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:

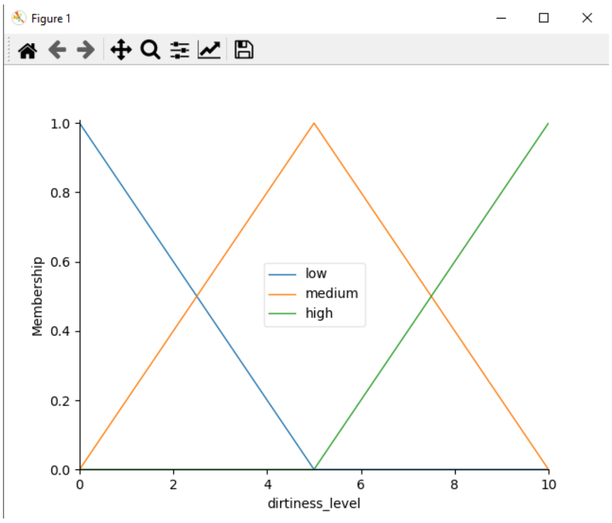


Figure 2:

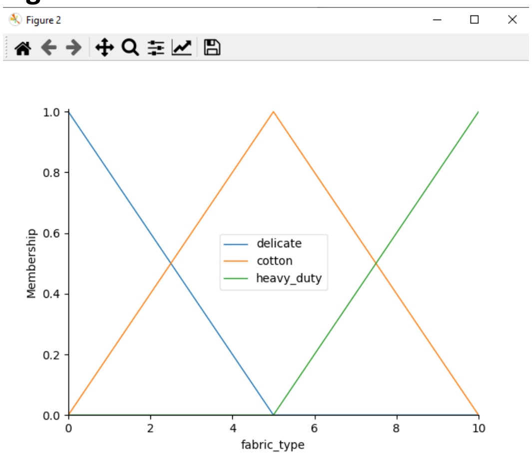


Figure 3:

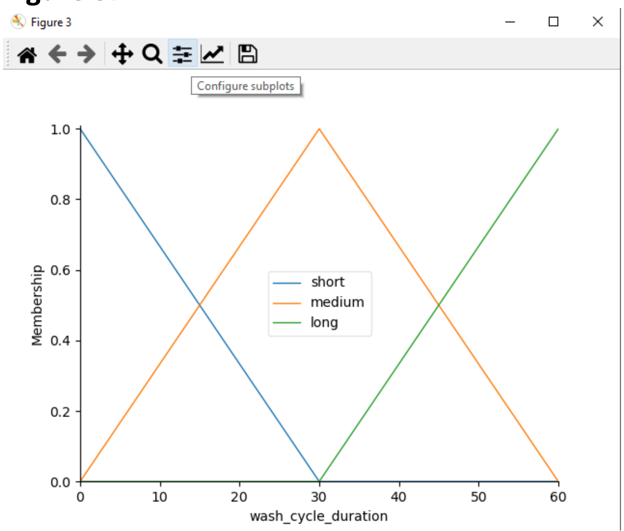


Figure 4:

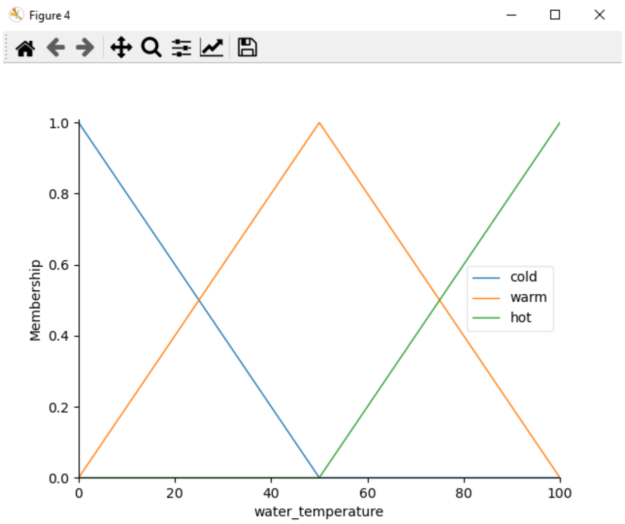
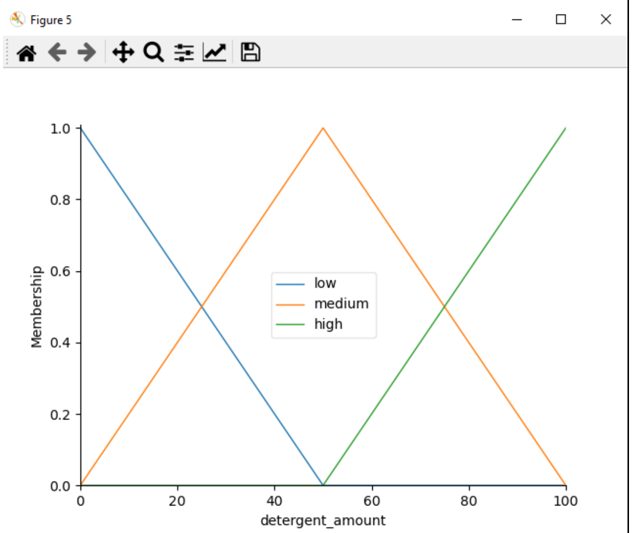


Figure 5:

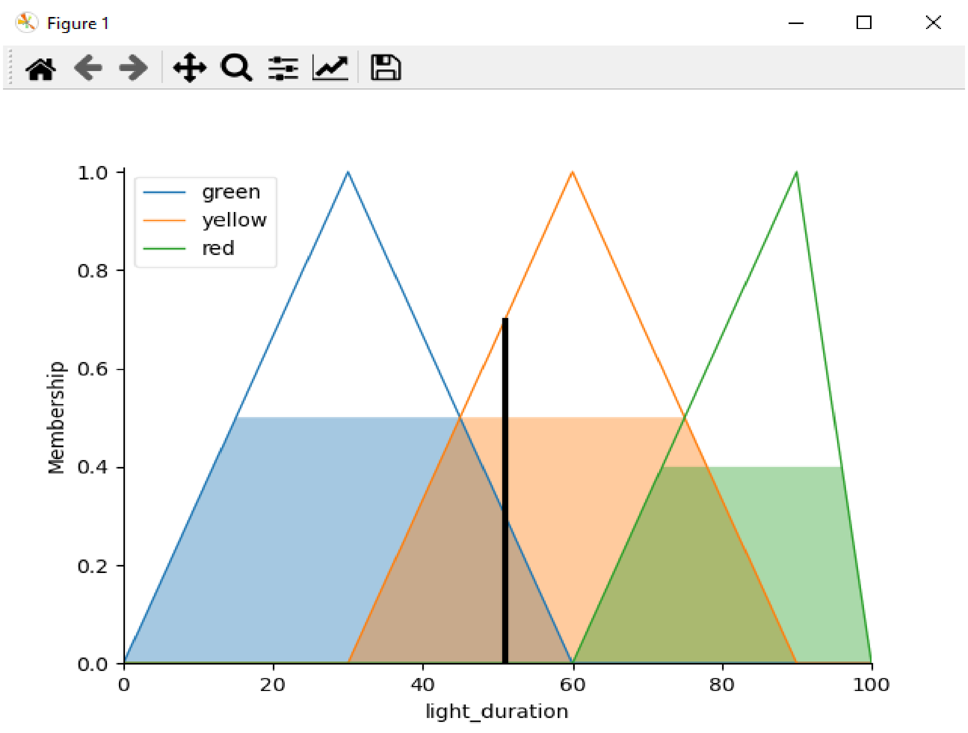


2.Develope a simulation of a fuzzy traffic light controller for a busy intersection. Define fuzzy sets and membership functions for traffic flow (e.g. low, medium, high) and waiting time. Design fuzzy rules to determine the duration of green, yellow, and red lights based on traffic flow and waiting time. 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]
```