**Министерство образования и науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(ВлГУ)**

Кафедра «Физика и прикладная математика»

**Лабораторная работа №1**

**по дисциплине «Неклассическая логика»**

**тема «Построение функций принадлежности»**

Выполнил:

ст. гр. ИТс-112

Власов И. Д.

Принял:

Абрахин С.И.

Владимир, 2014

Цель работы:

1. Изучить прямые методы построения функций принадлежности для одного эксперта.

2. Изучить и реализовать способ представления нечетких множеств в памяти компьютера (посредством связи между нечетким множеством универсального множества U и определенным образом устроенным семейством обычных его множеств).

Задача:

Разработать объект (класс) на любом объектно-ориентированном языке программирования для представления нечетких множеств (нечетких чисел), функция принадлежности которых строится на основе прямых методов для одного эксперта приведенных в табл. 1 теоретической части. Реализовать методы:

* разложение нечеткого множества (нечеткого числа), с видом функции принадлежности в соответствии с номером в табл. 1 на семейство обычных множеств, где число N уровней разложения указывается пользователем, с сохранением этого разложения внутри объекта, как динамической структуры, каждый элемент которой включает в себя тройку чисел { ai , Ai , Ai }, где ai – значение i-уровня разложения, Ai – нижняя граница обычного множества разложения, Ai – верхняя граница обычного множества разложения. (С точки зрения программирования этот метод будет являться конструктором разрабатываемого класса.)
* графическое отображение функции принадлежности.

Краткая теоретическая часть:

Основные группы методов построения функций принадлежности

В основании теории из любой области естествознания лежит очень важное, основополагающее для ее построения понятие элементарного объекта. Например, для механики – это материальная точка, для электродинамики – это вектор напряженности поля, для квантовой теории – понятие состояния. Для теории нечетких множеств основополагающим понятием является понятие нечеткого множества, которое характеризуется функцией принадлежности. Л.Заде предложил оценивать степень принадлежности числами из интервала [0,1].

Существует ряд методов построения по экспертным оценкам функции принадлежности нечеткого множества. Можно выделить две группы методов: прямые и косвенные методы.

Прямые методы определяются тем, что эксперт непосредственно задает правила определения значений функции принадлежности. В косвенных методах значения функции принадлежности выбираются таким образом, чтобы удовлетворить заранее сформулированным условиям. Экспертная информация является только исходной информацией для дальнейшей обработки. Дополнительные условия могут налагаться как на вид получаемой информации, так и на процедуру обработки.

Итак, нами выделены две основные группы методов построения функции принадлежности: прямые и косвенные. Однако функция принадлежности может отражать, как мнение группы экспертов, так и мнение одного (уникального) эксперта, следовательно, возможны, по крайней мере, четыре группы методов: прямые и косвенные для одного эксперта, прямые и косвенные для группы экспертов.

Требования к функциям принадлежности

Относительно функции принадлежности можно выдвинуть следующие условия:

1. Функция принадлежности должна быть положительной.

2. Если это не оговаривается дополнительно, функция принадлежности должна быть нормальной. Если условие нормальности принято, то запрещается использование функций принадлежности, не удовлетворяющих условию нормальности.

3. Функция принадлежности может задаваться как на непрерывном, так и на дискретном носителе.

Рассмотрим некоторые из наиболее простых методов построения функций принадлежности прямых методов для одного эксперта.

Прямые методы для одного эксперта

Прямые методы для одного (уникального) эксперта состоят в непосредственном назначении степени принадлежности для исследуемых объектов или непосредственном назначении функции (правила), позволяющей вычислять значения.

При непосредственном назначении функции (правила), позволяющей вычислять значения используют типовые функции принадлежности. К настоящему времени накоплен достаточно широкий набор различных вариантов функций принадлежности для самых разнообразных нечетких утверждений (см. методичку).

Безусловно, выбор функции принадлежности и их параметров определяется в большей степени опытом, интуицией и другими субъективными факторами, лица принимающего решения.

Для вычисления параметров функции принадлежности при известном аналитическом представлении можно предложить простую методику, которая вытекает из рассмотрения функций принадлежности, приведенных в методичке В.Г. Чернова «Нечеткие множества в задачах»

Выполнение:

package classes;

import java.awt.Graphics;

import java.util.ArrayList;

import java.util.List;

public abstract class FuzzySet {

private static int countExpansionLeavels = 100;

private List<Expansion> expansion;

private List<Double> parameters;

public FuzzySet(List<Expansion> expansion) {

this.expansion = expansion;

}

public FuzzySet(double ... parameters) {

this.parameters = new ArrayList<Double>();

for(double d : parameters)

this.parameters.add(d);

this.expansion = new ArrayList<Expansion>();

double alphaLevel = 1.0 / countExpansionLeavels;

for(int i = 0; i < countExpansionLeavels - 1; i++) {

expansion.add(new Expansion(alphaLevel,

calculatingLowerLimit(alphaLevel),

calculatingUpperLimit(alphaLevel)));

alphaLevel += 1.0 / countExpansionLeavels;

}

}

public static int getCountExpansionLeavels() {

return countExpansionLeavels;

}

public static void setCountExpansionLeavels(int countExpansionLeavels) {

FuzzySet.countExpansionLeavels = countExpansionLeavels;

}

public List<Double> getParameters() {

return parameters;

}

public void setParameters(List<Double> parameters) {

this.parameters = parameters;

}

protected abstract double calculatingLowerLimit(double alphaLevel);

protected abstract double calculatingUpperLimit(double alphaLevel);

public void draw(Graphics g) {

double maxValue = Double.MIN\_VALUE;

double minVlaue = Double.MAX\_VALUE;

int scale = 250;

int scaleWeight = 450;

for(int i = 0; i < expansion.size(); i++) {

double lowerLimit = expansion.get(i).getLowerLimit();

if (lowerLimit != -1) {

if (lowerLimit > maxValue) maxValue = lowerLimit;

if (lowerLimit < minVlaue) minVlaue = lowerLimit;

}

double upperLimit = expansion.get(i).getUpperLimit();

if (upperLimit != -1) {

if (upperLimit > maxValue) maxValue = upperLimit;

if (upperLimit < minVlaue) minVlaue = upperLimit;

}

}

double chartWidth = maxValue - minVlaue;

int [] xArrayLower = new int [expansion.size()];

int [] xArrayUpper = new int [expansion.size()];

int [] yArray = new int [expansion.size()];

double tmpLowerLimit = 0;

double tmpUpperLimit = 0;

boolean drawLowerLimit = false;

boolean drawUpperLimit = false;

for (int i = 0; i < expansion.size(); i++) {

if (expansion.get(i).getLowerLimit() != -1) {

tmpLowerLimit = expansion.get(i).getLowerLimit();

drawLowerLimit = true;

}

if (expansion.get(i).getUpperLimit() != -1) {

tmpUpperLimit = expansion.get(i).getUpperLimit();

drawUpperLimit = true;

}

xArrayLower[i] = (int)((tmpLowerLimit - minVlaue) / chartWidth \* scaleWeight) + 100;

xArrayUpper[i] = (int)((tmpUpperLimit - minVlaue) / chartWidth \* scaleWeight) + 100;

yArray[i] = (int)((1 - expansion.get(i).getAlphaLevel()) \* scale) + 100;

}

if (drawLowerLimit)

g.drawPolyline(xArrayLower, yArray, expansion.size());

if (drawUpperLimit)

g.drawPolyline(xArrayUpper, yArray, expansion.size());

}

public String toString() {

return this.getClass().getName() +

" (" + parameters.get(0) +

", " + parameters.get(1) +

", " + parameters.get(2) +

", " + parameters.get(3) + ")";

}

public FuzzySet add(FuzzySet fs) {

List<Expansion> expansions = new ArrayList<Expansion>();

for(int i = 0; i < this.expansion.size() - 1; i++) {

Expansion exp = new Expansion(this.expansion.get(i).getAlphaLevel(),

this.expansion.get(i).getLowerLimit() + fs.expansion.get(i).getLowerLimit(),

this.expansion.get(i).getUpperLimit() + fs.expansion.get(i).getUpperLimit());

expansions.add(exp);

}

return new OtherFuzzySet(expansions);

}

public FuzzySet sub(FuzzySet fs) {

List<Expansion> expansions = new ArrayList<Expansion>();

for(int i = 0; i < this.expansion.size() - 1; i++) {

Expansion exp = new Expansion(this.expansion.get(i).getAlphaLevel(),

this.expansion.get(i).getLowerLimit() - fs.expansion.get(i).getUpperLimit(),

this.expansion.get(i).getUpperLimit() - fs.expansion.get(i).getLowerLimit());

expansions.add(exp);

}

return new OtherFuzzySet(expansions);

}

public FuzzySet mul(FuzzySet fs) {

List<Expansion> expansions = new ArrayList<Expansion>();

for(int i = 0; i < this.expansion.size() - 1; i++) {

Expansion exp = new Expansion(this.expansion.get(i).getAlphaLevel(),

this.expansion.get(i).getLowerLimit() \* fs.expansion.get(i).getLowerLimit(),

this.expansion.get(i).getUpperLimit() \* fs.expansion.get(i).getUpperLimit());

expansions.add(exp);

}

return new OtherFuzzySet(expansions);

}

public FuzzySet div(FuzzySet fs) {

List<Expansion> expansions = new ArrayList<Expansion>();

for(int i = 0; i < this.expansion.size() - 1; i++) {

Expansion exp = new Expansion(this.expansion.get(i).getAlphaLevel(),

this.expansion.get(i).getLowerLimit() / fs.expansion.get(i).getUpperLimit(),

this.expansion.get(i).getUpperLimit() / fs.expansion.get(i).getLowerLimit());

expansions.add(exp);

}

return new OtherFuzzySet(expansions);

}

}

package classes;

public class FirstFunction extends FuzzySet {

public FirstFunction(double ... paraparameters) {

super(paraparameters);

}

double calcFirstFunction(double alphaLevel, double a, double b) {

if (alphaLevel < 0.5)

return a + Math.sqrt(alphaLevel \* Math.pow(b - a, 2) / 2);

else if (alphaLevel == 0.5)

return (a + b) / 2;

else

return b - Math.sqrt((1 - alphaLevel) \* Math.pow(b - a, 2) / 2);

}

@Override

protected double calculatingLowerLimit(double alphaLevel) {

double a = getParameters().get(0);

double b = getParameters().get(1);

return calcFirstFunction(alphaLevel, a, b);

}

@Override

protected double calculatingUpperLimit(double alphaLevel) {

return -1;

}

}

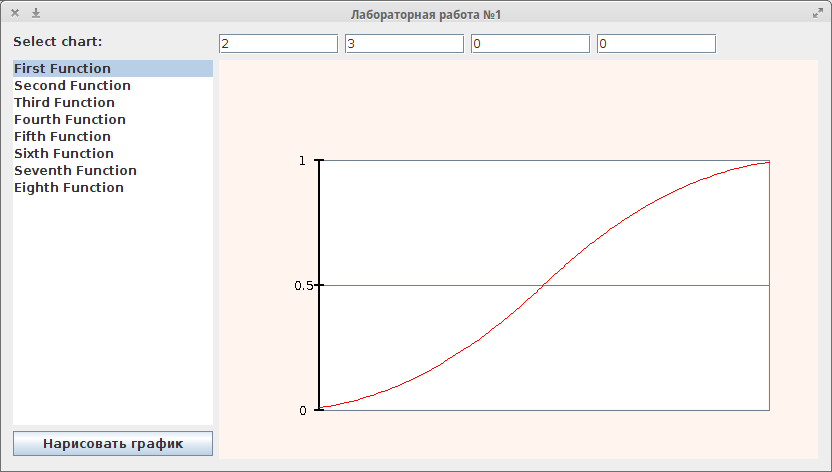


Рисунок 1 – Выполнение программы

Вывод:

1. Изучил прямые методы построения функций принадлежности для одного эксперта.

2. Изучил и реализовать способ представления нечетких множеств в памяти компьютера (посредством связи между нечетким множеством универсального множества U и определенным образом устроенным семейством обычных его множеств).