UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER SCIENCE

मुंबई विद्यापीठ

University of Mumbai

Re-accredited with A++ Grade

(CGPA 3.65) by NAAC (3rd Cycle 2021)

M.Sc. Computer Science – Semester III

Fuzzy Systems

JOURNAL

2024-2025

Seat No. ____________________

मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the work entered in this journal was done in the University Department of Computer Science laboratory by Mr./Ms.___________________________________ Seat No. ______________ for the course of M.Sc. Computer Science - Semester III (NEP 2020) during the academic year 2024- 2025 in a satisfactory manner.

____

Subject In-charge

____

Head of Department

____

External Examiner

INDEX

Note : Install necessary python modules with “pip install scikit-fuzzy numpy matplotlib scipy networkx” command.

Practical - 1

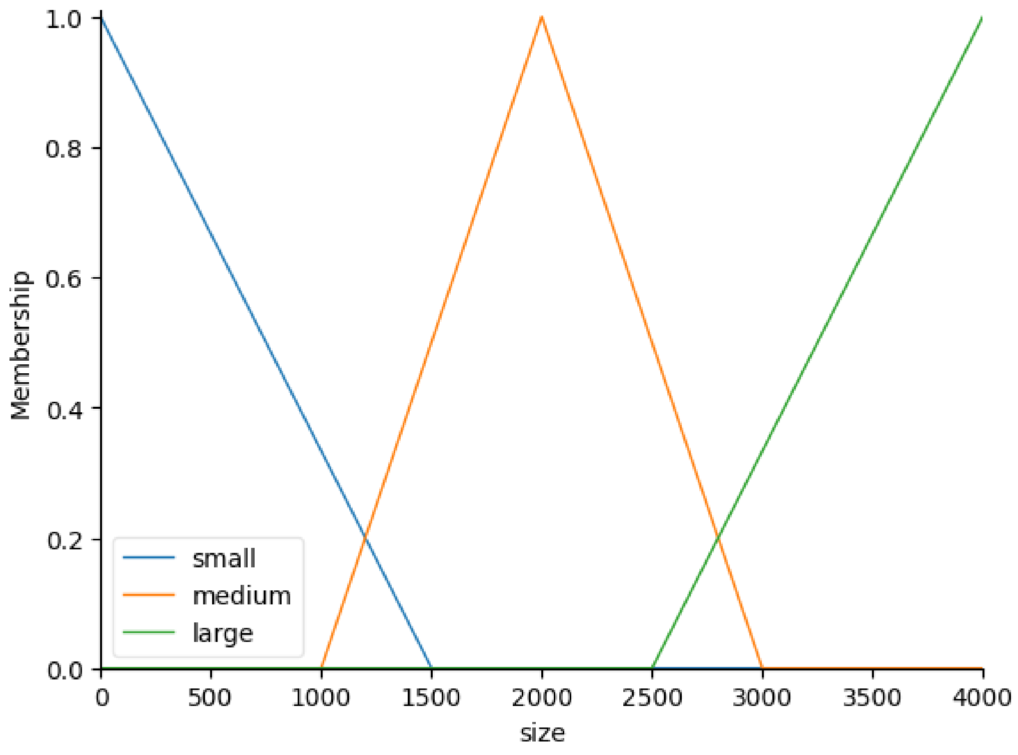
Aim: Develop a Fuzzy Sets model for House Pricing example

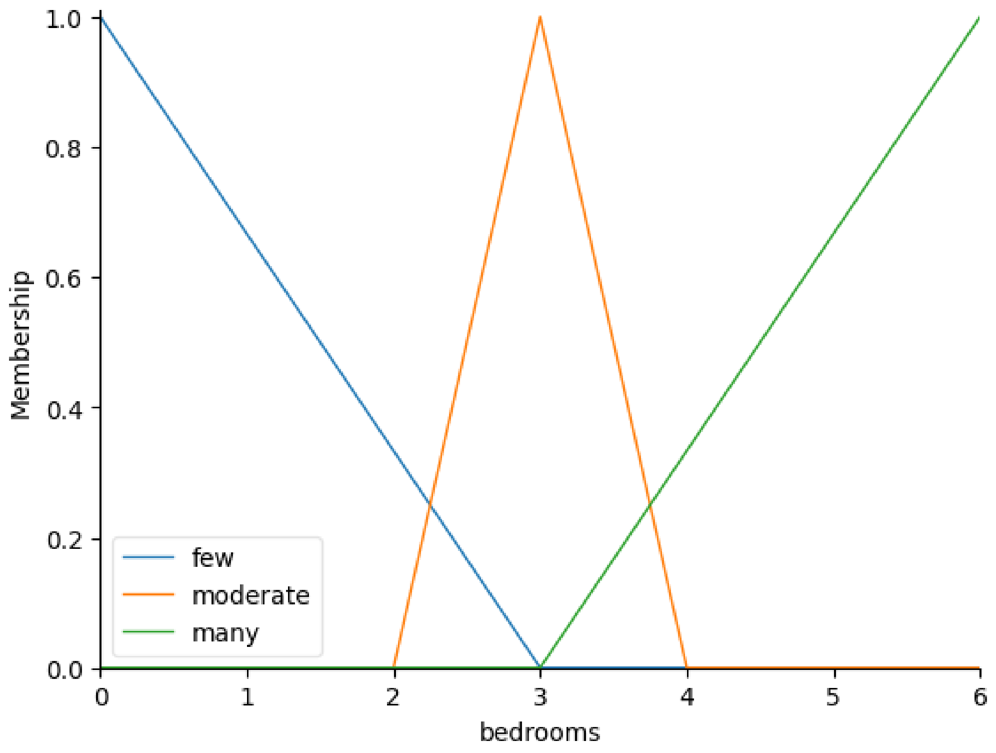
Program:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
# Define input variables (antecedents) and output (consequent)
size = ctrl.Antecedent(np.arange(0, 4001, 1), 'size')  # 0-4000 sq.ft
bedrooms = ctrl.Antecedent(np.arange(0, 7, 1), 'bedrooms')  # 0-6 bedrooms
location = ctrl.Antecedent(np.arange(0, 1.1, 0.1), 'location')  # 0 (rural) to 1 (urban)
price = ctrl.Consequent(np.arange(100000, 800001, 1000), 'price')  # $100k-$800k
# Define fuzzy membership functions
# --------------------------------------------------------------------
# Size: small, medium, large
size['small'] = fuzz.trimf(size.universe, [0, 0, 1500])
size['medium'] = fuzz.trimf(size.universe, [1000, 2000, 3000])
size['large'] = fuzz.trimf(size.universe, [2500, 4000, 4000])
# Bedrooms: few, moderate, many
bedrooms['few'] = fuzz.trimf(bedrooms.universe, [0, 0, 3])
bedrooms['moderate'] = fuzz.trimf(bedrooms.universe, [2, 3, 4])
bedrooms['many'] = fuzz.trimf(bedrooms.universe, [3, 6, 6])
# Location: rural, suburban, urban
location['rural'] = fuzz.trimf(location.universe, [0, 0, 0.4])
location['suburban'] = fuzz.trimf(location.universe, [0.3, 0.5, 0.7])
location['urban'] = fuzz.trimf(location.universe, [0.6, 1, 1])
# Price: low, medium, high
price['low'] = fuzz.trimf(price.universe, [100000, 100000, 350000])
price['medium'] = fuzz.trimf(price.universe, [250000, 450000, 650000])
price['high'] = fuzz.trimf(price.universe, [500000, 800000, 800000])
Visualize membership functions (optional)
size.view()
bedrooms.view()
location.view()
price.view()
plt.show()
# Define fuzzy rules
# --------------------------------------------------------------------
rule1 = ctrl.Rule(
    size['small'] & bedrooms['few'] & location['rural'],
    price['low']
)
rule2 = ctrl.Rule(
    size['medium'] & bedrooms['moderate'] & location['suburban'],
    price['medium']
)
rule3 = ctrl.Rule(
    size['large'] & bedrooms['many'] & location['urban'],
    price['high']
)
rule4 = ctrl.Rule(
    size['medium'] & bedrooms['moderate'] & location['urban'],
    price['high']
)
# Create control system
```

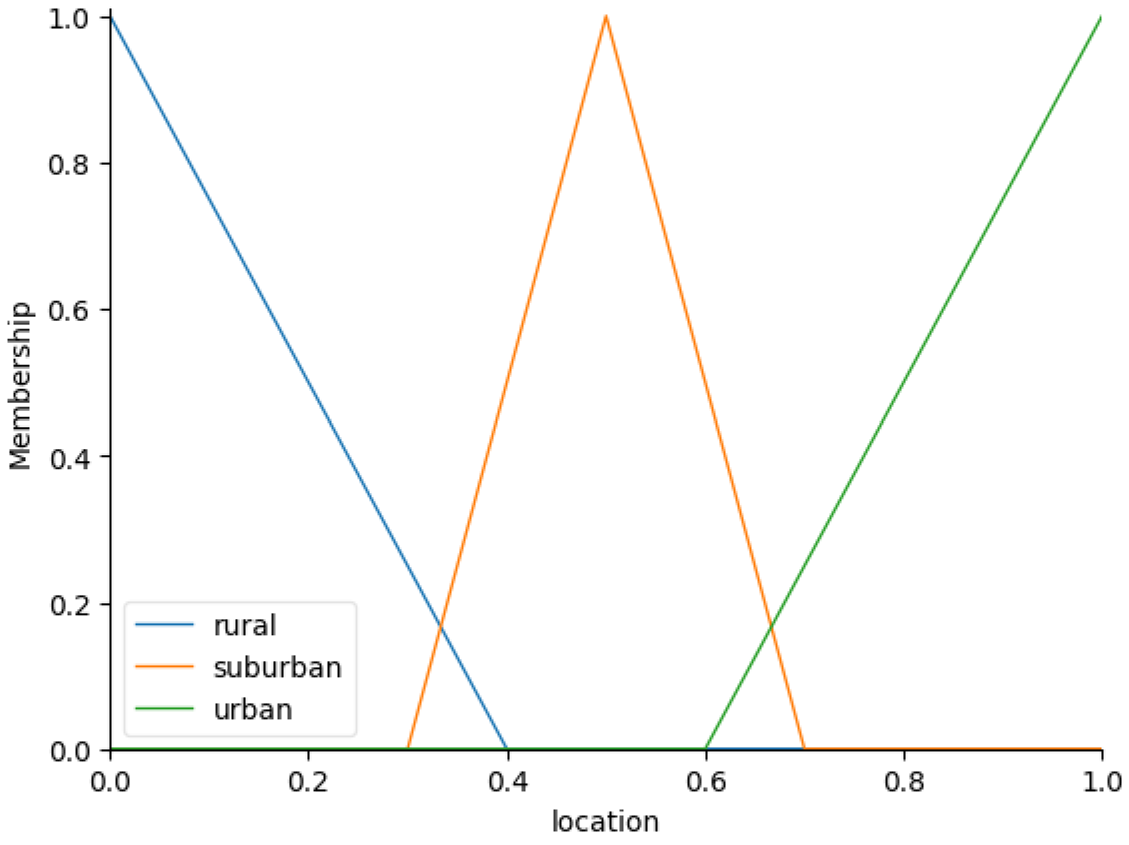
```
price_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4])
pricing = ctrl.ControlSystemSimulation(price_ctrl)
# Set input values and compute
# -------------------------------------------------------------------
pricing.input['size'] = 1200     # 1200 sq.ft
pricing.input['bedrooms'] = 3    # 3 bedrooms
pricing.input['location'] = 0.5  # Suburban (0.5)
pricing.compute()
# Print output
print("Estimated Price: $%.2f" % pricing.output['price'])
 price.view(sim=pricing)
 plt.show()
```

Output:

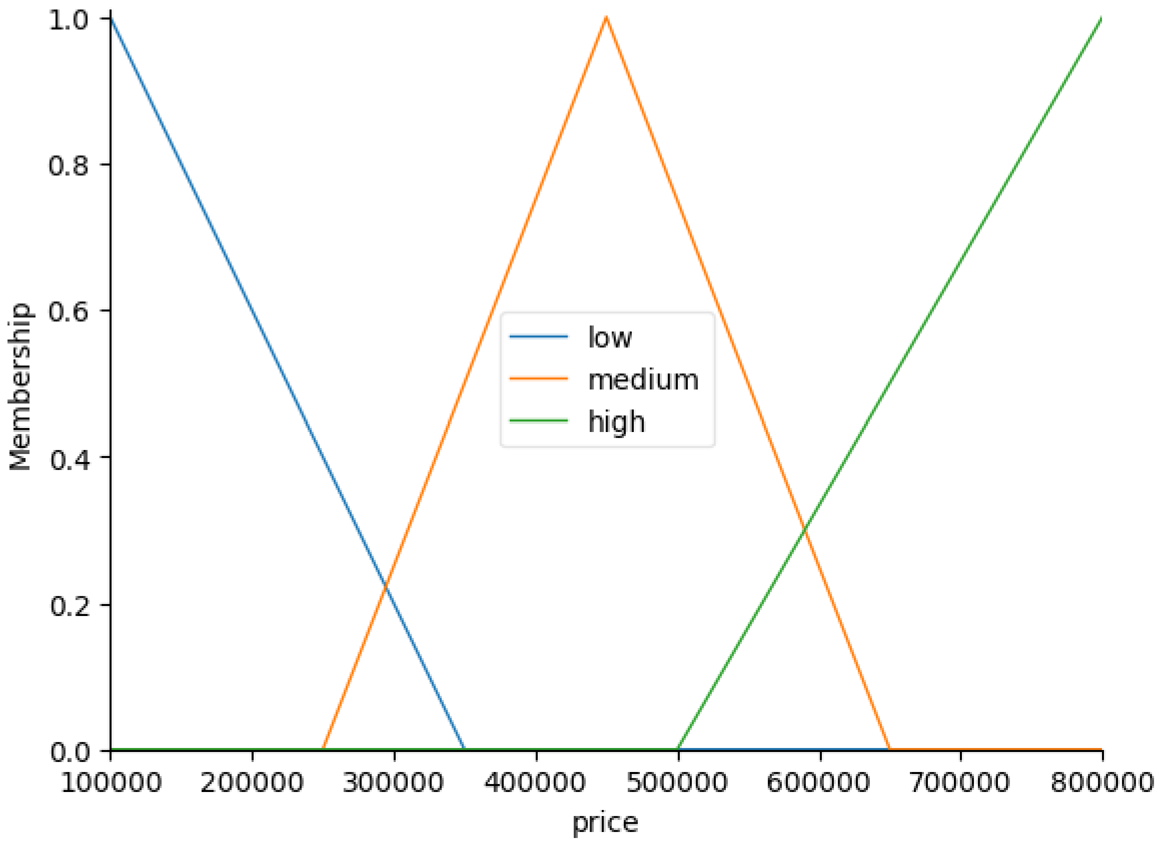




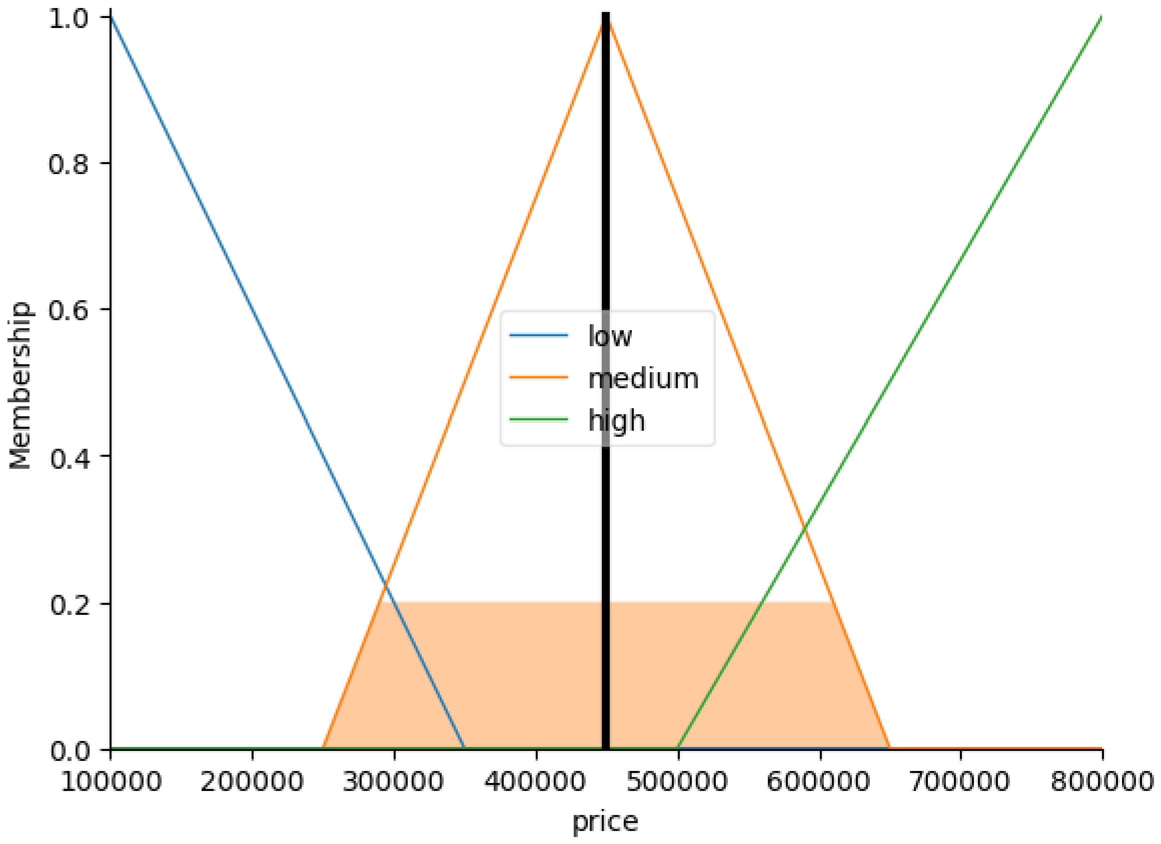
1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
1
2
3
4
5
6
bedrooms
few
moderate
many



1.0
0.8
0.6
0.4
0.2
0.0
Membership
0.0
0.2
0.4
0.6
0.8
1.0
location
rural
suburban
urban



```
Estimated Price: $450000.00
```



Practical - 2

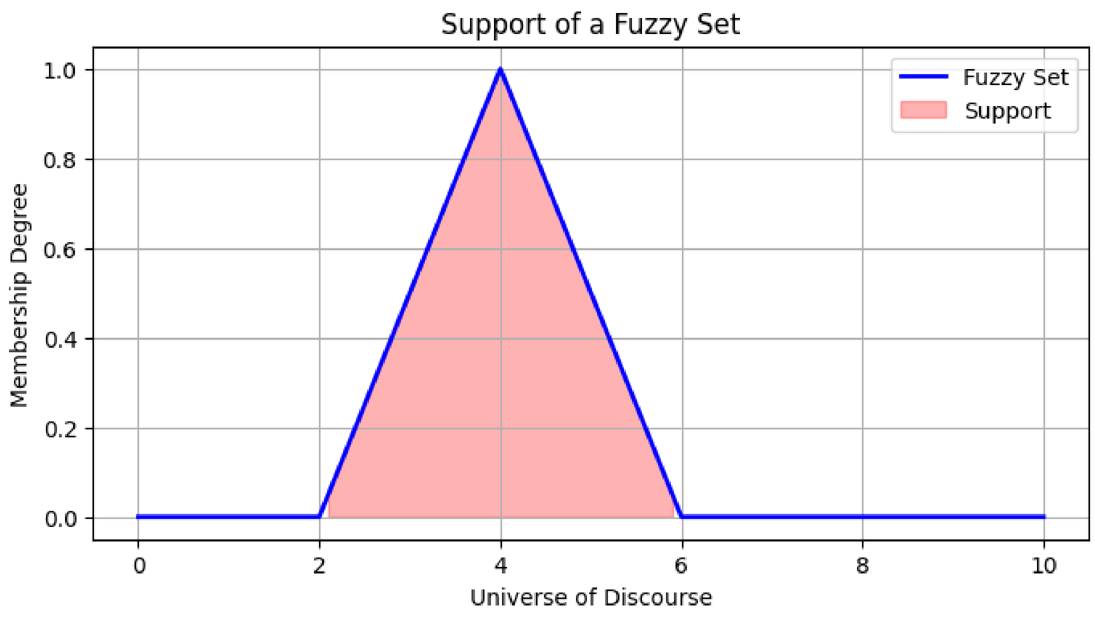
Aim: Calculate support in fuzzy logic.

Program:

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
# Define the universe of discourse (e.g., house sizes from 0 to 10)
universe = np.arange(0, 10.1, 0.1)
# Create a triangular fuzzy set (membership function)
membership = fuzz.trimf(universe, [2, 4, 6])
# Calculate the support (elements where membership > 0)
support = universe[membership > 0]
# Print the support range
print("Support of the fuzzy set:")
print(f"Start: {support[0]:.1f}, End: {support[-1]:.1f}")
print("All support points:", np.round(support, 1))
# Plot the fuzzy set and highlight the support
plt.figure(figsize=(8, 4))
plt.plot(universe, membership, 'b', linewidth=2, label='Fuzzy Set')
plt.fill_between(support, membership[membership > 0], color='red', alpha=0.3, label='Support')
plt.xlabel('Universe of Discourse')
plt.ylabel('Membership Degree')
plt.title('Support of a Fuzzy Set')
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Support of the fuzzy set:
Start: 2.1, End: 5.9
All support points: [2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.  3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.  4.1 4.2
 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.  5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9]
```



Practical - 3

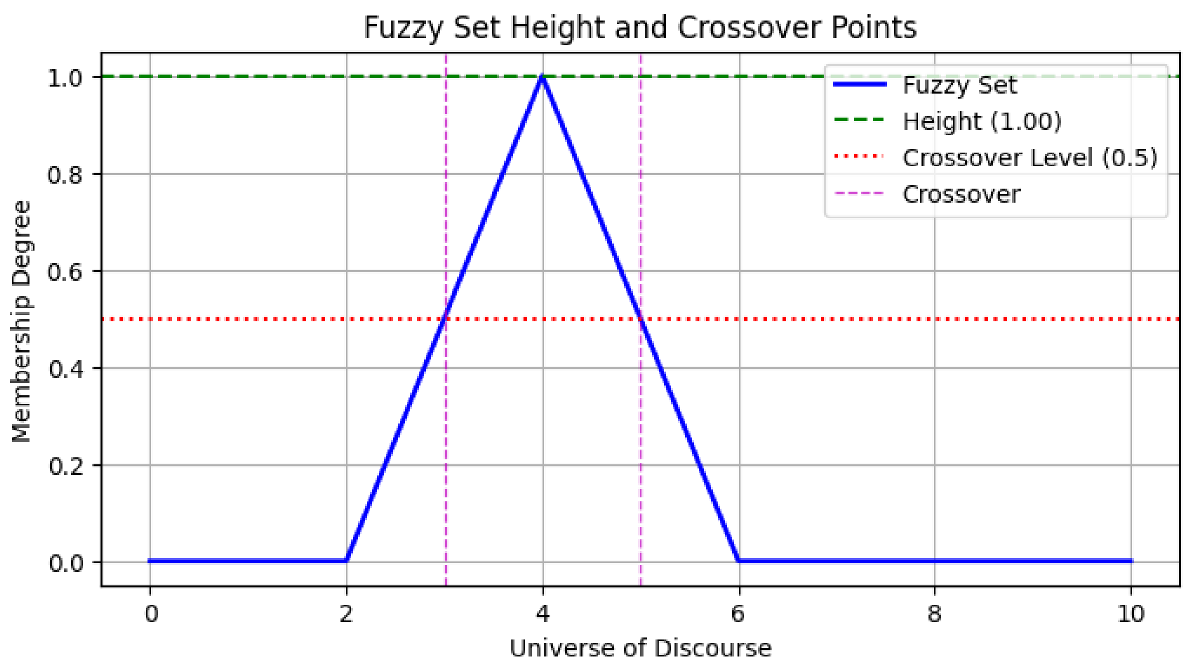
Aim: Calculate height and cross over in Fuzzy.

Program:

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
# Define the universe of discourse (e.g., house sizes from 0 to 10)
universe = np.arange(0, 10.01, 0.01)
# Create a triangular fuzzy set (membership function)
membership = fuzz.trimf(universe, [2, 4, 6])  # Parameters: left, peak, right
# Calculate height (maximum membership value)
height = np.max(membership)
print(f"Height of the fuzzy set: {height:.2f}")
# Calculate crossover points (where membership = 0.5)
crossovers = []
diff = membership - 0.5  # Difference from 0.5
# Detect sign changes in the difference array
for i in range(len(universe) - 1):
    if diff[i] * diff[i+1] <= 0:  # Sign change or exact zero
        x1, x2 = universe[i], universe[i+1]
        m1, m2 = membership[i], membership[i+1]
        # Linear interpolation to find exact crossover
        if m1 != m2:  # Avoid division by zero
            x_cross = x1 + (0.5 - m1) * (x2 - x1) / (m2 - m1)
            crossovers.append(x_cross)
        else:
            # Handle flat regions (membership constant at 0.5)
            if np.isclose(m1, 0.5):
                crossovers.extend([x1, x2])
# Remove duplicates and sort
crossovers = sorted(list(set(crossovers)))
print("Crossover points:", np.round(crossovers, 2))
# Plot the fuzzy set, height, and crossover points
plt.figure(figsize=(8, 4))
plt.plot(universe, membership, 'b', linewidth=2, label='Fuzzy Set')
plt.axhline(y=height, color='g', linestyle='--', label=f'Height ({height:.2f})')
plt.axhline(y=0.5, color='r', linestyle=':', label='Crossover Level (0.5)')
for x in crossovers:
    plt.axvline(x=x, color='m', linestyle='--', linewidth=1, alpha=0.7, label='Crossover' if x ==
crossovers[0] else "")
plt.xlabel('Universe of Discourse')
plt.ylabel('Membership Degree')
plt.title('Fuzzy Set Height and Crossover Points')
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Height of the fuzzy set: 1.00
Crossover points: [3. 5.]
```



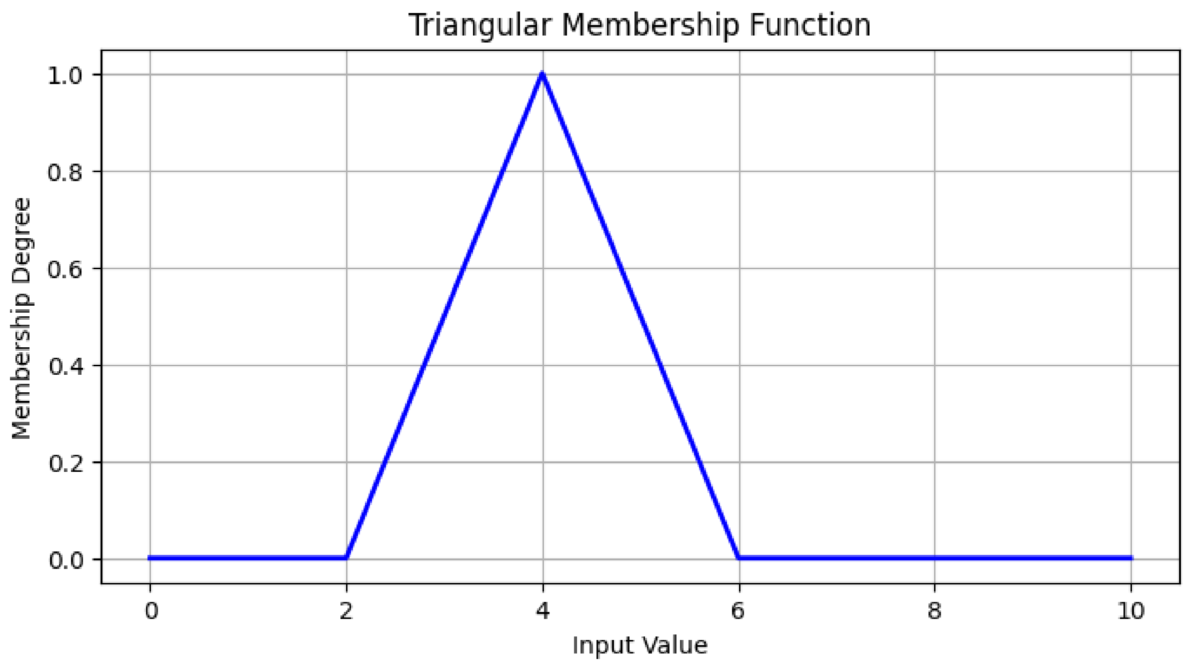
Practical – 4

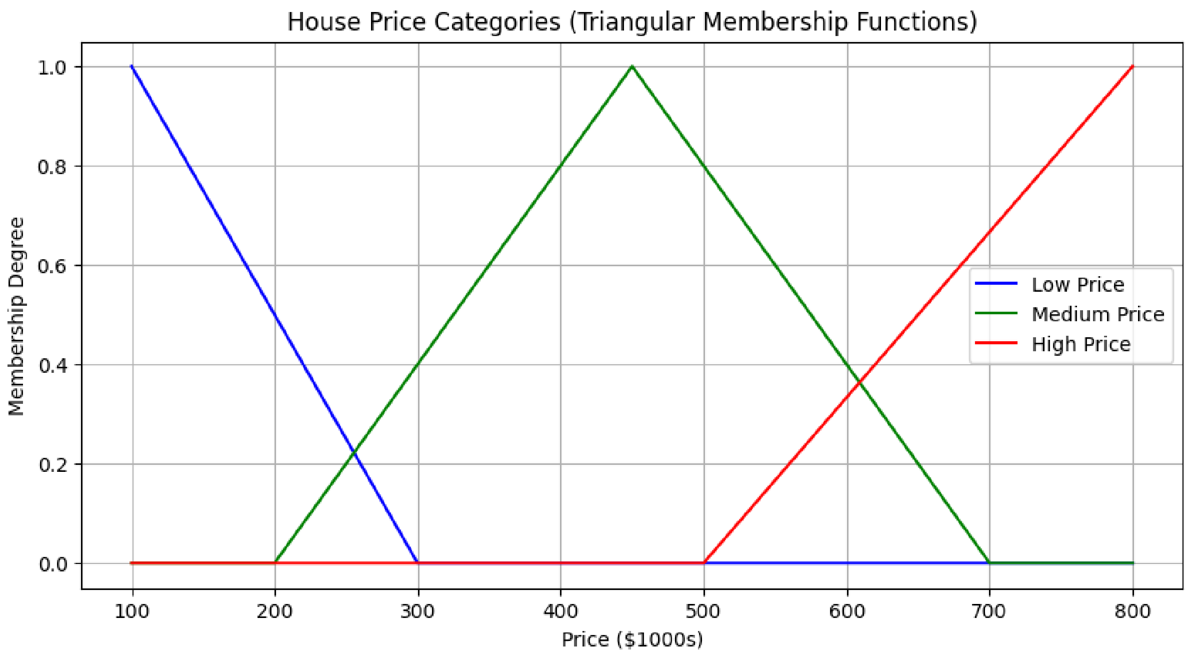
Aim: Create a triangular function using trimf.

Program:

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
# Define the universe of discourse (input range)
universe = np.arange(0, 10.1, 0.1)  # 0 to 10 in steps of 0.1
# Create a triangular membership function
# Parameters: [left-foot, peak, right-foot]
triangular_mf = fuzz.trimf(universe, [2, 4, 6])
# Plot the membership function
plt.figure(figsize=(8, 4))
plt.plot(universe, triangular_mf, 'b', linewidth=2)
plt.title('Triangular Membership Function')
plt.xlabel('Input Value')
plt.ylabel('Membership Degree')
plt.grid(True)
plt.show()
# Define universe (house prices in thousands)
price_universe = np.arange(100, 801, 1)  # $100k to $800k
# Create triangular membership functions
low_price = fuzz.trimf(price_universe, [100, 100, 300])    # Low price
medium_price = fuzz.trimf(price_universe, [200, 450, 700]) # Medium price
high_price = fuzz.trimf(price_universe, [500, 800, 800])   # High price
# Plot all membership functions
plt.figure(figsize=(10, 5))
plt.plot(price_universe, low_price, 'b', label='Low Price')
plt.plot(price_universe, medium_price, 'g', label='Medium Price')
plt.plot(price_universe, high_price, 'r', label='High Price')
plt.title('House Price Categories (Triangular Membership Functions)')
plt.xlabel('Price ($1000s)')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()
```

Output:


Triangular Membership Function
1.0
0.8
0.6
0.4
0.2
0.0
Membership Degree
0
2
4
6
8
10
Input Value


House Price Categories (Triangular Membership Functions)
1.0
0.8
0.6
0.4
0.2
0.0
Membership Degree
100
200
300
400
500
600
700
800
Price ($1000s)
Low Price
Medium Price
High Price

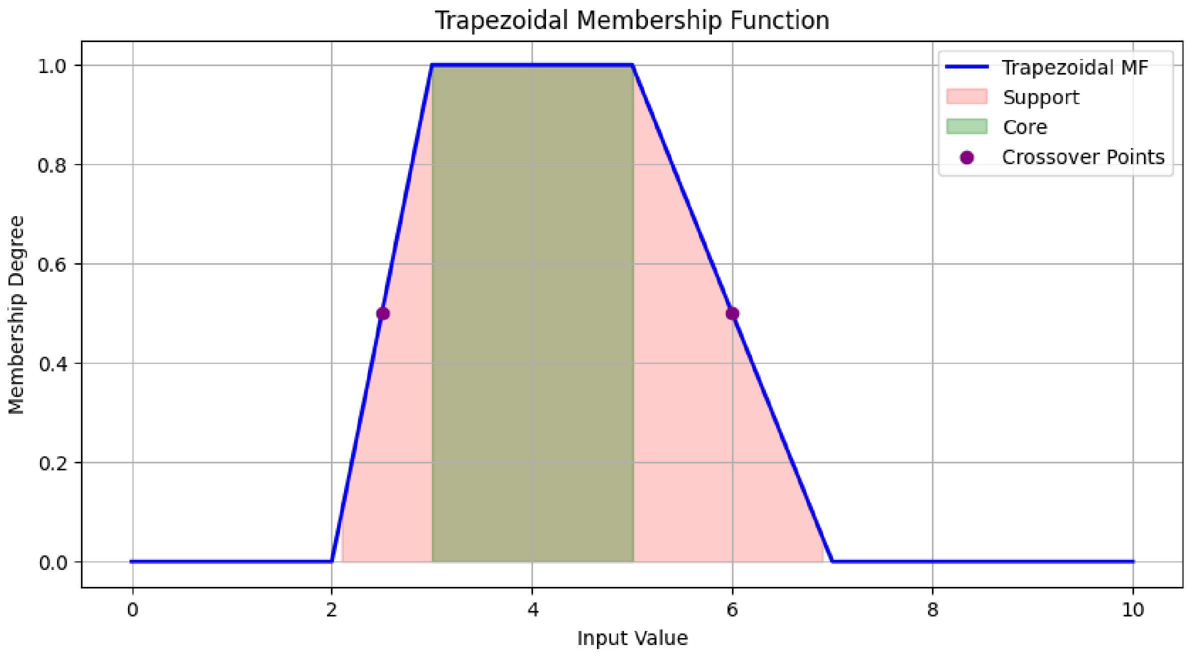
Practical - 5

Aim: Implement fuzzy_trapezodial_membership-function

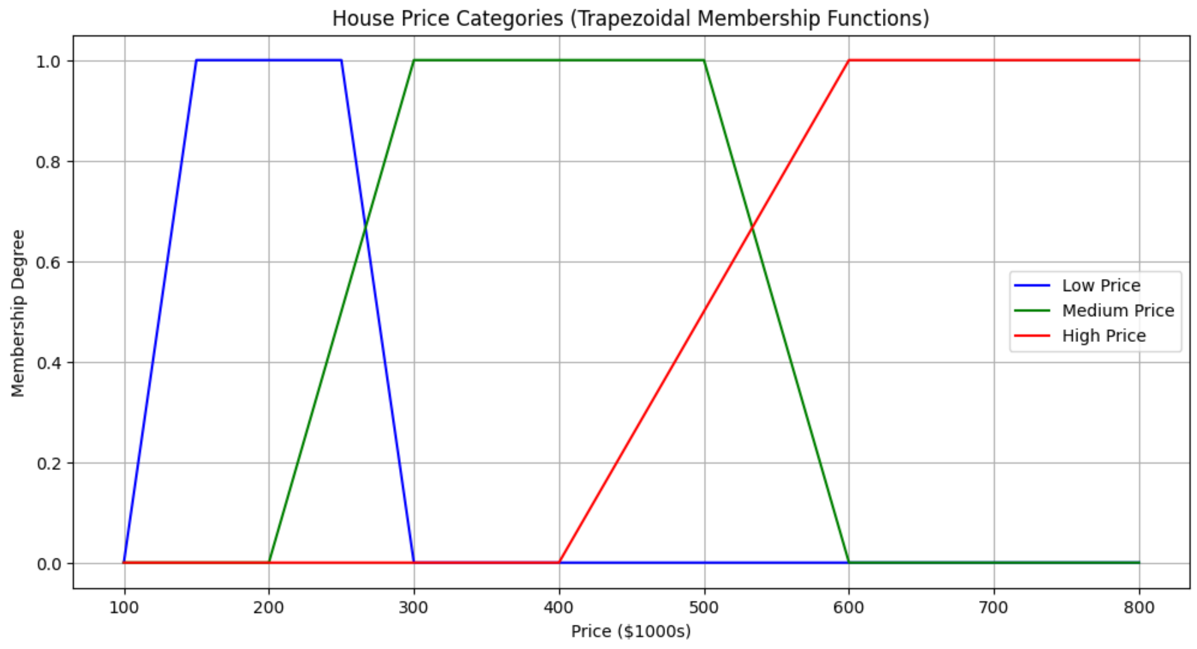
Program:

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
# Define the universe of discourse (input range)
universe = np.arange(0, 10.1, 0.1)  # 0 to 10 in steps of 0.1
# Create a trapezoidal membership function
# Parameters: [left_foot, left_shoulder, right_shoulder, right_foot]
trapezoidal_mf = fuzz.trapmf(universe, [2, 3, 5, 7])
# Calculate properties
support = universe[trapezoidal_mf > 0]          # All points with membership > 0
core = universe[trapezoidal_mf == 1]            # All points with membership = 1
crossovers = universe[np.isclose(trapezoidal_mf, 0.5, atol=0.01)]  # Points ≈ 0.5
# Plot the membership function
plt.figure(figsize=(10, 5))
plt.plot(universe, trapezoidal_mf, 'b', linewidth=2, label='Trapezoidal MF')
plt.fill_between(support, trapezoidal_mf[trapezoidal_mf > 0], color='red', alpha=0.2, label='Support')
plt.fill_between(core, 1, color='green', alpha=0.3, label='Core')
plt.scatter(crossovers, [0.5]*len(crossovers), color='purple', zorder=5, label='Crossover Points')
plt.title('Trapezoidal Membership Function')
plt.xlabel('Input Value')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()
# Print properties
print("Support:", np.round(support[[0, -1]], 2))
print("Core:", np.round(core[[0, -1]], 2))
print("Crossover Points:", np.round(crossovers, 2))
# Define universe (house prices in $1000s)
price_universe = np.arange(100, 801, 1)
# Create trapezoidal membership functions
low_price = fuzz.trapmf(price_universe, [100, 150, 250, 300])
medium_price = fuzz.trapmf(price_universe, [200, 300, 500, 600])
high_price = fuzz.trapmf(price_universe, [400, 600, 800, 800])
# Plot all membership functions
plt.figure(figsize=(12, 6))
plt.plot(price_universe, low_price, 'b', label='Low Price')
plt.plot(price_universe, medium_price, 'g', label='Medium Price')
plt.plot(price_universe, high_price, 'r', label='High Price')
plt.title('House Price Categories (Trapezoidal Membership Functions)')
plt.xlabel('Price ($1000s)')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



```
Support: [2.1 6.9]
Core: [3. 5.]
Crossover Points: [2.5 6. ]
```



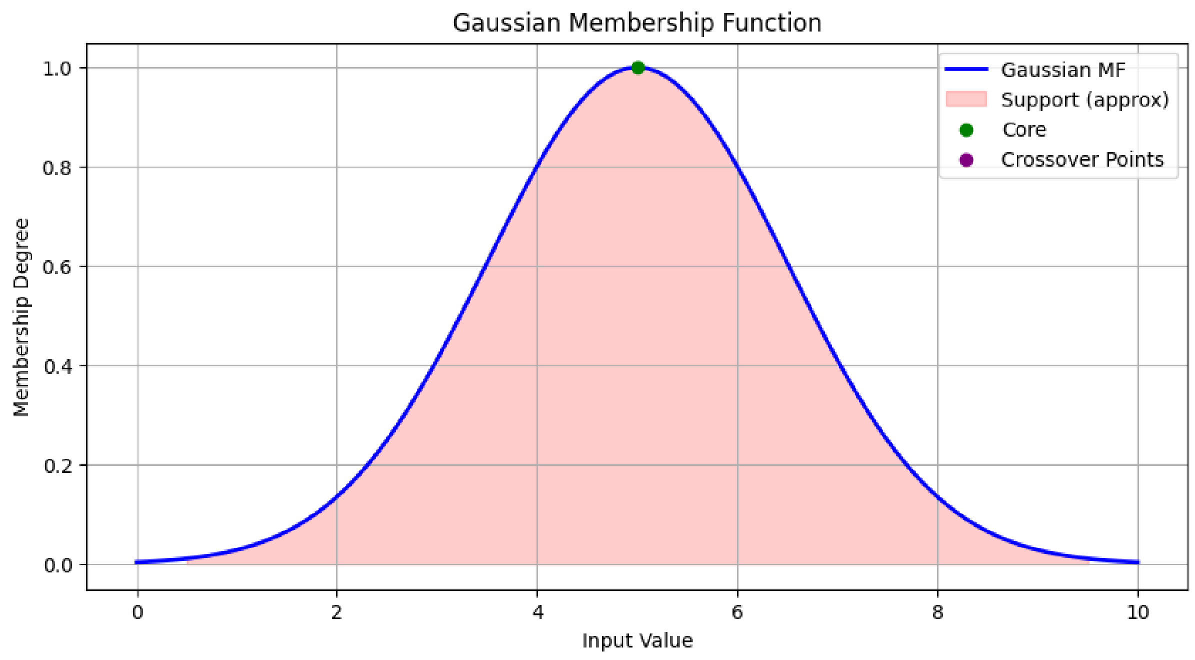
Practical - 6

Aim: Implement fuzzy_gaussian_membership_function

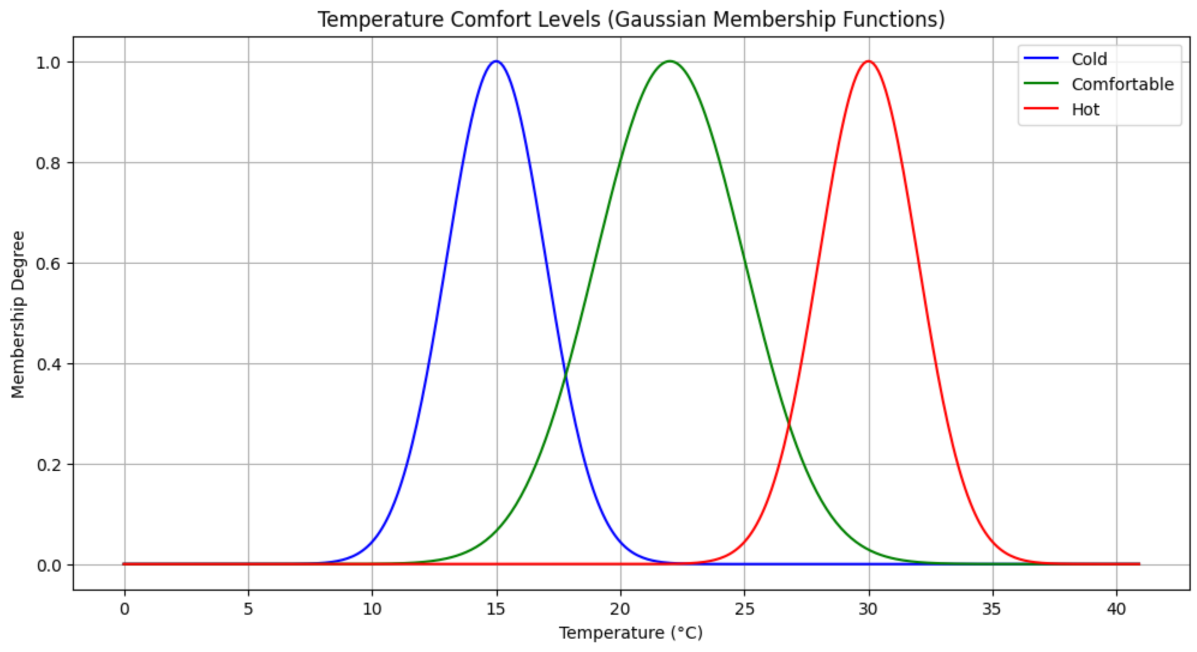
Program:

```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
# Define the universe of discourse (input range)
universe = np.arange(0, 10.1, 0.1)  # 0 to 10 in steps of 0.1
# Create a Gaussian membership function
# Parameters: [mean (center), sigma (spread)]
mean = 5.0    # Center of the Gaussian
sigma = 1.5   # Width (spread) of the curve
gaussian_mf = fuzz.gaussmf(universe, mean, sigma)
# Calculate properties
support = universe[gaussian_mf > 0.01]          # Approximate support (μ > 0.01)
core = universe[np.isclose(gaussian_mf, 1.0)]   # Core (μ = 1.0)
crossovers = universe[np.isclose(gaussian_mf, 0.5, atol=0.01)]  # μ ≈ 0.5
# Plot the membership function
plt.figure(figsize=(10, 5))
plt.plot(universe, gaussian_mf, 'b', linewidth=2, label='Gaussian MF')
plt.fill_between(support, gaussian_mf[gaussian_mf > 0.01], color='red', alpha=0.2, label='Support
(approx)')
plt.scatter(core, [1.0]*len(core), color='green', zorder=5, label='Core')
plt.scatter(crossovers, [0.5]*len(crossovers), color='purple', zorder=5, label='Crossover Points')
plt.title('Gaussian Membership Function')
plt.xlabel('Input Value')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()
# Print properties
print("Mean (Center):", mean)
print("Sigma (Spread):", sigma)
print("Crossover Points:", np.round(crossovers, 2))
# Define universe (temperature in °C)
temp_universe = np.arange(0, 41, 0.1)
# Create Gaussian membership functions
cold = fuzz.gaussmf(temp_universe, 15, 2)
comfortable = fuzz.gaussmf(temp_universe, 22, 3)
hot = fuzz.gaussmf(temp_universe, 30, 2)
# Plot all membership functions
plt.figure(figsize=(12, 6))
plt.plot(temp_universe, cold, 'b', label='Cold')
plt.plot(temp_universe, comfortable, 'g', label='Comfortable')
plt.plot(temp_universe, hot, 'r', label='Hot')
plt.title('Temperature Comfort Levels (Gaussian Membership Functions)')
plt.xlabel('Temperature (°C)')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



```
Mean (Center): 5.0
Sigma (Spread): 1.5
Crossover Points: []
```



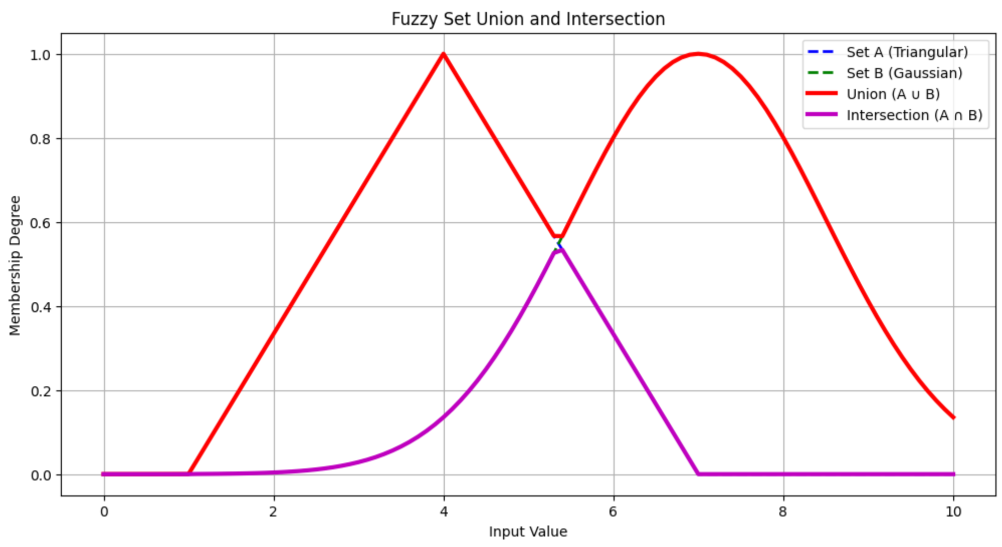
Practical - 7

Aim: Implement Fuzzy Sets Union Intersection

Program:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the universe of discourse (input range)
universe = np.arange(0, 10.1, 0.1)  # 0 to 10 in steps of 0.1
# Create two fuzzy sets (membership functions)
# -------------------------------------------
# Fuzzy Set A: Triangular membership function
set_A = np.clip(1 - np.abs(universe - 4) / 3, 0, 1)  # Peak at 4, width 3
# Fuzzy Set B: Gaussian membership function
set_B = np.exp(-(universe - 7)**2 / (2 * 1.5**2))  # Mean=7, sigma=1.5
# Fuzzy Union (A ∪ B) = max(A, B)
fuzzy_union = np.maximum(set_A, set_B)
# Fuzzy Intersection (A ∩ B) = min(A, B)
fuzzy_intersection = np.minimum(set_A, set_B)
# Plot the results
# -------------------------------------------
plt.figure(figsize=(12, 6))
# Plot Fuzzy Set A and B
plt.plot(universe, set_A, 'b--', linewidth=2, label='Set A (Triangular)')
plt.plot(universe, set_B, 'g--', linewidth=2, label='Set B (Gaussian)')
# Plot Union and Intersection
plt.plot(universe, fuzzy_union, 'r', linewidth=3, label='Union (A ∪ B)')
plt.plot(universe, fuzzy_intersection, 'm', linewidth=3, label='Intersection (A ∩ B)')
plt.title('Fuzzy Set Union and Intersection')
plt.xlabel('Input Value')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



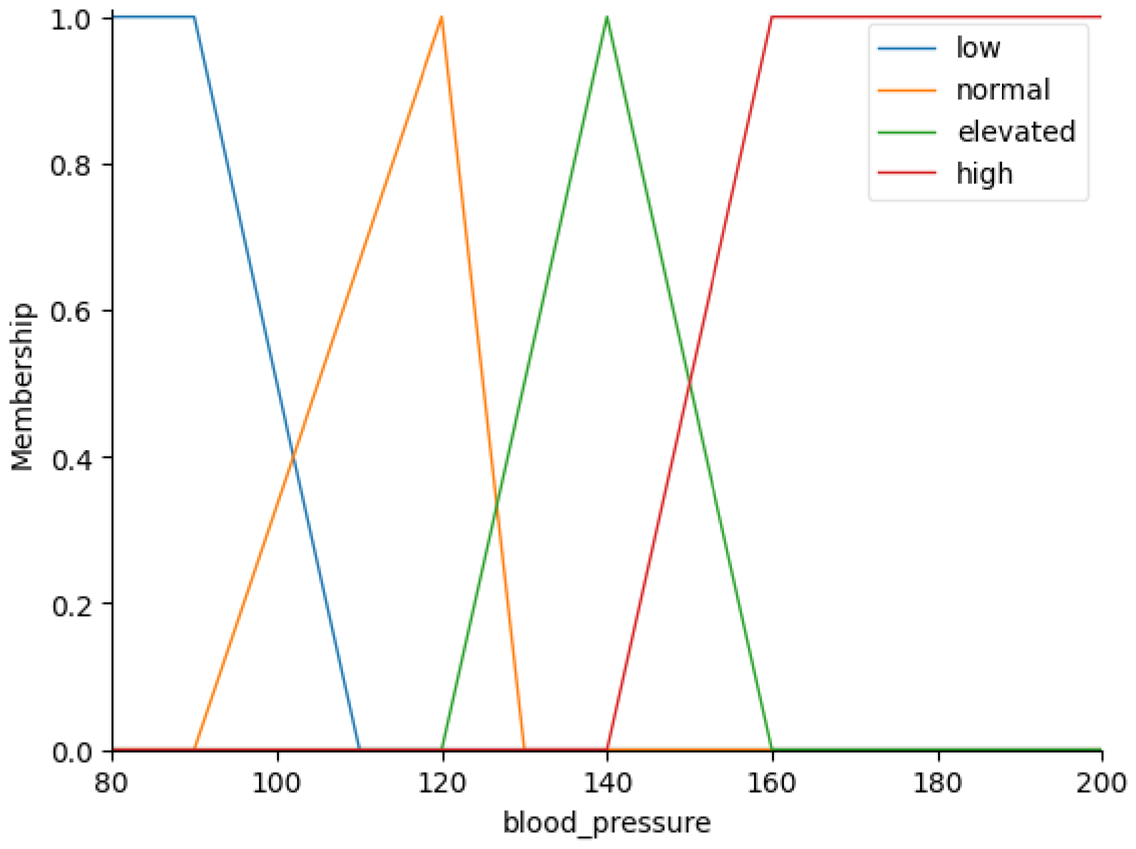
Practical - 8

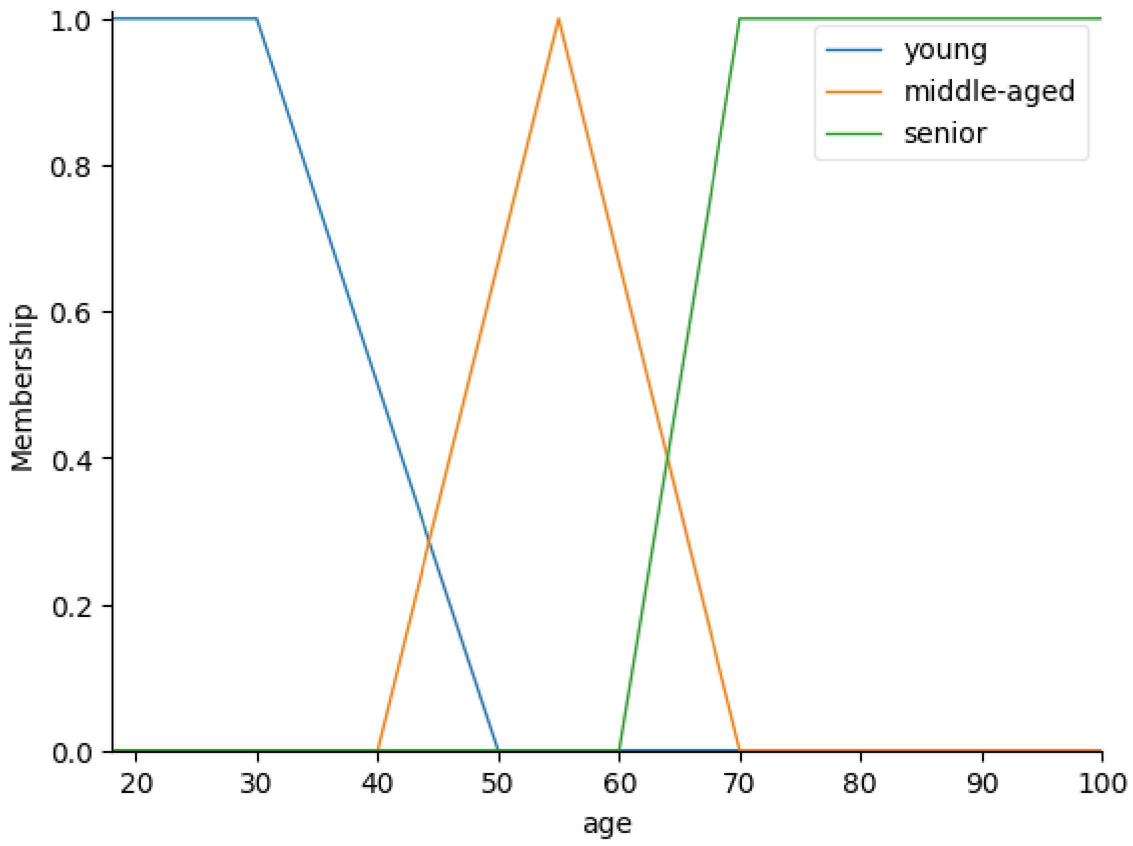
Aim: Implement fuzzy logic medical exam regarding the blood pressure and age

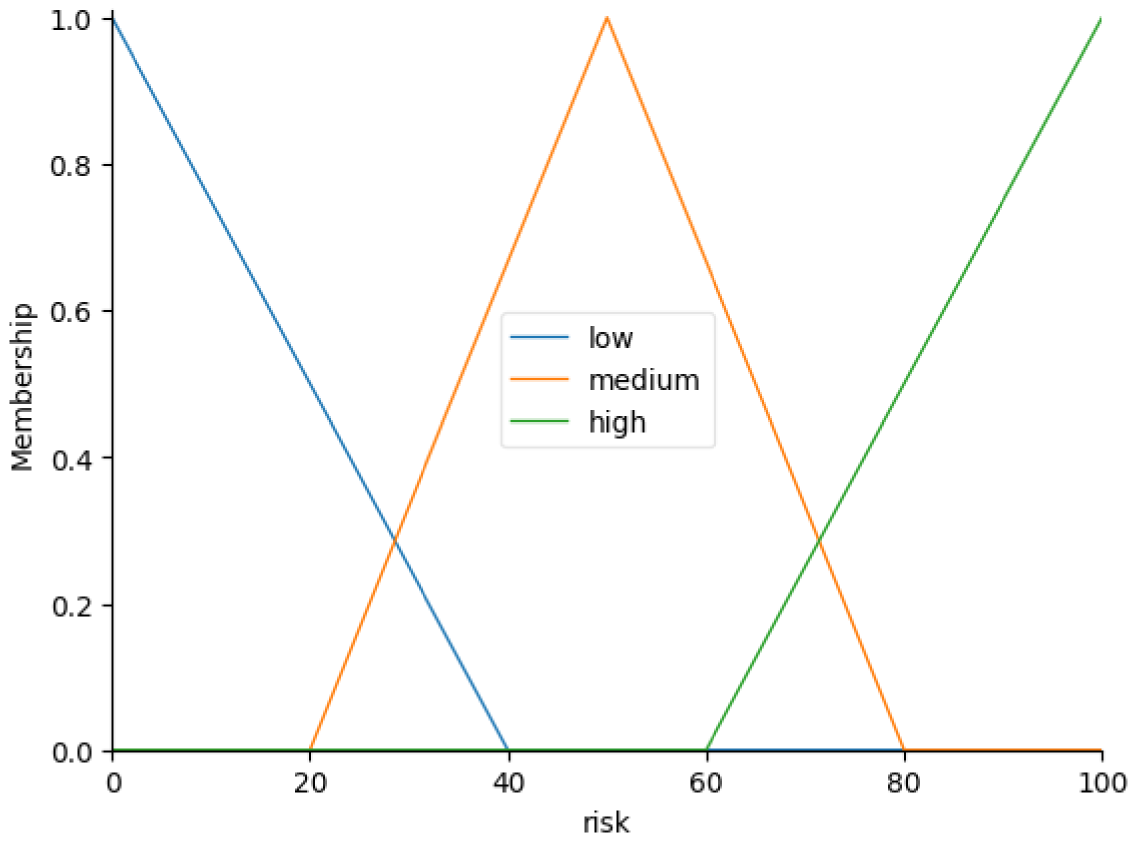
Program:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
# Define input and output variables
# --------------------------------------------------------------------
# Antecedents (Inputs)
blood_pressure = ctrl.Antecedent(np.arange(80, 201, 1), 'blood_pressure')  # 80-200 mmHg
age = ctrl.Antecedent(np.arange(18, 101, 1), 'age')                        # 18-100 years
# Consequent (Output)
risk = ctrl.Consequent(np.arange(0, 101, 1), 'risk')                       # 0-100%
# Define fuzzy membership functions
# --------------------------------------------------------------------
# Blood Pressure Categories
blood_pressure['low'] = fuzz.trapmf(blood_pressure.universe, [80, 80, 90, 110])
blood_pressure['normal'] = fuzz.trimf(blood_pressure.universe, [90, 120, 130])
blood_pressure['elevated'] = fuzz.trimf(blood_pressure.universe, [120, 140, 160])
blood_pressure['high'] = fuzz.trapmf(blood_pressure.universe, [140, 160, 200, 200])
# Age Categories
age['young'] = fuzz.trapmf(age.universe, [18, 18, 30, 50])
age['middle-aged'] = fuzz.trimf(age.universe, [40, 55, 70])
age['senior'] = fuzz.trapmf(age.universe, [60, 70, 100, 100])
# Risk Level Categories
risk['low'] = fuzz.trimf(risk.universe, [0, 0, 40])
risk['medium'] = fuzz.trimf(risk.universe, [20, 50, 80])
risk['high'] = fuzz.trimf(risk.universe, [60, 100, 100])
# Visualize membership functions (optional)
blood_pressure.view()
age.view()
risk.view()
plt.show()
# Define fuzzy rules
# --------------------------------------------------------------------
rule1 = ctrl.Rule(
    blood_pressure['low'] & age['young'],
    risk['low']
)
rule2 = ctrl.Rule(
    blood_pressure['normal'] & (age['young'] | age['middle-aged']),
    risk['low']
)
rule3 = ctrl.Rule(
    blood_pressure['elevated'] & age['middle-aged'],
    risk['medium']
)
rule4 = ctrl.Rule(
    blood_pressure['high'] & (age['middle-aged'] | age['senior']),
    risk['high']
)
rule5 = ctrl.Rule(
    blood_pressure['elevated'] & age['senior'],
    risk['high']
)
# Create control system
# -------------------------------------------------------------------
risk_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5])
risk_assessment = ctrl.ControlSystemSimulation(risk_ctrl)
# Example 1: Young patient with normal BP
risk_assessment.input['blood_pressure'] = 115
risk_assessment.input['age'] = 28
risk_assessment.compute()
risk_level = risk_assessment.output['risk']
print(f"Patient Age: 28, BP: 115 mmHg")
print(f"Risk Level: {risk_level:.2f}% (Low Risk)")
risk.view(sim=risk_assessment)
plt.show()
# Example 2: Senior patient with high BP
risk_assessment.input['blood_pressure'] = 180
risk_assessment.input['age'] = 72
risk_assessment.compute()
risk_level = risk_assessment.output['risk']
print(f"\nPatient Age: 72, BP: 180 mmHg")
print(f"Risk Level: {risk_level:.2f}% (High Risk)")
risk.view(sim=risk_assessment)
plt.show()
```

Output:

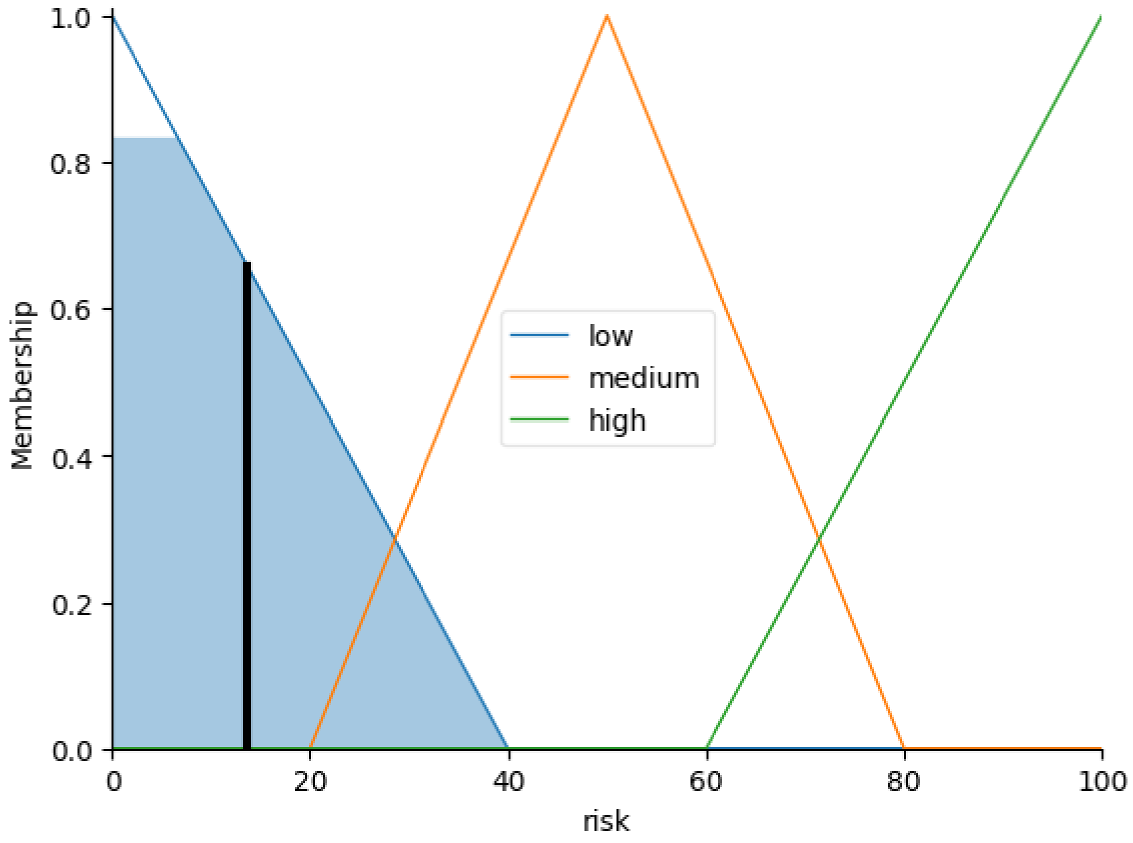




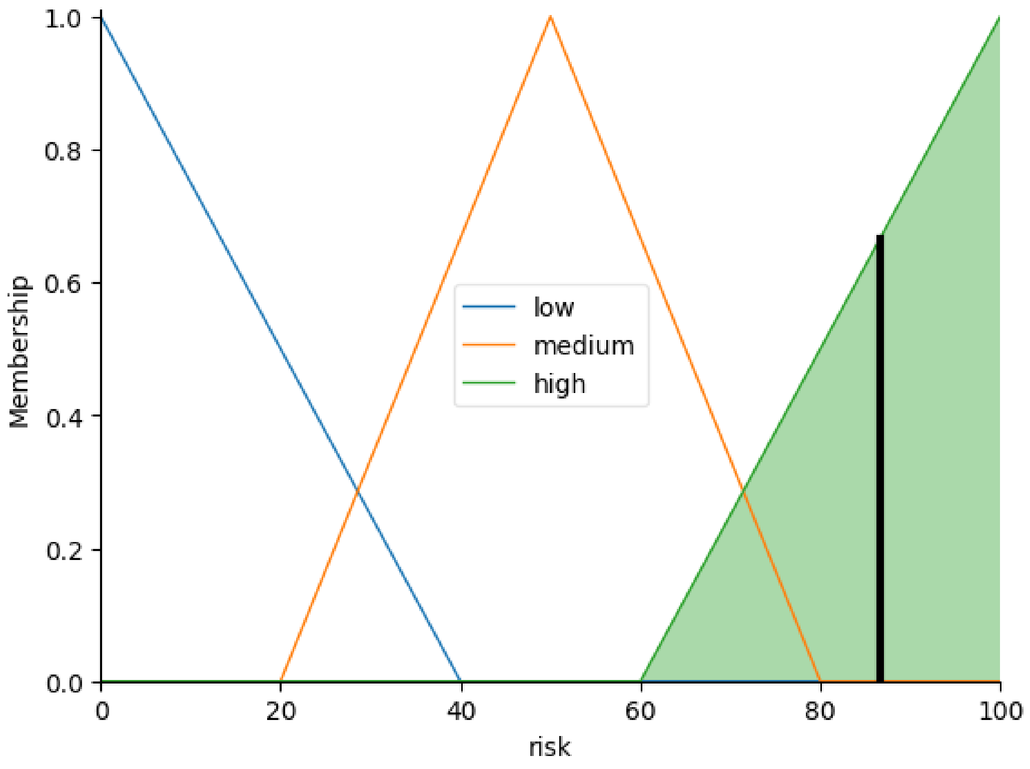


Patient Age: 28, BP: 115 mmHg

Risk Level: 13.65% (Low Risk)



Patient Age: 72, BP: 180 mmHg
Risk Level: 86.67% (High Risk)



Practical – 9

Aim: Implement Tipping Problem without Fuzzy Logic

Program:

```
def calculate_tip(service_rating, food_rating):

    # Validate inputs (0-10 scale)
    service_rating = max(0, min(10, service_rating))
    food_rating = max(0, min(10, food_rating))
    # Categorize service and food quality
    def categorize(rating):
        if rating <= 3:
            return "poor"
        elif 4 <= rating <= 6:
            return "average"
        else:
            return "excellent"
    service_category = categorize(service_rating)
    food_category = categorize(food_rating)
    # Define tipping rules
    tip_rules = {
        ("poor", "poor"): 5,
        ("average", "average"): 12,
        ("excellent", "excellent"): 20,
    }
    # Check for exact matches
    if (service_category, food_category) in tip_rules:
        return tip_rules[(service_category, food_category)]
    else:
        # Weighted average for mixed categories
        service_weight = (service_rating / 10) * 0.6  # Service contributes 60%
        food_weight = (food_rating / 10) * 0.4        # Food contributes 40%
        tip = 5 + (15 * (service_weight + food_weight))  # Tip range: 5%–20%
        return round(tip, 1)
# Example 1: Poor service and food
print("Example 1 (Poor Service & Food):")
tip = calculate_tip(2, 3)
print(f"Tip: {tip}%")  # Output: 5%
# Example 2: Excellent service and food
print("\nExample 2 (Excellent Service & Food):")
tip = calculate_tip(9, 8)
print(f"Tip: {tip}%")  # Output: 20%
# Example 3: Mixed ratings
print("\nExample 3 (Mixed Service & Food):")
tip = calculate_tip(5, 7)
print(f"Tip: {tip}%")  # Output: 13.2%
```

Output:

```
Example 1 (Poor Service & Food):
Tip: 5%
Example 2 (Excellent Service & Food):
Tip: 20%
Example 3 (Mixed Service & Food):
Tip: 13.7%
```

Practical – 10

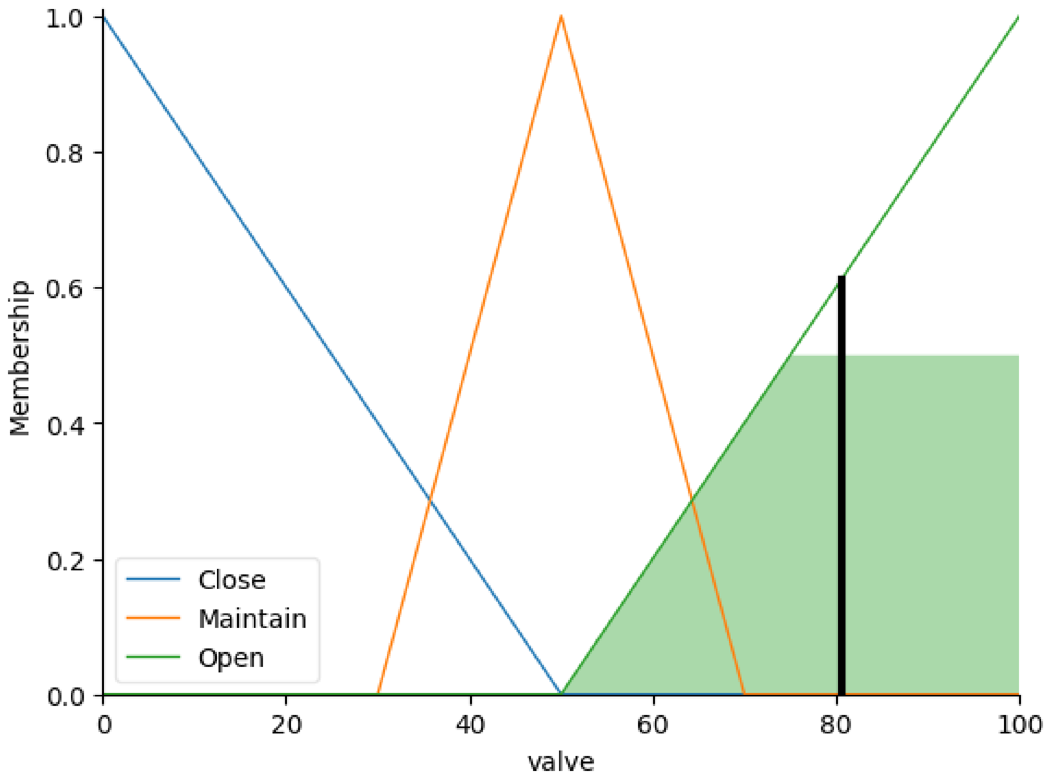
Aim: Implement Water Level Control in a Tank using Fuzzy Logic

Program:

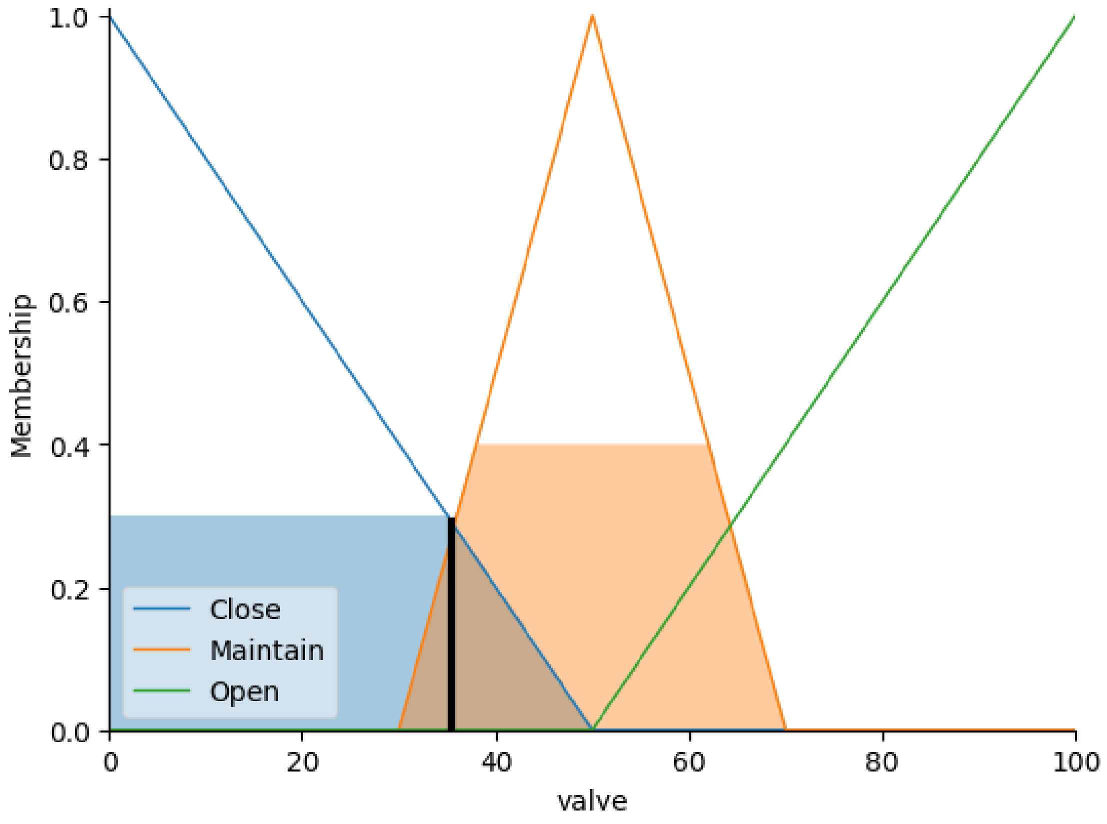
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
# Define input and output variables
# --------------------------------------------------------------------
# Input 1: Level Error (current level - setpoint)
level_error = ctrl.Antecedent(np.arange(-10, 11, 1), 'level_error')  # -10 cm to +10 cm
# Input 2: Rate of Change (derivative of level)
rate = ctrl.Antecedent(np.arange(-5, 5.1, 0.1), 'rate')            # -5 to +5 cm/min
# Output: Valve adjustment
valve = ctrl.Consequent(np.arange(0, 101, 1), 'valve')             # 0-100%
# Define fuzzy membership functions
# --------------------------------------------------------------------
# Level Error: Negative (below setpoint), Zero, Positive (above setpoint)
level_error['Negative'] = fuzz.trimf(level_error.universe, [-10, -10, 0])
level_error['Zero'] = fuzz.trimf(level_error.universe, [-5, 0, 5])
level_error['Positive'] = fuzz.trimf(level_error.universe, [0, 10, 10])
# Rate of Change: Decreasing, Steady, Increasing
rate['Decreasing'] = fuzz.trimf(rate.universe, [-5, -5, 0])
rate['Steady'] = fuzz.trimf(rate.universe, [-2.5, 0, 2.5])
rate['Increasing'] = fuzz.trimf(rate.universe, [0, 5, 5])
# Valve Adjustment: Close, Maintain, Open
valve['Close'] = fuzz.trimf(valve.universe, [0, 0, 50])
valve['Maintain'] = fuzz.trimf(valve.universe, [30, 50, 70])
valve['Open'] = fuzz.trimf(valve.universe, [50, 100, 100])
# Visualize membership functions (optional)
# level_error.view()
# rate.view()
# valve.view()
# plt.show()
# Define fuzzy rules
# --------------------------------------------------------------------
rule1 = ctrl.Rule(
    level_error['Negative'] & rate['Decreasing'],
    valve['Open']
)
rule2 = ctrl.Rule(
    level_error['Negative'] & rate['Steady'],
    valve['Open']
)
rule3 = ctrl.Rule(
    level_error['Negative'] & rate['Increasing'],
    valve['Maintain']
)
rule4 = ctrl.Rule(
    level_error['Zero'] & rate['Decreasing'],
    valve['Close']
)
rule5 = ctrl.Rule(
    level_error['Zero'] & rate['Steady'],
    valve['Maintain']
)
rule6 = ctrl.Rule(
    level_error['Zero'] & rate['Increasing'],
    valve['Open']
)
rule7 = ctrl.Rule(
    level_error['Positive'] & rate['Decreasing'],
    valve['Maintain']
)
rule8 = ctrl.Rule(
    level_error['Positive'] & rate['Steady'],
    valve['Close']
)
rule9 = ctrl.Rule(
    level_error['Positive'] & rate['Increasing'],
    valve['Close']
)
# Create control system
# ----------------------------------------------------------------------
tank_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9])
tank_sim = ctrl.ControlSystemSimulation(tank_ctrl)
# Example 1: Level is 5 cm below setpoint and decreasing
tank_sim.input['level_error'] = -5
tank_sim.input['rate'] = -2.5
tank_sim.compute()
valve_adjustment = tank_sim.output['valve']
print(f"Level Error: -5 cm, Rate: -2.5 cm/min")
print(f"Valve Adjustment: {valve_adjustment:.2f}% (Open)")
valve.view(sim=tank_sim)
plt.show()
# Example 2: Level is 3 cm above setpoint and steady
tank_sim.input['level_error'] = 3
tank_sim.input['rate'] = 0
tank_sim.compute()
valve_adjustment = tank_sim.output['valve']
print(f"\nLevel Error: +3 cm, Rate: 0 cm/min")
print(f"Valve Adjustment: {valve_adjustment:.2f}% (Close)")
valve.view(sim=tank_sim)
plt.show()
```

Output:

```
Level Error: -5 cm, Rate: -2.5 cm/min
Valve Adjustment: 80.56% (Open)
```



Level Error: +3 cm, Rate: 0 cm/min
Valve Adjustment: 35.41% (Close)


Membership
valve
Close
Maintain
Open