fuzzy-roksana-jandura-eng

May 14, 2025

1 A fuzzy logic-based advisory system that supports decision-making on whether to increase the production of a given product.

[2]:
```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
from skfuzzy import control as ctrl
```

Such systems are becoming increasingly important in modern industry and business, where production decisions must be made dynamically and often under conditions of uncertainty. In situations where data is imprecise, fuzzy, or difficult to interpret clearly, traditional decision-making models may be insufficient. That is why fuzzy logic is widely used as a tool to support intelligent decision-making.

The significance of this type of advisory system in the market is substantial, especially for manufacturing companies that need to balance growing demand, limited storage capacity, and fluctuating production costs. Implementing such an approach can lead to better production planning, cost optimization, and the avoidance of losses related to overproduction or stock shortages.

In the designed system, I used **three input variables** that play a key role in making decisions about the scale of production:

- **Market demand** – indicates how high the demand for a given product is. High demand may suggest the need to increase production, while low demand may lead to its reduction.
- **Inventory level** – provides information about the current stock of finished products. Low inventory may require rapid replenishment, whereas excessively high levels indicate a risk of overproduction.
- **Unit production cost** – reflects the cost-efficiency of manufacturing. High costs may discourage increasing production, even in the case of high demand, while low costs encourage expansion.

Output variable in the system is a recommendation regarding the adjustment of the production level. Based on the values of the input variables (demand, inventory, production cost), the system determines the degree of need to adjust the production scale.

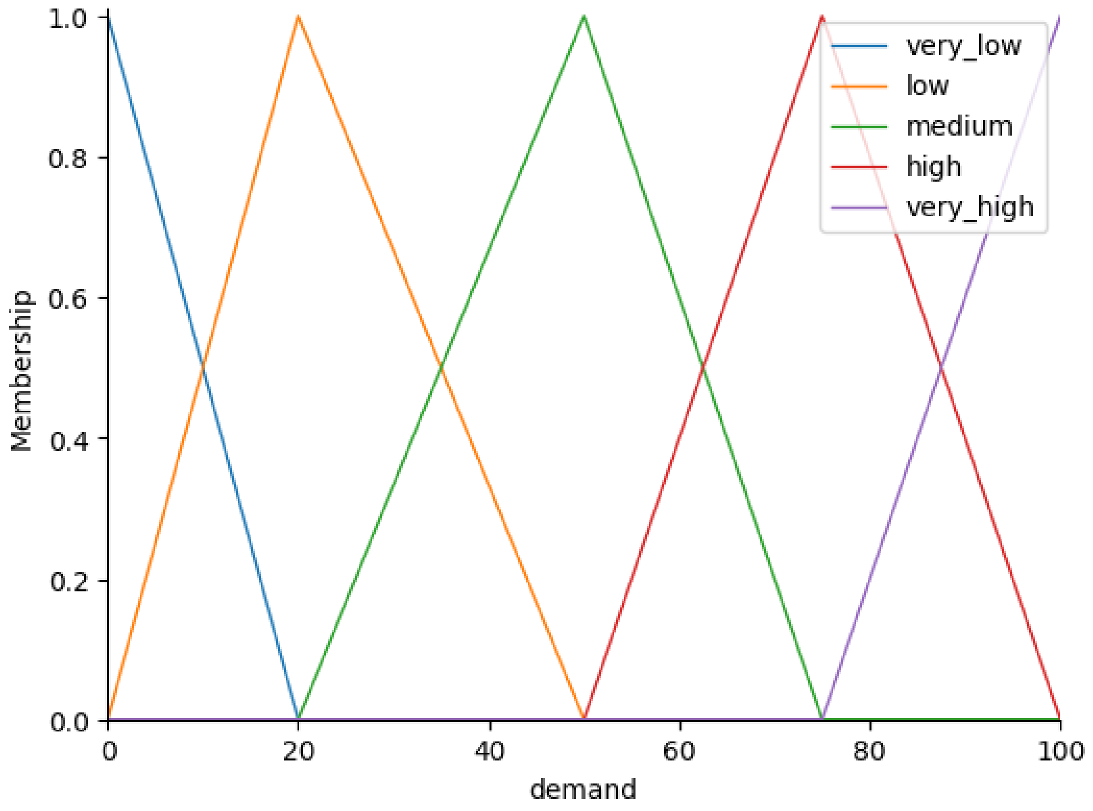
I decided to use five output levels to better capture the nuances of real-world decisions. The range was extended to values from -10 to 10, where:

- negative values indicate the need to decrease production,
- positive values suggest the need to increase production,
- and a value of 0 represents a decision to maintain the current level.

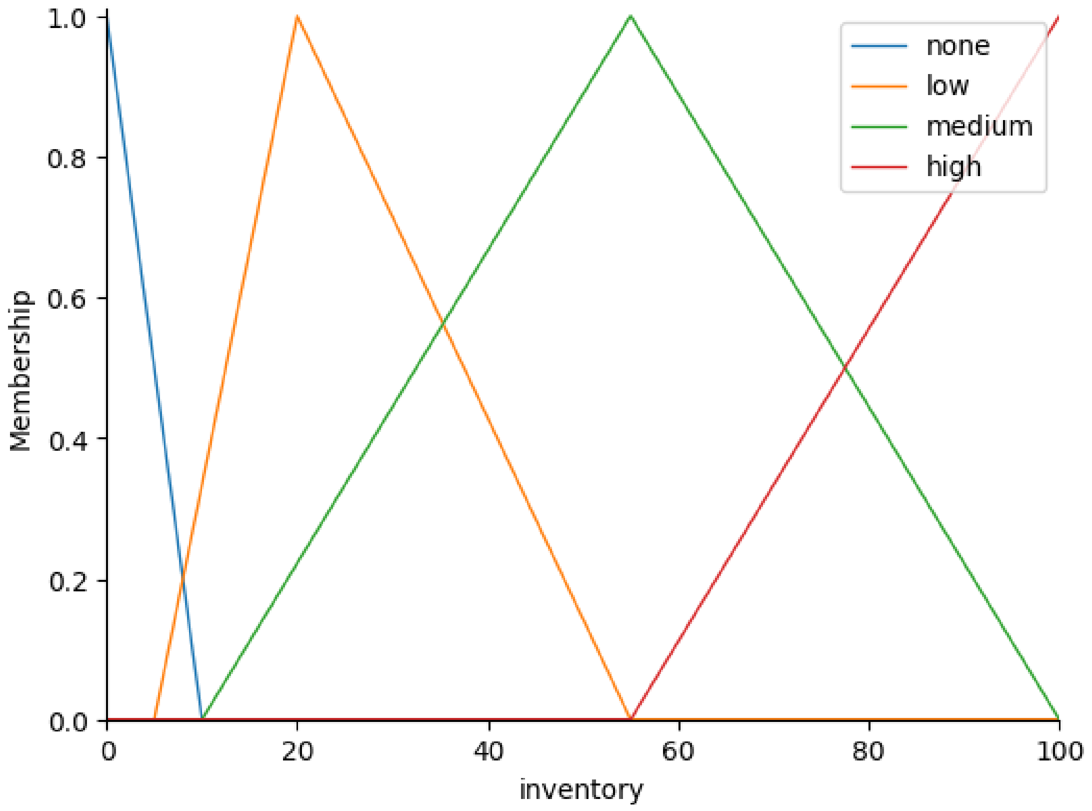
This setup allows for a natural interpretation of the final result – for example, a value of −6 can be interpreted as "decrease production by 60%", while +8 means "increase production by 80%".

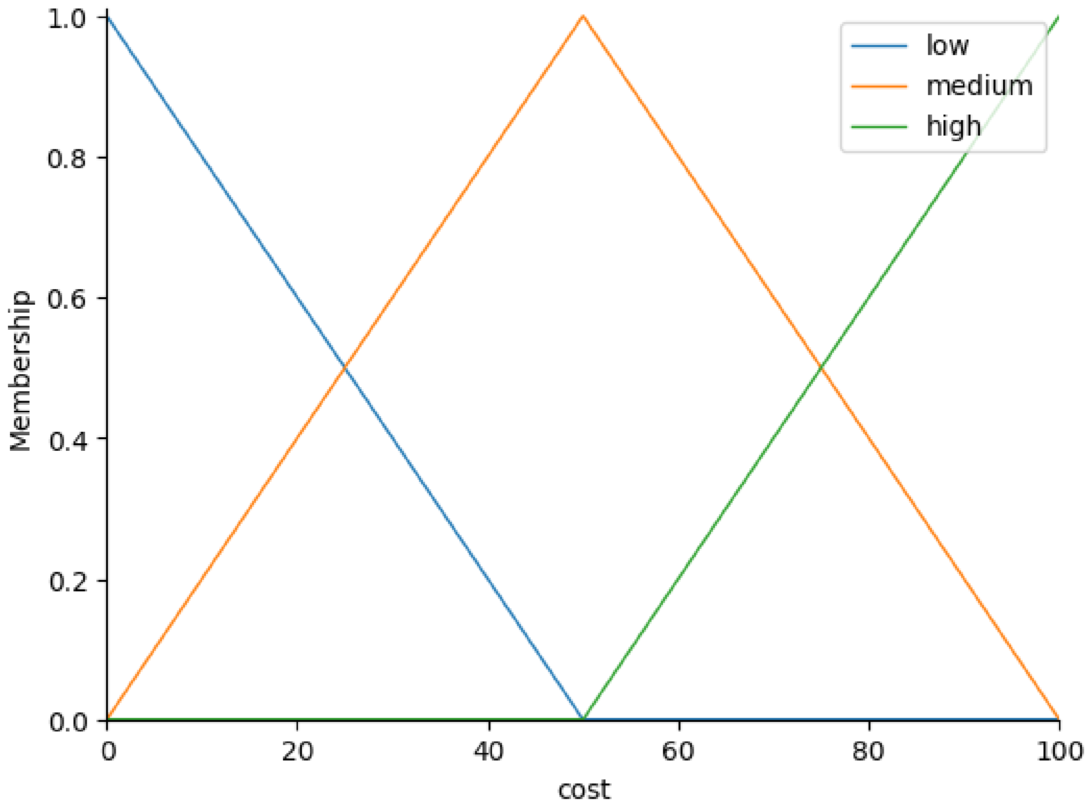
[3]:
```python
# Input variables (Antecedents)
demand = ctrl.Antecedent(np.arange(0, 101, 1), 'demand')
inventory = ctrl.Antecedent(np.arange(0, 101, 1), 'inventory')
cost = ctrl.Antecedent(np.arange(0, 101, 1), 'cost')

# Fuzzy membership functions for demand
demand['very_low'] = fuzz.trimf(demand.universe, [0, 0, 20])
demand['low'] = fuzz.trimf(demand.universe, [0, 20, 50])
demand['medium'] = fuzz.trimf(demand.universe, [20, 50, 75])
demand['high'] = fuzz.trimf(demand.universe, [50, 75, 100])
demand['very_high'] = fuzz.trimf(demand.universe, [75, 100, 100])

demand.view()
plt.legend(loc=1)
plt.show()

# Fuzzy membership functions for inventory
inventory['none'] = fuzz.trimf(inventory.universe, [0, 0, 10])
inventory['low'] = fuzz.trimf(inventory.universe, [5, 20, 55])
inventory['medium'] = fuzz.trimf(inventory.universe, [10, 55, 100])
inventory['high'] = fuzz.trimf(inventory.universe, [55, 100, 100])

inventory.view()
plt.legend(loc=1)
plt.show()

# Fuzzy membership functions for cost
cost['low'] = fuzz.trimf(cost.universe, [0, 0, 50])
cost['medium'] = fuzz.trimf(cost.universe, [0, 50, 100])
cost['high'] = fuzz.trimf(cost.universe, [50, 100, 100])

cost.view()
plt.legend(loc=1)
plt.show()
```

```
C:\Users\Dell\AppData\Local\Programs\Python\Python311\Lib\site-
packages\skfuzzy\control\fuzzyvariable.py:125: UserWarning: FigureCanvasAgg is
non-interactive, and thus cannot be shown
  fig.show()
```

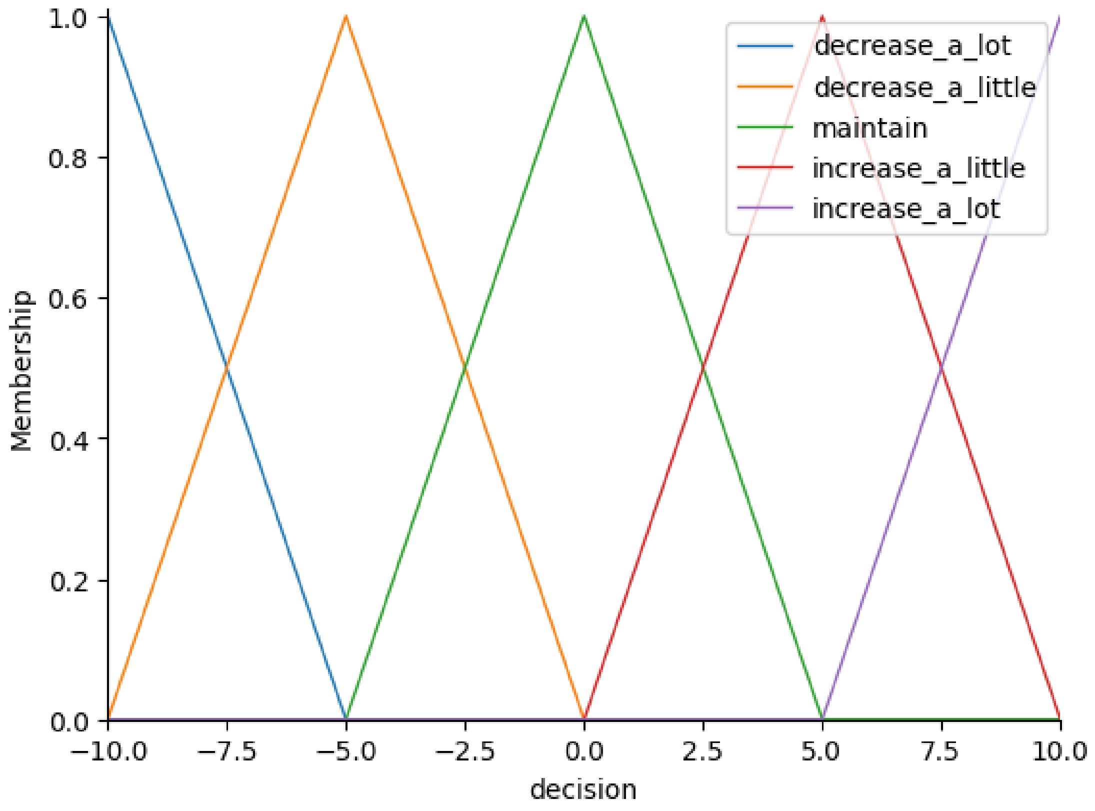


very_low
low
medium
high
very_high
Membership
demand
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
20
40
60
80
100
inventory
none
low
medium
high



[4]:
```python
# Output variable (Consequent)
decision = ctrl.Consequent(np.arange(-10, 10.1, 0.1), 'decision')

decision['decrease_a_lot'] = fuzz.trimf(decision.universe, [-10, -10, -5])
decision['decrease_a_little'] = fuzz.trimf(decision.universe, [-10, -5, 0])
decision['maintain'] = fuzz.trimf(decision.universe, [-5, 0, 5])
decision['increase_a_little'] = fuzz.trimf(decision.universe, [0, 5, 10])
decision['increase_a_lot'] = fuzz.trimf(decision.universe, [5, 10, 10])

decision.view()
plt.legend(loc=1)
plt.show()
```



1.0.1 Rules

```
[5]: rules = [
        ctrl.Rule(demand['very_low'] & inventory['none'] & cost['low'],␣
     ↪decision['maintain']),
        ctrl.Rule(demand['very_low'] & inventory['none'] & cost['medium'],␣
     ↪decision['maintain']),
        ctrl.Rule(demand['very_low'] & inventory['none'] & cost['high'],␣
     ↪decision['decrease_a_lot']),
        ctrl.Rule(demand['very_low'] & inventory['low'] & cost['low'],␣
     ↪decision['maintain']),
        ctrl.Rule(demand['very_low'] & inventory['low'] & cost['medium'],␣
     ↪decision['maintain']),
        ctrl.Rule(demand['very_low'] & inventory['low'] & cost['high'],␣
     ↪decision['maintain']),
        ctrl.Rule(demand['very_low'] & inventory['medium'] & cost['low'],␣
     ↪decision['maintain']),
        ctrl.Rule(demand['very_low'] & inventory['medium'] & cost['medium'],␣
     ↪decision['maintain']),
    ctrl.Rule(demand['very_low'] & inventory['medium'] & cost['high'],␣
↪decision['decrease_a_little']),
    ctrl.Rule(demand['very_low'] & inventory['high'] & cost['low'],␣
↪decision['decrease_a_little']),
    ctrl.Rule(demand['very_low'] & inventory['high'] & cost['medium'],␣
↪decision['decrease_a_little']),
    ctrl.Rule(demand['very_low'] & inventory['high'] & cost['high'],␣
↪decision['decrease_a_lot']),

    ctrl.Rule(demand['low'] & inventory['none'] & cost['low'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['none'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['none'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['low'] & cost['low'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['low'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['low'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['medium'] & cost['low'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['medium'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['medium'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['low'] & inventory['high'] & cost['low'],␣
↪decision['decrease_a_little']),
    ctrl.Rule(demand['low'] & inventory['high'] & cost['medium'],␣
↪decision['decrease_a_little']),
    ctrl.Rule(demand['low'] & inventory['high'] & cost['high'],␣
↪decision['decrease_a_lot']),

    ctrl.Rule(demand['medium'] & inventory['none'] & cost['low'],␣
↪decision['increase_a_little']),
    ctrl.Rule(demand['medium'] & inventory['none'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['medium'] & inventory['none'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['medium'] & inventory['low'] & cost['low'],␣
↪decision['increase_a_little']),
    ctrl.Rule(demand['medium'] & inventory['low'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['medium'] & inventory['low'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['medium'] & inventory['medium'] & cost['low'],␣
↪decision['maintain']),
    ctrl.Rule(demand['medium'] & inventory['medium'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['medium'] & inventory['medium'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['medium'] & inventory['high'] & cost['low'],␣
↪decision['decrease_a_little']),
    ctrl.Rule(demand['medium'] & inventory['high'] & cost['medium'],␣
↪decision['decrease_a_little']),
    ctrl.Rule(demand['medium'] & inventory['high'] & cost['high'],␣
↪decision['decrease_a_little']),

    ctrl.Rule(demand['high'] & inventory['none'] & cost['low'],␣
↪decision['increase_a_lot']),
    ctrl.Rule(demand['high'] & inventory['none'] & cost['medium'],␣
↪decision['increase_a_little']),
    ctrl.Rule(demand['high'] & inventory['none'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['high'] & inventory['low'] & cost['low'],␣
↪decision['increase_a_lot']),
    ctrl.Rule(demand['high'] & inventory['low'] & cost['medium'],␣
↪decision['increase_a_little']),
    ctrl.Rule(demand['high'] & inventory['low'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['high'] & inventory['medium'] & cost['low'],␣
↪decision['increase_a_little']),
    ctrl.Rule(demand['high'] & inventory['medium'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['high'] & inventory['medium'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['high'] & inventory['high'] & cost['low'],␣
↪decision['maintain']),
    ctrl.Rule(demand['high'] & inventory['high'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['high'] & inventory['high'] & cost['high'],␣
↪decision['maintain']),

    ctrl.Rule(demand['very_high'] & inventory['none'] & cost['low'],␣
↪decision['increase_a_lot']),
    ctrl.Rule(demand['very_high'] & inventory['none'] & cost['medium'],␣
↪decision['increase_a_little']),
    ctrl.Rule(demand['very_high'] & inventory['none'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['very_high'] & inventory['low'] & cost['low'],␣
↪decision['increase_a_lot']),
    ctrl.Rule(demand['very_high'] & inventory['low'] & cost['medium'],␣
↪decision['increase_a_little']),
    ctrl.Rule(demand['very_high'] & inventory['low'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['very_high'] & inventory['medium'] & cost['low'],␣
↪decision['increase_a_little']),
    ctrl.Rule(demand['very_high'] & inventory['medium'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['very_high'] & inventory['medium'] & cost['high'],␣
↪decision['maintain']),
    ctrl.Rule(demand['very_high'] & inventory['high'] & cost['low'],␣
↪decision['maintain']),
    ctrl.Rule(demand['very_high'] & inventory['high'] & cost['medium'],␣
↪decision['maintain']),
    ctrl.Rule(demand['very_high'] & inventory['high'] & cost['high'],␣
↪decision['maintain']),
]
```

As part of the advisory system, 60 fuzzy rules were designed to cover all possible combinations of the levels of the three input variables: market demand (5 levels), inventory level (4 levels), and production cost (3 levels).

The rules were developed based on intuitive business relationships, such as:

- high demand and low inventory → increase production,
- low demand and high inventory → decrease production,
- high production cost → more conservative decisions.

The project uses defuzzification by the centroid method, which is the default strategy in scikit-fuzzy. This method calculates the final recommendation as a weighted average of the centroids of the activated output membership functions. As a result, the system can return intermediate values (e.g., 6.86), allowing for more flexible and realistic decision recommendations.

1.0.2 Comparison of Defuzzification Methods in the Fuzzy System

To compare the impact of different defuzzification methods, an analysis was conducted for a specific set of inputs: demand = 80, inventory = 20, cost = 60. This set represents a scenario with high demand, low inventory, and medium production cost — a realistic situation where increasing production may be justified, but the scale of the decision depends on the interpretation of the rules.

Defuzzification is the final stage of inference in a fuzzy system, which involves converting the fuzzy output into a specific numerical value. The analysis compared five commonly used methods:

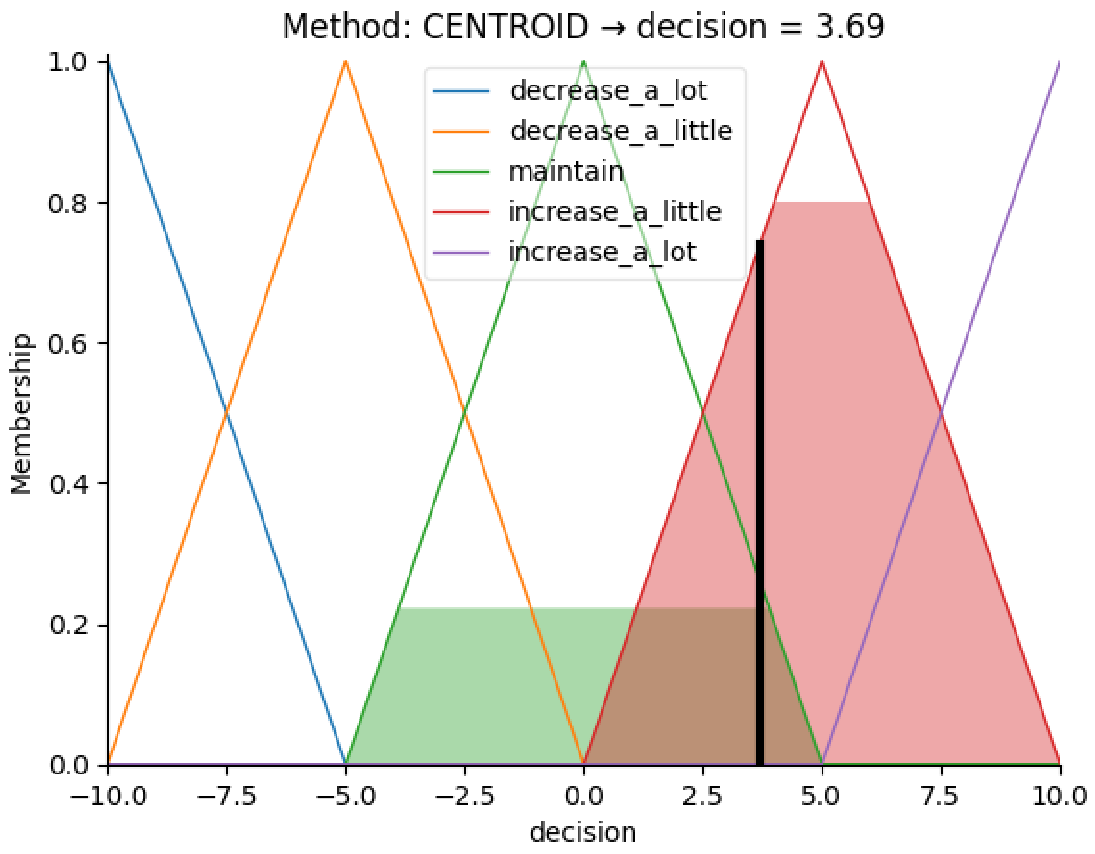
- Centroid (center of gravity): calculates the weighted average of all activated membership functions. This is the most precise and widely used method in practice.

- Bisector (middle of area): finds the point that divides the total output area into two equal parts.
- MOM (mean of maximum): returns the average of all values for which the membership degree reaches its maximum.
- SOM (smallest of maximum): selects the smallest value among those with the maximum membership degree.
- LOM (largest of maximum): selects the largest value among those with the maximum membership degree.

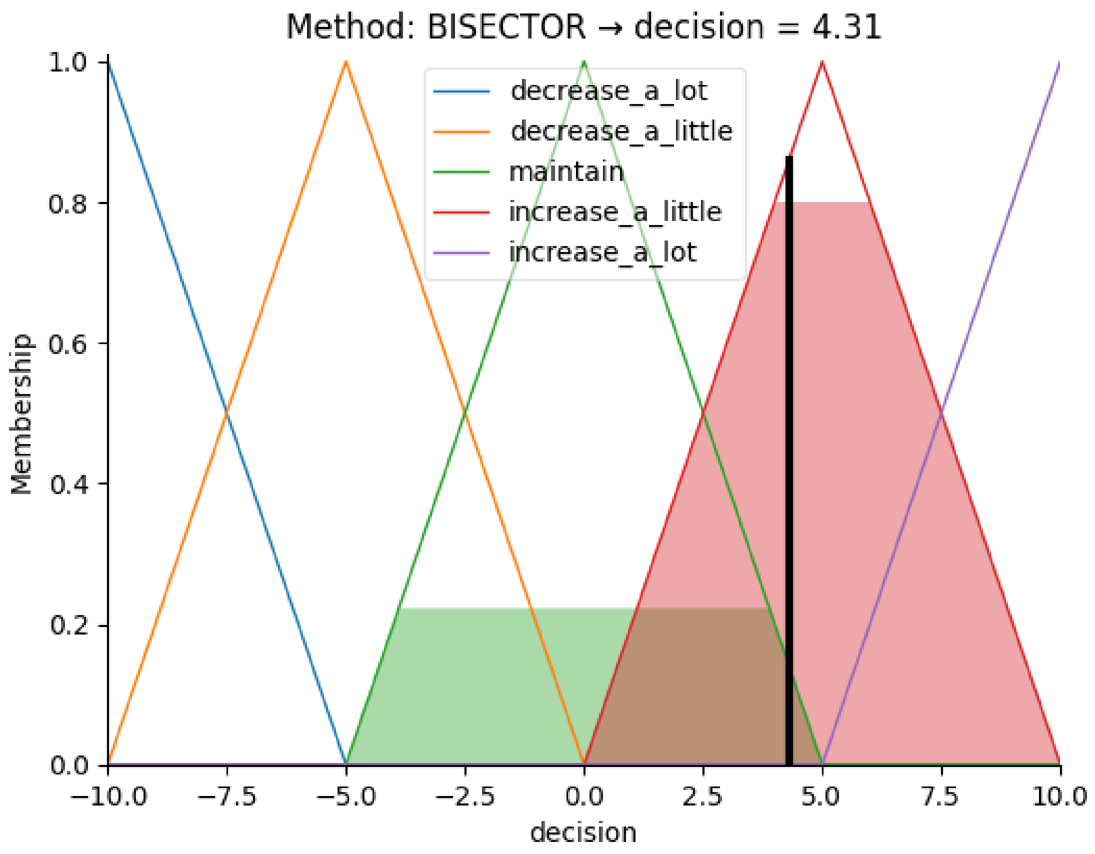
Each method may lead to a slightly different result.

```
[6]: methods = ['centroid', 'bisector', 'mom', 'som', 'lom']

     inputs = {
         'demand': 80,
         'inventory': 20,
         'cost': 60
     }

     for method in methods:
         print(f"\nDefuzzification method: {method.upper()}")

         decision.defuzzify_method = method

         production_system = ctrl.ControlSystem(rules)
         sim = ctrl.ControlSystemSimulation(production_system)

         for variable, value in inputs.items():
             sim.input[variable] = value

         sim.compute()

         print(f"Recommended decision: {sim.output['decision']:.2f}")

         decision.view(sim=sim)

         plt.title(f"Method: {method.upper()} → decision = {sim.output['decision']:.
       ↪2f}")
         plt.show()
```

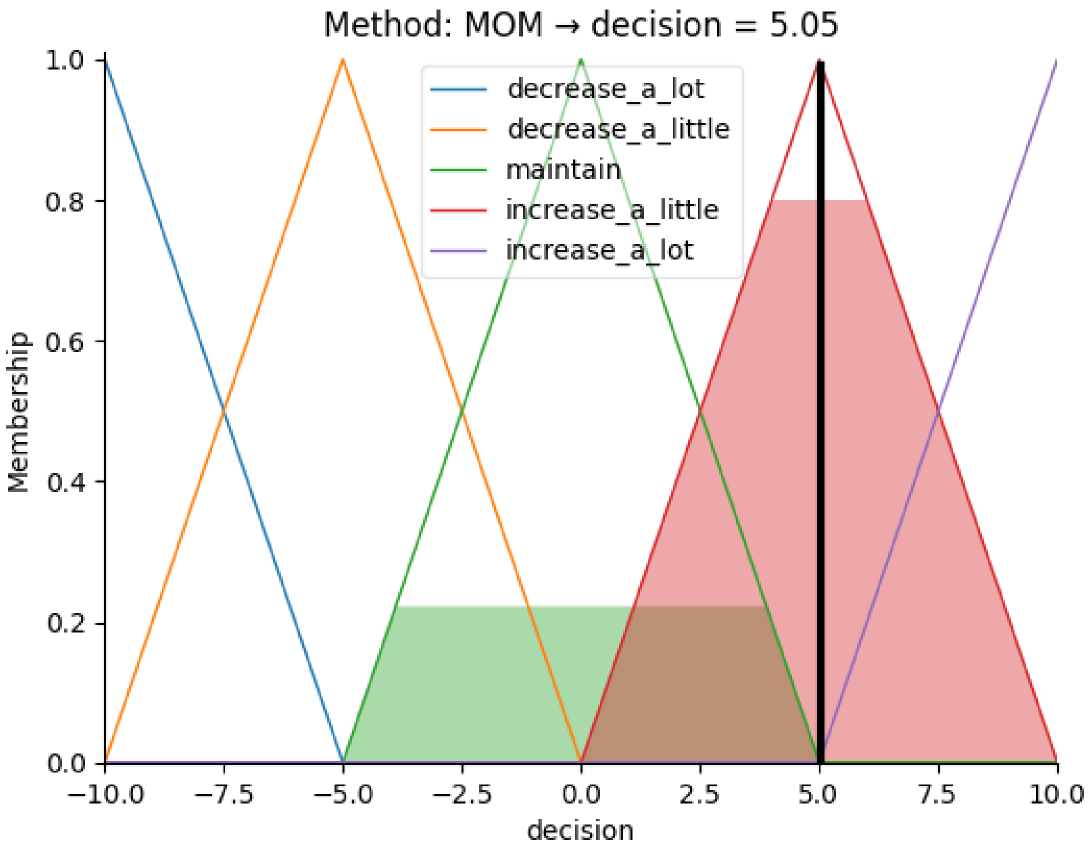
```
Defuzzification method: CENTROID
Recommended decision: 3.69
```



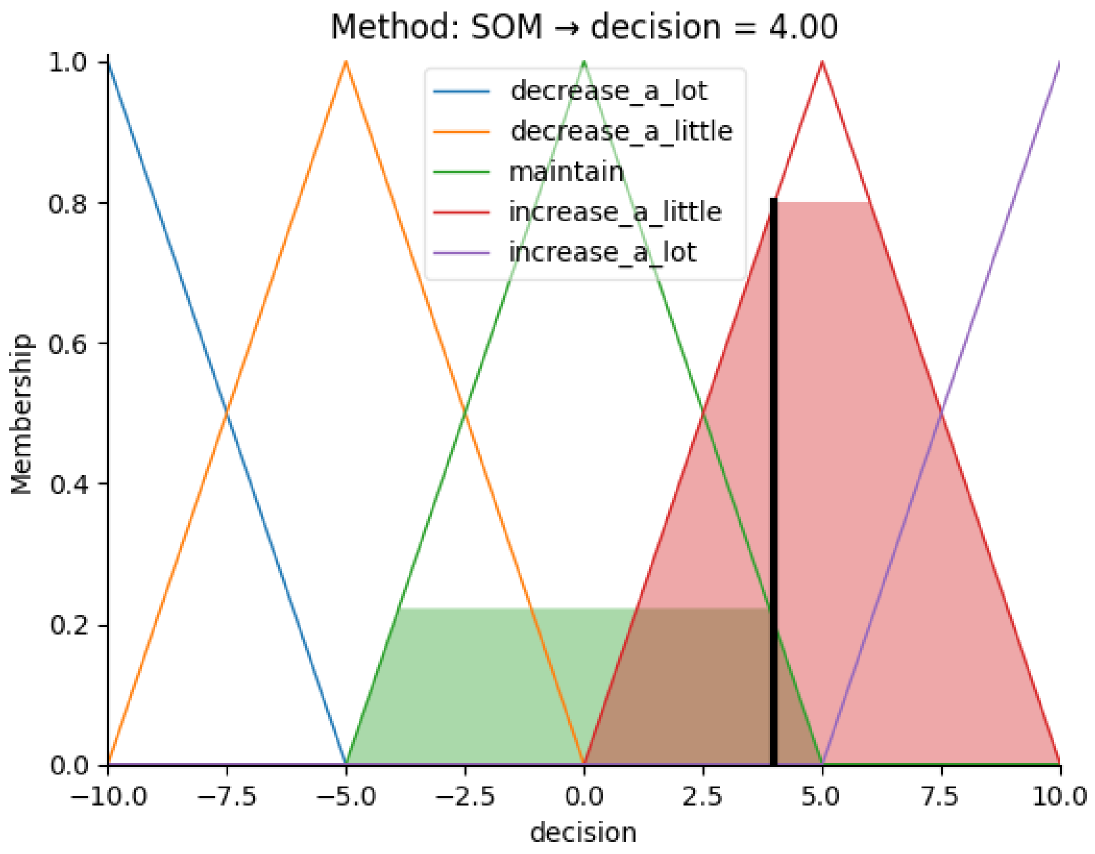
```
Defuzzification method: BISECTOR
Recommended decision: 4.31
```



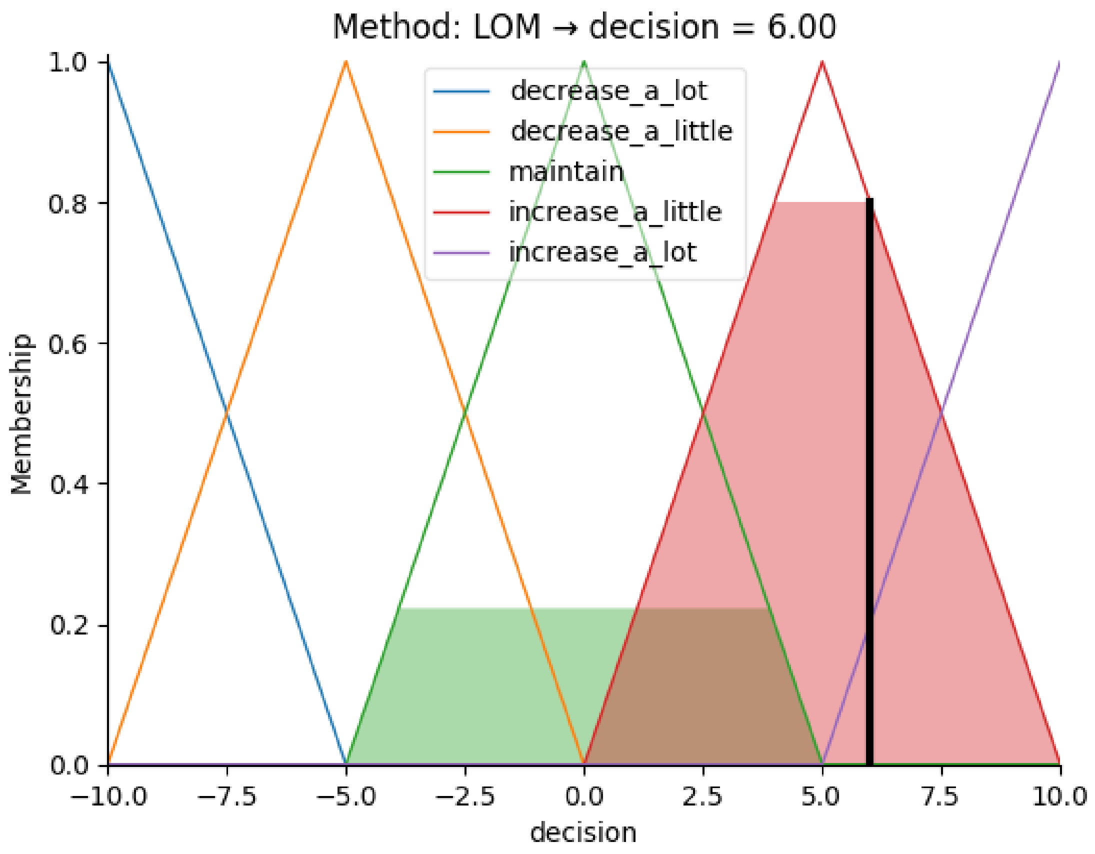
```
Defuzzification method: MOM
Recommended decision: 5.05
```



```
Defuzzification method: SOM
Recommended decision: 4.00
```



```
Defuzzification method: LOM
Recommended decision: 6.00
```



1.0.3 Conclusions:

- **Centroid method (center of gravity)** returned a value between the "maintain" and "increase a little" functions. This means the system recommends a moderate increase in production. It is a compromise decision — taking into account both high demand and low inventory levels, as well as rising production costs.
  The centroid method considers all activated rules proportionally, making it one of the most stable and balanced defuzzification methods.
- **Bisector method** divides the area activated by the membership functions into two equal parts by surface area. In this case, the result shows a stronger tendency toward increasing production compared to the centroid method, as the center of mass is shifted toward rules promoting growth.
- **MOM (Mean of Maximum)** selects the arithmetic mean of all x-values for which the membership functions reach their maximum activation. In this case, the "increase a little" function dominates, so the final decision clearly indicates an increase in production.
- **SOM (Smallest of Maximum)** chooses the smallest x-value for which the membership function reaches its maximum activation level. Here, the "increase a little" function was maximally activated, so the system selected the lowest possible value in that range.

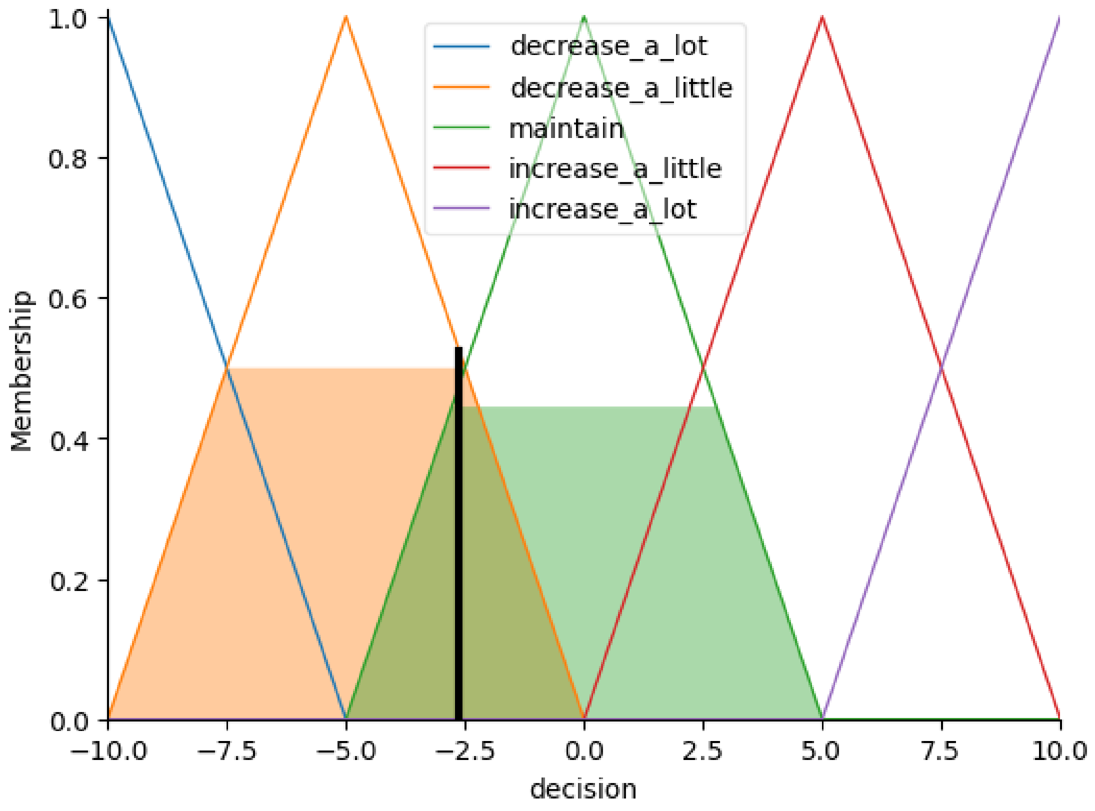
This method is characterized by caution — unlike MOM, it does not pick the average value but the lower boundary.

- **LOM (Largest of Maximum)** selects the largest x-value for which the membership function reaches its maximum activation level. In this situation, the decision was set at the upper boundary of the "increase a little" function.

1.0.4 Examples

[7]:
```python
decision.defuzzify_method = 'centroid'

production_system = ctrl.ControlSystem(rules)
sim = ctrl.ControlSystemSimulation(production_system)

sim.input['demand'] = 10
sim.input['inventory'] = 80
sim.input['cost'] = 40

sim.compute()

print(f"Recommended production decision: {sim.output['decision']:.2f}")
decision.view(sim=sim)
```

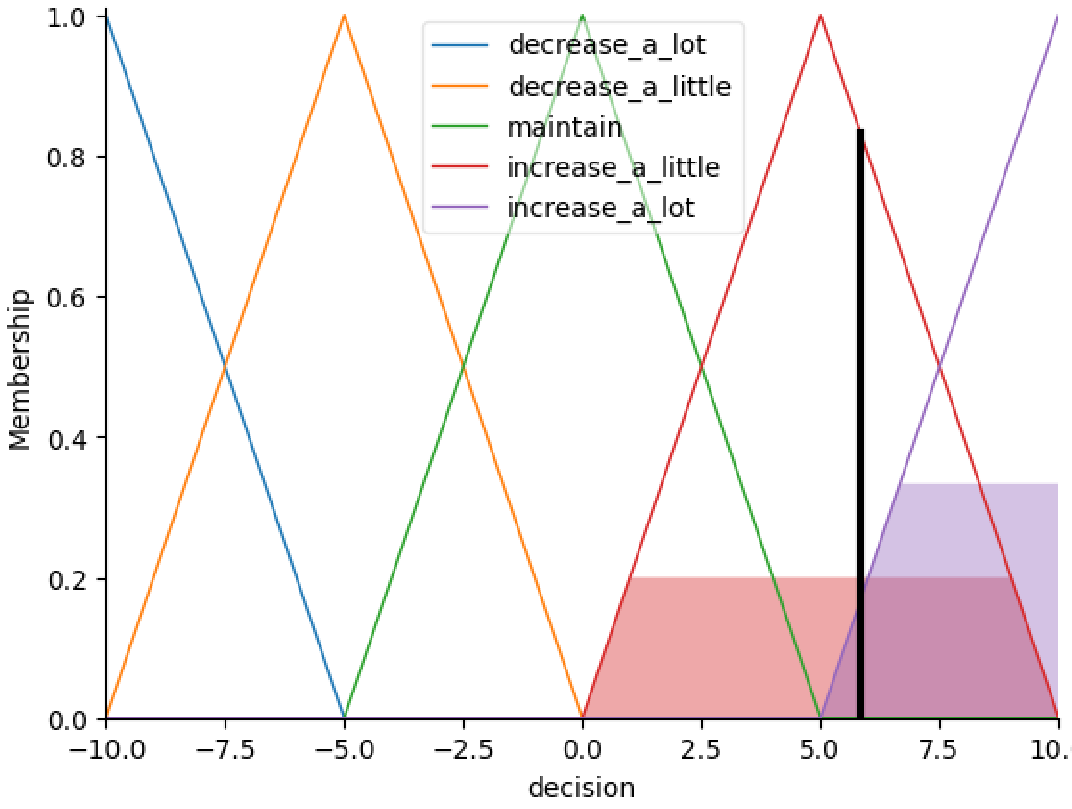
```
Recommended production decision: -2.62
```



For the case where the demand is very low (10%), the inventory level is very high (80%), and the production cost is moderate (40%), the fuzzy system using the centroid method returned a decision of −2.62, which means that production should be reduced by approximately 20–25%. This is a reasonable and proportionate response to market conditions, helping to limit overproduction while maintaining cost control.

[8]:
```python
sim.input['demand'] = 90
sim.input['inventory'] = 10
sim.input['cost'] = 10
sim.compute()

print(f"Recommended production decision: {sim.output['decision']:.2f}")
decision.view(sim=sim)
```

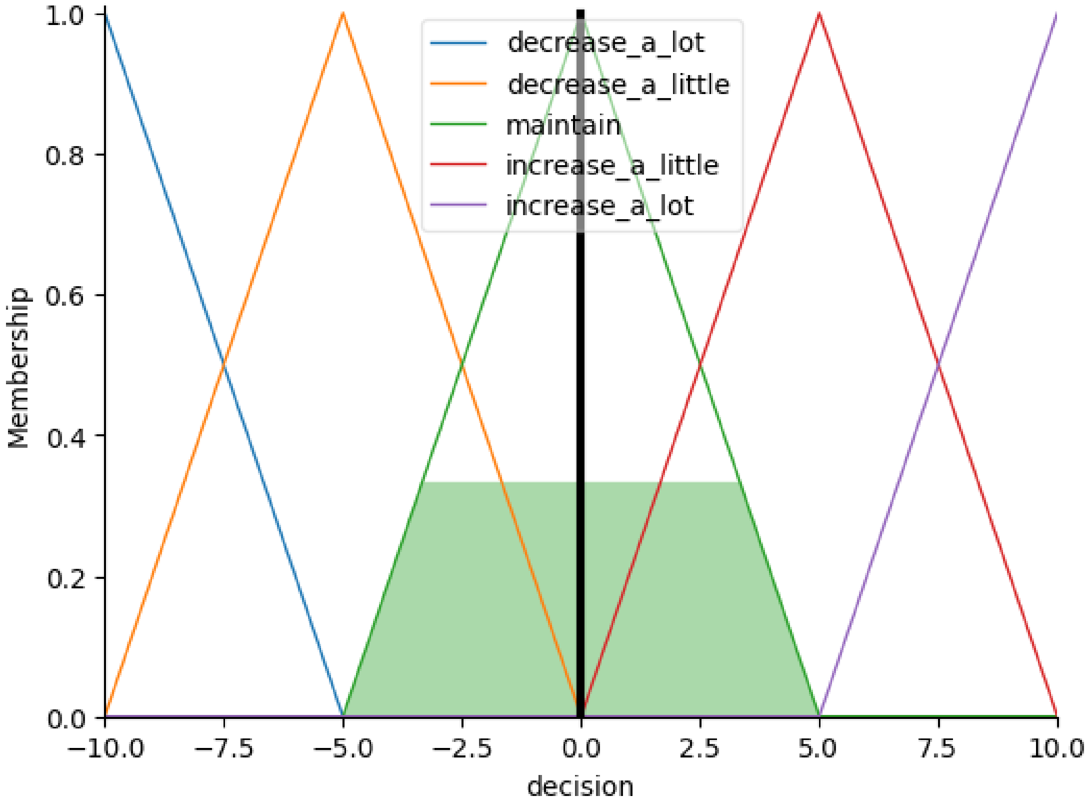
```
Recommended production decision: 5.84
```



For the case where the demand is very high (90%), the inventory level is very low (10%), and the production cost is low (10%), the fuzzy system using the centroid method returned a decision of 5.84, which means that production should be increased by approximately 60%. This is a response consistent with expectations, reflecting high market demand and low cost risk, enabling a rapid reaction to market needs.

[9]:
```
sim.input['demand'] = 30
sim.input['inventory'] = 10
sim.input['cost'] = 90

sim.compute()

print(f"Recommended production decision: {sim.output['decision']:.2f}")
decision.view(sim=sim)
```

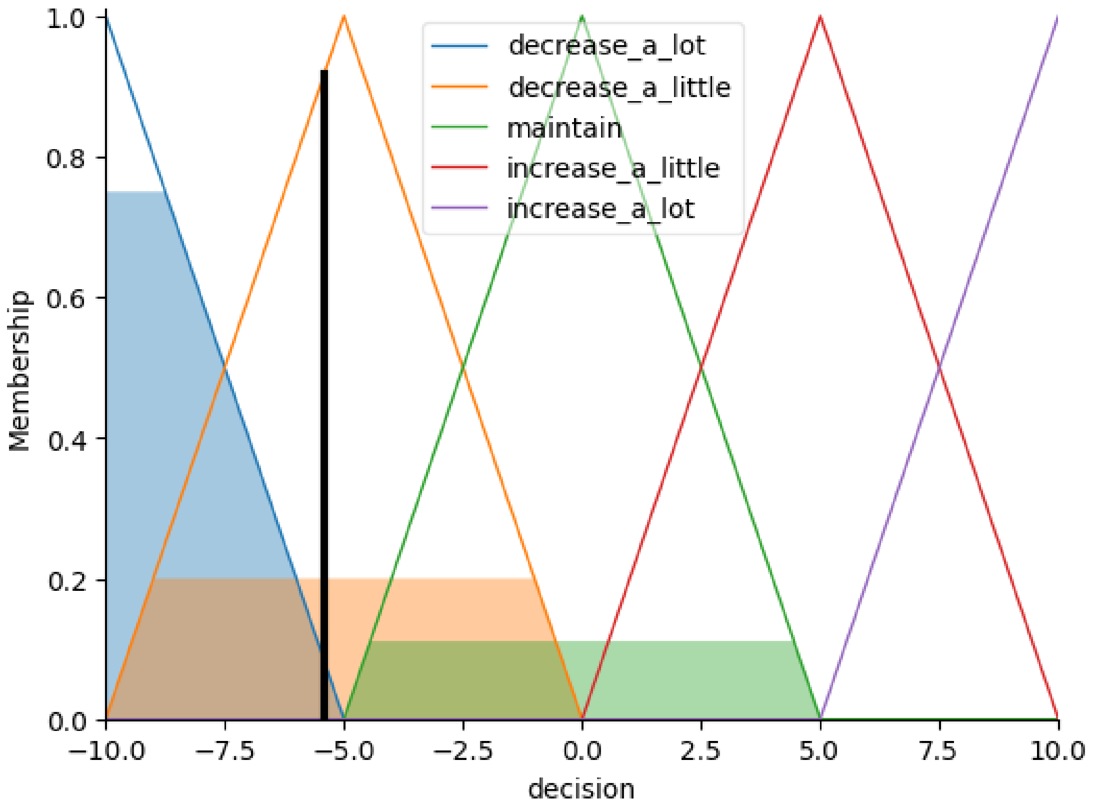
```
Recommended production decision: 0.00
```



For the case where the demand is moderately low (30%), the inventory level is very low (10%), and the production cost is very high (90%), the fuzzy system using the centroid method returned a decision of 0.00, indicating that the current level of production should be maintained. This result reflects a compromise between the pressure to increase production due to low inventory and the high costs that limit expansion. The system responded neutrally, avoiding the risk of both overproduction and shortage.

[10]:
```python
sim.input['demand'] =5
sim.input['inventory'] = 95
sim.input['cost'] = 90
sim.compute()
print(f"Recommended production decision: {sim.output['decision']:.2f}")
decision.view(sim=sim)
```

```
Recommended production decision: -5.41
```



For the case where the demand is very low (5%), the inventory level is very high (95%), and the production cost is very high (90%), the fuzzy system using the centroid method returned a decision of −5.41, indicating that production should be reduced by approximately 50%. Such a strong recommendation to cut production is fully justified — low demand generates no need for additional output, inventory is already accumulated, and the cost of further production is not economically viable.

[ ]: