ANKARA UNIVERSITY

FACULTY OF ENGINEERING

COM2536 - FUZZY LOGIC

Final Project

25.05.2025

Efe ATEŞ

22290588

PURPOSE

Aim of this project is to learn and implement fuzzy logic methods: fuzzification, Rule Evaluation, Aggregation, and defuzzification. A fuzzy logic system developed for bank officers that determines the amount of housing credit for a person who is applicant. Mamdani inference method will be used to evaluate housing credit.

THEORY

Introduction

The Mamdani fuzzy inference system was proposed as the first attempt to control a steam engine and boiler combination by a set of linguistic control rules obtained from experienced human operators. Figure 1 is an illustration of how a two-rule Mamdani fuzzy inference system derives the overall output z when subjected to two crisp inputs x and y.

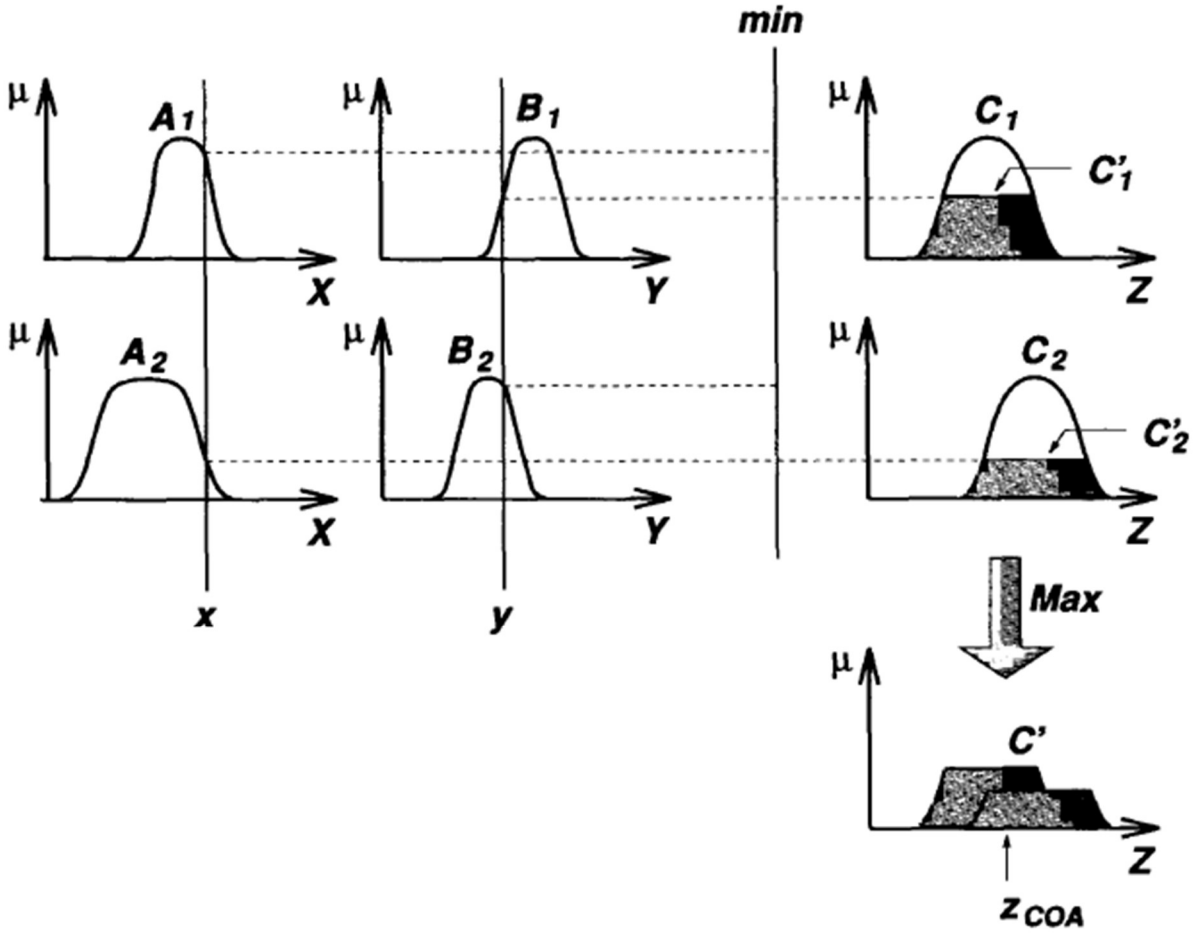


Figure 1.1: The Mamdani fuzzy inference system using min and max for T-norm and T-conorm operators, respectively

If we adopt max and algebraic product as our choice for the T-norm and T-conorm operators, respectively, and use max-product composition instead of the original max-min composition, then the resulting fuzzy reasoning is shown in Figure2, where the inferred output of each rule is a fuzzy set scaled down by its firing strength via algebraic product. Although this type of fuzzy reasoning was not employed in Mamdani's original paper, it has often been used in the literature. Other variations are possible if we use different T-norm and T-conorm operators.

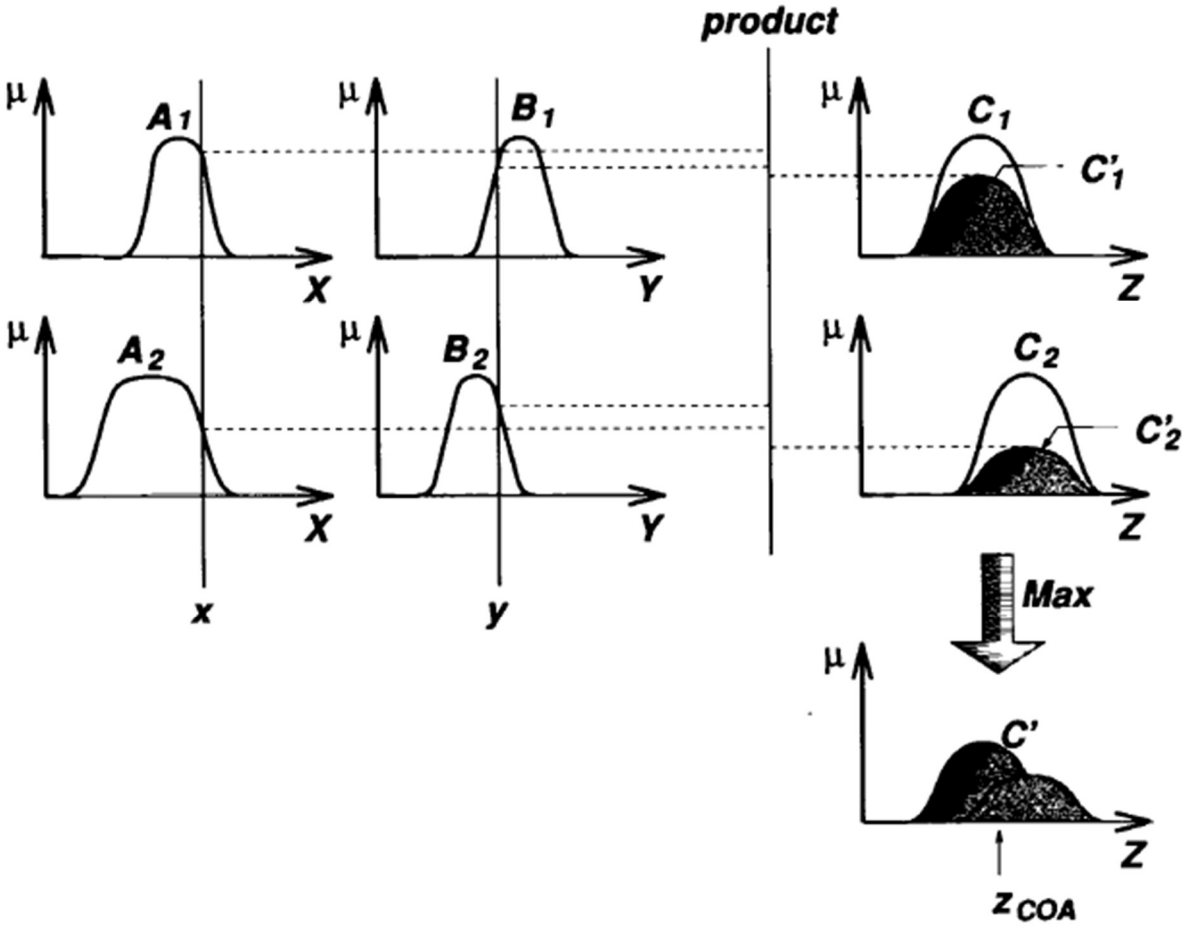


Figure 1.2: The Mamdani fuzzy inference system using product and max for T-norm and T-conorm operators, respectively

In Mamdani's application [1], two fuzzy inference systems were used as two controllers to generate the heat input to the boiler and throttle opening of the engine cylinder, respectively, to regulate the steam pressure in the boiler and the speed of the engine. Since the plant takes only crisp values as inputs, we have to use a defuzzifier to convert a fuzzy set to a crisp value.

Defuzzification

Defuzzification refers to the way a crisp value is extracted from a fuzzy set as a representative value. In general, there are five methods for defuzzifier a fuzzy set A of a universe of discourse Z, as shown in Figure 3. (Here the fuzzy set A is usually represented by an aggregated output MF, such as C' in Figures 4.2 and 4.3).

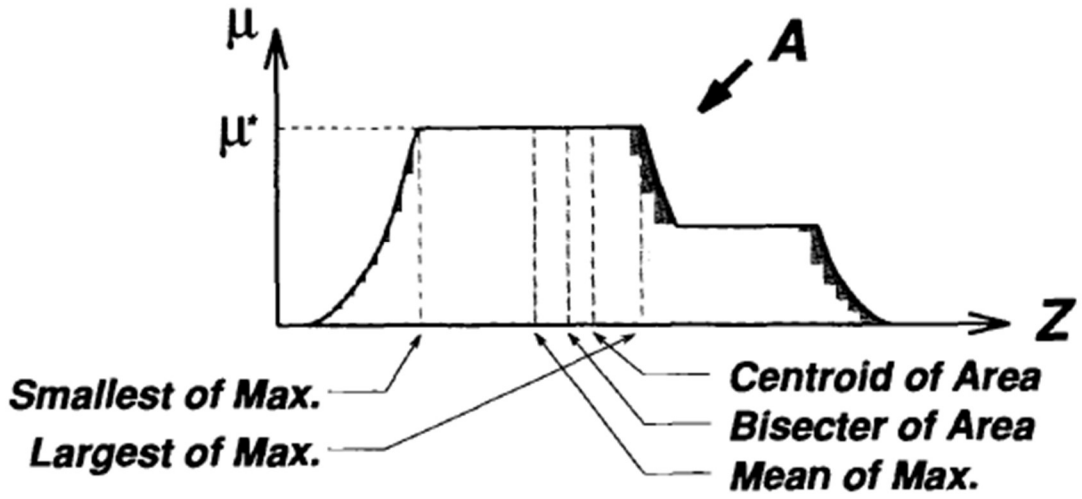


Figure 1.3: Various defuzzification schemes for obtaining a crisp output

A brief explanation of each defuzzification strategy are shown as follows:

- **Centroid of area** ZCOA:

$$z_{\text{COA}} = \frac{\int_Z \mu_A(z) z \, dz}{\int_Z \mu_A(z) \, dz},$$

  where $\mu_A(z)$ is the aggregation output MF. This is the most widely adopted defuzzification strategy, which is reminiscent of the calculation of expected values of probability distributions.

- **Bisector of area** ZBOA: ZBOA satisfies

$$\int_{\alpha}^{z_{\text{BOA}}} \mu_A(z) \, dz = \int_{z_{\text{BOA}}}^{\beta} \mu_A(z) \, dz,$$

where $\alpha=\min\{z|z \in Z\}$ and $\beta=\max\{z|z \in Z\}$. That is, the vertical line $z=z_{BOA}$ partitions the region between $z=\alpha$, $z=\beta$, $y=0$ and $y=\mu_A(z)$ into two regions with the same area.

- **Mean of maximum** z_{MOM}: z_{MOM} is the average of the maximizing z at which the MF reach a maximum μ^*. In symbols

$$z_{MOM} = \frac{\int_{Z'} z\, dz}{\int_{Z'} dz},$$

where $Z' = \{z \mid \mu_A(z)=\mu^*\}$. In particular, if $\mu_A(z)$ has a single maximum at $z=z^*$, then $z_{MOM}=z^*$. Moreover, if $\mu_A(z)$ reaches its maximum whenever $z \in [z_{left}, z_{right}]$ (This is the case in Figure 3), then $z_{MOM} = (z_{left} + z_{right})/2$. The mean of maximum is the defuzzification strategy employed in Mamdani's fuzzy logic controllers.

- **Smallest of maximum** z_{SOM}: z_{SOM} is the minimum (in terms of magnitude) of the maximizing z.
- **Largest of maximum** z_{LOM}: z_{LOM} is the maximum (in terms of magnitude) of the maximizing z. Because of their obvious bias, z_{SOM} and z_{LOM} are not used as often as the other three defuzzification methods.

The calculation needed to carry out any of these five defuzzification operations is time-consuming unless special hardware support is available. Furthermore, these defuzzification operations are not easily subject to rigorous mathematical analysis, so most of the studies are based on experimental results.

RESULT

Firstly I defined functions and their corresponding resultants for example:

```
#Market
def market_value_low(price):
    if price <= 0:
        return 0
    elif 0 < price <= 50:
        return price / 50
    elif 50 < price < 100:
        return (100 - price) / 50
    else:
        return 0
```

Figure 2.1 that evaluations house evaluation membership degree using Mamdani's fuzzy inference method

This held the correct assignment of membership degree for house's price. I defined these methods for 5 functions which are Market Value, Location, Asset, Income, and Interest membership functions.

Then for the three remaining methods I defined a mamdani fuzzy inference system which is provided on rules in the manual. We are doing taking minimum and maximum approach here:

```
def houseEval(price,loc):

    very_low_activation = 0
    low_activation = 0
    medium_activation = 0
    high_activation = 0
    very_high_activation = 0

    rule1 = market_value_low(price)
    low_activation = max(low_activation, rule1)

    rule2 = location_bad(loc)
    low_activation = max(low_activation, rule2)

    rule3 = min(location_bad(loc), market_value_low(price))
    very_low_activation = max(very_low_activation, rule3)

    rule4 = min(location_bad(loc), market_value_medium(price))
    low_activation = max(low_activation, rule4)

    rule5 = min(location_bad(loc), market_value_high(price))
    medium_activation = max(medium_activation, rule5)

    rule6 = min(location_bad(loc), market_value_very_high(price))
    high_activation = max(high_activation, rule6)

    rule7 = min(location_fair(loc), market_value_low(price))
    low_activation = max(low_activation, rule7)

    rule8 = min(location_fair(loc), market_value_medium(price))
    medium_activation = max(medium_activation, rule8)

    rule9 = min(location_fair(loc), market_value_high(price))
    high_activation = max(high_activation, rule9)

    rule10 = min(location_fair(loc), market_value_very_high(price))
    very_high_activation = max(very_high_activation, rule10)

    rule11 = min(location_excellent(loc), market_value_low(price))
    medium_activation = max(medium_activation, rule11)

    rule12 = min(location_excellent(loc), market_value_medium(price))
    high_activation = max(high_activation, rule12)

    rule13 = min(location_excellent(loc), market_value_high(price))
    very_high_activation = max(very_high_activation, rule13)

    rule14 = min(location_excellent(loc), market_value_very_high(price))
    very_high_activation = max(very_high_activation, rule14)

    house_evaluations = {
        "Very_low": very_low_activation,
        "Low": low_activation,
        "Medium": medium_activation,
        "High": high_activation,
        "Very_high": very_high_activation
    }

    return house_evaluations
```

Figure 2.2 that evaluations house evaluation membership degree using Mamdani's fuzzy inference method

The hierarchical structure of the program we will be consider membership degree from 6 fuzzy functions, that is assigned to us in Project Manual:

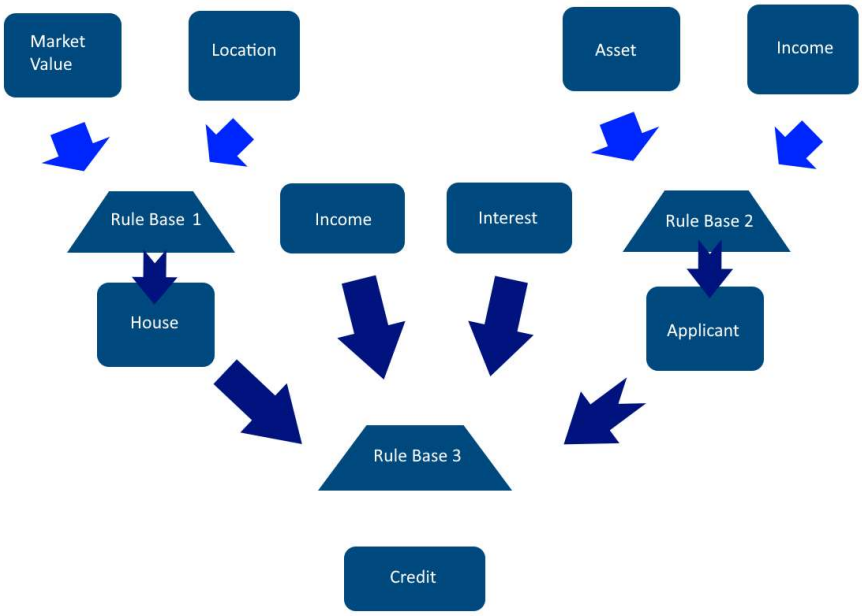
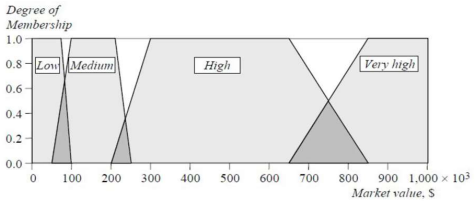
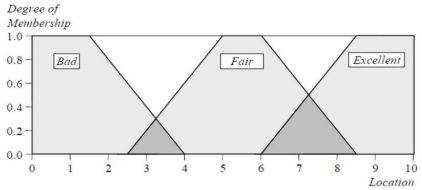
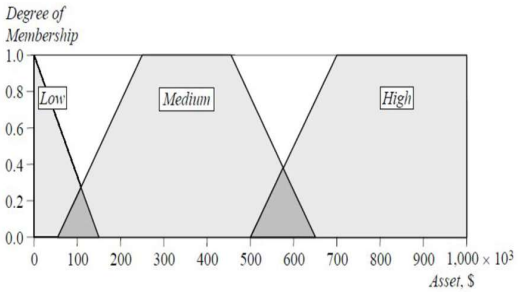


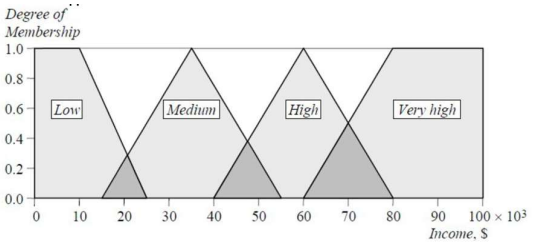
Figure 2.3 that shows programme hierarchy, that includes more than one usage of income variable

Membership functions of market value, location, asset, income, interest, credit, house, and applicant in order:









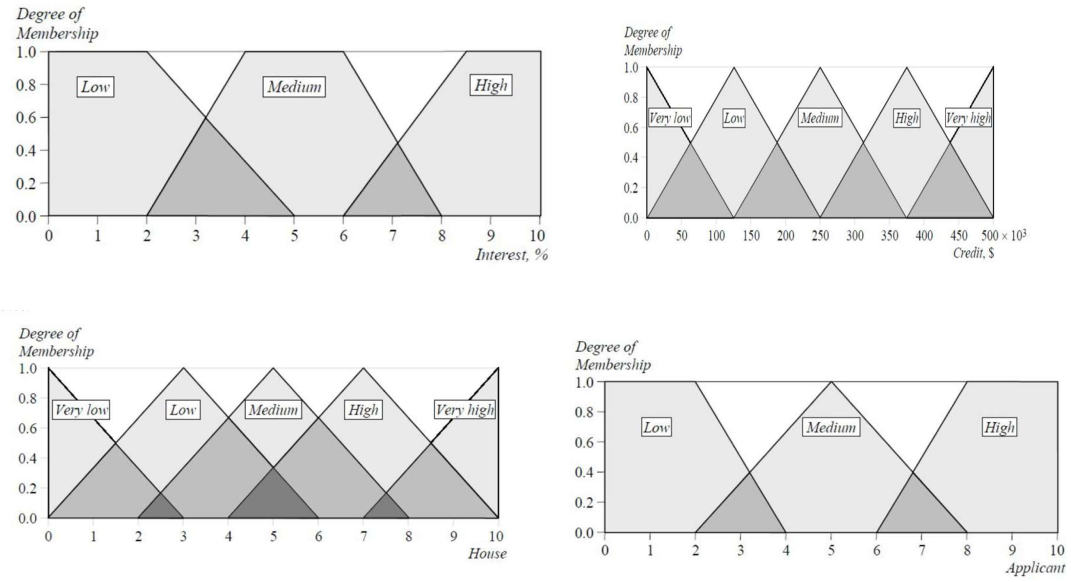


Figure 2.4: Degree of Membership for shown function in the hierarchical structure of the program

```
if __name__ == "__main__":
    person1 = evaluateThePerson(market_value=600, location=4, asset=1000, income=890, interest=3)
    person2 = evaluateThePerson(market_value=900, location=8, asset=10000, income=8900, interest=1)
    person3 = evaluateThePerson(market_value=200, location=1, asset=10000, income=900, interest=10)


    print("------------------------------\nPERSON 1",end="\n------------------------------\n")
    print("House Evaluation |*|  Applicant Score |*| Credit Score |*| Crisp Credit Score")
    print(str(person1['House Evaluation']).strip('{}'),str(person1['Applicant Evaluation']).strip('{}'),str(person1['Credit Evaluation']).strip('{}'),(person1['Crisp Credit Score']),sep=" |*| ")

    print("------------------------------\nPERSON 2",end="\n------------------------------\n")
    print("House Evaluation |*| Applicant Score |*| Credit Score |*| Crisp Credit Score")
    print(str(person2['House Evaluation']).strip('{}'),str(person2['Applicant Evaluation']).strip('{}'),str(person2['Credit Evaluation']).strip('{}'),(person2['Crisp Credit Score']),sep=" |*| ")

    print("------------------------------\nPERSON 3",end="\n------------------------------\n")
    print("House Evaluation |*| Applicant Score |*|  Credit Score |*| Crisp Credit Score")
    print(str(person3['House Evaluation']).strip('{}'),str(person3['Applicant Evaluation']).strip('{}'),str(person3['Credit Evaluation']).strip('{}'),(person3['Crisp Credit Score']),sep=" |*| ")
```

Figure 2.5: Assigning applicants' data

In the main function part it evaluates 3 different applicants' Crisp credit score.

First person has 600k $ of House market value, location is defined as 4 and his asset is 1000 k $ with income of 890 k $ annually. Interest rate is 3%.

Second person has 900k $ of House market value, location is defined as 8 and his asset is 10000 k $ with income of 8.90 M $ annual. Interest rate is 1%.

Third person has 200k $ of House market value, location is defined as 1 and his asset is 10000 k $ with income of 900 k $ annually. Interest rate is 10%.

Thus that gave the corresponding answer from terminal output which is defined below.

```
PS C:\Users\lenovo\Desktop\New folder> C:/Users/lenovo/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/lenovo/Desktop/New folder/22290588.py"
------------------------------
PERSON 1
------------------------------
House Evaluation |*|  Applicant Score |*| Credit Score |*| Crisp Credit Score
'Very_low': 0, 'Low': 0, 'Medium': 0, 'High': 0.6666666666666666, 'Very_high': 0 |*| 'Low': 0, 'Medium': 0, 'High': 1 |*| 'Very_low': 0, 'Low': 0, 'Medium': 0, 'High': 0.6666666666666666, 'Very_high': 0 |*| 75.0
------------------------------
PERSON 2
------------------------------
House Evaluation |*| Applicant Score |*| Credit Score |*| Crisp Credit Score
'Very_low': 0, 'Low': 0, 'Medium': 0, 'High': 0, 'Very_high': 1 |*| 'Low': 0, 'Medium': 0, 'High': 1 |*| 'Very_low': 0, 'Low': 0, 'Medium': 0, 'High': 0, 'Very_high': 1 |*| 100.0
------------------------------
PERSON 3
------------------------------
House Evaluation |*| Applicant Score |*|  Credit Score |*| Crisp Credit Score
'Very_low': 0, 'Low': 1, 'Medium': 0, 'High': 0, 'Very_high': 0 |*| 'Low': 0, 'Medium': 0, 'High': 1 |*| 'Very_low': 0, 'Low': 0, 'Medium': 1, 'High': 0, 'Very_high': 0 |*| 50.0

Press any key to continue . . .
PS C:\Users\lenovo\Desktop\New folder>
```

Figure 2.6: Output from inputs

It gave the expected result based on theory.

CONCLUSION and COMMENT

The code worked as expected and gave results in float numbers between 0 and 1. Everything was according to theory. Only the OS library is used as an import because when the code runs on the terminal it will close immediately. It will show output but since the computer calculates it fast it will terminate the terminal immediately. To stop this, I used System(“pause”) to hold output until the user presses a key.

This project successfully implemented a fuzzy logic system for housing credit evaluation. The system effectively mimics human decision-making processes by using linguistic variables and fuzzy inference to assess credit applications. Through the Mamdani inference method, the program evaluates multiple factors including property characteristics (market value, location), applicant qualifications (assets, income), and loan parameters (interest rates).

This fuzzy logic approach proved particularly suitable for this application domain as it addresses the inherent ambiguity in credit evaluation criteria. Unlike traditional binary logic systems, this implementation can handle partial truths and gradual transitions between categories (e.g., "low" to "medium" income). This implementation demonstrates how fuzzy logic can bridge the gap between rigid computational methods and the nuanced judgments typically requiring human expertise, making it a valuable tool for financial decision support systems.

APPENDIX

Main.py:

```
"""
Name: Efe ATEŞ
Student No: 22290588

"""

import os


#Market
def market_value_low(price):
    if price <= 0:
        return 0
    elif 0 < price <= 50:
        return price / 50
    elif 50 < price < 100:
        return (100 - price) / 50
    else:
        return 0

def market_value_medium(price):
    if price <= 50 or price >= 250:
        return 0
    elif 50 < price < 100:
        return (price - 50) / 50
    elif 100 <= price <= 200:
        return 1
    else:
        return (250 - price)

def market_value_high(price):
    if price <= 200 or price >= 700:
        return 0
    elif 200 < price < 300:
        return (price - 200) / 100
    elif 300 <= price <= 600:
        return 1
    else:  # 600 < price < 700
        return (700 - price) / 100

def market_value_very_high(price):
    if price <= 600:
        return 0
    elif 600 < price < 700:
        return (price - 600) / 100
    else:
        return 1

#Location
def location_bad(loc):
    if loc <= 0:
        return 1
    elif 0 < loc <= 2:
        return 1
    elif 2 < loc < 3:
        return (3 - loc)
    else:
        return 0

def location_fair(loc):
    if loc <= 2 or loc >= 8:
        return 0
    elif 2 < loc < 5:
        return (loc - 2) / 3
```

```
    else:  # 5 < loc < 8
        return (8 - loc) / 3
def location_excellent(loc):
    if loc <= 7:
        return 0
    elif 7 < loc < 8:
        return (loc - 7)
    else:  # loc >= 8
        return 1


#Person's Asset

def asset_low(asset):
    if asset <= 0:
        return 1
    elif 0 < asset < 200:
        return (200 - asset) / 200
    else:
        return 0

def asset_medium(asset):
    if asset <= 100 or asset >= 600:
        return 0
    elif 100 < asset < 200:
        return (asset - 100) / 100
    elif 200 <= asset <= 500:
        return 1
    else:
        return (600 - asset) / 100

def asset_high(asset):
    if asset <= 500:
        return 0
    elif 500 < asset < 600:
        return (asset - 500) / 100
    else:
        return 1
#Person's Income
def income_low(income):
    if income <= 10:
        return 1
    elif 10 < income < 20:
        return (20 - income) / 10
    else:
        return 0

def income_medium(income):
    if income <= 20 or income >= 50:
        return 0
    elif 20 < income < 30:
        return (income - 20) / 10
    elif income == 30:
        return 1
    else:
        return (50 - income) / 20

def income_high(income):
    if income <= 40 or income >= 80:
        return 0
    elif 40 < income < 60:
        return (income - 40) / 20
    elif income == 60:
        return 1
    else:
        return (80 - income) / 20

def income_very_high(income):
    if income <= 70:
        return 0
```

```
    elif 70 < income < 80:
        return (income - 70) / 10
    else:
        return 1
#Interest
def interest_low(interest):
    if interest <= 2:
        return 1
    elif 2 < interest < 4:
        return (4 - interest) / 2
    else:
        return 0

def interest_medium(interest):
    if interest <= 2 or interest >= 8:
        return 0
    elif 2 < interest < 4:
        return (interest - 2) / 2
    elif 4 <= interest <= 6:
        return 1
    else:
        return (8 - interest) / 2

def interest_high(interest):
    if interest <= 6:
        return 0
    elif 6 < interest < 8:
        return (interest - 6) / 2
    else:
        return 1


def houseEval(price,loc):

    very_low_activation = 0
    low_activation = 0
    medium_activation = 0
    high_activation = 0
    very_high_activation = 0

    rule1 = market_value_low(price)
    low_activation = max(low_activation, rule1)

    rule2 = location_bad(loc)
    low_activation = max(low_activation, rule2)

    rule3 = min(location_bad(loc), market_value_low(price))
    very_low_activation = max(very_low_activation, rule3)

    rule4 = min(location_bad(loc), market_value_medium(price))
    low_activation = max(low_activation, rule4)

    rule5 = min(location_bad(loc), market_value_high(price))
    medium_activation = max(medium_activation, rule5)

    rule6 = min(location_bad(loc), market_value_very_high(price))
    high_activation = max(high_activation, rule6)

    rule7 = min(location_fair(loc), market_value_low(price))
    low_activation = max(low_activation, rule7)

    rule8 = min(location_fair(loc), market_value_medium(price))
    medium_activation = max(medium_activation, rule8)

    rule9 = min(location_fair(loc), market_value_high(price))
    high_activation = max(high_activation, rule9)

    rule10 = min(location_fair(loc), market_value_very_high(price))
    very_high_activation = max(very_high_activation, rule10)
```

```
    rule11 = min(location_excellent(loc), market_value_low(price))
    medium_activation = max(medium_activation, rule11)

    rule12 = min(location_excellent(loc), market_value_medium(price))
    high_activation = max(high_activation, rule12)

    rule13 = min(location_excellent(loc), market_value_high(price))
    very_high_activation = max(very_high_activation, rule13)

    rule14 = min(location_excellent(loc), market_value_very_high(price))
    very_high_activation = max(very_high_activation, rule14)

    house_evaluations = {
        "Very_low": very_low_activation,
        "Low": low_activation,
        "Medium": medium_activation,
        "High": high_activation,
        "Very_high": very_high_activation
    }

    return house_evaluations

def applicantEval(asset, income):
    very_low_activation = 0
    low_activation = 0
    medium_activation = 0
    high_activation = 0
    very_high_activation= 0

    #If (Asset is Low) and (Income is Low) then (Applicant is Low)
    rule1 = min(asset_low(asset), income_low(income))
    low_activation = max(low_activation, rule1)

    #If (Asset is Low) and (Income is Medium) then (Applicant is Low)
    rule2 = min(asset_low(asset), income_medium(income))
    low_activation = max(low_activation, rule2)

    #If (Asset is Low) and (Income is High) then (Applicant is Medium)
    rule3 = min(asset_low(asset), income_high(income))
    medium_activation = max(medium_activation, rule3)

    #If (Asset is Low) and (Income is Very_high) then (Applicant is High)
    rule4 = min(asset_low(asset), income_very_high(income))
    high_activation = max(high_activation, rule4)

    #If (Asset is Medium) and (Income is Low) then (Applicant is Low)
    rule5 = min(asset_medium(asset), income_low(income))
    low_activation = max(low_activation, rule5)

    #If (Asset is Medium) and (Income is Medium) then (Applicant is Medium)
    rule6 = min(asset_medium(asset), income_medium(income))
    medium_activation = max(medium_activation, rule6)

    #If (Asset is Medium) and (Income is High) then (Applicant is High)
    rule7 = min(asset_medium(asset), income_high(income))
    high_activation = max(high_activation, rule7)

    #If (Asset is Medium) and (Income is Very_high) then (Applicant is High)
    rule8 = min(asset_medium(asset), income_very_high(income))
    high_activation = max(high_activation, rule8)

    #If (Asset is High) and (Income is Low) then (Applicant is Medium)
    rule9 = min(asset_high(asset), income_low(income))
    medium_activation = max(medium_activation, rule9)

    #If (Asset is High) and (Income is Medium) then (Applicant is Medium)
    rule10 = min(asset_high(asset), income_medium(income))
    medium_activation = max(medium_activation, rule10)

    #If (Asset is High) and (Income is High) then (Applicant is High)
```

```
    rule11 = min(asset_high(asset), income_high(income))
    high_activation = max(high_activation, rule11)

    #If (Asset is High) and (Income is Very_high) then (Applicant is High)
    rule12 = min(asset_high(asset), income_very_high(income))
    high_activation = max(high_activation, rule12)

    applicant_evaluations = {
        "Low": low_activation,
        "Medium": medium_activation,
        "High": high_activation
    }

    return applicant_evaluations


def amountCredit(house_eval, applicant_eval, income, interest):
    very_low_activation= 0
    low_activation = 0
    medium_activation = 0
    high_activation = 0
    very_high_activation= 0

    rule1 = min(income_low(income),interest_medium(interest))
    very_low_activation = max(very_low_activation,rule1)

    rule2 = min(income_low(income),interest_high(interest))
    very_low_activation = max(very_low_activation,rule2)

    rule3 = min(income_medium(income), interest_high(interest))
    low_activation = max(low_activation, rule3)

    #dictionary search
    rule4 = applicant_eval["Low"]
    very_low_activation = max(very_low_activation, rule4)

    rule5 = house_eval["Very_low"]
    very_low_activation = max(very_low_activation, rule5)

    rule6 = min(applicant_eval["Medium"], house_eval["Very_low"])
    low_activation = max(low_activation, rule6)

    rule7 = min(applicant_eval["Medium"], house_eval["Low"])
    low_activation = max(low_activation, rule7)

    rule8 = min(applicant_eval["Medium"], house_eval["Medium"])
    medium_activation = max(medium_activation, rule8)

    rule9 = min(applicant_eval["Medium"], house_eval["High"])
    high_activation = max(high_activation, rule9)

    rule10 = min(applicant_eval["Medium"], house_eval["Very_high"])
    high_activation = max(high_activation, rule10)

    rule11 = min(applicant_eval["High"], house_eval["Very_low"])
    low_activation = max(low_activation, rule11)

    rule12 = min(applicant_eval["High"], house_eval["Low"])
    medium_activation = max(medium_activation, rule12)

    rule13 = min(applicant_eval["High"], house_eval["Medium"])
    high_activation = max(high_activation, rule13)

    rule14 = min(applicant_eval["High"], house_eval["High"])
    high_activation = max(high_activation, rule14)

    rule15 = min(applicant_eval["High"], house_eval["Very_high"])
    very_high_activation = max(very_high_activation, rule15)

    credit_evaluations = {
```

```
        "Very_low": very_low_activation,
        "Low": low_activation,
        "Medium": medium_activation,
        "High": high_activation,
        "Very_high": very_high_activation
    }

    return credit_evaluations



def evaluateThePerson(market_value, location, asset, income, interest):
    HouseResult = houseEval(market_value, location)
    appResult = applicantEval(asset, income)
    creditResult =  amountCredit(HouseResult, appResult, income, interest)


    #defuzzyfication part 2 crisp
    credit_values = {"Very_low": 0, "Low": 25, "Medium": 50, "High": 75, "Very_high": 100}
    numerator = 0
    denominator = 0

    for category, activation in creditResult.items():
        numerator += credit_values[category] * activation
        denominator += activation

    if denominator == 0:
        crisp_result = 0
    else:
        crisp_result = numerator / denominator

    return {
        "House Evaluation": HouseResult,
        "Applicant Evaluation": appResult,
        "Credit Evaluation": creditResult,
        "Crisp Credit Score": crisp_result
    }









if __name__ == "__main__":
    person1 = evaluateThePerson(market_value=600, location=4, asset=1000, income=890, interest=3)
    print("-------------------------------\nPERSON 1",end="\n-------------------------------\n")
    print("House Evaluation \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t Applicant Score
\t\t\t\t\t\t\t\t\t\t\t\t Credit Score\t\t\t\t\t\t Crisp Credit Score")
    print(str(person1['House Evaluation']).strip('{}'),str(person1['Applicant
Evaluation']).strip('{}'),str(person1['Credit Evaluation']).strip('{}'),(person1['Crisp Credit
Score']),sep=" |*| ")

    person2 = evaluateThePerson(market_value=900, location=8, asset=10000, income=8900,
interest=1)
    print("-------------------------------\nPERSON 2",end="\n-------------------------------\n")
    print("House Evaluation \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t Applicant Score \t\t\t\t\t\t\t\t\t\t\t
Credit Score\t\t\t\t\t\t\t Crisp Credit Score")
    print(str(person2['House Evaluation']).strip('{}'),str(person2['Applicant
Evaluation']).strip('{}'),str(person2['Credit Evaluation']).strip('{}'),(person2['Crisp Credit
Score']),sep=" |*| ")



    person3 = evaluateThePerson(market_value=200, location=1, asset=10000, income=900,
interest=10)
    print("-------------------------------\nPERSON 3",end="\n-------------------------------\n")
```

```
    print("House Evaluation \t\t\t\t\t\t\t\t\t\t\t\t\t Applicant Score \t\t\t\t\t\t\t\t\t\t\t\t
Credit Score\t\t\t\t\t\t\t Crisp Credit Score")
    print(str(person3['House Evaluation']).strip('{}'),str(person3['Applicant
Evaluation']).strip('{}'),str(person3['Credit Evaluation']).strip('{}'),(person3['Crisp Credit
Score']),sep=" |*| ")



print()
os.system("pause")
```

length : 411 rows

The rule set:

<u>1. House Evaluation</u>

1. If (Market_value is Low) then (House is Low)

2. If (Location is Bad) then (House is Low)

3. If (Location is Bad) and (Market_value is Low) then (House is Very_low)

4. If (Location is Bad) and (Market_value is Medium) then (House is Low)

5. If (Location is Bad) and (Market_value is High) then (House is Medium)

6. If (Location is Bad) and (Market_value is Very_high) then (House is High)

7. If (Location is Fair) and (Market_value is Low) then (House is Low)

8. If (Location is Fair) and (Market_value is Medium) then (House is Medium)

9. If (Location is Fair) and (Market_value is High) then (House is High)

10.If (Location is Fair) and (Market_value is Very_high) then (House is Very_high)

11.If (Location is Excellent) and (Market_value is Low) then (House is Medium)

12.If (Location is Excellent) and (Market_value is Medium) then (House is High)

13.If (Location is Excellent) and (Market_value is High) then (House is Very_high)

14.If (Location is Excellent) and (Market_value is Very_high) then (House is Very_high)

<u>2. Applicant Evaluation</u>

1. If (Asset is Low) and (Income is Low) then (Applicant is Low)

2. If (Asset is Low) and (Income is Medium) then (Applicant is Low)

3. If (Asset is Low) and (Income is High) then (Applicant is Medium)

4. If (Asset is Low) and (Income is Very_high) then (Applicant is High)

5. If (Asset is Medium) and (Income is Low) then (Applicant is Low)

6. If (Asset is Medium) and (Income is Medium) then (Applicant is Medium)

7. If (Asset is Medium) and (Income is High) then (Applicant is High)

8. If (Asset is Medium) and (Income is Very_high) then (Applicant is High)

9. If (Asset is High) and (Income is Low) then (Applicant is Medium)

10.If (Asset is High) and (Income is Medium) then (Applicant is Medium)

11.If (Asset is High) and (Income is High) then (Applicant is High)

12.If (Asset is High) and (Income is Very_high) then (Applicant is High)

3. Evaluation of the Amount of Credit

1. If (Income is Low) and (Interest is Medium) then (Credit is Very_low)

2. If (Income is Low) and (Interest is High) then (Credit is Very_low)

3. If (Income is Medium) and (Interest is High) then (Credit is Low)

4. If (Applicant is Low) then (Credit is Very_low)

5. If (House is Very_low) then (Credit is Very_low)

6. If (Applicant is Medium) and (House is Very_low) then (Credit is Low)

7. If (Applicant is Medium) and (House is Low) then (Credit is Low)

8. If (Applicant is Medium) and (House is Medium) then (Credit is Medium)

9. If (Applicant is Medium) and (House is High) then (Credit is High)

10.If (Applicant is Medium) and (House is Very_high) then (Credit is High)

11.If (Applicant is High) and (House is Very_low) then (Credit is Low)

12.If (Applicant is High) and (House is Low) then (Credit is Medium)

13.If (Applicant is High) and (House is Medium) then (Credit is High)

14.If (Applicant is High) and (House is High) then (Credit is High)

15.If (Applicant is High) and (House is Very_high) then (Credit is Very_high)

REFERENCES

1. Lect. MUHAMMED SAADETDİN KAYA, COM2536 - Project Manual, (Access Date: 26/05/2025) https://ekampus.ankara.edu.tr/pluginfile.php/286331/mod_assign/introattachment/0/COM2536%20-%20Project.docx?forcedownload=1
2. Mamdani Inference Method (Acces Date 26/05/2025): Mamdani Fuzzy Model