**1.Design a command-line calculator that performs arithmetic operations (addition, subtraction, multiplication, division) on fuzzy numbers. Implement fuzzy arithmetic operations using appropriate fuzzy logic rules and membership functions. Test the calculator with different fuzzy numbers and evaluate the accuracy of the results.**

import numpy as np

class FuzzyNumber:

def \_\_init\_\_(self, value, membership\_function):

self.value = value

self.membership\_function = membership\_function

def fuzzy\_addition(a, b):

result\_value = a.value + b.value

result\_membership\_function = np.minimum(a.membership\_function, b.membership\_function)

return FuzzyNumber(result\_value, result\_membership\_function)

def fuzzy\_subtraction(a, b):

result\_value = a.value - b.value

result\_membership\_function = np.minimum(a.membership\_function, 1 - b.membership\_function)

return FuzzyNumber(result\_value, result\_membership\_function)

def fuzzy\_multiplication(a, b):

result\_value = a.value \* b.value

result\_membership\_function = np.minimum(np.maximum(a.membership\_function, b.membership\_function),

np.minimum(a.membership\_function, b.membership\_function))

return FuzzyNumber(result\_value, result\_membership\_function)

def fuzzy\_division(a, b):

if b.value == 0:

raise ValueError("Division by zero is undefined.")

result\_value = a.value / b.value

result\_membership\_function = np.minimum(a.membership\_function, 1 / b.membership\_function)

return FuzzyNumber(result\_value, result\_membership\_function)

**# Example usage**

a = FuzzyNumber(5, np.array([0, 0.4, 0.8, 1, 0.6, 0.2, 0]))

b = FuzzyNumber(3, np.array([0, 0.2, 0.6, 1, 0.4, 0, 0]))

result\_addition = fuzzy\_addition(a, b)

result\_subtraction = fuzzy\_subtraction(a, b)

result\_multiplication = fuzzy\_multiplication(a, b)

result\_division = fuzzy\_division(a, b)

print("Fuzzy Addition:", result\_addition.value)

print("Fuzzy Subtraction:", result\_subtraction.value)

print("Fuzzy Multiplication:", result\_multiplication.value)

print("Fuzzy Division:", result\_division.value)

**Output:-**

**Fuzzy Addition: 8**

**Fuzzy Subtraction: 2**

**Fuzzy Multiplication: 15**

**Fuzzy Division: 1.6666666666666667**

**4.Design and implement a single-layer perceptron from scratch using Python. Train the**

**perceptron on a binary classification problem**

import numpy as np

class SingleLayerPerceptron:

def \_\_init\_\_(self, input\_size):

**# Initialize weights and bias**

self.weights = np.zeros(input\_size)

self.bias = 0

def predict(self, inputs):

**# Calculate the weighted sum and apply step function**

linear\_output = np.dot(inputs, self.weights) + self.bias

prediction = np.where(linear\_output >= 0, 1, 0)

return prediction

def train(self, inputs, labels, learning\_rate=0.1, epochs=100):

for epoch in range(epochs):

for input\_data, label in zip(inputs, labels):

prediction = self.predict(input\_data)

**# Update weights and bias based on prediction error**

update = learning\_rate \* (label - prediction)

self.weights += update \* input\_data

self.bias += update

**# Print accuracy for each epoch**

accuracy = self.evaluate(inputs, labels)

print(f"Epoch {epoch + 1}/{epochs}, Accuracy: {accuracy:.2%}")

def evaluate(self, inputs, labels):

predictions = np.array([self.predict(input\_data) for input\_data in inputs])

accuracy = np.mean(predictions == labels)

return accuracy

**# Example usage**

if \_\_name\_\_ == "\_\_main\_\_":

**# Generating some random data for binary classification**

np.random.seed(42)

input\_size = 2

num\_samples = 100

inputs = np.random.rand(num\_samples, input\_size)

labels = np.random.randint(2, size=num\_samples)

**# Instantiate and train the perceptron**

perceptron = SingleLayerPerceptron(input\_size)

perceptron.train(inputs, labels, learning\_rate=0.1, epochs=100)

**# Evaluate accuracy on the training data**

accuracy = perceptron.evaluate(inputs, labels)

print(f"Final Accuracy: {accuracy:.2%}")

**Output:-**

Epoch 1/100, Accuracy: 51.00%

Epoch 2/100, Accuracy: 55.00%

Epoch 3/100, Accuracy: 55.00%

Epoch 4/100, Accuracy: 53.00%

……

……

……

……

……

Epoch 99/100, Accuracy: 53.00%

**Final Accuracy: 54.00%**

**5. Develop a Multi-Layer Perceptron (MLP) for any real world problem.**

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score

**# Load Iris dataset**

iris = load\_iris()

X = iris.data

y = iris.target

**# Split the dataset into training and testing sets**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**# Standardize features**

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**# Create and train MLP model**

mlp = MLPClassifier(hidden\_layer\_sizes=(5, 1), max\_iter=1000, random\_state=42)

mlp.fit(X\_train, y\_train)

**# Make predictions on the test set**

y\_pred = mlp.predict(X\_test)

**# Evaluate accuracy**

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

**output:-Accuracy: 0.9333333333333333**

**3.Design a fuzzy logic controller for a washing machine that adjusts the wash cycle based on the level of dirtiness and fabric type. Define fuzzy sets and membership functions for dirtiness level (e.g. low, medium, high) and fabric type (e.g., delicate, cotton, heavy-duty). Create fuzzy rules to determine the wash cycle duration, water temperature, and detergent amount based on dirtiness level and fabric type. Implement the fuzzy logic controller and evaluate its effectiveness in achieving clean and undamaged clothes in python.**

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

import matplotlib.pyplot as plt

# Define input variables

dirtiness\_level = ctrl.Antecedent(np.arange(0, 11, 1), 'dirtiness\_level')

fabric\_type = ctrl.Antecedent(np.arange(0, 11, 1), 'fabric\_type')

# Define output variables

wash\_cycle\_duration = ctrl.Consequent(np.arange(0, 61, 1), 'wash\_cycle\_duration')

water\_temperature = ctrl.Consequent(np.arange(0, 101, 1), 'water\_temperature')

detergent\_amount = ctrl.Consequent(np.arange(0, 101, 1), 'detergent\_amount')

# Define fuzzy sets and membership functions

dirtiness\_level['low'] = fuzz.trimf(dirtiness\_level.universe, [0, 0, 5])

dirtiness\_level['medium'] = fuzz.trimf(dirtiness\_level.universe, [0, 5, 10])

dirtiness\_level['high'] = fuzz.trimf(dirtiness\_level.universe, [5, 10, 10])

fabric\_type['delicate'] = fuzz.trimf(fabric\_type.universe, [0, 0, 5])

fabric\_type['cotton'] = fuzz.trimf(fabric\_type.universe, [0, 5, 10])

fabric\_type['heavy\_duty'] = fuzz.trimf(fabric\_type.universe, [5, 10, 10])

# Define fuzzy sets and membership functions for outputs

wash\_cycle\_duration['short'] = fuzz.trimf(wash\_cycle\_duration.universe, [0, 0, 30])

wash\_cycle\_duration['medium'] = fuzz.trimf(wash\_cycle\_duration.universe, [0, 30, 60])

wash\_cycle\_duration['long'] = fuzz.trimf(wash\_cycle\_duration.universe, [30, 60, 60])

water\_temperature['cold'] = fuzz.trimf(water\_temperature.universe, [0, 0, 50])

water\_temperature['warm'] = fuzz.trimf(water\_temperature.universe, [0, 50, 100])

water\_temperature['hot'] = fuzz.trimf(water\_temperature.universe, [50, 100, 100])

detergent\_amount['low'] = fuzz.trimf(detergent\_amount.universe, [0, 0, 50])

detergent\_amount['medium'] = fuzz.trimf(detergent\_amount.universe, [0, 50, 100])

detergent\_amount['high'] = fuzz.trimf(detergent\_amount.universe, [50, 100, 100])

# Define fuzzy rules

rule1 = ctrl.Rule(dirtiness\_level['low'] & fabric\_type['delicate'],

(wash\_cycle\_duration['short'], water\_temperature['cold'], detergent\_amount['low']))

rule2 = ctrl.Rule(dirtiness\_level['medium'] & fabric\_type['delicate'],

(wash\_cycle\_duration['medium'], water\_temperature['warm'], detergent\_amount['medium']))

rule3 = ctrl.Rule(dirtiness\_level['high'] & fabric\_type['delicate'],

(wash\_cycle\_duration['long'], water\_temperature['hot'], detergent\_amount['high']))

rule4 = ctrl.Rule(dirtiness\_level['low'] & fabric\_type['cotton'],

(wash\_cycle\_duration['short'], water\_temperature['warm'], detergent\_amount['medium']))

rule5 = ctrl.Rule(dirtiness\_level['medium'] & fabric\_type['cotton'],

(wash\_cycle\_duration['medium'], water\_temperature['hot'], detergent\_amount['high']))

rule6 = ctrl.Rule(dirtiness\_level['high'] & fabric\_type['cotton'],

(wash\_cycle\_duration['long'], water\_temperature['hot'], detergent\_amount['high']))

rule7 = ctrl.Rule(dirtiness\_level['low'] & fabric\_type['heavy\_duty'],

(wash\_cycle\_duration['medium'], water\_temperature['warm'], detergent\_amount['high']))

rule8 = ctrl.Rule(dirtiness\_level['medium'] & fabric\_type['heavy\_duty'],

(wash\_cycle\_duration['long'], water\_temperature['hot'], detergent\_amount['high']))

rule9 = ctrl.Rule(dirtiness\_level['high'] & fabric\_type['heavy\_duty'],

(wash\_cycle\_duration['long'], water\_temperature['hot'], detergent\_amount['high']))

# Create control system

washing\_machine\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9])

washing\_machine\_simulation = ctrl.ControlSystemSimulation(washing\_machine\_ctrl)

# Run simulation with specific inputs

washing\_machine\_simulation.input['dirtiness\_level'] = 7

washing\_machine\_simulation.input['fabric\_type'] = 8

washing\_machine\_simulation.compute()

# Print simulation results

print("Wash Cycle Duration:", washing\_machine\_simulation.output['wash\_cycle\_duration'])

print("Water Temperature:", washing\_machine\_simulation.output['water\_temperature'])

print("Detergent Amount:", washing\_machine\_simulation.output['detergent\_amount'])

# Visualize membership functions

dirtiness\_level.view()

fabric\_type.view()

wash\_cycle\_duration.view()

water\_temperature.view()

detergent\_amount.view()

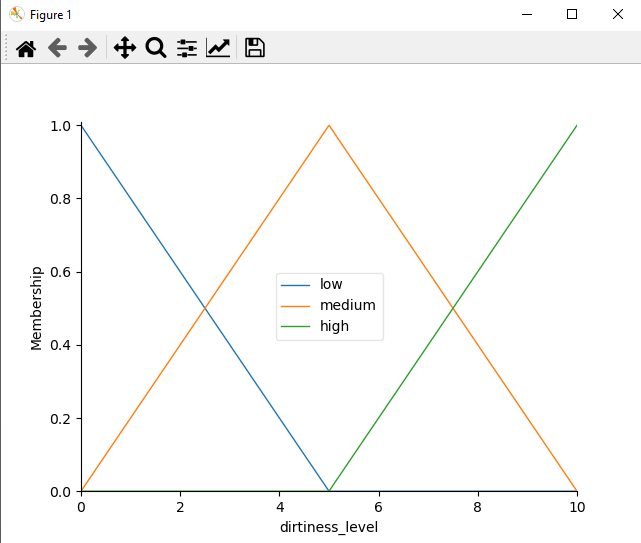
plt.show()

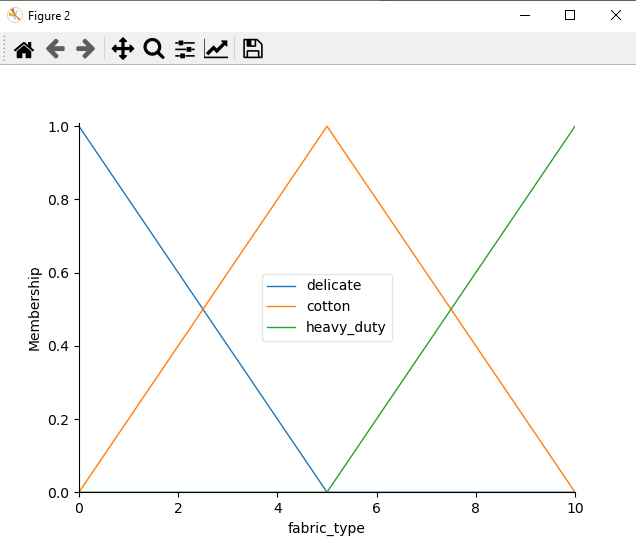
**Output:-** **Wash Cycle Duration: 35.268292682926806**

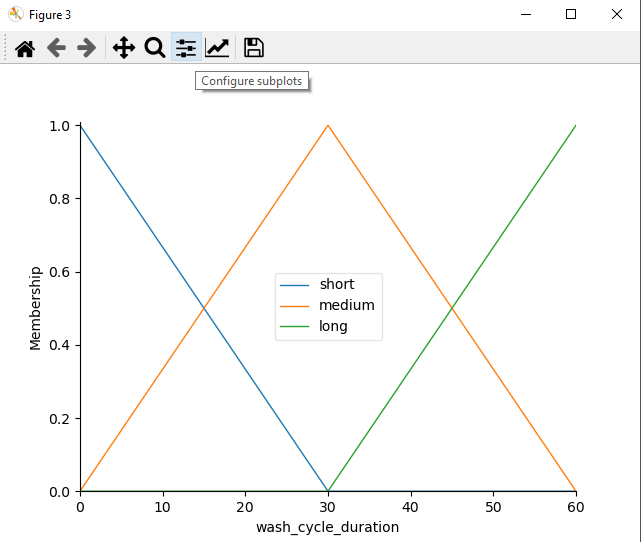
**Water Temperature: 81.42857142857139**

**Detergent Amount: 81.42857142857139**

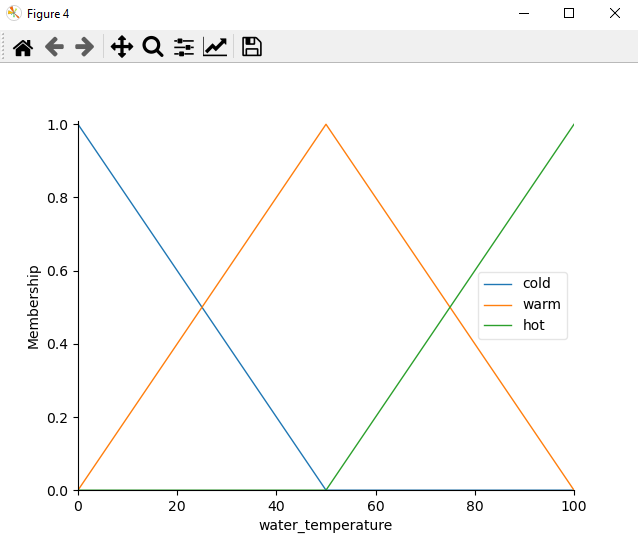
**Figure 1:**

****

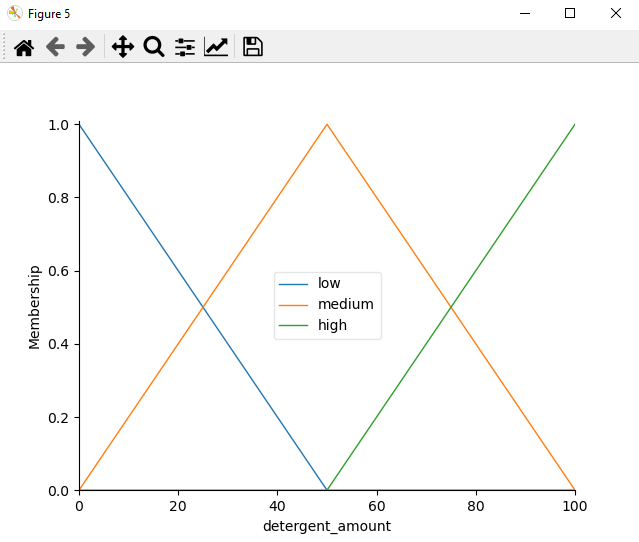
**Figure 2: **

**Figure 3:**

**Figure 4:**

****

**Figure 5:**

****

**2.Develope a simulation of a fuzzy traffic light controller for a busy intersection. Define fuzzy sets and membership functions for traffic flow (e.g. low, medium, high) and waiting time. Design fuzzy rules to determine the duration of green, yellow, and red lights based on traffic flow and waiting time. Simulate the traffic light controller and analyze its performance in terms of traffic congestion and waiting times.**

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

import matplotlib.pyplot as plt

# Define input variables

traffic\_flow = ctrl.Antecedent(np.arange(0, 101, 1), 'traffic\_flow')

waiting\_time = ctrl.Antecedent(np.arange(0, 61, 1), 'waiting\_time')

# Define output variable

light\_duration = ctrl.Consequent(np.arange(0, 101, 1), 'light\_duration')

# Define fuzzy sets and membership functions

traffic\_flow['low'] = fuzz.trimf(traffic\_flow.universe, [0, 20, 40])

traffic\_flow['medium'] = fuzz.trimf(traffic\_flow.universe, [20, 40, 60])

traffic\_flow['high'] = fuzz.trimf(traffic\_flow.universe, [40, 60, 100])

waiting\_time['low'] = fuzz.trimf(waiting\_time.universe, [0, 10, 20])

waiting\_time['medium'] = fuzz.trimf(waiting\_time.universe, [10, 20, 30])

waiting\_time['high'] = fuzz.trimf(waiting\_time.universe, [20, 30, 60])

light\_duration['green'] = fuzz.trimf(light\_duration.universe, [0, 30, 60])

light\_duration['yellow'] = fuzz.trimf(light\_duration.universe, [30, 60, 90])

light\_duration['red'] = fuzz.trimf(light\_duration.universe, [60, 90, 100])

# Define fuzzy rules

rule1 = ctrl.Rule(traffic\_flow['low'] | waiting\_time['low'], light\_duration['green'])

rule2 = ctrl.Rule(traffic\_flow['medium'] & waiting\_time['medium'], light\_duration['yellow'])

rule3 = ctrl.Rule(traffic\_flow['high'] | waiting\_time['high'], light\_duration['red'])

# Create control system

traffic\_light\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

traffic\_light\_simulation = ctrl.ControlSystemSimulation(traffic\_light\_ctrl)

# Run simulation with specific inputs

traffic\_light\_simulation.input['traffic\_flow'] = 48

traffic\_light\_simulation.input['waiting\_time'] = 15

traffic\_light\_simulation.compute()

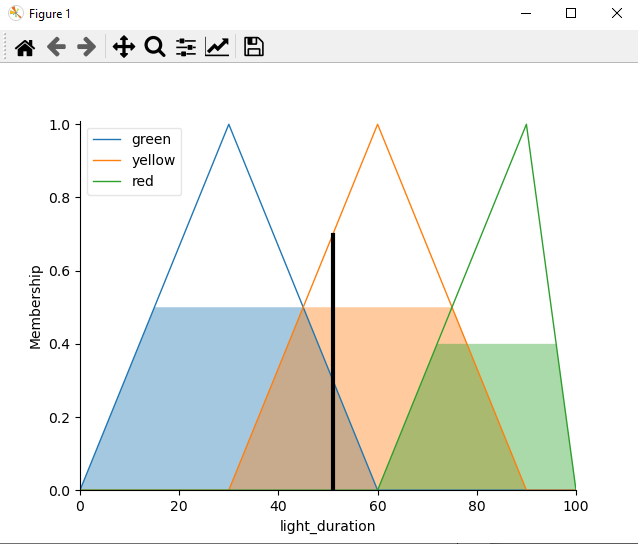
# Print simulation results

print("Light Duration:", traffic\_light\_simulation.output['light\_duration'])

light\_duration.view(sim=traffic\_light\_simulation)

plt.show()

**Output :-** **Light Duration: 50.92730085073477**



**6.Application of genetics algorithm to real world problems Generate Population**.

#GeneratePopulation

import numpy as np

# Define function to generate initial population

def generate\_population(pop\_size, chromosome\_length, gene\_range):

population = []

for \_ in range(pop\_size):

individual = np.random.randint(gene\_range[0], gene\_range[1] + 1, size=chromosome\_length)

population.append(individual)

return population

# Example usage

pop\_size = 4 # Population size

chromosome\_length = 3 # Length of each chromosome

gene\_range = (0, 3) # Range of possible values for genes

population = generate\_population(pop\_size, chromosome\_length, gene\_range)

for i, individual in enumerate(population):

print("Individual", i + 1, ":", individual)

**#Crossover**

import numpy as np

# Define crossover function

def crossover(parent1, parent2):

# Choose a random crossover point

crossover\_point = np.random.randint(1, len(parent1))

# Perform crossover

child1 = parent1[:crossover\_point] + parent2[crossover\_point:]

child2 = parent2[:crossover\_point] + parent1[crossover\_point:]

return child1, child2

# Example usage

parent1 = [1, 2, 3, 4, 5]

parent2 = [5, 4, 3, 2, 1]

child1, child2 = crossover(parent1, parent2)

print("Parent 1:", parent1)

print("Parent 2:", parent2)

print("Child 1 after crossover:", child1)

print("Child 2 after crossover:", child2)

**#Mutation**

import numpy as np

# Define mutation function

def mutate(individual, mutation\_rate):

mutated\_individual = individual.copy()

for i in range(len(mutated\_individual)):

if np.random.rand() < mutation\_rate:

# Mutate the gene at index i

mutated\_individual[i] = np.random.randint(0, 10) # Example: mutation within the range [0, 9]

return mutated\_individual

# Example usage

individual = [1, 2, 3, 4, 5]

mutation\_rate = 0.1 # Example: mutation rate of 10%

mutated\_individual = mutate(individual, mutation\_rate)

print("Original individual:", individual)

print("Mutated individual:", mutated\_individual)

Output:-

Original individual: [1, 2, 3, 4, 5]

Mutated individual: [1, 9, 3, 8, 5]