МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Кафедра «информатика и вычислительная техника»

Отчет по лабораторной работе

по дисциплине: «Технологии разработки ПО»

РУКОВОДИТЕЛЬ:

Чернобаев И.Д.

(подпись)

Выполнили:

Разумовский С.

Пухов А.

Новичков В.

Егоров. А. Д.

М-19-ИВТ-3

Нижний Новгород

2020 г.

**Содержание**

[**Задание** 3](#_Toc59217740)

[**Введение** 3](#_Toc59217741)

[Чёткие множества и логика: 3](#_Toc59217742)

[Нечёткие множества: 5](#_Toc59217743)

[Системы нечеткого вывода: 7](#_Toc59217744)

[**Архитектура системы:** 10](#_Toc59217745)

[**Реализация классов** 10](#_Toc59217746)

[Класс FuzzySet 10](#_Toc59217747)

[Классы Fuzzy Variable (нечётких переменных) 13](#_Toc59217748)

[Класс FuzzyRules (нечёткие правила) 14](#_Toc59217749)

[Fuzzy System Class. 15](#_Toc59217750)

[Добавление нового правила в FuzzySystem 16](#_Toc59217751)

[**Пример использования:** 17](#_Toc59217752)

[Нечёткие правила 18](#_Toc59217753)

# **Задание**

Нечёткая классификация состояний объекта в диагностике.

# **Введение**

Многие задачи просты для людей, но они создают постоянную проблему для машин. Примеры таких систем включают прогулку по местности с множеством препятствий, подъем хрупких предметов или парковку автомобиля. Способность людей работать с неточными данными облегчает нам выполнение таких задач. Следовательно, если мы стремимся воспроизвести управляющие действия человека-оператора, мы должны иметь возможность моделировать действия оператора, а не самого предприятия. Наша модель должна быть построена таким образом, чтобы она могла работать с расплывчатой информацией.

Системы, основанные на нечеткой логике, делают именно это; они превосходны там, где системы особенно сложны, и успешно используются во многих приложениях, от распознавания голоса и рукописного ввода до контроля скорости движения поездов в метро.

## Чёткие множества и логика:

Классическая логика основана на четком множестве, где группа отдельных объектов рассматривается как коллекция. Например, белый и красный цвета являются отдельными объектами сами по себе, но их можно рассматривать как коллекцию, используя обозначение {красный, белый}. Чёткие множества условно обозначаются заглавной буквой, поэтому приведенный выше пример можно описать следующим образом:

F = {красный, белый}

Четкое подмножество можно определить из более обширного набора, где элементы набора принадлежат подмножеству в соответствии с некоторым условием. Например, набор A может быть определен как набор чисел, которые больше или равны 4 и меньше или равны 12. Этот оператор можно описать с помощью следующих обозначений:

A = {i | i - целое число и 4 <= i <= 12}

Графическое представление указанного выше подмножества возможно, если мы введем понятие характеристической или индикаторной функции множества, то есть, в данном случае, функция, определенная для множества целых чисел, которую мы будем называть X, будет указывать на принадлежность элементов из подмножества A в X. Это достигается путем присвоения значения 1 элементам X в A и значения 0 элементам X не в A. Следовательно, в нашем примере индикаторная функция для этого множества:

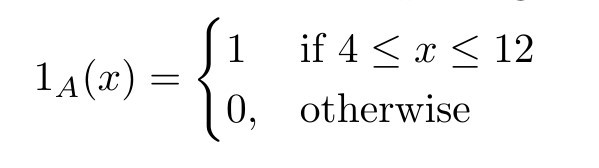


Рис 1. Индикаторная функция

Графически можно изобразить так:

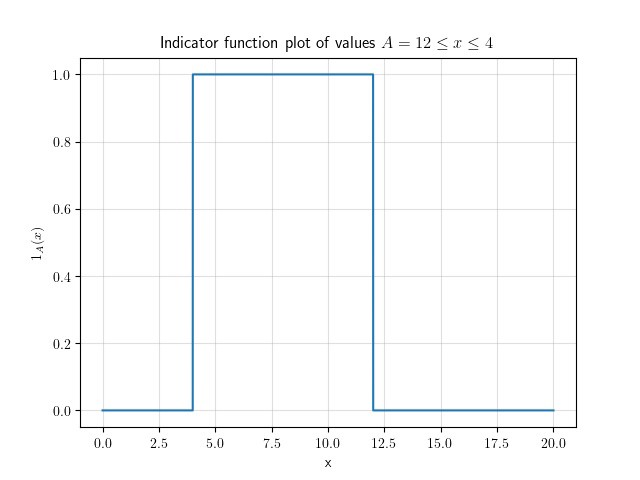


Рис 2. Графическое изображение индикаторной функции

Пересечение двух множеств — это множество, содержащее все элементы, общие для обоих множеств. Объединение двух множеств — это множество, содержащее все элементы, входящие в любое из множеств.

Отрицание множества A — это множество, содержащее все элементы, которых нет в A.

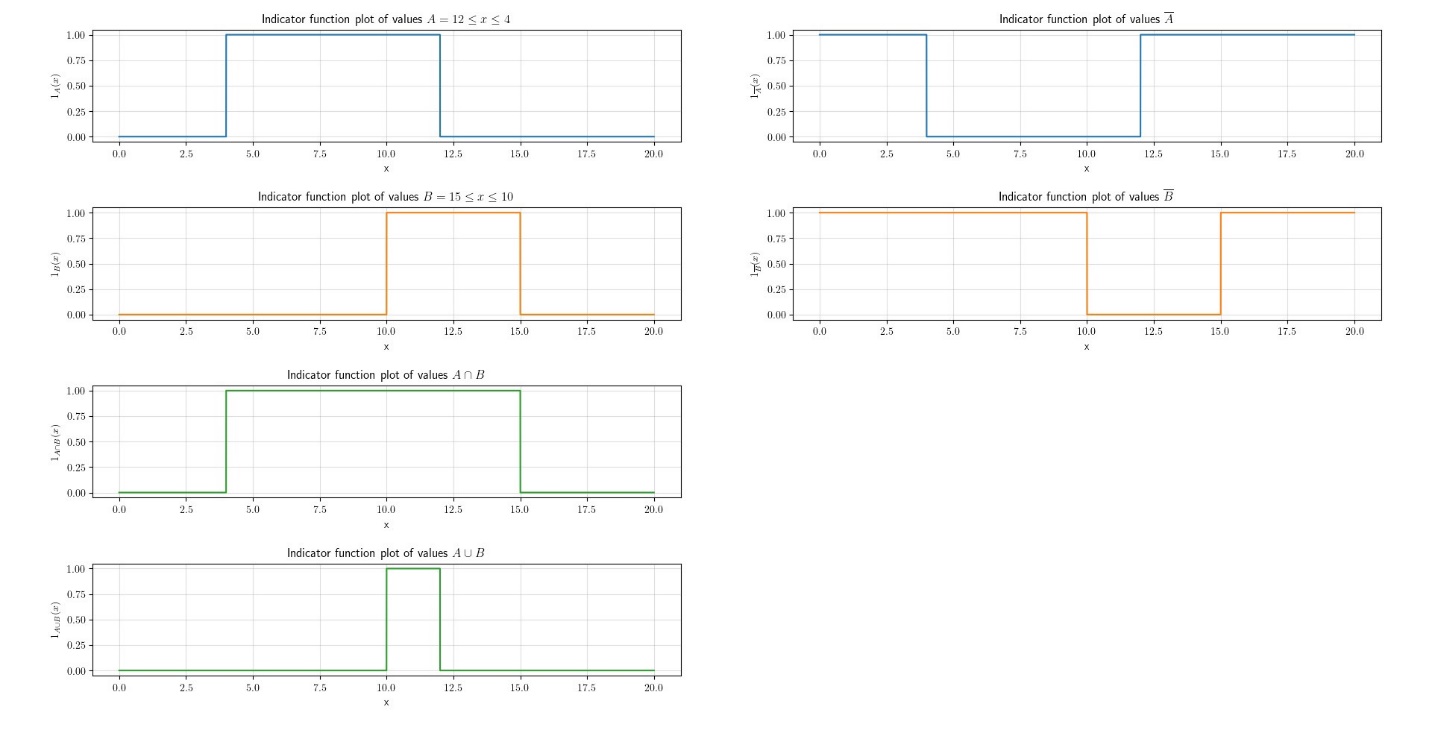


Рис 3. Операции с множествами

## Нечёткие множества:

В отличие от четких множеств, нечеткое множество допускает частичную принадлежность к множеству, которое определяется степенью принадлежности, обозначаемой µ, которая может принимать любое значение от 0 (вообще не принадлежит) до 1 (полностью принадлежит).

Очевидно, что, если мы удалим все значения принадлежности, кроме 0 и 1, нечеткое множество схлопнется до четкого множества, которое было описано ранее.

Функция принадлежности множества — это отношение между элементами множества и их степенью принадлежности. Ниже показано, как функции принадлежности могут применяться к температуре.

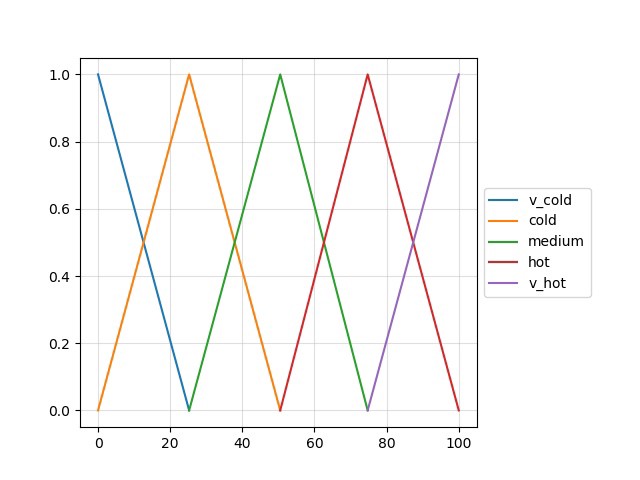


Рис. 4 Пример функций принадлежности

В приведенном выше примере нечеткие множества описывают температуру двигателя в диапазоне от очень холодного до очень горячего. Значение µ — это количество членов множества. Можно заметить, например, что при температуре 80 градусов двигатель можно охарактеризовать как очень горячий (v\_hot) с коэффициентом 0,2, а просто горячий (hot) с коэффициентом 0,8.

В предыдущем разделе обсуждались операторы объединения, пересечения и отрицания четких множеств, поскольку они обеспечивают способ выражения конъюнкции и дизъюнкции (и / или), которые имеют решающее значение для получения результата.

Наиболее распространенный метод вычисления объединения двух нечетких множеств - применение оператора максимума или минимума к множествам. Существуют и другие методы, включая использование оператора произведения для двух множеств.

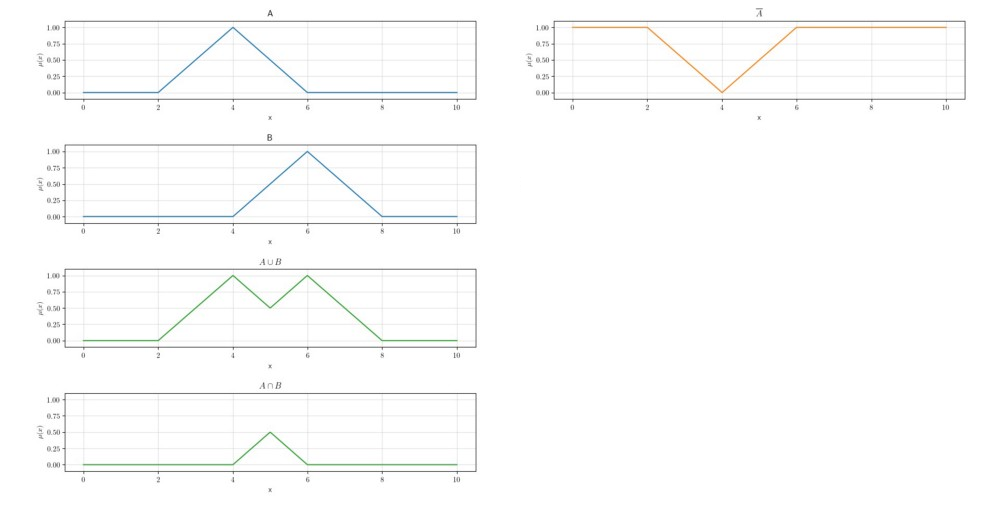


Рис. 5 Операции с множествами

## Системы нечеткого вывода:

Нечеткая система — это хранилище нечетких экспертных знаний, при помощи которых можно типизировать данные в расплывчатых терминах вместо точной булевой логики. Экспертные знания — это набор нечетких функций принадлежности и множество нечетких правил, известных как база правил, имеющее форму:

**IF** (выполнены условия) **THEN** (предполагаются последствия)

Базовая конфигурация нечеткой системы показана ниже:

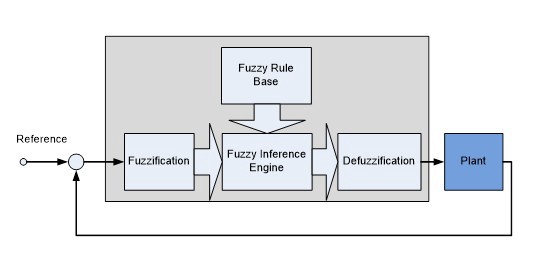


Рис. 6 Конфигурация нечеткой системы

Типичную нечеткую систему можно разделить на четыре основные части, а именно фаззификатор (Fuzzification), базу знаний (Fuzzy Rule Base), механизм вывода (Fuzzy Inference Engine) и дефаззификатор (Defuzzification);

Фаззификатор сопоставляет реальный четкий ввод с нечеткой функцией, тем самым определяя «степень принадлежности» ввода нечеткой концепции. Значения входных переменных преобразуются в диапазон значений, соответствующий вселенной дискурса (universe of discourse). Диапазон и разрешение входных нечетких множеств, и их влияние на процесс фаззификации рассматриваются как факторы, влияющие на общую производительность.

База знаний включает в себя знания о предметной области и соответствующие соотношения с данными. Её можно разделить на базу данных определений и базу правил. Интуитивно понятно, что база знаний — это основной элемент нечеткой системы, поскольку он будет содержать всю информацию, необходимую для выполнения задач. Для формирования базы знаний многие используют искусственный интеллект, например, генетические алгоритмы или нейронные сети.

Механизм вывода обеспечивает логику принятия решений. Он выводит нечеткие управляющие воздействия, используя нечеткие импликации и нечеткие правила вывода. Во многих аспектах это можно рассматривать как имитацию принятия решений человеком.

Процесс дефаззификации преобразует нечеткие контрольные значения в четкие величины, то есть связывает одну точку с нечетким множеством, учитывая, что точка принадлежит опоре нечеткого множества. Существует множество техник дефаззификации, наиболее известными из которых являются определение центра площади или центра тяжести.

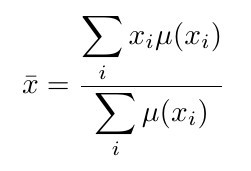


Рис. 7 Дефаззификация методом центра тяжести

Другие методы дефаззификации включают: первое из максимума и среднее из максимумов.

В логическом выводе Мамдани антецеденты и следствия нечеткого правила являются нечеткими множествами. Вывод основан на Generalized Modus Ponens, который утверждает, что степень истинности консеквента нечеткого правила — это степень истинности антецедента. В случае, когда присутствует более чем одно предшествующее предложение, отдельные степени членства объединяются с использованием оператора min t-norm. Если нечеткая система содержит несколько правил, их выходные данные объединяются с использованием оператора max s-norm. Дефаззификация необходима для того, чтобы последующее действие можно было выразить в терминах четкого значения. Графическое представление этого процесса показано ниже.

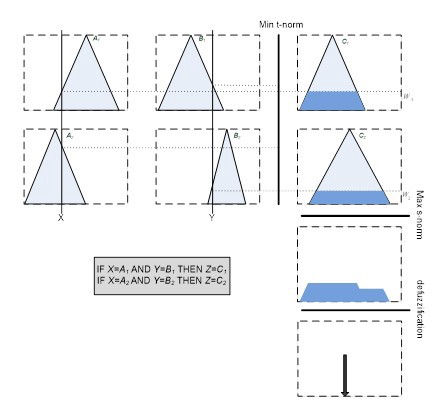


Рис. 8 Графическое представление дефаззификации

В модели вывода Такаги-Сугено консеквенты — это функции, которые преобразуют четкие входные значения в четкие выходные данные правила. Следовательно, нечеткие правила имеют вид:

IF x IS X and y IS Y THEN z=f(x,y)

где f обычно является линейной функцией от X и Y. В отличие от нечетких систем Мамдани, правила не объединяются с использованием max -оператора, а объединяются путем нахождения средневзвешенного значения, где вес данного правила является степенью принадлежности своего антецедента. Поэтому системы Такаги-Сугено не требуют дефаззификации.

# **Архитектура системы:**

На схеме ниже показана структура приложения. Дизайн основан на нескольких соображениях о системах нечеткого вывода, некоторые из которых:

* Система нечеткого вывода потребует входных и выходных переменных и набора нечетких правил.
* Как входные, так и выходные переменные будут содержать набор нечетких множеств, если система нечеткого вывода относится к типу Мамдани.
* Входные и выходные переменные очень похожи, но используются нечеткими правилами по-разному. Во время выполнения входные переменные используют входные значения в систему для нечеткого определения своих наборов, то есть они определяют степень принадлежности этого входного значения ко всем нечетким наборам переменной. Каждое правило в некоторой степени влияет на выходные переменные; совокупность этих правил будет определять производительность системы.
* Нечеткие правила имеют структуру

if {причина} then {следствие}

Поэтому правило будет содержать несколько причин и несколько следствий. Пункты будут иметь форму:

{имя переменной} is {имя множества}

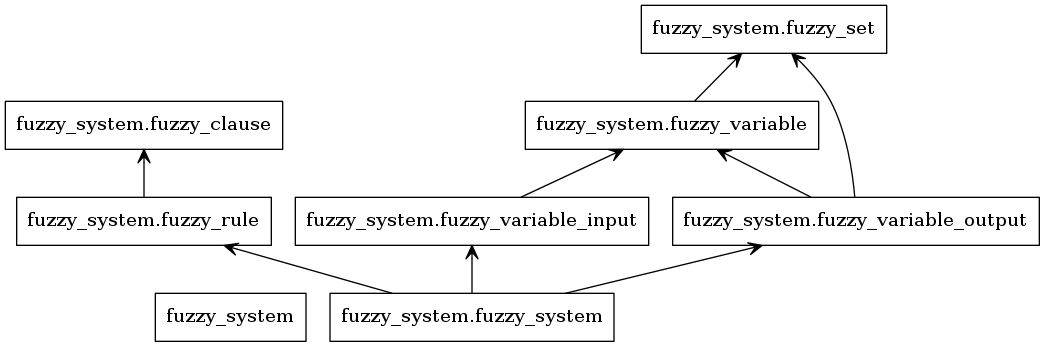


Рис. 9 Структура приложения.

## **Реализация классов**

### Класс FuzzySet

Для класса FuzzySet необходимы следующие параметры:

* name - название множества
* minimum value - минимальное значение множества
* maximum value - максимальное значение множества
* resolution - количество шагов между минимальным и максимальным значением

Таким образом, можно представить нечеткий набор, используя два numpy массива: один, который будет содержать значения домена, а другой, который будет содержать значения степени принадлежности. Если известны все переменные, то промежуточные значения можно вычислить. В данном проекте в качестве примера был выбран numpy массив, чтобы код был более читабельным и простым.

def create\_triangular(cls, name, domain\_min, domain\_max, res, a, b, c):

t1fs = cls(name, domain\_min, domain\_max, res)

a = t1fs.\_adjust\_domain\_val(a)

b = t1fs.\_adjust\_domain\_val(b)

c = t1fs.\_adjust\_domain\_val(c)

t1fs.\_dom = np.round(np.maximum(np.minimum((t1fs.\_domain-a)/(b-a), (c-t1fs.\_domain)/(c-b)), 0), t1fs.\_precision)

Рис. 10 Инициализация FuzzySet

В контексте нечеткой переменной все наборы будут иметь одинаковые минимальное, максимальное значения и значения разрешения.

Поскольку мы имеем дело с дискретным доменом, необходимо будет скорректировать любое значение, для соответствия множеству или взять степень принадлежности до ближайшего значения в массиве домена.

def \_adjust\_domain\_val(self, x\_val):

return self.\_domain[np.abs(self.\_domain-x\_val).argmin()]

Рис. 11 Корректировка значений домена

Класс содержит методы, с помощью которых можно построить множество заданной формы с заданным числом параметров. В случае треугольного множества, например, нужны три параметра: два из которых, определяют размеры множеств, и один для вершины. Используя эти три параметра, можно построить треугольный набор, как показано на рисунке ниже.

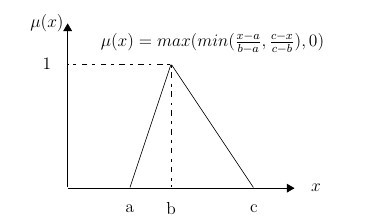


Рис. 12 Представление треугольного множества

Поскольку множества основаны на массивах numpy, выражение выше может быть преобразовано в код, представленный ниже. Множества, имеющие разную форму, построены одним и тем же методом.

def create\_triangular(cls, name, domain\_min, domain\_max, res, a, b, c):

t1fs = cls(name, domain\_min, domain\_max, res)

a = t1fs.\_adjust\_domain\_val(a)

b = t1fs.\_adjust\_domain\_val(b)

c = t1fs.\_adjust\_domain\_val(c)

t1fs.\_dom = np.round(np.maximum(np.minimum((t1fs.\_domain-a)/(b-a), (c-t1fs.\_domain)/(c-b)), 0), t1fs.\_precision)

Рис. 13 Создание треугольного множества

Класс FuzzySet также содержит операторы объединения, пересечения и отрицания, необходимые для выполнения логического вывода. Все методы операторов возвращают новый нечеткий набор с результатом выполненной операции.

def union(self, f\_set):

result = FuzzySet(f'({self.\_name}) union ({f\_set.\_name})', self.\_domain\_min, self.\_domain\_max, self.\_res)

result.\_dom = np.maximum(self.\_dom, f\_set.\_dom)

return result

Рис 14. Объединение двух объектов

Наконец, мы реализовали возможность получения четких результатов из нечеткого набора с помощью метода центра тяжести, о котором подробно говорилось на этапе разбора теории. Важно отметить, что в литературе доступно большое количество методов дефаззификации. Тем не менее, поскольку метод центра тяжести чрезвычайно популярен, он используется в этой реализации.

def cog\_defuzzify(self):

num = np.sum(np.multiply(self.\_dom, self.\_domain))

den = np.sum(self.\_dom)

return num/den

Рис. 15 Дефаззификация методом центра тяжести

### Классы Fuzzy Variable (нечётких переменных)

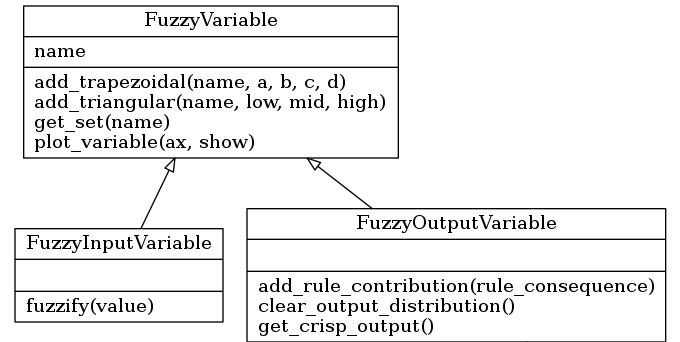


Рис. 16 Классы Fuzzy Variable

Как обсуждалось ранее, переменные могут быть входными или выходными, при этом различие влияет на расчет нечеткого вывода. FuzzyVariable — это коллекция множеств, которая хранится в словаре Python, имеющем имя набора в качестве ключа.

Для входных переменных фаззификация выполняется путем извлечения степени принадлежности всех наборов в переменной для данного значения домена. Степень принадлежности сохраняется в наборе, поскольку она будет требоваться правилами при их оценке.

def fuzzify(self, val):

# get dom for each set and store it -

# it will be required for each rule

for set\_name, f\_set in self.\_sets.items():

f\_set.last\_dom\_value = f\_set[val]

Рис. 17 Фаззификация входной переменной

Выходные переменные в конечном итоге дадут результат итерации нечеткого вывода. Это означает, что для систем типа Мамдани, которые мы тут используем, выходные переменные будут содержать объединение нечетких результатов всех правил и впоследствии будут дефаззифицировать этот результат, чтобы получить четкое значение, которое можно использовать в реальных приложениях.

Следовательно, выходные переменные потребуют дополнительного атрибута FuzzySet, который будет содержать выходное распределение для этой переменной, где будет результат, полученный после применения каждого правила и добавленный с помощью оператора объединения множества. Затем результат дефаззификации можно получить, вызвав метод центра тяжести для выходного набора распределения.

class FuzzyOutputVariable(FuzzyVariable):

def \_\_init\_\_(self, name, min\_val, max\_val, res):

super().\_\_init\_\_(name, min\_val, max\_val, res)

self.\_output\_distribution = FuzzySet(name, min\_val, max\_val, res)

def add\_rule\_contribution(self, rule\_consequence):

self.\_output\_distribution = self.\_output\_distribution.union(rule\_consequence)

def get\_crisp\_output(self):

return self.\_output\_distribution.cog\_defuzzify()

Рис. 18 Распределение нечёткого вывода и метод дефаззификации

### Класс FuzzyRules (нечёткие правила)

Класс нечёткого выражения требует две переменные: нечёткое значение и нечёткое множество, таким образом утверждение

variable is set

может быть создано. Выражения используются для реализации операторов, которые можно связать вместе, чтобы сформировать предшествующую и последующую части правила.

При использовании в качестве предшествующего предложения FuzzyClause возвращает последнее значение степени принадлежности множеству, которое вычисляется на этапе фаззификации.

Правило будет объединять значения степени принадлежности из различных предшествующих предложений с помощью оператора min, получая правило активации, которое затем используется вместе с последующими предложениями для получения вклада правила в выходные переменные. Эта операция состоит из двух этапов:

* Значение активации комбинируется с последующим FuzzySet множеством с использованием оператора min, который будет действовать как порог для значений степени принадлежности FuzzySet.
* Результирующее множество FuzzySet объединяется с множествами FuzzySet, полученными из других правил, с помощью оператора объединения, получая выходное распределение для этой переменной.

# execution methods for a FuzzyClause

# that comtains a FuzzyVariable; \_variable

# and a FuzzySet; \_set

def evaluate\_antecedent(self):

return self.\_set.last\_dom\_value

def evaluate\_consequent(self, activation):

self.\_variable.add\_rule\_contribution(self.\_set.min\_scalar(activation))

Рис. 19 Методы обработки предыдущего или последующего нечёткого предложения

Поэтому классу FuzzyRule потребуются два атрибута:

* список, содержащий предшествующие предложения
* список, содержащий следующие предложения

Во время выполнения FuzzyRule выполняется описанная выше процедура. FuzzyRule координирует все задачи, используя при необходимости все различные FuzzyClauses.

def evaluate(self):

# rule activation initialize to 1 as min operator will be performed

rule\_activation = 1

# execute all antecedent clauses, keeping the minimum of the returned doms to determine the activation

for ante\_clause in self.\_antecedent:

rule\_activation = min(ante\_clause.evaluate\_antecedent(), rule\_activation)

# execute consequent clauses, each output variable will update its output\_distribution set

for consequent\_clause in self.\_consequent:

consequent\_clause.evaluate\_consequent(rule\_activation)

Рис. 20 Выполнение правила

### Fuzzy System Class.

На самом верхнем уровне этой архитектуры у нас есть класс FuzzySystem, который координирует все действия между FuzzyVariables и FuzzyRules. Следовательно, класс содержит входные и выходные переменные, которые хранятся в словарях Python с использованием имен переменных в качестве ключей и списка правил.

Одна из проблем, представленных на этом этапе, — это метод, который пользователь будет использовать для добавления правил, который в идеале должен абстрагироваться от деталей реализации классов FuzzyClause. Реализованный метод состоит из предоставления двух словарей Python, которые будут содержать предшествующие и последующие предложения правила в следующем формате:

variable name : set name

Более удобный для пользователя метод - предоставить правило в виде строки, а затем проанализировать эту строку для создания правила, но это казалось ненужными накладными расходами для демонстрационного приложения.

def add\_rule(self, antecedent\_clauses, consequent\_clauses):

'''

adds a new rule to the system.

TODO: add checks

Arguments:

-----------

antecedent\_clauses -- dict, having the form {variable\_name:set\_name, ...}

consequent\_clauses -- dict, having the form {variable\_name:set\_name, ...}

'''

# create a new rule

# new\_rule = FuzzyRule(antecedent\_clauses, consequent\_clauses)

new\_rule = FuzzyRule()

for var\_name, set\_name in antecedent\_clauses.items():

# get variable by name

var = self.get\_input\_variable(var\_name)

# get set by name

f\_set = var.get\_set(set\_name)

# add clause

new\_rule.add\_antecedent\_clause(var, f\_set)

for var\_name, set\_name in consequent\_clauses.items():

var = self.get\_output\_variable(var\_name)

f\_set = var.get\_set(set\_name)

new\_rule.add\_consequent\_clause(var, f\_set)

# add the new rule

self.\_rules.append(new\_rule)

Рис. 21 Создание правила

### Добавление нового правила в FuzzySystem

Выполнение процесса вывода может быть достигнуто с помощью нескольких строк кода с данной структурой, где выполняются следующие шаги;

* Наборы выходных распределений всех выходных переменных очищаются.
* Входные значения в систему передаются в соответствующие входные переменные, так что каждый набор в переменной может определять свою степень принадлежности для этого входного значения.
* Происходит выполнение нечетких правил, что означает, что наборы выходных распределений всех выходных переменных теперь будут содержать объединение вкладов каждого правила.

# clear the fuzzy consequences as we are evaluating a new set of inputs.

# can be optimized by comparing if the inputs have changes from the previous

# iteration.

self.\_clear\_output\_distributions()

# Fuzzify the inputs. The degree of membership will be stored in

# each set

for input\_name, input\_value in input\_values.items():

self.\_input\_variables[input\_name].fuzzify(input\_value)

# evaluate rules

for rule in self.\_rules:

rule.evaluate()

# finally, defuzzify all output distributions to get the crisp outputs

output = {}

for output\_var\_name, output\_var in self.\_output\_variables.items():

output[output\_var\_name] = output\_var.get\_crisp\_output()

return output

Рис. 22 Вывод результата

Реализованная здесь система нечеткого вывода содержит дополнительные функции для построения нечетких множеств и переменных, а также для получения информации о выполнении шага вывода.

# **Пример использования:**

В качестве примера реализовано исследование скорости вращения вентилятора от влажности и температуры помещения.

Нечеткая система начинается с рассмотрения входных и выходных переменных и разработки нечетких множеств для объяснения этой переменной.

Для переменных потребуется нижний и верхний предел и, поскольку мы будем иметь дело с дискретными нечеткими множествами, диапазон значений. Следовательно, определение переменной будет выглядеть следующим образом

temp = FuzzyInputVariable('Temperature', 10, 40, 100)

где переменная «Temperature» находится в диапазоне от 10 до 40 градусов и дискретизируется в 100 ячейках.

Нечеткие множества, определенные для переменной, потребуют разных параметров в зависимости от их формы. В случае треугольных множеств, необходимы три параметра: два для нижних значений, имеющих степень принадлежности 0, и один для вершины, имеющей степень принадлежности 1. Определение треугольного множества для переменной «Temperature», будет выглядеть следующим образом;

temp.add\_triangular('Cold', 10, 10, 25)

где набор под названием «Cold» имеет крайние значения 10 и 25 и вершину 10 градусов. В нашей системе мы рассмотрели две входные переменные, «Temperature» и «Humidity», и одну выходную переменную «Speed». Каждая переменная описывается тремя нечеткими множествами. Определение выходной переменной «Speed» выглядит следующим образом:

motor\_speed = FuzzyOutputVariable('Speed', 0, 100, 100) motor\_speed.add\_triangular('Slow', 0, 0, 50) motor\_speed.add\_triangular('Moderate', 10, 50, 90) motor\_speed.add\_triangular('Fast', 50, 100, 100)

Как мы видели ранее, нечеткая система — это объект, который будет содержать эти переменные и нечеткие правила. Следовательно, переменные необходимо добавить в систему следующим образом:

system = FuzzySystem()

system.add\_input\_variable(temp)

system.add\_input\_variable(humidity)

system.add\_output\_variable(motor\_speed)

## Нечёткие правила

Нечеткая система выполняет нечеткие правила для работы с выражением:

if x1 is S and x2 is M then y is S

где часть If содержит несколько предшествующих предложений, а раздел then будет включать несколько последующих предложений. Для простоты предположим, что правила, требующие предшествующего предложения от каждой входной переменной, связаны только с помощью оператора «и». Операторы могут быть связаны с помощью «или», а также могут содержать такой оператор как «not».

Самый простой способ добавить нечеткое правило в нашу систему - предоставить список предшествующих и последующих предложений. Один из способов сделать это - использовать словарь Python, содержащий

Variable:Set

записи для наборов предложений. Следовательно, приведенное выше правило может быть реализовано следующим образом:

system.add\_rule(

{ 'Temperature':'Cold',

'Humidity':'Wet' },

{ 'Speed':'Slow'})

Выполнение системы включает ввод значений для всех входных переменных и получение значений для выходных значений взамен. Опять же, это достигается за счет использования словарей, которые используют имена переменных в качестве ключей.

output = system.evaluate\_output({

'Temperature':18,

'Humidity':60 })

Система вернет словарь, содержащий имена выходных переменных в качестве ключей и дефаззифицированный результат в виде значений.

{'Speed': 37.24647662394699}