found to be:
Optimal solution:
x = [0.8123; 12.3104]
Optimal objective function value:
f(x) = 9.1270e+04
Output details:
Additional information regarding the optimization process, such as the number of iterations and algorithm details.

5.7 Conclusion

Write what conclusion you have drawn from the experiments performed.

Experiment 6

Implementation of Artificial Neural Networks to Solve Optimization Problem

6.1 Aim of the Experiment

To implement and solve an optimization problem using Artificial Neural Networks (ANNs) in MATLAB. Develop an ANN model to find the optimal solution for a given problem.

6.2 Requirements:

- MATLAB software with Neural Network Toolbox installed
- Basic knowledge of neural networks and optimization techniques

6.3 Theory:

Artificial Neural Networks: Artificial Neural Networks (ANNs) are computational models inspired by the human brain. They consist of interconnected processing elements called neurons that work together to solve specific problems. ANNs are particularly useful for pattern recognition, regression, classification, and optimization tasks.

1. Components of ANNS:

a. Neurons: The basic units of an ANN, which process input signals and produce output signals.
b. Layers: ANNs are organized into layers: input layer, hidden layers, and output layer.
c. Weights: Connections between neurons have associated weights that are adjusted during training.
d. Activation Function: A function that determines the output of a neuron based on its input.

2. Training ANNs

Training an ANN involves adjusting the weights to minimize the difference between the predicted output and the actual output. This is typically done using a back propagation algorithm and gradient descent optimization.

6.4 Procedure:

Step I: Define the Optimization Problem**:** Consider an optimization problem where we want to approximate the function /(z) = sin (x) over the interval [0, 2z] using an ANN.

Step 2: Generate Training Data: Generate input and output data for training the ANN.

```
x = linspace(0, 2*pi, 100)';
y = sin(x);
```

Step 3: Create and Train the ANN:

a. Create the ANN: Use the *'feedforwardnet'* function to create a feedforward neural network with one hidden layer.

```
hiddenLayerSize = 10;
```

net = feedforwardnet(hiddenLayerSize);

b. Configure the ANN: Configure the ANN with the training data.
net = configure(net, x, y);

c. Train the ANN: Train the ANN using the training data.
[net, tr] = train(net, x, y);

Step 4: Evaluate the ANN**:**

a. Simulate the ANN: Use the trained ANN to predict outputs for new inputs.
y_pred = net(x);

b. Plot Results: Plot the original function and the ANN's approximation.

```
figure;
plot(x, y, 'b-', x, y_pred, 'r--');
legend('True Function', 'ANN Approximation');
title('Function Approximation using ANN');
```

6.5 Complete Code:

```
%Generate Training Data
x = linspace(0, 2*pi, 100)';
y = sin(x);
% Create and Configuration the ANN
hiddenLayerSize = 10;
net = feedforwardnet(hiddenLayerSize);
net = configure(net, x, y);
% Train the ANN
[net, tr] = train(net, x, y);
% Simulate the ANN
y_pred = net(x);
% Plot Results
figure;
plot(x, y, 'b-', x, y_pred, 'r--');
legend('True Function', 'ANN Approximation');
title('Function Approximation using ANN');
```

6.6 Conclusion:

In this experiment, we implemented an Artificial Neural Network to solve an optimization problem using MATLAB. By generating training data, creating and configuring the ANN, and training it with the data, we were able to approximate a given function. This experiment demonstrated the effectiveness of ANNs in solving complex optimization problems, showcasing their ability to learn and generalize from data.

Experiment 7

Implementation of Particle Swarm Optimization to solve optimization problems.

7.1 Aim of the Experiment

Solution of single objective optimization problem using particle swarm algorithm in MATLAB.

7.2 Software/tools required:

MATLAB/ optimization tool

7.3 Theory

In computational science, particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

x = particleswarm (fun, nvars)

x = particleswarm (fun, nvars, lb, ub)

x = particleswarm(fun, nvars, lb, ub, options)

x = particleswarm (problem)

[x, fval, exitflag, output, points] = particleswarm (___)

Description

x = particleswarm (fun, nvars) attempts to find a vector x that achieves a local minimum of fun. nvars is the dimension (number of design variables) of fun.

7.4 Problem statement

Minimize a function of two variables

$$f(x) = \frac{20 + 500x_2}{{x_1}^{1.2+3x_2}}$$

Constraints:

1. $1 \leq x_1 \leq 4$
2. $0.1 \leq x_2 \leq 0.4$

7.5 MATLAB code

```
% Define the objective function
f = @(x)(20 + 500*x(2)). / x(1).^(1.2 + 3*x(2));
% Define lower and upper bounds
lb = [1 0.1];
ub = [4 0.4];
% Number of variables
nvars = 2;
rng default   % for reproducibility
% Use particleswarm to minimize the function
[x, fval] = particleswarm (f, nvars, lb, ub);
% Display the results
disp ('Optimal solution found :');
disp(x);
disp ('Objective function value at the optimal solution :');
disp (fval);
%further customize the optimization by adjusting PSO parameters or by using a hybrid approach. For example, you might want to use 'fmincon' to refine the solution after PSO converges
options = optimoptions ('particleswarm', 'HybridFcn', @fmincon);
[x, fval] = particleswarm (f, nvars, lb, ub, options);
```

7.6 Result

Optimal solution found:

4.0000 0.4000

Objective function value at the optimal solution:

7.8973

7.7 Conclusion

Write what conclusion you have drawn from the experiments performed.