**БИБЛИОТЕКИ ДЛЯ РЕАЛИЗАЦИИ МАТРИЧНОГО НЕЧЁТКОГО ЛОГИЧЕСКОГО ВЫВОДА НА PYTHON**

**Руководство пользователя**

На 10 листах

Епишина Наталья

epishina.nata@bk.ru

Пермь, 2023

Оглавление

[1 Введение 3](#_Toc133368355)

[1.1 Область применения 3](#_Toc133368356)

[1.2 Краткое описание возможностей 3](#_Toc133368357)

[1.3 Уровень подготовки пользователя 3](#_Toc133368358)

[2 НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ 4](#_Toc133368359)

[2.1 Виды деятельности, функции 4](#_Toc133368360)

[2.2 Программные и аппаратные требования к системе 4](#_Toc133368361)

[3 ПОДГОТОВКА К РАБОТЕ 5](#_Toc133368362)

[3.1 Запуск системы 5](#_Toc133368363)

[3.2 Проверка работоспособности системы 5](#_Toc133368364)

[4 ОПИСАНИЕ ОПЕРАЦИЙ 6](#_Toc133368365)

[4.1 Задание объекта класса FuzzyInferenceSystem 6](#_Toc133368366)

[4.2 Выбор алгоритма 6](#_Toc133368367)

[4.3 Выбор метода получения четкого результата 6](#_Toc133368368)

[4.4 Создание признака или лингвистической переменной 6](#_Toc133368369)

[4.5 Создание предиката или терма лингвистической переменной 7](#_Toc133368370)

[4.6 Создание правил 7](#_Toc133368371)

[4.7 Вычисление значений выходных параметров 7](#_Toc133368372)

[4.8 Вывод графиков принадлежности 7](#_Toc133368373)

[5 Пример 8](#_Toc133368374)

# Введение

## Область применения

Нечёткий вывод используется для решения задач, где необходимо работать с нечеткими, невыразительными или неопределенными данными. Например, в задачах управления процессами, где точные математические модели могут быть слишком сложными. Нечеткий вывод позволяет работать с нечеткими данными и принимать решения на основе лингвистических переменных, которые не имеют четкого числового значения.

## Краткое описание возможностей

Библиотека позволяет:

* использовать матричные нечёткие векторы, предикаты;
* задавать входные и выходные параметры алгоритма;
* задавать графики функций истинности предикатов выбором из заранее настроенного набора;
* создавать правила базы знаний;
* производить нечёткий логический вывод с использованием созданной базы знаний в соответствии с матричным алгоритмом или алгоритмом Мамдани.

## Уровень подготовки пользователя

Базовое умение работы с ПК, наличие среды разработки, которая поддерживает язык программирования python.

# НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ

## Виды деятельности, функции

Данная библиотека подходит для решения задач систем управления. С ее помощью можно вычислить значения выходных параметров по заданным значениям входных.

## Программные и аппаратные требования к системе

Система – библиотека на python. Обязательно наличие среды разработки, которая поддерживает язык программирования python.

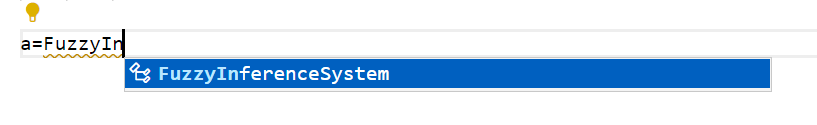
# ПОДГОТОВКА К РАБОТЕ

## Запуск системы

Установка библиотеки происходит через терминал с помощью команды pip install fuzzy\_matrix\_mamdani.

## Проверка работоспособности системы

Если всё прошло успешно, вам станет доступен основной класс (FuzzyInferenceSystem) для взаимодействия.



# ОПИСАНИЕ ОПЕРАЦИЙ

## Задание объекта класса FuzzyInferenceSystem

Для начала работы вам нужно задать объект класса FuzzyInferenceSystem, который будет далее полностью описывать вашу систему.

Например: fis = FuzzyInferenceSystem()

## Выбор алгоритма

Для работы доступны два алгоритма – Мамдани и матричный. Для выбора алгоритма нужно присвоить полю algorithm соответствующее строковое значение.

Например: fis.algorithm = 'Matrix'или fis.algorithm = 'Mamdani'

## Выбор метода получения четкого результата

Для работы доступны два метода – метод центра тяжести масс и упрощенный метод дефаззификации. Для выбора метода нужно присвоить полю defuzzification соответствующее строковое значение.

Например: fis.defuzzification = "Simple" или fis.defuzzification = " Centroid"

Рекомендуется при выборе алгоритма Мамдани использовать метод Centroid, а для матричного – Simple.

## Создание признака или лингвистической переменной

Для этого нужно выбрать метод create\_feature и передать ему 5 параметров: строковое значение с названием ЛП, строковое значение со системой измерения, значение типа float для указания минимального значения, значение типа float для указания максимального значения и логического значение, где true – входная переменная, а false – выходная.

Например: f\_temp = fis.create\_feature("Температура", "C", 0, 150, True)

## Создание предиката или терма лингвистической переменной

Для этого нужно выбрать метод create\_ predicate и передать ему 3 параметра: строковое значение с названием терма, функцию принадлежности. Доступно три функции принадлежности: треугольная Triangle, передаем три значения, трапециевидная Trapeze, передаем четыре значения, и гауссовская Gauss передаем два значения (координата максимума функции принадлежности и коэффициент концентрации функции принадлежности). Также если вы выбрали метод Simple, то вместо функции, следует передавать константу, которая будет означать центр тяжести фигуры, в задании термов для выходного параметра.

Например: p\_temp\_low = fis.create\_predicate(f\_temp, 'низкая', Trapeze(0, 50, 100, 150))

p\_temp\_low = fis.create\_predicate(f\_temp, 'низкая', const=75)

## Создание правил

Для этого нужно выбрать метод create\_rule и передать ему список условий, заключение и вес правила. Условия в правиле соединены между собой связкой “и”, поэтому если в правиле присутствует связка “или”, правило следует разбить на несколько правил.

Например: r\_1 = fis.create\_rule([p\_temp\_low, p\_flow\_low], p\_pressure\_low, 1)

## Вычисление значений выходных параметров

Для этого помещаем в список, переменные в которые запишем результат, и вызываем метод predict, передавая ему значения входных параметров, в том порядке, в котором вводили ЛП.

Например: [pressure] = fis.predict(85, 3.5)

## Вывод графиков принадлежности

Для этого нужно вызвать метод show\_view.

Например: fis.show\_view()

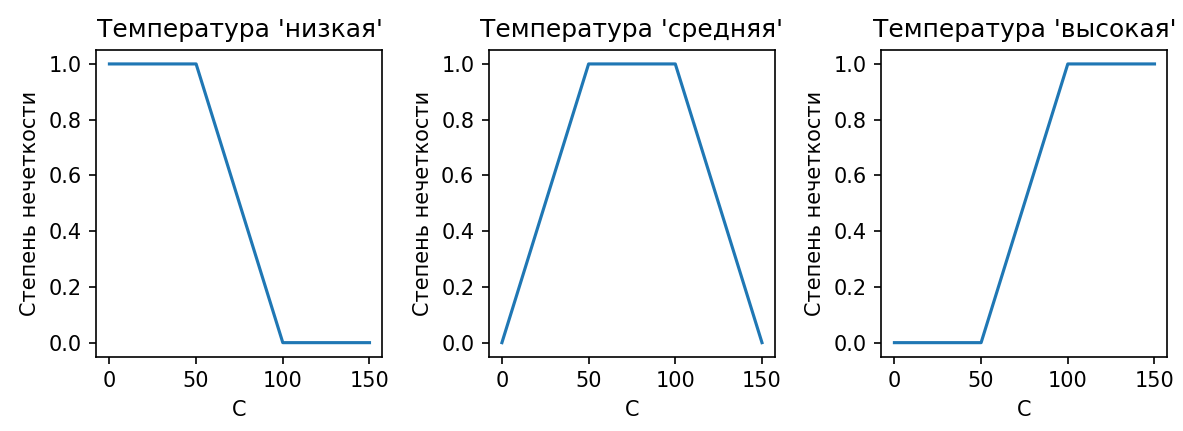
# Пример

Пусть у нас есть некоторая система, например реактор, описываемая тремя параметрами: температура, давление и расход рабочего вещества. Все показатели измеримы и множество возможных значений известно. Так же из опыта работы с системой известны некоторые правила, связывающие значения этих параметров. Предположим, что сломался датчик, измеряющий значение одного из параметров системы, но знать его значение хотя бы приблизительно необходимо. Тогда встаёт задача об отыскании значения этого неизвестного значения (пусть это будет давление) при известных значениях двух других параметров (температуры и расхода) и связи этих величин в виде следующих правил:

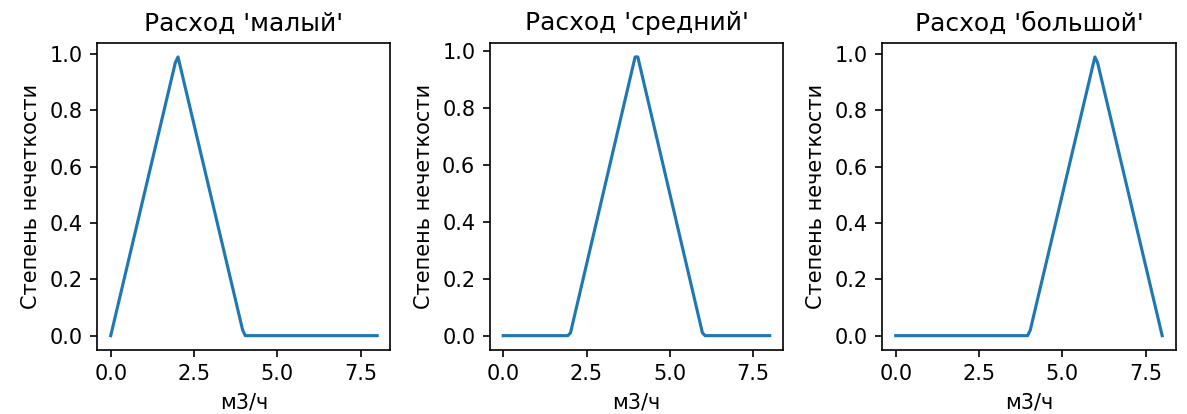
* Если Температура низкая и Расход малый, то Давление низкое;
* Если Температура средняя, то Давление среднее;
* Если Температура высокая, то Давление высокое;
* Если Расход большой, то Давление высокое.

В нашем случае Температура, Давление и Расход – лингвистические переменные. Опишем каждую из них:

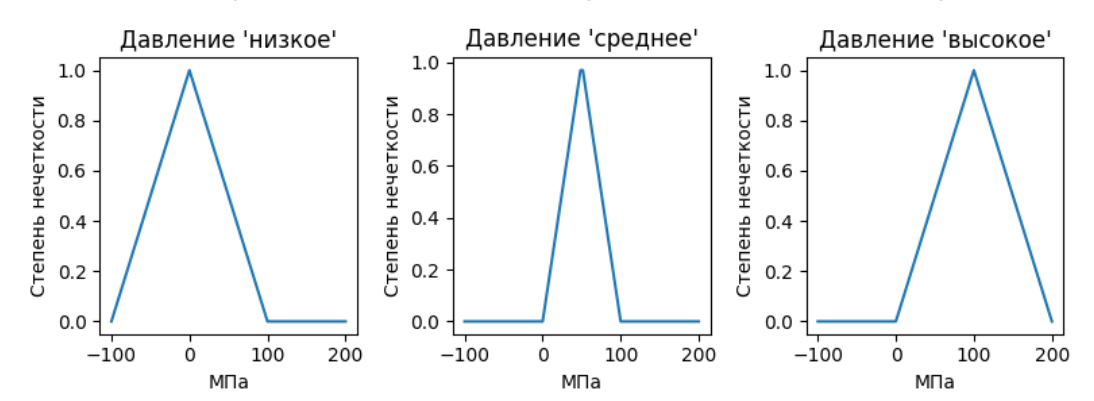
* Температура. Унивёрс (множество возможных значений) – отрезок [0,150]. Начальное множество термов {Высокая, Средняя, Низкая}. Функции принадлежности термов имеют следующий вид:



* Расход. Унивёрс – отрезок [0,8]. Начальное множество термов {Большой, Средний, Малый} Функции принадлежности термов имеют следующий вид:



* Давление. Унивёрс – отрезок [-100,200]. Начальное множество термов {Высокое, Среднее, Низкое} Функции принадлежности термов имеют следующий вид:



Пусть известны значения Температура 85 и Расход 3,5. Произведём расчёт значения давления.

Пример кода для работы с данной библиотекой:

fis = FuzzyInferenceSystem()

fis.algorithm = 'Matrix'

fis.defuzzification = "Simple"

f\_temp = fis.create\_feature("Температура", "C", 0, 150, True)

f\_flow =  fis.create\_feature("Расход", "м3/ч", 0, 8, True)

f\_pressure  = fis.create\_feature("Давление", "МПа", -100, 200, False)

p\_temp\_low = fis.create\_predicate(f\_temp, 'низкая', Trapeze(0, 0, 50, 100))

p\_temp\_normal = fis.create\_predicate(f\_temp, 'средняя', Trapeze(0, 50, 100,150))

p\_temp\_high = fis.create\_predicate(f\_temp, 'высокая', Trapeze(50, 100, 150, 150))

p\_flow\_low = fis.create\_predicate(f\_flow, 'малый', Triangle(0, 2, 4))

p\_flow\_normal = fis.create\_predicate(f\_flow, 'средний', Triangle(2, 4, 6))

p\_flow\_high = fis.create\_predicate(f\_flow, 'большой', Triangle(4, 6, 8))

p\_pressure\_low = fis.create\_predicate(f\_pressure, 'низкое', const=0)

p\_pressure\_normal = fis.create\_predicate(f\_pressure, 'среднее', const=50)

p\_pressure\_high = fis.create\_predicate(f\_pressure, 'высокое', const=100)

r\_1 = fis.create\_rule([p\_temp\_low, p\_flow\_low], p\_pressure\_low, 1)

r\_2 = fis.create\_rule([p\_temp\_normal], p\_pressure\_normal, 1)

r\_3 = fis.create\_rule([p\_temp\_high], p\_pressure\_high, 1)

r\_4 = fis.create\_rule([p\_flow\_high], p\_pressure\_high, 1)

[pressure] = fis.predict(85, 3.5)

print(f"Значение давления: {pressure}")

Результат работы: “Значение давления: 67.60563380281691”