Aim: To design a Fuzzy control system using Fuzzy tool / library.

Theory:

A Fuzzy Control System (FCS) is a decision-making framework that uses the concepts of fuzzy logic to control processes or systems that are difficult to model using precise mathematical equations. Instead of working only with exact numerical data, a fuzzy control system uses linguistic rules (e.g., "If temperature is high and humidity is low, then fan speed is medium") to mimic human reasoning.

Fuzzy logic is particularly effective for nonlinear, time-varying, or uncertain systems where conventional control methods struggle. It is based on fuzzy set theory, which allows elements to have varying degrees of membership between 0 (completely false) and 1 (completely true).

Components of a Fuzzy Control System

A fuzzy control system typically consists of the following blocks:

1. Fuzzifier
    - Converts crisp numerical inputs into fuzzy values using membership functions.
    - Example: Temperature = 30°C → “Medium” with membership value 0.7.

2. Knowledge Base
    - Stores the definitions of fuzzy sets (membership functions) and the rule base containing IF–THEN rules.

3. Inference Engine
    - Evaluates the fuzzy rules and determines the fuzzy output by applying fuzzy reasoning methods (e.g., Mamdani, Sugeno).

4. Defuzzifier
    - Converts the fuzzy output back into a crisp numerical value for use in the real-world system.

Design Steps for a Fuzzy Control System

1. Identify variables – Select the system’s input(s) and output(s) with their numerical ranges (universe of discourse).
2. Define fuzzy sets – Assign linguistic terms (Low, Medium, High) and their corresponding membership functions.
3. Formulate fuzzy rules – Define expert-based IF–THEN rules relating inputs to outputs.
4. Select inference method – Choose a reasoning mechanism (commonly Mamdani).
5. Defuzzification – Apply a method such as Centroid of Area to get the final crisp output.
6. Simulation & tuning – Run the system using a fuzzy tool/library (e.g., scikit-fuzzy) and adjust parameters for better performance.

Mathematical Equations in Fuzzy Control

1. Fuzzy Set Definition

A fuzzy set A on universe X is:

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X\}$$

where $\mu_{\tilde{A}}(x) \in [0,1]$ is the membership degree.

2. Mamdani Rule Representation

For a rule:

$$\text{IF } x \text{ is } \tilde{A}_i \text{ AND } y \text{ is } \tilde{B}_i \text{ THEN } z \text{ is } \tilde{C}_i$$

- Firing strength:
$$\mu_{\tilde{C}_i'}(z) = \min(w_i,\ \mu_{\tilde{C}_i}(z))$$
- Implication (clipping):
$$\mu_{\tilde{C}}(z) = \max_i \mu_{\tilde{C}_i'}(z)$$
- Aggregation:
$$\mu_{\tilde{C}}(z) = \max_i \mu_{\tilde{C}_i'}(z)$$

Defuzzification (Centroid Method)

$$z^* = \frac{\sum_k z_k \cdot \mu_{\tilde{C}}(z_k)}{\sum_k \mu_{\tilde{C}}(z_k)}$$

where z^* is the crisp output.

Advantages of Fuzzy Control

- Can model complex, ill-defined, and nonlinear systems.
- Uses expert knowledge without requiring exact mathematical models.
- Produces smooth control actions.

Limitations

- Requires careful tuning of membership functions and rules.
- Performance depends heavily on the quality of the rule base.

Code:

```
# Install required library
!pip install scikit-fuzzy
```

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
```

```
# Cloud cover in percentage
cloud_cover = ctrl.Antecedent(np.arange(0, 101, 1), 'cloud_cover')
```

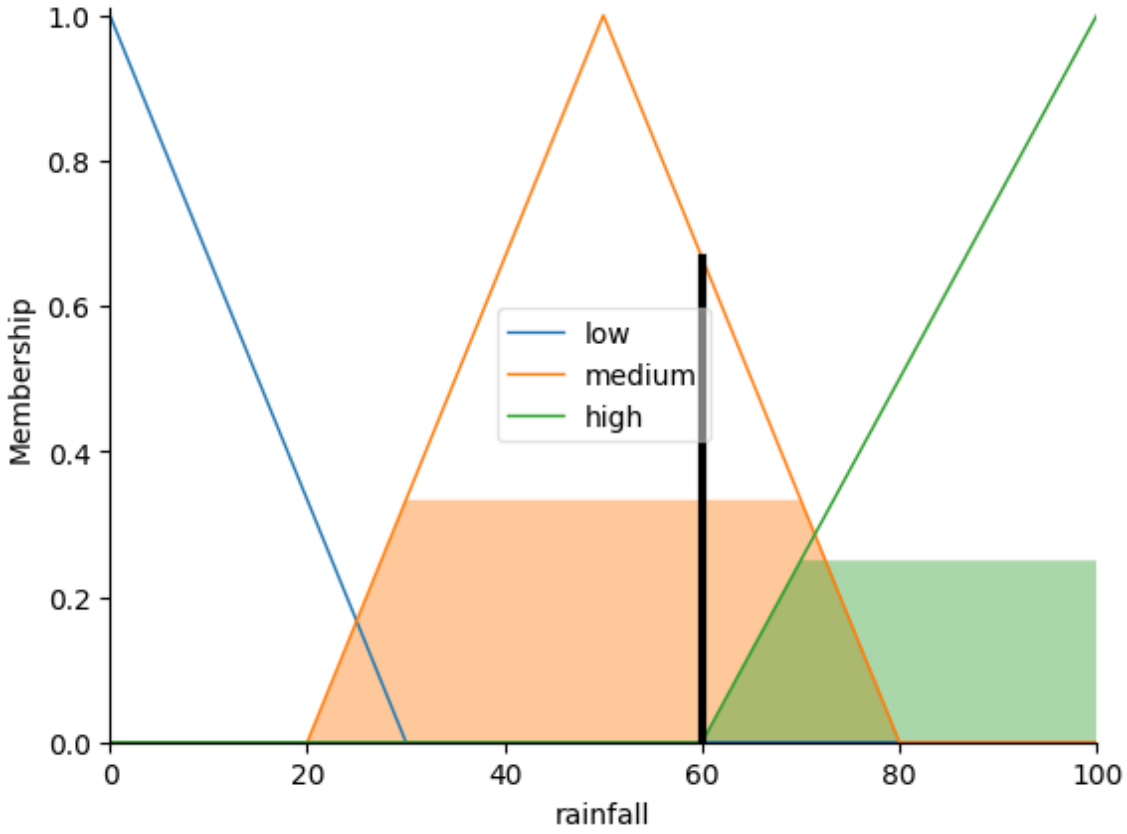
```
# Humidity in percentage
humidity = ctrl.Antecedent(np.arange(0, 101, 1), 'humidity')
```

```
# Rainfall in mm
```

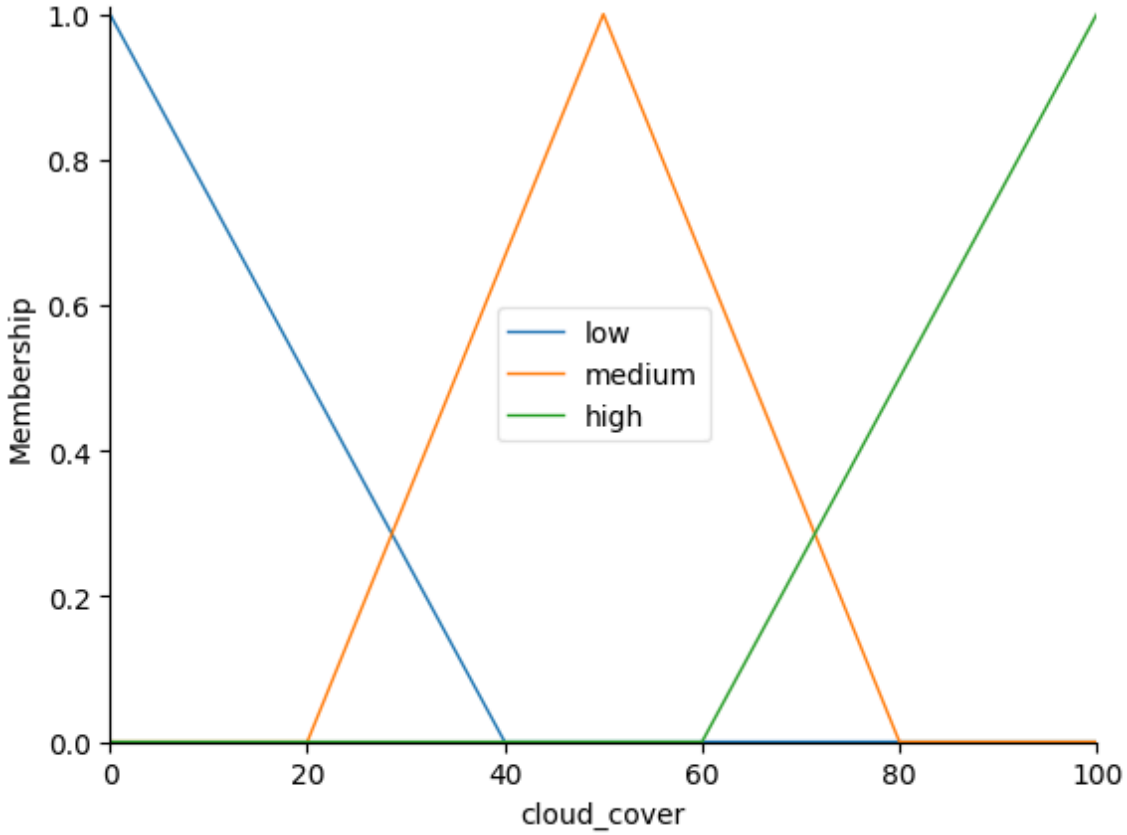
```
rainfall = ctrl.Consequent(np.arange(0, 101, 1), 'rainfall')

# Cloud Cover
cloud_cover['low'] = fuzz.trimf(cloud_cover.universe, [0, 0, 40])
cloud_cover['medium'] = fuzz.trimf(cloud_cover.universe, [20, 50, 80])
cloud_cover['high'] = fuzz.trimf(cloud_cover.universe, [60, 100, 100])

# Humidity
humidity['low'] = fuzz.trimf(humidity.universe, [0, 0, 40])
humidity['medium'] = fuzz.trimf(humidity.universe, [20, 50, 80])
humidity['high'] = fuzz.trimf(humidity.universe, [60, 100, 100])

# Rainfall Output
rainfall['low'] = fuzz.trimf(rainfall.universe, [0, 0, 30])
rainfall['medium'] = fuzz.trimf(rainfall.universe, [20, 50, 80])
rainfall['high'] = fuzz.trimf(rainfall.universe, [60, 100, 100])

rule1 = ctrl.Rule(cloud_cover['high'] & humidity['high'], rainfall['high'])
rule2 = ctrl.Rule(cloud_cover['medium'] & humidity['high'], rainfall['medium'])
rule3 = ctrl.Rule(cloud_cover['high'] & humidity['medium'], rainfall['medium'])
rule4 = ctrl.Rule(cloud_cover['low'] & humidity['high'], rainfall['medium'])
rule5 = ctrl.Rule(cloud_cover['medium'] & humidity['medium'], rainfall['low'])
rule6 = ctrl.Rule(cloud_cover['low'] & humidity['low'], rainfall['low'])

rainfall_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6])
rainfall_sim = ctrl.ControlSystemSimulation(rainfall_ctrl)

# Example: Cloud Cover = 70%, Humidity = 80%
rainfall_sim.input['cloud_cover'] = 70
rainfall_sim.input['humidity'] = 80

# Compute output
rainfall_sim.compute()

print("Predicted Rainfall (mm):", rainfall_sim.output['rainfall'])
```

```
Predicted Rainfall (mm): 59.990785108735686
```

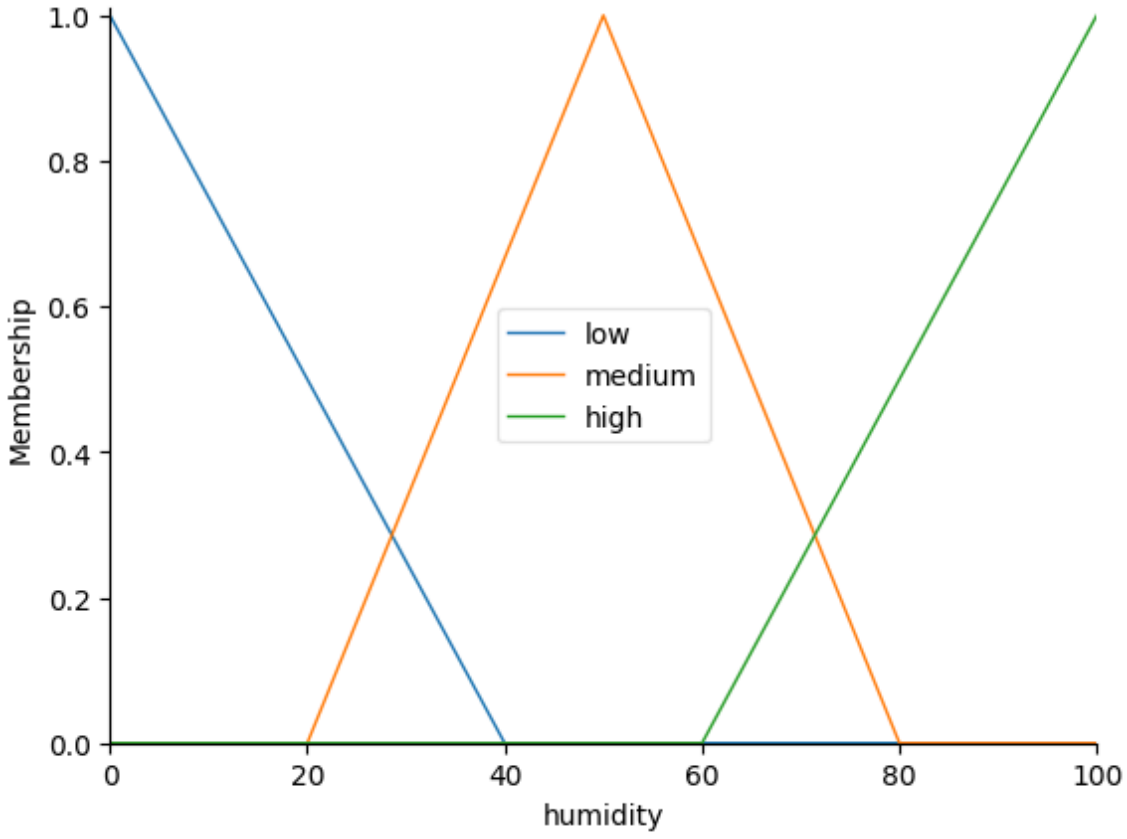
```
# Visualize Result
rainfall.view(sim=rainfall_sim)
plt.show()
```



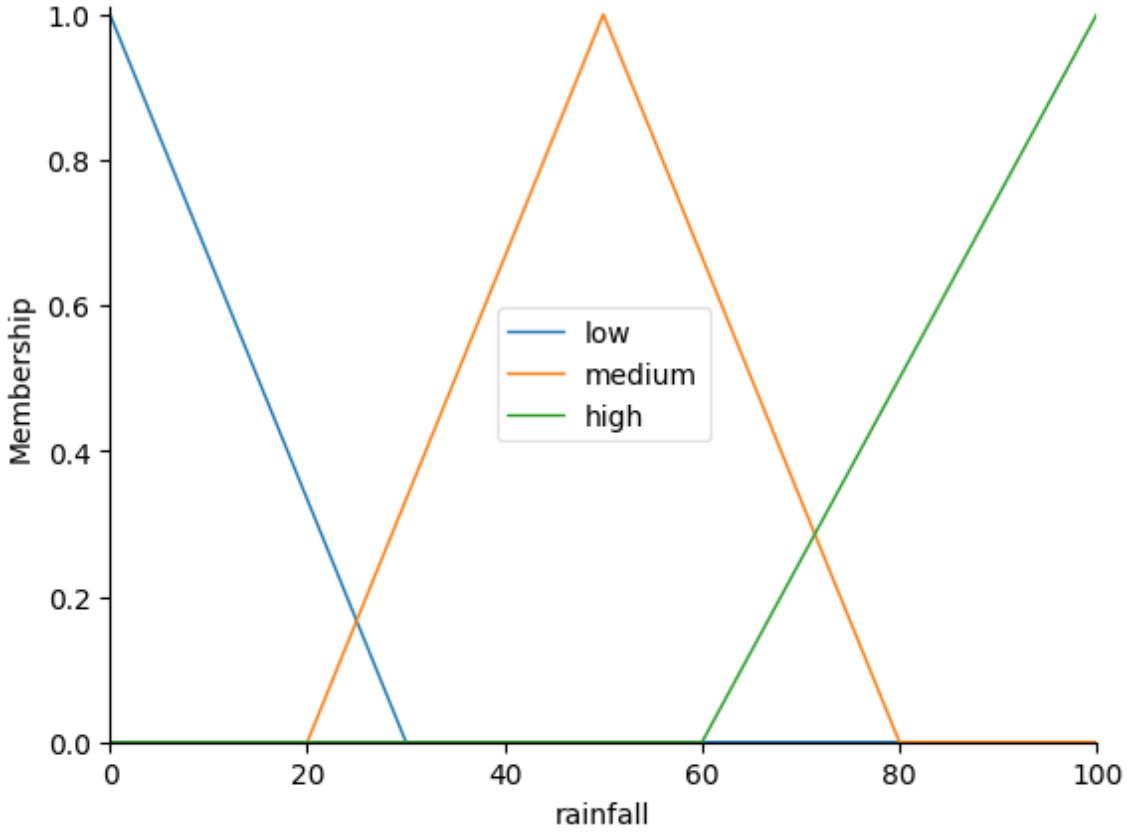

```
# View Membership Functions
cloud_cover.view()
```



```
humidity.view()
```




```
rainfall.view()
```



Conclusion:

In this experiment, we successfully designed and implemented a fuzzy control system using a fuzzy logic library. By defining input and output variables, assigning membership functions, and formulating a rule base, the system was able to process crisp inputs and generate a meaningful crisp output through fuzzification, inference, and defuzzification. This demonstrated how fuzzy control can effectively handle uncertain and nonlinear problems in a human-like decision-making manner.

AI&DS2_Expt_08