Fuzzy-Expert Documentation

Release 0.1.0

Juan D. Velasquez

June 24, 2021

CONTENTS

Author

Prof. Juan David Velásquez-Henao, MSc, PhD
Universidad Nacional de Colombia, Sede Medellín.
jdvelasq@unal.edu.co

What is it?

Fuzzy-Expert is a Python package for building Mamdani Fuzzy Inference Systems. **Fuzzy-Expert** admits a mix of fuzzy and crisp values and associated certainty values. The inference system returns crisp values and certainty factors for the conclusions of the rules.

Main Features

- Specification of membership functions using standard functions or as a list of points.
- Different functions for specifying T-norms and Conorms.
- Different implication operators.
- Composition using *max-min* or *max-prod* rules.
- Most common functions for production aggregation.
- Different operators to defuzzificate fuzzy sets.

Release Information

- Date: June 24, 2021 **Version**: 0.1.0
- Binary Installers: https://pypi.org/project/fuzzy_expert
- Source Repository: https://github.com/jdvelasq/fuzzy-expert
- Documentation: https://jdvelasq.github.io/fuzzy-expert/

CHAPTER

ONE

INSTALLATION

1.1 Requeriments

Fuzzy-Expert was developed in Python 3.9, and requires *Numpy* version 1.20 or higher.

1.2 How to install

The current stable version can be installed from the command line using:

```
$ pip install fuzzy_expert
```

The current development version can be installed by clonning the GitHub repo https://github.com/jdvelasq/fuzzy-expert and executing

```
$ python3 setup.py install develop
```

at the command prompt.

CHAPTER

TWO

TUTORIALS

2.1 Tutorial 1 - Bank decision loan problem with crisp inputs

In this tutorial a fuzzy inference system for loan approbation is builded. The problem has three input variables: score, ratio, and credit; and one output variable: decision.

[1]:
```
import os

os.chdir('/workspaces/fuzzy-expert')
```

[2]:
```
import warnings

warnings.filterwarnings("ignore")
```

2.1.1 Variable specification

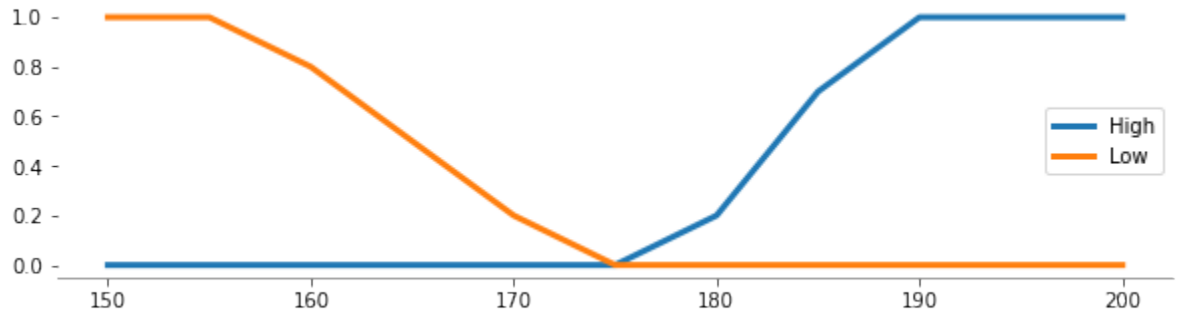
In the following code, a dictionary containing the variables of the problem is defined. The keys of the dictionary are the names of the variables in the rules. For each variable is defined the limits of the universe of discourse, the terms, and the membership function for each term. Finally, the variable `score` is ploted.

[3]:
```
import matplotlib.pyplot as plt
import numpy as np

from fuzzy_expert.variable import FuzzyVariable

variables = {
    "score": FuzzyVariable(
        universe_range=(150, 200),
        terms={
            "High": [(175, 0), (180, 0.2), (185, 0.7), (190, 1)],
            "Low": [(155, 1), (160, 0.8), (165, 0.5), (170, 0.2), (175, 0)],
        },
    ),
    "ratio": FuzzyVariable(
```

(continues on next page)

(continued from previous page)

```
        universe_range=(0.1, 1),
        terms={
            "Goodr": [(0.3, 1), (0.4, 0.7), (0.41, 0.3), (0.42, 0)],
            "Badr": [(0.44, 0), (0.45, 0.3), (0.5, 0.7), (0.7, 1)],
        },
    ),
    #
    "credit": FuzzyVariable(
        universe_range=(0, 10),
        terms={
            "Goodc": [(2, 1), (3, 0.7), (4, 0.3), (5, 0)],
            "Badc": [(5, 0), (6, 0.3), (7, 0.7), (8, 1)],
        },
    ),
    #
    "decision": FuzzyVariable(
        universe_range=(0, 10),
        terms={
            "Approve": [(5, 0), (6, 0.3), (7, 0.7), (8, 1)],
            "Reject": [(2, 1), (3, 0.7), (4, 0.3), (5, 0)],
        },
    ),
}

plt.figure(figsize=(10, 2.5))
variables["score"].plot()
```



2.1.2 Rule specification

The fuzzy inference system has two rules. They are directly stored in a list.

[4]:
```
from fuzzy_expert.rule import FuzzyRule

rules = [
    FuzzyRule(
        premise=[
            ("score", "High"),
            ("AND", "ratio", "Goodr"),
            ("AND", "credit", "Goodc"),
        ],
        consequence=[("decision", "Approve")],
    ),
    FuzzyRule(
        premise=[
            ("score", "Low"),
            ("AND", "ratio", "Badr"),
            ("OR", "credit", "Badc"),
        ],
        consequence=[("decision", "Reject")],
    )
]

print(rules[0])
print()
print(rules[1])
```

```
IF  score IS High
    AND ratio IS Goodr
    AND credit IS Goodc
THEN
    decision IS Approve
CF = 1.00
Threshold-CF = 0.00


IF  score IS Low
    AND ratio IS Badr
    OR credit IS Badc
THEN
    decision IS Reject
CF = 1.00
Threshold-CF = 0.00
```

2.1.3 Inference system specification and computations

Finally, the fuzzy inference system is specified. The model is used to evaluate the following crisp values for the input variables: `score=190`, `ratio=0.39`, and `credit=1.5`. The model returns a dictionary with the values of the variables in the consequence of the rules and the certainty factor of the conclusion.

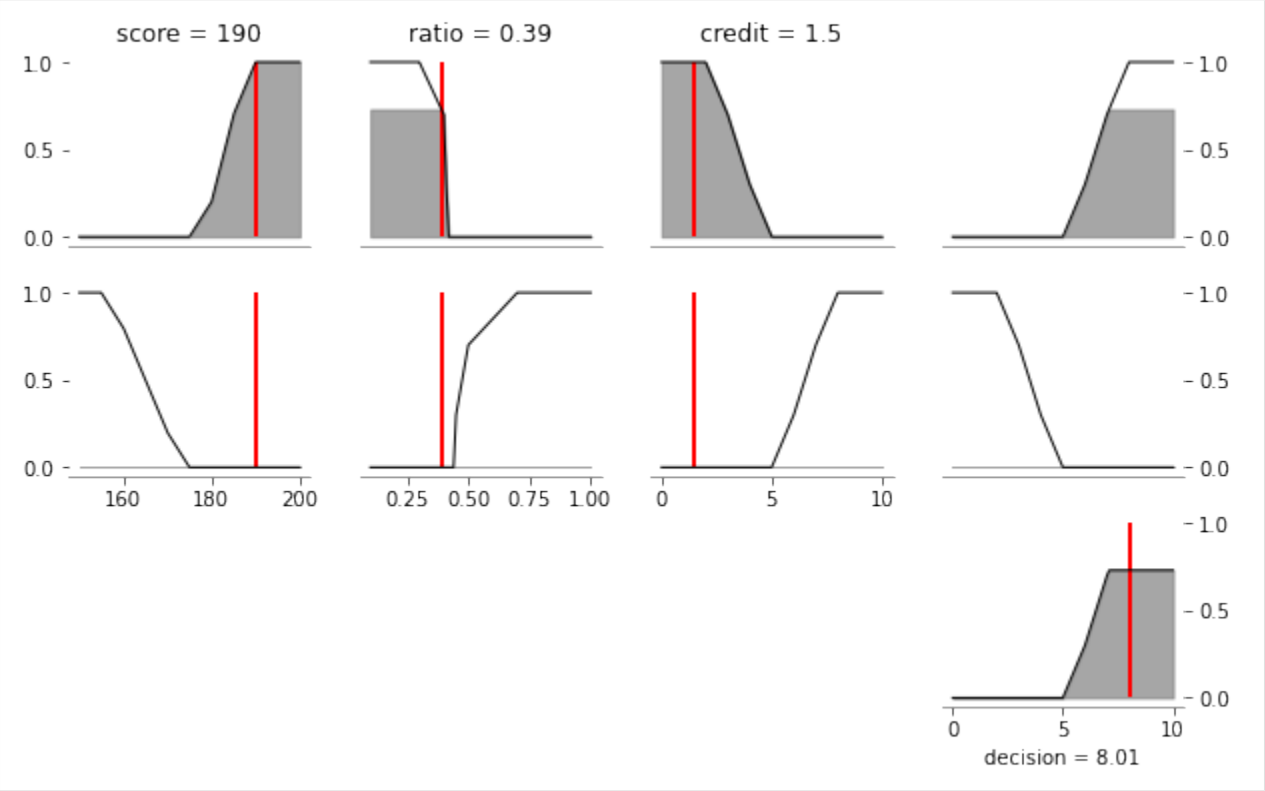
[5]:
```
from fuzzy_expert.inference import DecompositionalInference

model = DecompositionalInference(
    and_operator="min",
    or_operator="max",
    implication_operator="Rc",
    composition_operator="max-min",
    production_link="max",
    defuzzification_operator="cog",
)

model(
    variables=variables,
    rules=rules,
    score=190,
    ratio=0.39,
    credit=1.5,
)
```

[5]:
```
({'decision': 8.010492631084489}, 1.0)
```

2.1.4 Visualization of the results

The function plot can be used to obtain a graphical represntation of the results of the inference systems. It uses the same parameters used to the evaluation of the system.

[6]:
```
plt.figure(figsize=(10, 6))
model.plot(
    variables=variables,
    rules=rules,
    score=190,
    ratio=0.39,
    credit=1.5,
)
```



2.1.5 User interaction

Using the `ipywidgets` package in Jupyter Lab it is possible to obtain an interactive user interface for the user. The funtion `demo` is used to plot the results; following, the function `interact` is used to create the user interface.

[7]:
```
from ipywidgets import interact, widgets

def demo(score, ratio, credit):
    plt.figure(figsize=(10,6))
    model.plot(
        variables=variables,
        rules=rules,
        score=score,
        ratio=ratio,
        credit=credit,
    )

interact(
    demo,
    score=widgets.FloatSlider(min=150, max=200),
    ratio=widgets.FloatSlider(min=0.1, max=1),
    credit=widgets.FloatSlider(min=0, max=10),
```

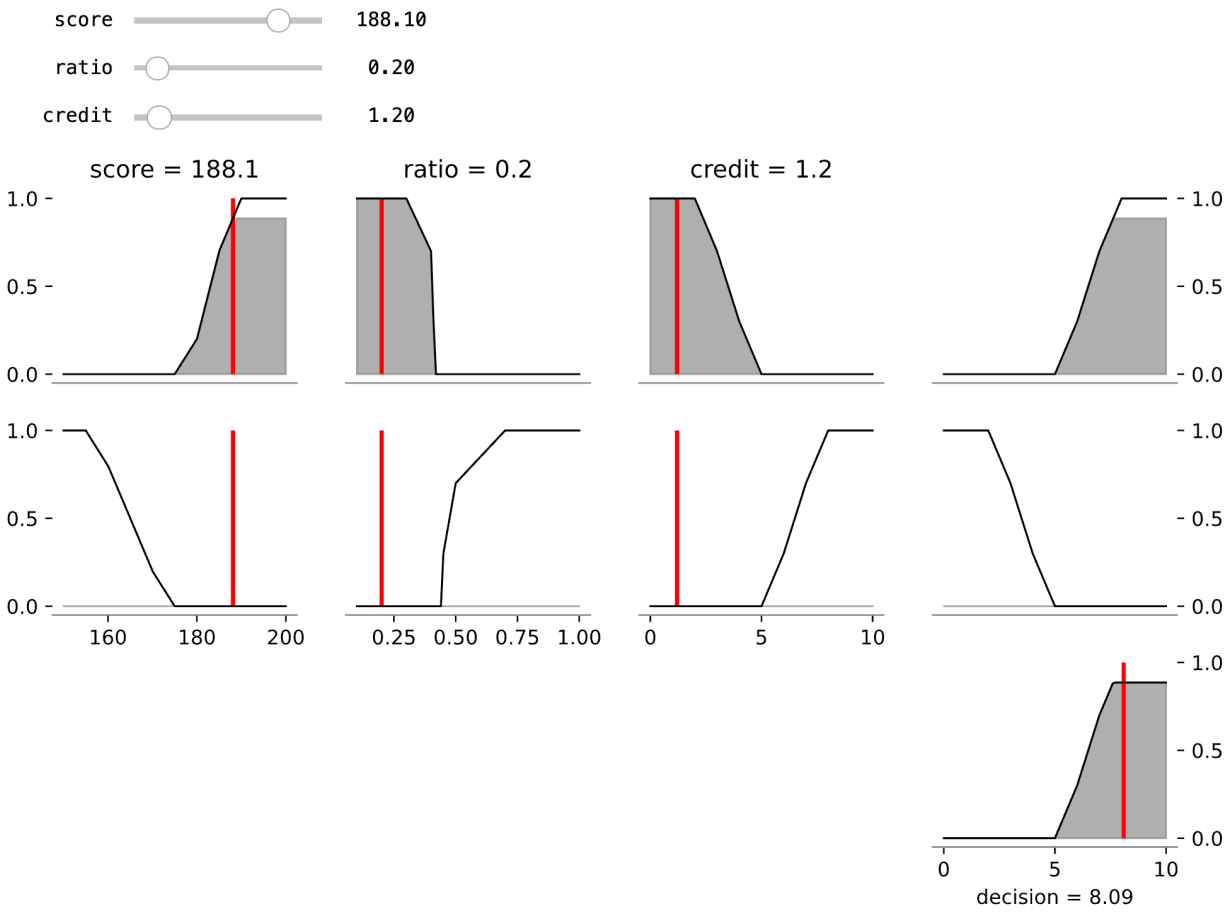
(continues on next page)

(continued from previous page)

```
)
```

```
interactive(children=(FloatSlider(value=150.0, description='score', max=200.0,␣
↪min=150.0), FloatSlider(value=0...
```

[7]:
```
<function __main__.demo(score, ratio, credit)>
```



2.2 Tutorial 2 - Bank decision loan problem with fuzzy inputs

This tutorial uses the fuzzy inference system developed in Tutorial 1.

[1]:
```
import os
import warnings

os.chdir('/workspaces/fuzzy-expert')
warnings.filterwarnings("ignore")
```

2.2.1 Specification of the fuzzy inference system

[2]:
```
import matplotlib.pyplot as plt
import numpy as np

from fuzzy_expert.variable import FuzzyVariable
from fuzzy_expert.rule import FuzzyRule
from fuzzy_expert.inference import DecompositionalInference

variables = {
    "score": FuzzyVariable(
        universe_range=(150, 200),
        terms={
            "High": [(175, 0), (180, 0.2), (185, 0.7), (190, 1)],
            "Low": [(155, 1), (160, 0.8), (165, 0.5), (170, 0.2), (175, 0)],
        },
    ),
    "ratio": FuzzyVariable(
        universe_range=(0.1, 1),
        terms={
            "Goodr": [(0.3, 1), (0.4, 0.7), (0.41, 0.3), (0.42, 0)],
            "Badr": [(0.44, 0), (0.45, 0.3), (0.5, 0.7), (0.7, 1)],
        },
    ),
    #
    "credit": FuzzyVariable(
        universe_range=(0, 10),
        terms={
            "Goodc": [(2, 1), (3, 0.7), (4, 0.3), (5, 0)],
            "Badc": [(5, 0), (6, 0.3), (7, 0.7), (8, 1)],
        },
    ),
    #
    "decision": FuzzyVariable(
        universe_range=(0, 10),
        terms={
            "Approve": [(5, 0), (6, 0.3), (7, 0.7), (8, 1)],
            "Reject": [(2, 1), (3, 0.7), (4, 0.3), (5, 0)],
        },
    ),
}

rules = [
    FuzzyRule(
        premise=[
            ("score", "High"),
```

(continues on next page)

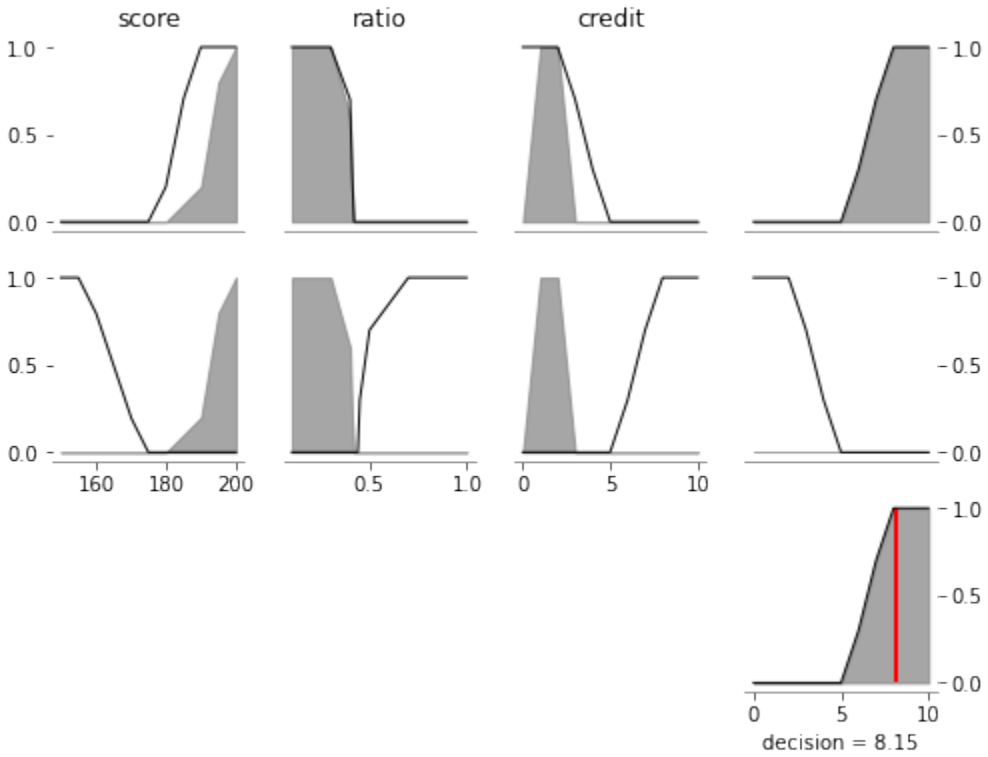
(continued from previous page)

```
            ("AND", "ratio", "Goodr"),
            ("AND", "credit", "Goodc"),
        ],
        consequence=[("decision", "Approve")],
    ),
    FuzzyRule(
        premise=[
            ("score", "Low"),
            ("AND", "ratio", "Badr"),
            ("OR", "credit", "Badc"),
        ],
        consequence=[("decision", "Reject")],
    )
]

model = DecompositionalInference(
    and_operator="min",
    or_operator="max",
    implication_operator="Rc",
    composition_operator="max-min",
    production_link="max",
    defuzzification_operator="cog",
)
```

2.2.2 Computation with fuzzy inputs

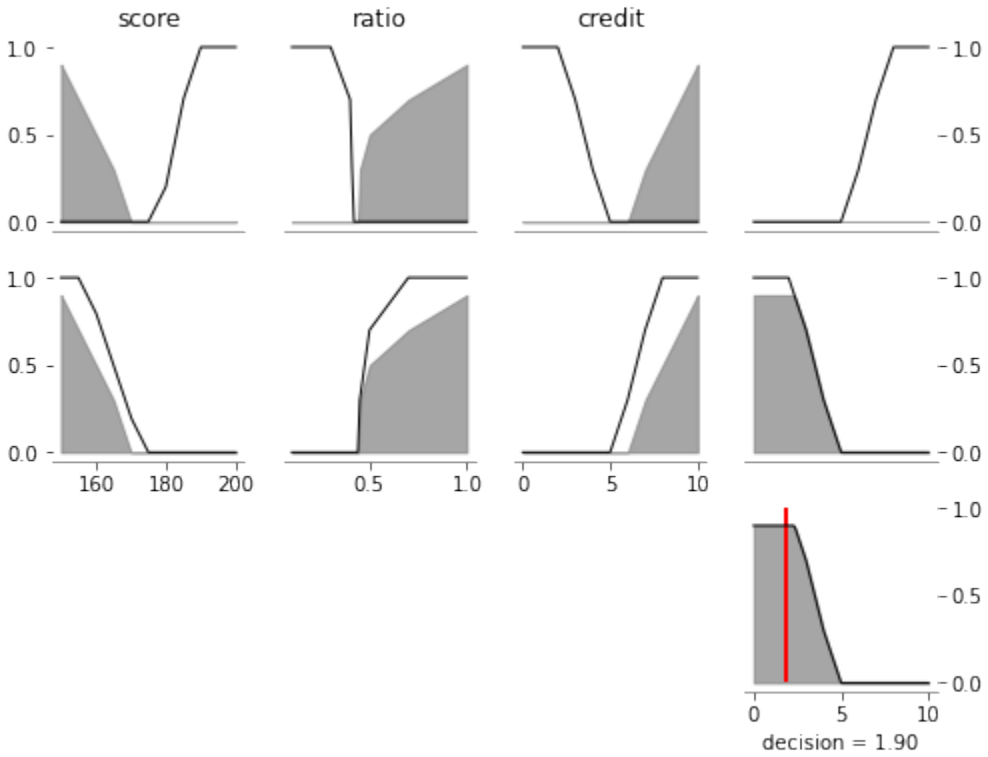
Fuzzy inputs are specified as a list of points (x, u), where x is a point in the universe of discourse and u is the corresponding value of the membership function. In the first case, the fuzziness of inputs are considered; however, values are according with an approval decision.

[3]:
```
plt.figure(figsize=(10,6))

model.plot(
    variables=variables,
    rules=rules,
    score=[(180, 0.0), (190, 0.2), (195, 0.8), (200, 1.0)],
    ratio=[(0.1, 1), (0.3, 1), (0.4, 0.6), (0.41, 0.2),  (0.42, 0)],
    credit=[(0, 0), (1, 1), (2, 1), (3, 0.0), (4, 0.0)],
)
```



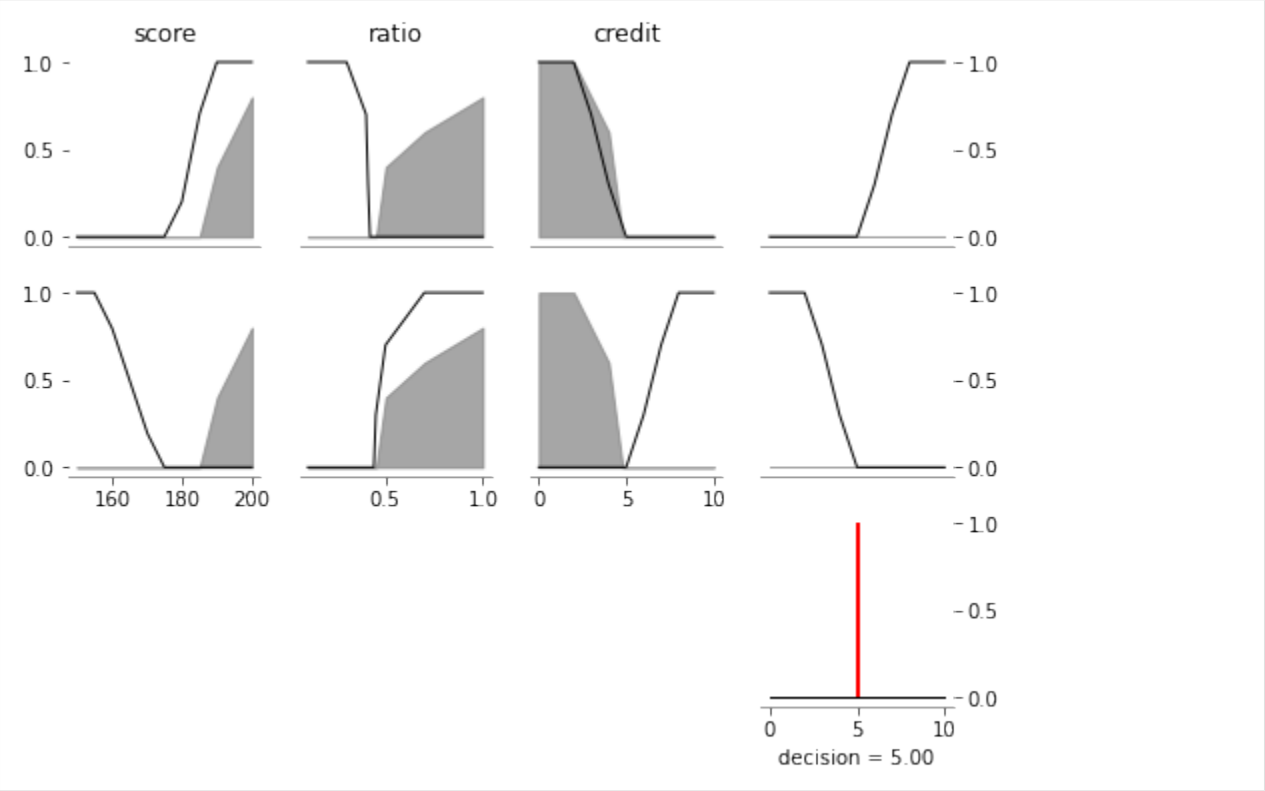
In the second case, the values are clearly related to a reject decision.

[10]:
```
plt.figure(figsize=(10,6))

model.plot(
    variables=variables,
    rules=rules,
    score=[(150, 0.9), (155, 0.7), (160, 0.5), (165, 0.3), (170, 0.0)],
    ratio=[(0.44, 0), (0.45, 0.3), (0.5, 0.5), (0.7, 0.7),  (1, 0.9)],
    credit=[(6, 0), (7, 0.3), (8, 0.5), (9, 0.7), (10, 0.9)],
)
```



In the third case, two variables have good values and the third a bad value, causing indetermination in the result.

[13]:
```
plt.figure(figsize=(10,6))

model.plot(
    variables=variables,
    rules=rules,
    score=[(185, 0.0), (190, 0.4), (195, 0.6), (200, 0.8)],
    ratio=[(0.45, 0), (0.5, 0.4), (0.7, 0.6), (1, 0.8)],
    credit=[(2, 1), (3, 0.8), (4, 0.6), (4.8, 0.0)],
)
```



2.3 Tutorial 3 - Standard membership functions

This tutorial uses the fuzzy inference system developed in Tutorial 1.

[1]:
```
import os
import warnings

os.chdir('/workspaces/fuzzy-expert')
warnings.filterwarnings("ignore")
```

2.3.1 Specification of the fuzzy variables with standard membership functions

In the following code, fuzzy sets are specified using standard membership functions, which are described in the function reference section.

Fuzzy sets in variables `score` and `ratio` are specified using the `smf` and `zmf` functions. Fuzzy sets for variables `credit` and `decision` are specified using the `trapmf` function.

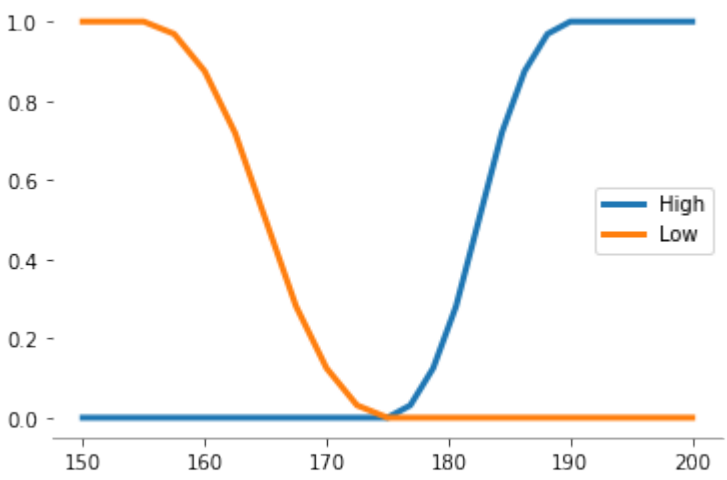
[2]:
```
import matplotlib.pyplot as plt
import numpy as np
```

(continues on next page)

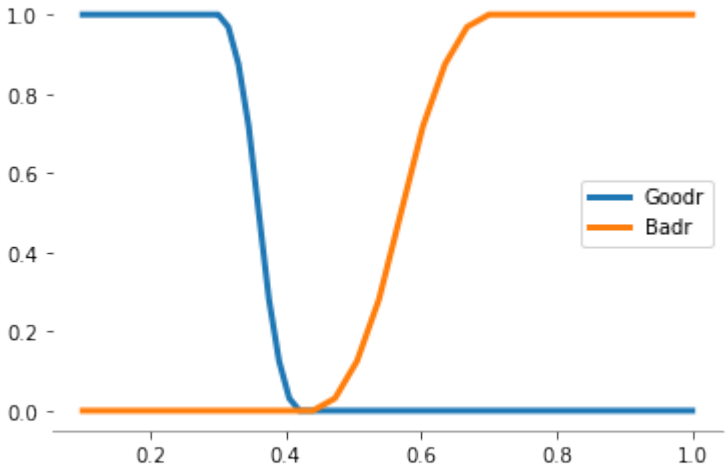
(continued from previous page)

```
from fuzzy_expert.variable import FuzzyVariable

variables = {
    "score": FuzzyVariable(
        universe_range=(150, 200),
        terms={
            "High": ('smf', 175, 190),
            "Low": ('zmf', 155, 175),
        },
    ),
    "ratio": FuzzyVariable(
        universe_range=(0.1, 1),
        terms={
            "Goodr": ('zmf', 0.3, 0.42),
            "Badr": ('smf', 0.44, 0.7),
        },
    ),
    #
    "credit": FuzzyVariable(
        universe_range=(0, 10),
        terms={
            "Goodc": ('trapmf', 0, 0, 2, 5),
            "Badc":  ('trapmf', 5, 8, 10, 10),
        },
    ),
    #
    "decision": FuzzyVariable(
        universe_range=(0, 10),
        terms={
            "Approve": ('trapmf', 5, 8, 10, 10),
            "Reject": ('trapmf', 0, 0, 2, 5),
        },
    ),
}
```

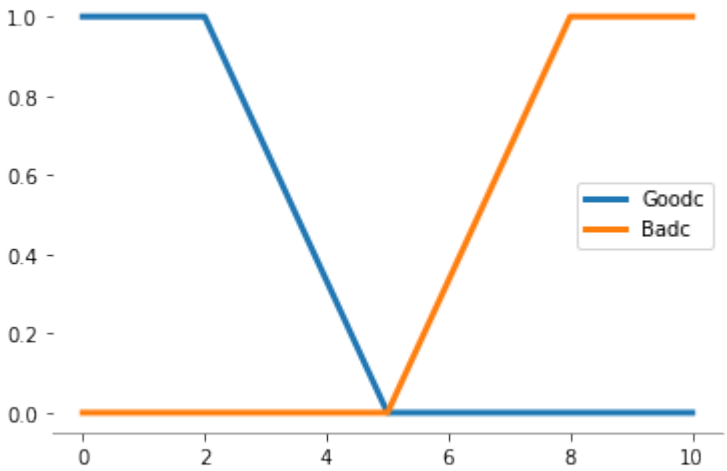
[3]:
```
variables['score'].plot()
```



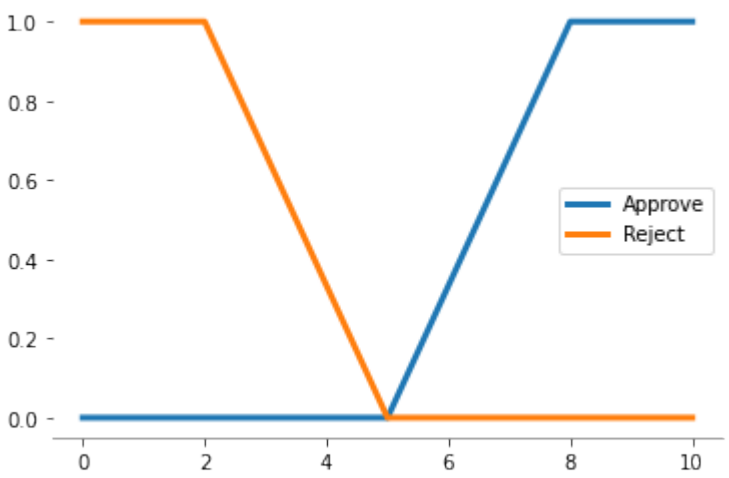
[4]:
```
variables['ratio'].plot()
```



[5]:
```
variables['credit'].plot()
```



[6]:
```
variables['decision'].plot()
```



2.4 Tutorial 4 - Certainty factors

This tutorial uses the fuzzy inference system developed in Tutorial 1.

[5]:
```
import os
import warnings

os.chdir('/workspaces/fuzzy-expert')
warnings.filterwarnings("ignore")
```

2.4.1 Fuzzy Variables

[6]:
```
import matplotlib.pyplot as plt
import numpy as np

from fuzzy_expert.variable import FuzzyVariable
from fuzzy_expert.rule import FuzzyRule
from fuzzy_expert.inference import DecompositionalInference

variables = {
    "score": FuzzyVariable(
        universe_range=(150, 200),
        terms={
            "High": [(175, 0), (180, 0.2), (185, 0.7), (190, 1)],
            "Low": [(155, 1), (160, 0.8), (165, 0.5), (170, 0.2), (175, 0)],
        },
    ),
```

(continues on next page)

(continued from previous page)

```
    "ratio": FuzzyVariable(
        universe_range=(0.1, 1),
        terms={
            "Goodr": [(0.3, 1), (0.4, 0.7), (0.41, 0.3), (0.42, 0)],
            "Badr": [(0.44, 0), (0.45, 0.3), (0.5, 0.7), (0.7, 1)],
        },
    ),
    #
    "credit": FuzzyVariable(
        universe_range=(0, 10),
        terms={
            "Goodc": [(2, 1), (3, 0.7), (4, 0.3), (5, 0)],
            "Badc": [(5, 0), (6, 0.3), (7, 0.7), (8, 1)],
        },
    ),
    #
    "decision": FuzzyVariable(
        universe_range=(0, 10),
        terms={
            "Approve": [(5, 0), (6, 0.3), (7, 0.7), (8, 1)],
            "Reject": [(2, 1), (3, 0.7), (4, 0.3), (5, 0)],
        },
    ),
}
```

2.4.2 Fuzzy rules with certainty factors

It is possible to assign a certainty factor (`cf`) to each rule. If this value is not specified, it has assumed to be equal to 1.0. In addition, the `threshold_cf` is the minimum certainty factor required to consider the rule fired; this is, rules with a computed certainty factor below the threshold are not considering for computing the output of the system. The first rule has a certainty factor of `0.9`, while the second rule has a certainty factor of `1.0` (by default).

[7]:
```
rules = [
    FuzzyRule(
        cf=0.9,
        premise=[
            ("score", "High"),
            ("AND", "ratio", "Goodr"),
            ("AND", "credit", "Goodc"),
        ],
        consequence=[("decision", "Approve")],
    ),
    FuzzyRule(
        premise=[
```

(continues on next page)

(continued from previous page)

```
            ("score", "Low"),
            ("AND", "ratio", "Badr"),
            ("OR", "credit", "Badc"),
        ],
        consequence=[("decision", "Reject")],
    )
]
```

2.4.3 Facts with certainty factors

In addition, also it is possible to assign certainty factors to the facts. When a certainty factor not is specified by the user, it has a default value or 1.0. In the following code, the variables `score`, `ratio`, and `credit` have certainty factors of `0.9`, `1.0`, and `0.95` respectively. The conclusion is `decision=8.01` with a certainty factor of `0.95`.

[8]:
```
from fuzzy_expert.inference import DecompositionalInference

model = DecompositionalInference(
    and_operator="min",
    or_operator="max",
    implication_operator="Rc",
    composition_operator="max-min",
    production_link="max",
    defuzzification_operator="cog",
)

model(
    variables=variables,
    rules=rules,
    score=(190, 0.9),
    ratio=(0.39, 1.0),
    credit=(1.5, 0.95),
)
```

[8]:
```
({'decision': 8.010492631084489}, 0.95)
```

CHAPTER

THREE

LIBRARY REFERENCE

Description of the functions, classes and modules contained within **Fuzzy-Expert**.

3.1 Modifiers and operators

fuzzy_expert.operators.**apply_modifiers**(*membership*, *modifiers*)

Apply a list of modifiers or hedges to a numpy array.

Parameters

- **membership** (*npt.ArrayLike*) – Membership values to be modified.
- **modifiers** (*List[str]*) – List of modifiers or hedges.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import apply_modifiers
>>> x = [0.0, 0.25, 0.5, 0.75, 1]
>>> apply_modifiers(x, ('not', 'very'))
array([1.    , 0.9375, 0.75  , 0.4375, 0.    ])
```

fuzzy_expert.operators.**bounded_diff**(*memberships*)

Applies the element-wise function max(0, u - v).

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import bounded_diff
>>> x = [0, 0.25, 0.5, 0.75, 1]
>>> y = [0, 0.25, 0.5, 0.6, 0.7]
>>> bounded_diff([x, y])
array([0.  , 0.  , 0.  , 0.15, 0.3 ])
```

fuzzy_expert.operators.**bounded_prod**(*memberships*)

Applies the element-wise function max(0, u + v - 1).

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import bounded_prod
>>> x = [0.1, 0.25, 0.5, 0.75, 1]
>>> bounded_prod([x, x])
array([0.  , 0.  , 0.  , 0.5 , 1.  ])
```

fuzzy_expert.operators.**bounded_sum**(*memberships*)

Applies the element-wise function min(1, u + v).

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import bounded_sum
>>> x = [0, 0.25, 0.5, 0.75, 1]
>>> bounded_sum([x, x, x])
array([0.  , 0.75, 1.  , 1.  , 1.  ])
```

fuzzy_expert.operators.**defuzzificate**(*universe*, *membership*, *operator='cog'*)

Computes a representative crisp value for the fuzzy set.

Parameters

- **universe** – Array of values representing the universe of discourse.
- **membership** – Array of values representing the membership function.
- **operator** – Method used for computing the crisp representative value of the fuzzy set.
  - *"cog"*: Center of gravity.
  - *"boa"*: Bisector of area.
  - *"mom"*: Mean of the values for which the membership function is maximum.
  - *"lom"*: Largest value for which the membership function is maximum.
  - *"som"*: Smallest value for which the membership function is minimum.

Return type dict

```
>>> from fuzzy_expert.operators import defuzzificate
>>> u = [0, 1,   2, 3, 4]
>>> m = [0, 0, 0.5, 1, 1]
>>> defuzzificate(u, m, "cog")
2.9166666666666665
```

```
>>> defuzzificate(u, m, "boa")
3.0
```

```
>>> defuzzificate(u, m, "mom")
3.5
```

```
>>> defuzzificate(u, m, "lom")
4
```

```
>>> defuzzificate(u, m, "som")
3
```

fuzzy_expert.operators.**drastic_prod**(*memberships*)

Applies the element-wise function f(u, v) = u if v == 0 else v if u == 1 else 0.

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import drastic_prod
>>> x = [0, 0.25, 0.5, 0.75, 1]
>>> y = [1, 0.75, 0.5, 0.25, 0]
>>> drastic_prod([x, y])
array([0., 0., 0., 0., 1.])
```

fuzzy_expert.operators.**drastic_sum**(*memberships*)

Applies the element-wise function f(u, v) = u if v == 0 else v if u == 0 else 1.

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import drastic_sum
>>> x = [0.1, 0.25, 0.5, 0.75, 0.3]
>>> y = [0, 0.75, 0.5, 0.25, 0]
>>> drastic_sum([x, y])
array([0.1, 1. , 1. , 1. , 0.3])
```

fuzzy_expert.operators.**extremely**(*membership*)

Applies the element-wise function fn(u) = u^3.

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import extremely
>>> extremely([0, 0.25, 0.5, 0.75, 1])
array([0.      , 0.015625, 0.125   , 0.421875, 1.      ])
```

fuzzy_expert.operators.**intensify**(*membership*)

Applies the element-wise function fn(u) = u^2 if u <= 0.5 else 1 - 2 * (1 - u)^2.

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import intensify
>>> intensify([0, 0.25, 0.5, 0.75, 1])
array([0.    , 0.0625, 0.25  , 0.875 , 1.    ])
```

fuzzy_expert.operators.**maximum**(*memberships*)

Applies the element-wise function f(u, v) = max(u, v).

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import maximum
>>> x = [0.1, 0.25, 0.5, 0.75, 0.3]
>>> y = [0, 0.75, 0.5, 0.25, 0]
>>> maximum([x, y])
array([0.1 , 0.75, 0.5 , 0.75, 0.3 ])
```

fuzzy_expert.operators.**minimum**(*memberships*)

Applies the element-wise function f(u, v) = min(u, v).

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import minimum
>>> x = [0.1, 0.25, 0.5, 0.75, 0.3]
>>> y = [0, 0.75, 0.5, 0.25, 0]
>>> minimum([x, y])
array([0.  , 0.25, 0.5 , 0.25, 0.  ])
```

fuzzy_expert.operators.**more_or_less**(*membership*)

Applies the element-wise function fn(u) = u^(1/2).

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import more_or_less
>>> more_or_less([0, 0.25, 0.5, 0.75, 1])
array([0.        , 0.5       , 0.70710678, 0.8660254 , 1.        ])
```

fuzzy_expert.operators.**norm**(*membership*)

Applies the element-wise function fn(u) = u / max(u).

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import norm
>>> norm([0, 0.25, 0.5])
array([0. , 0.5, 1. ])
```

fuzzy_expert.operators.**not_**(*membership*)

Applies the element-wise function fn(u) = 1 - u.

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import not_
>>> not_([0, 0.25, 0.5, 0.75, 1])
array([1.  , 0.75, 0.5 , 0.25, 0.  ])
```

fuzzy_expert.operators.**plus**(*membership*)

Applies the element-wise function fn(u) = u^1.25.

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import plus
>>> plus([0, 0.25, 0.5, 0.75, 1])
array([0.        , 0.1767767 , 0.42044821, 0.69795364, 1.        ])
```

fuzzy_expert.operators.**prob_or**(*memberships*)

Applies the element-wise function fn(u, v) = u + v - u * v. Also known as algebraic-sum.

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import prob_or
>>> x = [0.1, 0.25, 0.5, 0.75, 0.3]
>>> y = [0, 0.75, 0.5, 0.25, 0]
>>> prob_or([x, y])
array([0.1   , 0.8125, 0.75  , 0.8125, 0.3   ])
```

fuzzy_expert.operators.**product**(*memberships*)

Applies the element-wise function f(u, v) = u * v.

Parameters **memberships** (*List[npt.ArrayLike]*) – List of arrays of membership values.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import product
>>> x = [0, 0.25, 0.5, 0.75, 1]
>>> product([x, x, x])
array([0.      , 0.015625, 0.125   , 0.421875, 1.      ])
```

fuzzy_expert.operators.**slightly**(*membership*)

Applies the element-wise function fn(u) = u^(1/2).

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import slightly
>>> slightly([0, 0.25, 0.5, 0.75, 1])
array([0.        , 0.16326531, 0.99696182, 1.        , 0.        ])
```

fuzzy_expert.operators.**somewhat**(*membership*)

Applies the element-wise function fn(u) = u^(1/3).

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import somewhat
>>> somewhat([0, 0.25, 0.5, 0.75, 1])
array([0.        , 0.62996052, 0.79370053, 0.9085603 , 1.        ])
```

fuzzy_expert.operators.**very**(*membership*)

Applies the element-wise function fn(u) = u^2.

Parameters **membership** (*npt.ArrayLike*) – Membership function to be modified.

Return type npt.ArrayLike

```
>>> from fuzzy_expert.operators import very
>>> very([0, 0.25, 0.5, 0.75, 1])
array([0.    , 0.0625, 0.25  , 0.5625, 1.    ])
```

3.2 Membership Functions

Functions in this module returns a standard membership function specificaion as a list of points (x_i, u_i).

class fuzzy_expert.mf.**MembershipFunction**(*n_points=9*)

Bases: object

Membership function constructor.

Parameters **n_points** (*int*) – Number base point for building the approximations.

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=3)
>>> mf(('gaussmf', 5, 1))
[(2, 0), (3.0, 0.1353352832366127), (3.8, 0.48675225595997157), (4.6, 0.
↪9231163463866356), (5.0, 1.0), (5.4, 0.9231163463866356), (6.2, 0.
↪48675225595997157), (7.0, 0.1353352832366127), (8, 0)]
```

gaussmf(*center*, *sigma*)

Gaussian membership function.

Parameters

- **center** (*float*) – Defines the center of the membership function.
- **sigma** (*float*) – Defines the width of the membership function, where a larger value creates a wider membership function.

Return type List[Tuple[float, float]]

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=3)
>>> mf.gaussmf(center=5, sigma=1)
[(2, 0), (3.0, 0.1353352832366127), (3.8, 0.48675225595997157), (4.6, 0.
↪9231163463866356), (5.0, 1.0), (5.4, 0.9231163463866356), (6.2, 0.
↪48675225595997157), (7.0, 0.1353352832366127), (8, 0)]
```

gbellmf(*center*, *width*, *shape*)

Generalized bell-shaped membership function.

Parameters

- **center** (*float*) – Defines the center of the membership function.
- **width** (*float*) – Defines the width of the membership function, where a larger value creates a wider membership function.
- **shape** (*float*) – Defines the shape of the curve on either side of the central plateau, where a larger value creates a more steep transition.

Return type List[Tuple[float, float]]

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=3)
>>> mf.gbellmf(center=5, width=1, shape=0.5)
[(-1, 0), (0.0, 0.16666666666666666), (1.0, 0.2), (2.0, 0.25), (3.0, 0.
↪3333333333333333), (3.8, 0.45454545454545453), (4.0, 0.5), (4.6, 0.
↪7142857142857141), (5.0, 1.0), (5.4, 0.7142857142857141), (6.0, 0.5),␣
↪(6.2, 0.45454545454545453), (7.0, 0.3333333333333333), (8.0, 0.25),␣
↪(9.0, 0.2), (10.0, 0.16666666666666666), (11, 0)]
```

pimf(*left_feet*, *left_peak*, *right_peak*, *right_feet*)

Pi-shaped membership function.

Parameters

- **left_feet** (*float*) – Defines the left feet of the membership function.
- **left_peak** (*float*) – Defines the left peak of the membership function.
- **right_peak** (*float*) – Defines the right peak of the membership function.
- **right_feet** (*float*) – Defines the right feet of the membership function.

Return type List[Tuple[float, float]]

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=4)
>>> mf.pimf(left_feet=1, left_peak=2, right_peak=3, right_feet=4)
[(1.0, 0.0), (1.3333333333333333, 0.22222222222222213), (1.
↪6666666666666665, 0.7777777777777776), (2.0, 1.0), (3.0, 1.0), (3.
↪3333333333333335, 0.7777777777777776), (3.6666666666666665, 0.
↪22222222222222243), (4.0, 0.0)]
```

sigmf(*center*, *width*)

Sigmoidal membership function.

Parameters

- **center** (*float*) – Defines the center of the membership function.
- **width** (*float*) – Defines the width of the membership function.

Return type List[Tuple[float, float]]

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=3)
>>> mf.sigmf(center=5, width=1)
[(-1, 0), (0.0, 0.0066928509242848554), (2.0, 0.04742587317756678), (4.
↪0, 0.2689414213699951), (5.0, 0.5), (6.0, 0.7310585786300049), (8.0,␣
↪0.9525741268224334), (10.0, 0.9933071490757153), (11, 1)]
```

smf(*foot*, *shoulder*)

S-shaped membership function.

Parameters

- **foot** (*float*) – Defines the foot of the membership function.
- **shoulder** (*float*) – Defines the shoulder of the membership function.

Return type List[Tuple[float, float]]

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=4)
>>> mf.smf(foot=1, shoulder=2)
[(1.0, 0.0), (1.3333333333333333, 0.22222222222222213), (1.
↪6666666666666665, 0.7777777777777776), (2.0, 1.0)]
```

trapmf(*left_feet*, *left_peak*, *right_peak*, *right_feet*)
Trapezoidal membership function.

Parameters

- **left_feet** (*float*) – Defines the left feet of the membership function.
- **left_peak** (*float*) – Defines the left peak of the membership function.
- **right_peak** (*float*) – Defines the right peak of the membership function.
- **right_feet** (*float*) – Defines the right feet of the membership function.

Return type List[Tuple[float, float]]

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=4)
>>> mf.trapmf(left_feet=1, left_peak=2, right_peak=3, right_feet=4)
[(1.0, 0.0), (2.0, 1.0), (3.0, 1.0), (4.0, 0.0)]
```

trimf(*left_feet*, *peak*, *right_feet*)
Triangular membership function.

Parameters

- **left_feet** (*float*) – Defines the left feet of the membership function.
- **peak** (*float*) – Defines the peak of the membership function.
- **right_feet** (*float*) – Defines the right feet of the membership function.

Return type List[Tuple[float, float]]

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=4)
>>> mf.trimf(left_feet=1, peak=2, right_feet=4)
[(1.0, 0.0), (2.0, 1.0), (4.0, 0.0)]
```

zmf(*shoulder*, *feet*)
Z-shaped membership function.

Parameters

- **shoulder** (*float*) – Defines the shoulder of the membership function.
- **feet** (*float*) – Defines the feet of the membership function.

Return type List[Tuple[float, float]]

```
>>> from fuzzy_expert.mf import MembershipFunction
>>> mf = MembershipFunction(n_points=4)
>>> mf.zmf(shoulder=1, feet=2)
[(1.0, 1.0), (1.3333333333333333, 0.7777777777777779), (1.
↪6666666666666665, 0.22222222222222243), (2.0, 0.0)]
```

3.3 Fuzzy Variables

class fuzzy_expert.variable.**FuzzyVariable**(*universe_range*, *terms=None*, *step=0.1*)

Bases: object

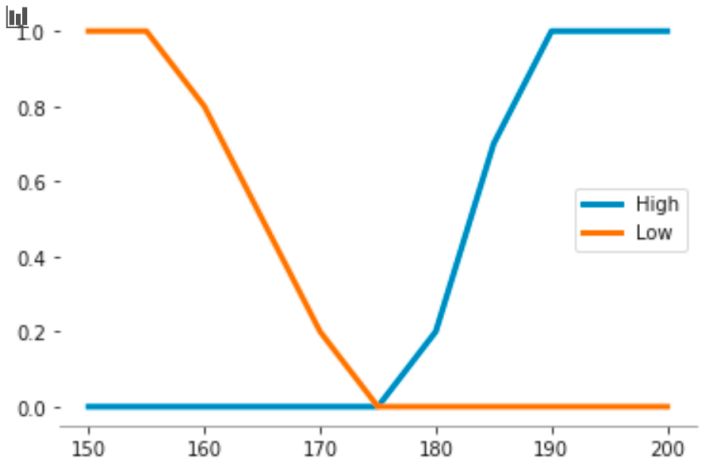
Creates a fuzzy variable.

Parameters

- **unverse_range** – Limits of the universe of discourse.
- **terms** (*Union[dict, None]*) – Dictionary where each term is the key of the dictionary, and the values is the membership function.
- **step** (*float*) – Value controling the resolution for the discrete representation of the universe.
- **universe_range** (*tuple[float, float]*) –

Return type None

```
>>> from fuzzy_expert.variable import FuzzyVariable
>>> v = FuzzyVariable(
...     universe_range=(150, 200),
...     terms={
...         "High": [(175, 0), (180, 0.2), (185, 0.7), (190, 1)],
...         "Low": [(155, 1), (160, 0.8), (165, 0.5), (170, 0.2), (175, 0)],
...     },
... )
>>> v.plot()
```



add_points_to_universe(*points*)

get_modified_membeship(*term*, *modifiers=None*)

Returns the membership modified values for the term.

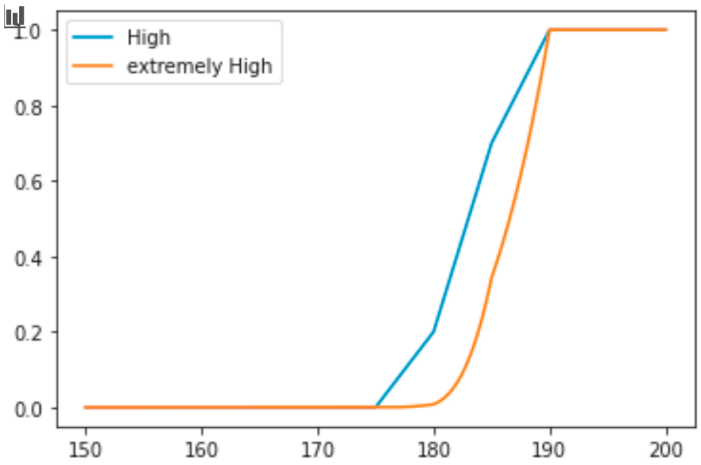
Parameters

- **term** (*str*) – Name of the fuzzy set.

- **modifiers** (*Union[None, List[str]]*) – List of modifiers.

Return type numpy.ndarray

```
>>> import matplotlib.pyplot as plt
>>> from fuzzy_expert.variable import FuzzyVariable
>>> v = FuzzyVariable(
...     universe_range=(150, 200),
...     terms={
...         "High": [(175, 0), (180, 0.2), (185, 0.7), (190, 1)],
...         "Low": [(155, 1), (160, 0.8), (165, 0.5), (170, 0.2), (175,␣
↪0)],
...     },
... )
>>> y = v.get_modified_membeship('High' ,['extremely'])
>>> _ = plt.plot(v.universe, v['High'], label='High')
>>> _ = plt.plot(v.universe, y, label='extremely High')
>>> _ = plt.legend()
>>> plt.show()
```



plot(*fmt='-'*, *linewidth=3*)

Plots a fuzzy variable.

Parameters

- **fmt** (*str*) – Format string passed to Matplotlib.pyplot.
- **linewidth** (*float*) – Width of lines.

Return type None

plot_input(*value*, *fuzzyset*, *view_xaxis=True*, *view_yaxis='left'*)

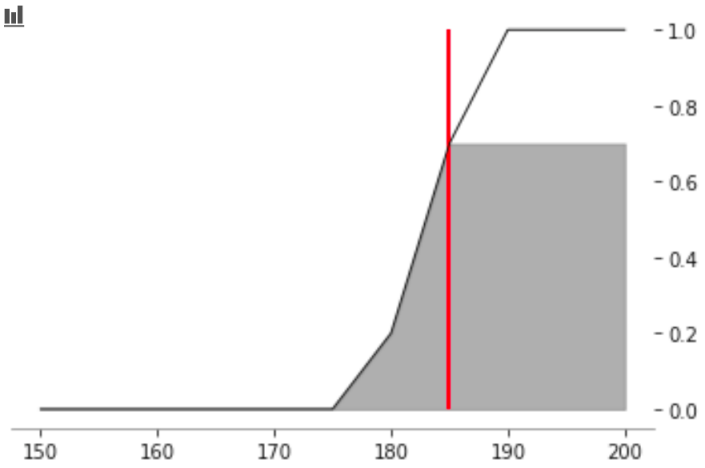
Plots the fuzzy set, and the input.

Parameters

- **value** – Crisp or fuzzy input value.
- **fuzzyset** – Term to plot.

- **view_xaxis** – Draw the x-axis of plot
- **view_yaxis** – Draw the y-axis of plot at left or right side.

```
>>> from fuzzy_expert.variable import FuzzyVariable
>>> v = FuzzyVariable(
...     universe_range=(150, 200),
...     terms={
...         "High": [(175, 0), (180, 0.2), (185, 0.7), (190, 1)],
...         "Low": [(155, 1), (160, 0.8), (165, 0.5), (170, 0.2), (175,␣
↪0)],
...     },
... )
>>> v.plot_input(value=185, fuzzyset='High', view_xaxis=True, view_
↪yaxis='right')
```



3.4 Zadeh-Mamdani Rules

class fuzzy_expert.rule.**FuzzyRule**(*premise*, *consequence*, *cf=1.0*, *threshold_cf=0*)

Bases: object

Creates a Zadeh-Mamdani fuzzy rule.

Parameters

- **premise** – List of propositions in rule premise.
- **consequence** – List of propositions in rule consequence.
- **cf** (*float*) – Certainty factor of the rule.
- **threshold_cf** (*float*) – Minimum certainty factor for rule firing.

```
>>> from fuzzy_expert.rule import FuzzyRule
>>> rule = FuzzyRule(
```

(continues on next page)

(continued from previous page)

```
...     premise=[
...         ("score", "High"),
...         ("AND", "ratio", "Goodr"),
...         ("AND", "credit", "Goodc"),
...     ],
...     consequence=[("decision", "Approve")],
... )
>>> rule
IF  score IS High
    AND ratio IS Goodr
    AND credit IS Goodc
THEN
    decision IS Approve
CF = 1.00
Threshold-CF = 0.00
```

3.5 Inference Method

class fuzzy_expert.inference.**DecompositionalInference**(*and_operator*, *or_operator*, *implication_operator*, *composition_operator*, *production_link*, *defuzzification_operator*)

Bases: `object`

Decompositional inference method.

Parameters

- **and_operator** – AND operator method for combining the compositions of propositions in a fuzzy rule premise, specified as one of the following:
  - *"min"*.
  - *"prod"*.
  - *"bunded_prod"*.
  - *"drastic_prod"*.
- **or_operator** – OR operator method for combining the compositions of propositions in a fuzzy rule premise, specified as one of the following:
  - *"max"*.
  - *"prob_or"*.
  - *"bounded_sum"*.
  - *"drastic_sum"*.

- **implication_operator** – method for computing the compositions of propositions in a fuzzy rule premise, specified as one of the following:
  - *"Ra"*.
  - *"Rm"*.
  - *"Rc"*.
  - *"Rb"*.
  - *"Rs"*.
  - *"Rg"*.
  - *"Rsg"*.
  - *"Rgs"*.
  - *"Rgg"*.
  - *"Rss"*.
- **production_link** – method for aggregating the consequences of the fuzzy rules, specified as one of the following:
  - *"min"*.
  - *"prod"*.
  - *"bunded_prod"*.
  - *"drastic_prod"*.
  - *"max"*.
  - *"prob_or"*.
  - *"bounded_sum"*.
  - *"drastic_sum"*.
- **defuzzification_operator** – Method for defuzzificate the resulting membership function, specified as one of the following:
  - *"cog"*: Center of gravity.
  - *"boa"*: Bisector of area.
  - *"mom"*: Mean of the values for which the membership function is maximum.
  - *"lom"*: Largest value for which the membership function is maximum.
  - *"som"*: Smallest value for which the membership function is minimum.

plot(*variables*, *rules*, ***facts*)

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

S

T

V

Z