assignment2

August 14, 2024

0.1 Ramesh Chandra Soren

Enrollment No: 2022CSB086
Department: Computer Science and Technology

1 Question 1

Let A(x) and B(x) be two fuzzy sets of speed limit on a highway, defined by the following membership functions:

1.1 a. Fuzzy set A: "Low speed limit"

```
A(x) = { 1,                 if x < 30 (low speed)
         (x-30)/(50-30),    if 30 < x < 50 (moderately low speed)
         (x-50)/(70-50),    if 50 < x < 70 (increasing speed)
         0,                 if x > 70 (not low speed)
       }
```

1.2 b. Fuzzy set B: "High speed limit"

```
B(x) = { 0,                 if x < 60 (Not high speed)
         (x-60)/(80-60),    if 60 < x < 80 (moderately high speed)
         (x-80)/(100-80),   if 80 < x < 100 (increasing speed)
         1,                 if x > 100 (high speed)
       }
```

Now, perform the following operations to obtain the resultant set and plot the same:

i. Union - (A B)x = max(A(x), B(x))

ii. Intersection - (A B)x = min(A(x), B(x))

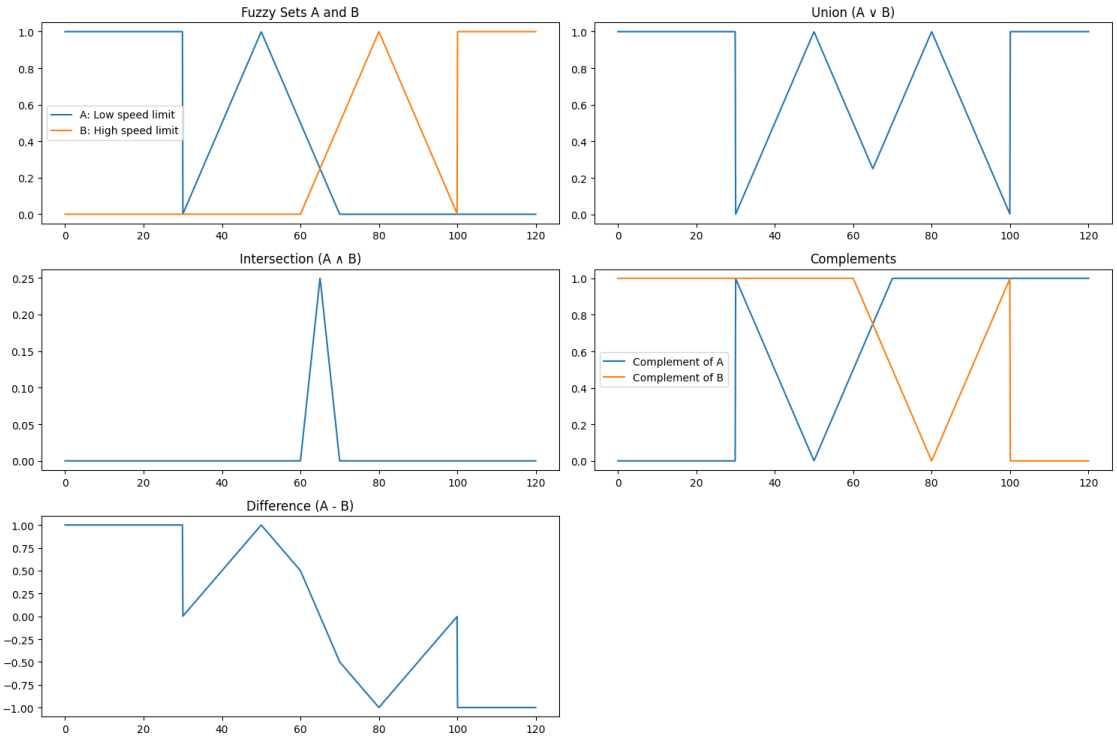
iii. Complement of both sets - A^(-1)(x) = 1 - A(x), B^(-1)(x) = 1 - B(x)

iv. Difference of the two sets – (A - B)x = (A(x) - B(x))

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def mu_A(x):
    if x < 30:
```

```
        return 1
    elif 30 <= x < 50:
        return (x - 30) / (50 - 30)
    elif 50 <= x < 70:
        return (70 - x) / (70 - 50)
    else:
        return 0

def mu_B(x):
    if x < 60:
        return 0
    elif 60 <= x < 80:
        return (x - 60) / (80 - 60)
    elif 80 <= x < 100:
        return (100 - x) / (100 - 80)
    else:
        return 1

# Generate x values
x = np.linspace(0, 120, 1000)

# Calculate membership values
y_A = [mu_A(xi) for xi in x]
y_B = [mu_B(xi) for xi in x]

# i. Union
union = np.maximum(y_A, y_B)

# ii. Intersection
intersection = np.minimum(y_A, y_B)

# iii. Complement
complement_A = 1 - np.array(y_A)
complement_B = 1 - np.array(y_B)

# iv. Difference
difference = np.array(y_A) - np.array(y_B)

# Plotting
plt.figure(figsize=(15, 10))

plt.subplot(3, 2, 1)
plt.plot(x, y_A, label='A: Low speed limit')
plt.plot(x, y_B, label='B: High speed limit')
plt.title('Fuzzy Sets A and B')
plt.legend()
```

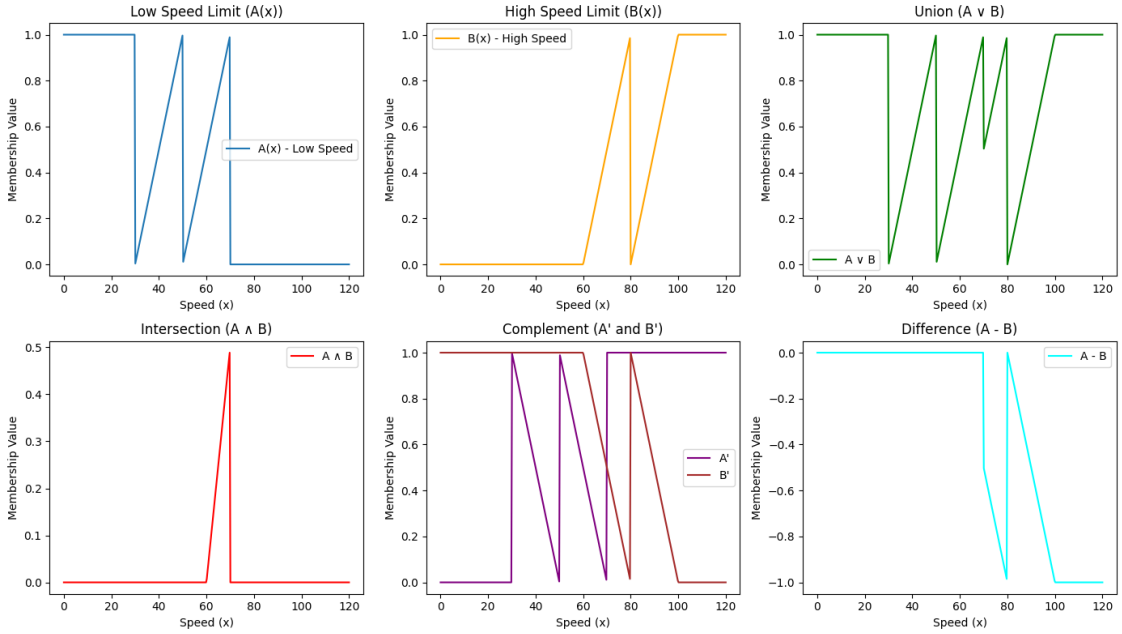
```
plt.subplot(3, 2, 2)
plt.plot(x, union)
plt.title('Union (A   B)')

plt.subplot(3, 2, 3)
plt.plot(x, intersection)
plt.title('Intersection (A   B)')

plt.subplot(3, 2, 4)
plt.plot(x, complement_A, label='Complement of A')
plt.plot(x, complement_B, label='Complement of B')
plt.title('Complements')
plt.legend()

plt.subplot(3, 2, 5)
plt.plot(x, difference)
plt.title('Difference (A - B)')

plt.tight_layout()
plt.show()
```



[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
# Membership function for low speed limit A(x)
def mu_A(x):
    if x < 30:
        return 1
    elif 30 <= x < 50:
        return (x - 30) / (50 - 30)
    elif 50 <= x < 70:
        return (x - 50) / (70 - 50)
    else:
        return 0

# Membership function for high speed limit B(x)
def mu_B(x):
    if x < 60:
        return 0
    elif 60 <= x < 80:
        return (x - 60) / (80 - 60)
    elif 80 <= x < 100:
        return (x - 80) / (100 - 80)
    else:
        return 1

# Defining x range (speed)
x_values = np.linspace(0, 120, 400)

# Calculate membership values
mu_A_values = np.array([mu_A(x) for x in x_values])
mu_B_values = np.array([mu_B(x) for x in x_values])

# Perform operations
union_values = np.maximum(mu_A_values, mu_B_values)
intersection_values = np.minimum(mu_A_values, mu_B_values)
complement_A_values = 1 - mu_A_values
complement_B_values = 1 - mu_B_values
difference_values = np.minimum(0,mu_A_values - mu_B_values)

# Plotting
plt.figure(figsize=(14, 8))

plt.subplot(2, 3, 1)
plt.plot(x_values, mu_A_values, label='A(x) - Low Speed')
plt.title("Low Speed Limit (A(x))")
plt.xlabel("Speed (x)")
plt.ylabel("Membership Value")
plt.legend()
```

```
plt.subplot(2, 3, 2)
plt.plot(x_values, mu_B_values, label='B(x) - High Speed', color='orange')
plt.title("High Speed Limit (B(x))")
plt.xlabel("Speed (x)")
plt.ylabel("Membership Value")
plt.legend()

plt.subplot(2, 3, 3)
plt.plot(x_values, union_values, label='A   B', color='green')
plt.title("Union (A   B)")
plt.xlabel("Speed (x)")
plt.ylabel("Membership Value")
plt.legend()

plt.subplot(2, 3, 4)
plt.plot(x_values, intersection_values, label='A   B', color='red')
plt.title("Intersection (A   B)")
plt.xlabel("Speed (x)")
plt.ylabel("Membership Value")
plt.legend()

plt.subplot(2, 3, 5)
plt.plot(x_values, complement_A_values, label='A\'', color='purple')
plt.plot(x_values, complement_B_values, label='B\'', color='brown')
plt.title("Complement (A' and B')")
plt.xlabel("Speed (x)")
plt.ylabel("Membership Value")
plt.legend()

plt.subplot(2, 3, 6)
plt.plot(x_values, difference_values, label='A - B', color='cyan')
plt.title("Difference (A - B)")
plt.xlabel("Speed (x)")
plt.ylabel("Membership Value")
plt.legend()

plt.tight_layout()
plt.show()
```



2 Fuzzy Set Temperature Analysis

2.1 Question 2

Let C(x) and D(x) be two fuzzy sets of temperature (Celsius) in a city, defined by the following membership functions:

2.1.1 a. Fuzzy set C: "Cold"

```
C(x) = { 0,            if x <= -10 (extremely cold)
         (x+10)/10,    if -10 < x < 0 (very cold)
         1,            if 0 <= x <= 5 (cold)
         0,            if x > 5 (not cold)
       }
```

2.1.2 b. Fuzzy set D: "Warm"

```
D(x) ={ 0,             if x < 25 (Not warm)
        (x-25)/10,     if 25 <= x < 35 (moderately warm)
        1,             if 35 <= x <= 40 (warm)
        0,             if x > 40 (extremely warm)
      }
```

2.2 Tasks

Perform the following operations to obtain the result:

1. Max-Min Composition: (C D)x = max(xi X min(C(xi), D(xi)))

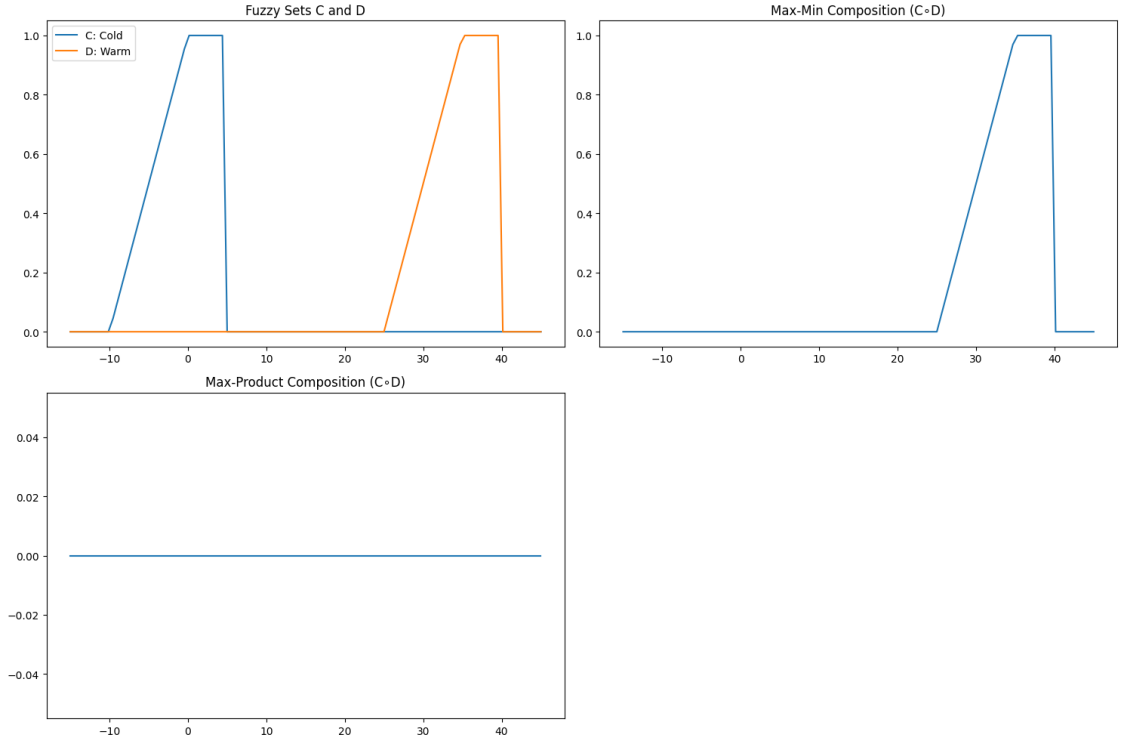
2. Max-Product Composition: (C D)x = max(xi X (C(xi) * D(xi)))

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

def mu_C(x):
    if x < -10:
        return 0
    elif -10 < x < 0:
        return (x + 10) / 10
    elif 0 < x < 5:
        return 1
    else:
        return 0

def mu_D(x):
    if x < 25:
        return 0
    elif 25 < x < 35:
        return (x - 25) / 10
    elif 35 < x < 40:
        return 1
    else:
        return 0

import numpy as np

# Generate x values covering both membership function ranges
x = np.linspace(-15, 45, 100)

# Calculate membership values
y_C = np.array([mu_C(xi) for xi in x])
y_D = np.array([mu_D(xi) for xi in x])

# i. Max-Min Composition
max_min_comp = np.zeros_like(x)
for i in range(len(x)):
    min_values = [min(mu_C(x[j]), mu_D(x[i])) for j in range(len(x))]
    max_min_comp[i] = max(min_values)

# ii. Max-Product Composition
max_prod_comp = np.maximum.reduce([y_C * y_D])

# Display results
print("Max-Min Composition values:")
print(max_min_comp)
```

```
print("\nMax-Product Composition values:")
print(max_prod_comp)


# Plotting
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.plot(x, y_C, label='C: Cold')
plt.plot(x, y_D, label='D: Warm')
plt.title('Fuzzy Sets C and D')
plt.legend()

plt.subplot(2, 2, 2)
plt.plot(x, max_min_comp)
plt.title('Max-Min Composition (C D)')

plt.subplot(2, 2, 3)
plt.plot(x, max_prod_comp)
plt.title('Max-Product Composition (C D)')

plt.tight_layout()
plt.show()
```

```
Max-Min Composition values:
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.06060606 0.12121212 0.18181818 0.24242424 0.3030303
 0.36363636 0.42424242 0.48484848 0.54545455 0.60606061 0.66666667
 0.72727273 0.78787879 0.84848485 0.90909091 0.96969697 1.
 1.         1.         1.         1.         1.         1.
 1.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]

Max-Product Composition values:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0.]
```

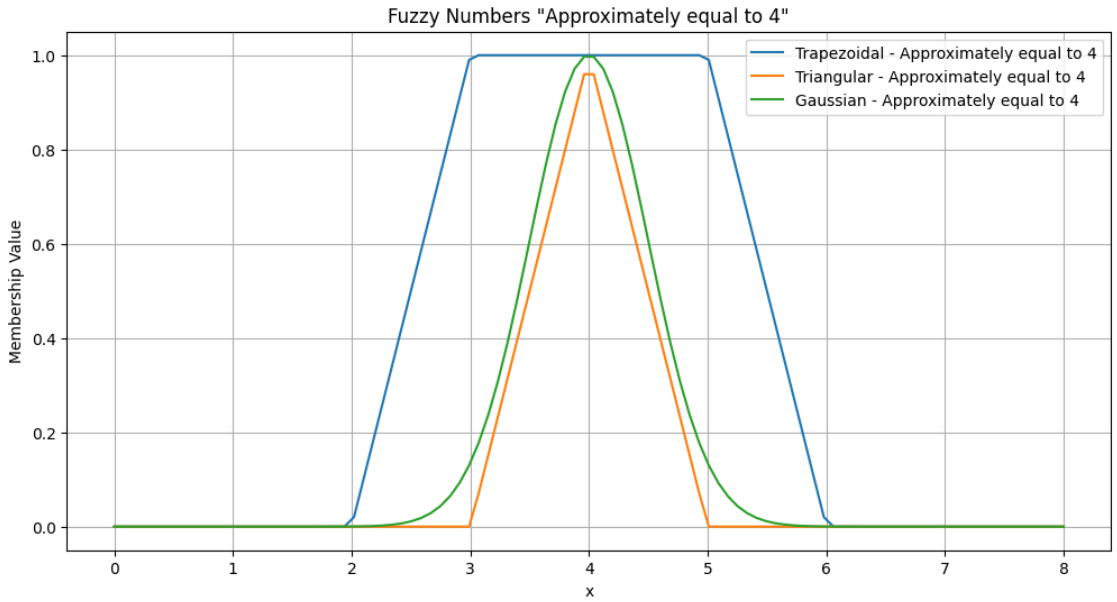


2 (c) Apply your intuition to develop fuzzy numbers "Approximately equal to 4" using the following membership functions having shapes:

(i) Trapezoidal
(ii) Triangular
(iii) Gaussian

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Trapezoidal Membership Function
def trapezoidal(x, a, b, c, d):
    if x <= a or x >= d:
        return 0
    elif a < x < b:
        return (x - a) / (b - a)
    elif b <= x <= c:
        return 1
    elif c < x < d:
        return (d - x) / (d - c)
```

```python
# Triangular Membership Function
def triangular(x, a, b, c):
    if x <= a or x >= c:
        return 0
    elif a < x <= b:
        return (x - a) / (b - a)
    elif b < x < c:
        return (c - x) / (c - b)

# Gaussian Membership Function
def gaussian(x, m, sigma):
    return np.exp(-((x - m) ** 2) / (2 * sigma ** 2))

# Generate x values
x = np.linspace(0, 8, 100)

# Calculate membership values
y_trapezoidal = np.array([trapezoidal(xi, 2, 3, 5, 6) for xi in x])
y_triangular = np.array([triangular(xi, 3, 4, 5) for xi in x])
y_gaussian = gaussian(x, 4, 0.5)

# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(x, y_trapezoidal, label='Trapezoidal - Approximately equal to 4')
plt.plot(x, y_triangular, label='Triangular - Approximately equal to 4')
plt.plot(x, y_gaussian, label='Gaussian - Approximately equal to 4')
plt.title('Fuzzy Numbers "Approximately equal to 4"')
plt.xlabel('x')
plt.ylabel('Membership Value')
plt.legend()
plt.grid(True)
plt.show()
```



2 (d) Design a Mamdani Fuzzy Inference System to control the FAN-SPEED of a furnace by inputting TEMPERATURE of a thermostat of a household.

Frame the If-Then rules using Linguistic variables input TEMPERATURE and output FAN-SPEED.

Sample rule: "IF TEMPERATURE is 85, THEN increase furnace FAN-SPEED to 300 RPM"

(i) Write fuzzy rules (as many as possible) using the linguistic variable TEMPERATURE and FAN-SPEED.

Consider Fuzzy variables for TEMPERATURE like "Risky", "Average", "Excellent" and FAN-SPEED like "Slow", "High", "Moderate" etc.

(ii) Fuzzify the input and output fuzzy variables with appropriate membership functions.

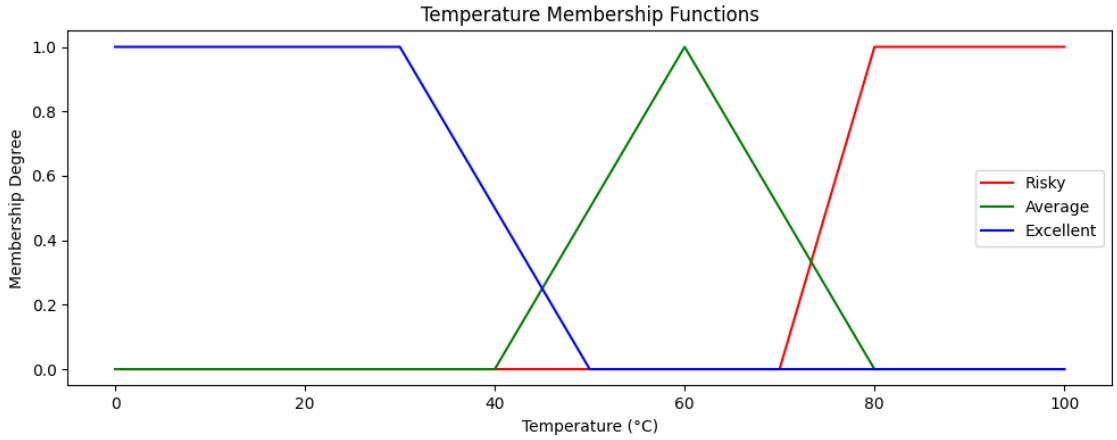
(iii) Aggregate the rules using Mamdani model

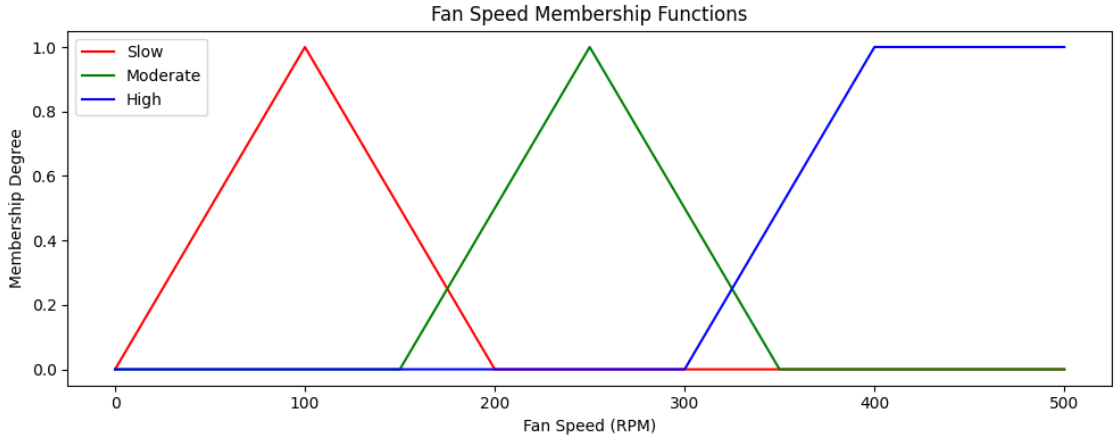
(iv) Apply Centroid defuzzification method to calculate the crisp value of FAN-SPEED.

[ ]:
```
# Need to install fuzzy library
!pip install scikit-fuzzy
```

```
Requirement already satisfied: scikit-fuzzy in /usr/local/lib/python3.10/dist-
packages (0.4.2)
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-fuzzy) (1.26.4)
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-fuzzy) (1.13.1)
Requirement already satisfied: networkx>=1.9.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-fuzzy) (3.3)
```

[ ]:
```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt

# Define the range for temperature and fan speed
temp_range = np.arange(0, 101, 1)
fan_speed_range = np.arange(0, 501, 1)

# Fuzzy membership functions for Temperature
temp_risky = fuzz.trapmf(temp_range, [70, 80, 100, 100])
temp_average = fuzz.trimf(temp_range, [40, 60, 80])
temp_excellent = fuzz.trapmf(temp_range, [0, 0, 30, 50])

# Fuzzy membership functions for Fan Speed
fan_slow = fuzz.trimf(fan_speed_range, [0, 100, 200])
fan_moderate = fuzz.trimf(fan_speed_range, [150, 250, 350])
fan_high = fuzz.trapmf(fan_speed_range, [300, 400, 500, 500])

# Plot the membership functions
plt.figure(figsize=(10, 8))

plt.subplot(2, 1, 1)
plt.plot(temp_range, temp_risky, 'r', label='Risky')
plt.plot(temp_range, temp_average, 'g', label='Average')
plt.plot(temp_range, temp_excellent, 'b', label='Excellent')
plt.title('Temperature Membership Functions')
plt.xlabel('Temperature (°C)')
plt.ylabel('Membership Degree')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(fan_speed_range, fan_slow, 'r', label='Slow')
plt.plot(fan_speed_range, fan_moderate, 'g', label='Moderate')
plt.plot(fan_speed_range, fan_high, 'b', label='High')
plt.title('Fan Speed Membership Functions')
plt.xlabel('Fan Speed (RPM)')
plt.ylabel('Membership Degree')
plt.legend()

plt.tight_layout()
plt.show()
```





[ ]:
```
# Example crisp input
crisp_temperature = 65

# Fuzzification
temp_risk_level = fuzz.interp_membership(temp_range, temp_risky,␣
 ↪crisp_temperature)
temp_avg_level = fuzz.interp_membership(temp_range, temp_average,␣
 ↪crisp_temperature)
temp_exc_level = fuzz.interp_membership(temp_range, temp_excellent,␣
 ↪crisp_temperature)

# Rule Evaluation
# Rule 1: If Temperature is Risky, then Fan Speed is High
rule1 = temp_risk_level
fan_activation_high = np.fmin(rule1, fan_high)

# Rule 2: If Temperature is Average, then Fan Speed is Moderate
rule2 = temp_avg_level
fan_activation_moderate = np.fmin(rule2, fan_moderate)
```

```
# Rule 3: If Temperature is Excellent, then Fan Speed is Slow
rule3 = temp_exc_level
fan_activation_slow = np.fmin(rule3, fan_slow)

# Aggregation
aggregated = np.fmax(fan_activation_slow, np.fmax(fan_activation_moderate,␣
 ↪fan_activation_high))

# Defuzzification
fan_speed = fuzz.defuzz(fan_speed_range, aggregated, 'centroid')
fan_speed_activation = fuzz.interp_membership(fan_speed_range, aggregated,␣
 ↪fan_speed)

print(f"Crisp output (Fan Speed): {fan_speed} RPM")
```

```
Crisp output (Fan Speed): 250.0000000000004 RPM
```