Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Институт информационных технологий и анализа данных

наименование института

ОТЧЕТ  
к лабораторной работе №2 по дисциплине

Интеллектуальные системы и технологии

«Разработка генетического алгоритма»

Наименование темы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы: | ИСМб-19-1 |  |  |  | Вовиков Д.Е. |
|  | Шифр группы |  | подпись |  | Фамилия ИО |
| Проверил |  |  |  |  | Массель А.Г. |
|  | должность |  | подпись |  | Фамилия ИО |
| Содержание отчета на: | |  | 23 | стр. | |

Иркутск 2022 г

Содержание

[1 Введение 3](#_Toc118559253)

[2 Проектирование классов 4](#_Toc118559254)

[2.1 Словесное описание классов 4](#_Toc118559255)

[2.2 Класс Controller 4](#_Toc118559256)

[2.3 Класс Individidual 4](#_Toc118559257)

[2.4 Класс Population 5](#_Toc118559258)

[2.5 Класс WorkWithNumbers 5](#_Toc118559259)

[3 Таблица тестов 6](#_Toc118559260)

[4 Результаты тестирования 7](#_Toc118559261)

[5 Исходный код 9](#_Toc118559262)

[5.1 Класс Controller 9](#_Toc118559263)

[5.2 Класс Individidual 10](#_Toc118559264)

[5.3 Класс Population 14](#_Toc118559265)

[5.4 Класс WorkWithNumbers 19](#_Toc118559266)

1 Введение

**Цель работы**

Создать на языке высокого уровня простейший генетический алгоритм, способный подстраиваться под условия среды.

**Варианты индивидуальных заданий**

Y=cos(x)

2 Проектирование классов

2.1 Словесное описание классов

Программа состоит из 4 классов. Описание классов:

1. Controller – логика графического интерфейса.
2. Individidual –скрещивание и мутация, вычисление шага для мутации.
3. Population – класс запускает генетический алгоритм, в цикле размножение, мутация и удаление особей.
4. WorkWithNumbers – класс для задачи функции и расчета эвклидова расстояния.

2.2 Класс Controller

Таблица 2.2 – Таблица методов класса Controller

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| start | void | Нет | public | Stage stage | Старт программы |

2.3 Класс Individidual

Таблица 2.3 – Таблица методов класса Individidual

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| Individidual |  | Нет | public | int numberOfSigns | Конструктор |
| setSigns | void | Нет | public | double[] signs | Получение размера |
| crossoverByTwoPoints | Individidual[] | Да | public | Individidual parent1, Individidual parent2 | Скрещивание |
| mutateInStepIncrease | void | Нет | public | double[] fitnessFunc | Мутация особи при помощи прибавления к признаку числа |
| getStep | double | Нет | public | double gen, double fitness | Функция для вычисления числа (шага), на которое нужно увеличить признак, чтобы приблизить его к искомой функции |
| getNumberOfSigns | int | Нет | public | - | Получение номера признака |
| getSigns | double[] | Нет | public | - | Возвращение размера |
| countEuclidDistanceForSignsWithRequiredFunc | void | Нет | public | double[] requiredFunction | Считаем эвклидово расстояние между признаками и искомой функцией |
| toString | String | Нет | public | - | Вывод |
| getEuclidDist | double | Нет | public | - | Возвращение эвклидова расстояния |
| compareTo | int | Нет | public | Individidual ind | Сравнивать две особи между собой |

2.4 Класс Population

Таблица 2.4 – Таблица методов класса Population

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| Population |  | Нет | public | - | Конструктор |
| startGeneticAlg | void | Нет | public | - | Запускает генетический алгоритм. Здесь в цикле размножение, мутация и удаление особей.  Остановка цикла в случае, если эвклидово рассояние <= заданного |
| createPopulation | void | Нет | public | - | Создаем начальную популяцию. Создаем 20 особей и заполняем признаки особей рандомными числами при помощи getRandomArray() |
| crossoverByTwoPointOperator | void | Нет | public | - | Размножение (Двухточечный ОПЕРАТОР КРОССИНГОВЕРА)  В каждой хромосоме определяются две точки оператора кроссинговера,  и хромосомы обмениваются участками, расположенными между двумя точками оператора кроссинговера.  Точки оператора кроссинговера в двухточечном операторе кроссинговера также определяются случайно. |
| addInd | void | Нет | public | Individidual ind | Метод для добавления особи в популяцию |
| mutatePopulation | void | Нет | public | - | Проводим мутацию для каждой особи |
| delete | void | Нет | public | - | Метод для поиска и удаления максимально отдаленных особей. Сначала сортируем по эвклидову расстоянию, а затем удаляем вторую часть с наибольшими значениями |
| sortByEuclid | void | Нет | public | - | Функция для сортировки особей по возрастанию по эвклидовому расстоянию между признаками особи и искомой функцией |
| getIndWithMinEuclidDist | Individidual | Нет | public | - | Выводит особь с минимальным эвклидовым расстоянием |

2.5 Класс WorkWithNumbers

Таблица 2.5 – Таблица методов класса WorkWithNumbers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя** | **Тип** | **Статический** | **Уровень доступа** | **Аргументы** | **Назначение** |
| getRandomFromTo | void | Да | public | int a, int b | Получить рандомное число от a до b |
| getEuclideanDistance | double | Да | public | double[] x, double[] y | Общая формула евклидова расстояния для n-мерного случая (n переменных) |
| getEuclideanDistance | double | Да | public | double x, double y | Общая формула евклидова расстояния для 2-мерного случая |
| getRandomArray | double[] | Да | public | int countOfElements, double rangeMin, double rangeMax | Создание массива double со случайными числами |
| getArrayFromTo | double[] | Да | public | double step, double start, int countOfElements | Создание массива с заданным шагом от точки start до точки end |
| getFunction | double[] | Да | public | double[] x | создаем искомую функцию |
| round | double | Да |  | double value, int precision | Округление |

3 Таблица тестов

Таблица 3.1 – таблица тестов

|  |  |  |
| --- | --- | --- |
| **№** | **Действие** | **Результат** |
| 1 | Задание эвклидова расстояния 100 | Поколение 1 |
| 2 | Задание эвклидова расстояния 0,5 | Поколение 1967 |
| 3 | Задание эвклидова расстояния 0,01 | Поколение 2923 |

4 Результаты тестирования

Результаты тестирования представлены на рисунках 1 – 3

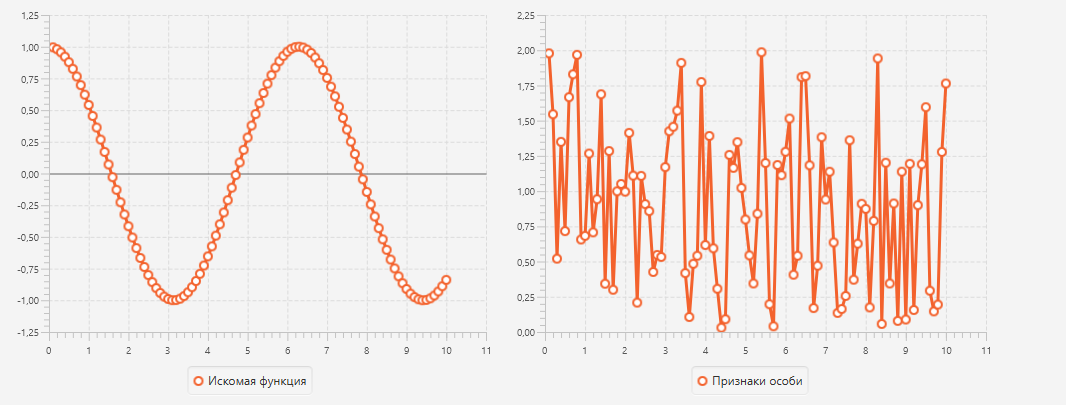


Рисунок 1 – Результат выполнения теста 1

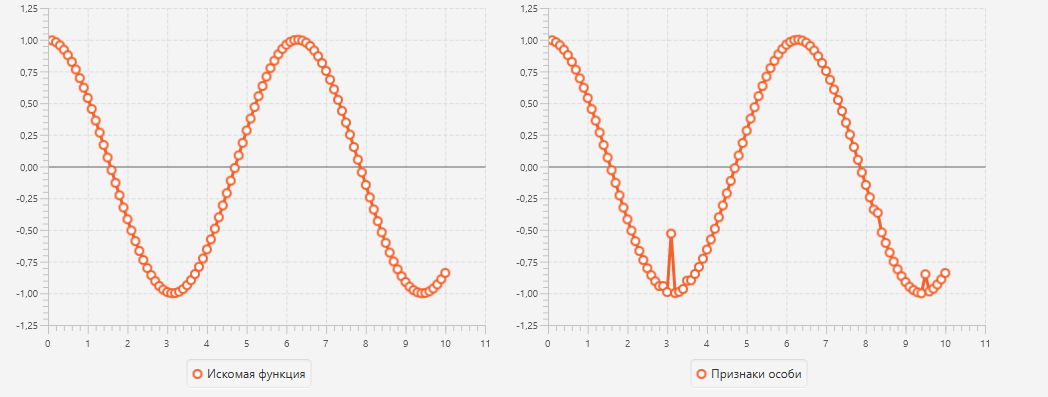


Рисунок 2 – Результат выполнения теста 2

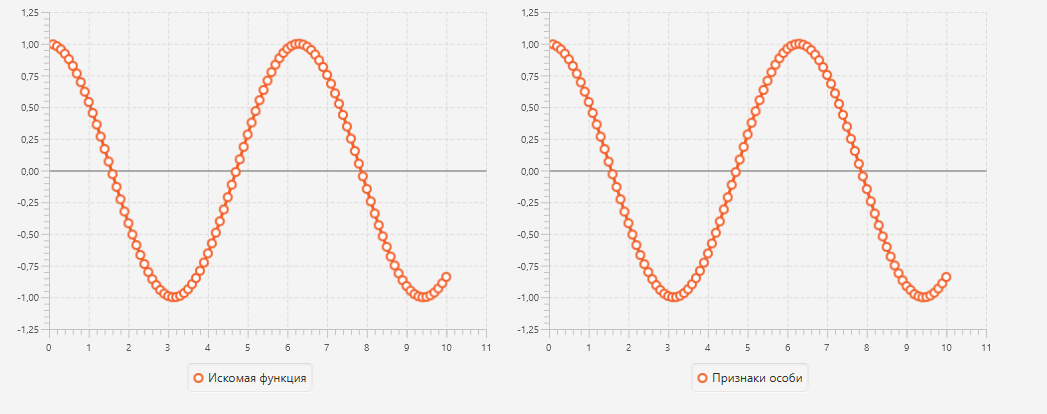


Рисунок 3 – Результат выполнения теста 3

5 Исходный код

5.1 Класс Controller

package com.company;

import javafx.application.Application;

import javafx.scene.layout.FlowPane;

import javafx.stage.Stage;

import javafx.scene.Scene;

import javafx.scene.chart.LineChart;

import javafx.scene.chart.NumberAxis;

import javafx.scene.chart.XYChart;

public class Controller extends Application {

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage stage) {

stage.setTitle("");

// определение осей

final NumberAxis xAxis = new NumberAxis();

final NumberAxis yAxis = new NumberAxis();

// создание графика

final LineChart<Number,Number> lineChart = new LineChart<Number,Number>(xAxis,yAxis);

// Делаем график искомой функции

XYChart.Series series1 = new XYChart.Series();

series1.setName("Искомая функция");

// создаем популяцию с 20ю особями, по 100 признаков у каждой

Population population = new Population(20, 100);

population.createPopulation(); // заполняем популяцию особями

double[] x = population.getxForRequiredFunc();

double[] y = population.getRequiredFunction();//ln

for (int i = 0; i < x.length; i++){

series1.getData().add(new XYChart.Data(x[i], y[i]));

}

// определение осей

final NumberAxis xAxis2 = new NumberAxis();

final NumberAxis yAxis2 = new NumberAxis();

// создание графика

final LineChart<Number,Number> lineChart2 = new LineChart<Number,Number>(xAxis2, yAxis2);

// выполняем генетический алгоритм

population.startGeneticAlg();

// Делаем график признаков особи с наименьшим эвклидовым расстоянием

XYChart.Series series2 = new XYChart.Series();

series2.setName("Признаки особи");

double[] y2 = population.getIndWithMinEuclidDist().getSigns();

double[] x2 = x;

// все х и у вбиваем в график поточечно

for (int i = 0; i < x.length; i++){

series2.getData().add(new XYChart.Data(x2[i], y2[i]));

}

FlowPane root = new FlowPane(lineChart, lineChart2);

Scene scene = new Scene(root);

lineChart.getData().add(series1);

lineChart2.getData().add(series2);

stage.setScene(scene);

stage.show();

}

}

5.2 Класс Individidual

package com.company;

import java.math.BigDecimal;

import java.util.Arrays;

public class Individidual implements Comparable<Individidual>{

int numberOfSigns; // количество признаков

double[] signs; // массив признаков для текущей особи

double euclidDist; // эвклидово расстояние между особью и искомой функцией

public Individidual(int numberOfSigns) {

this.numberOfSigns = numberOfSigns;

signs = new double[numberOfSigns];

}

public void setSigns(double[] signs) {

this.signs = signs;

}

/\*\*

\* СКРЕЩИВАНИЕ (двуточечный оператор кроссинговера)

\* В каждой хромосоме определяются две точки оператора кроссинговера,

\* хромосомы обмениваются участками, расположенными между двумя точками оператора кроссинговера.

\* Точки оператора кроссинговера в двухточечном операторе кроссинговера также определяются случайно.

\* @param parent1 - первый родитель

\* @param parent2 - второй родитель

\* @return individiduals - результат скрещивания (два потомка).

\*/

public static Individidual[] crossoverByTwoPoints(Individidual parent1, Individidual parent2){

int countOfSigns = parent1.getNumberOfSigns(); // количество признаков в особи

// точка разреза, после которой выполняется разрез хромосомы (получаем рандомно от 1 до последнего признака)

int idOfCuttingPoint1 = WorkWithNumbers.getRandomFromTo(1, countOfSigns);

int idOfCuttingPoint2 = WorkWithNumbers.getRandomFromTo(1, countOfSigns);

// определяем левую (меньшую) точку

int leftPoint, rightPoint;

if (idOfCuttingPoint1 < idOfCuttingPoint2){

leftPoint = idOfCuttingPoint1;

rightPoint = idOfCuttingPoint2;

} else if (idOfCuttingPoint1 == idOfCuttingPoint2){

leftPoint = idOfCuttingPoint1 - 1;

rightPoint = idOfCuttingPoint2;

} else {

leftPoint = idOfCuttingPoint2;

rightPoint = idOfCuttingPoint1;

}

// создаем два потомка

Individidual child1 = new Individidual(countOfSigns);

Individidual child2 = new Individidual(countOfSigns);

// до точки разрыва признаки оставляем те же

for (int i = 0; i < leftPoint; i++){

child1.signs[i] = parent1.signs[i];

child2.signs[i] = parent2.signs[i];

}

// После точки разрыва до следующей точки признаки меняем

for (int i = leftPoint; i < rightPoint; i++){

child1.signs[i] = parent2.signs[i];

child2.signs[i] = parent1.signs[i];

}

// После правой точки разрыва признаки оставляем те же

for (int i = rightPoint; i < countOfSigns; i++){

child1.signs[i] = parent1.signs[i];

child2.signs[i] = parent2.signs[i];

}

// Записываем детей в массив и возвращаем

return new Individidual[]{child1, child2};

}

/\*\*

\* Мутация особи при помощи прибавления к признаку числа, т.е.

\* увеличиваем/уменьшаем признак на шаг step, чтобы приблизить его к искомой функции.

\* @param fitnessFunc - искомая функция (с ней мы сравниваем признаки)

\*/

public void mutateInStepIncrease(double[] fitnessFunc) {

// для каждого признака особи:

for (int i = 0; i < numberOfSigns; ++i) {

double step = getStep(signs[i], fitnessFunc[i]); // вычисляем шаг, который будем прибавлять к признаку

signs[i] += step; // увеличиваем признак на найденный шаг

signs[i] = WorkWithNumbers.round(signs[i], 10); // округляем

}

}

/\*\*

\* Функция для вычисления числа (шага), на которое нужно увеличить признак, чтобы приблизить его к искомой функции.

\* Здесь определяем, в какую сторону будем двигать точку (признак) - вверх или вниз.

\* Если step отрицательный, то будем двигать вниз (уменьшать значение признака)

\* Если step положительный, то будем двигать вверх (увеличивать значение признака)

\* @param gen - признак, который будем сравнивать с точкой искомой функции

\* @param fitness - точка искомой функции

\* @return step - число, которое будем прибавлять к признаку для того, чтобы приблизить его к точке искомой функции

\*/

public double getStep(double gen, double fitness){

double euclid = WorkWithNumbers.getEuclideanDistance(gen, fitness);// эвклидово расст между признаками и искомой функцией

double step; // число, которое будем прибавлять к признаку для того, чтобы приблизить его к точке искомой функции

// сначала ставим 00.1, чтобы проверить: уменьшится ли эвклидово расстояние после того, как признак увеличится на это число

// это нужно для того, чтобы понять - в какую сторону изменять признак (уменьшать или увеличивать)

step = 0.001;

// считаем эвклид расст между точкой искомой функции и признаком, увеличенным на шаг step

double euclidWithStep = WorkWithNumbers.getEuclideanDistance(gen + step, fitness);

// сравниваем эвклидовые расстояние до и после увеличения на step

if (euclidWithStep < euclid){ // если увеличенное значение меньше исходного, тогда будем прибавлять (идем вверх по оси У)

step = 0.001;

} else if (euclid == euclidWithStep){ // если равны, то оставляем шаг 0, так как изменений нет

step = 0;

} else { // если увеличенное значение меньше исходного, тогда будем уменьшать (идем вниз по оси У)

step = -0.001;

}

return step;

}

public int getNumberOfSigns() {

return numberOfSigns;

}

public double[] getSigns() {

return signs;

}

/\*\*

\* Считаем эвклидово расстояние между признаками и искомой функцией

\* @param requiredFunction - искомая функция

\*/

public void countEuclidDistanceForSignsWithRequiredFunc(double[] requiredFunction) {

this.euclidDist = WorkWithNumbers.getEuclideanDistance(this.signs, requiredFunction);

}

@Override

public String toString() {

return "Individidual{" +

"signs=" + Arrays.toString(signs) +

", euclidDist=" + euclidDist +

'}';

}

public double getEuclidDist() {

return WorkWithNumbers.round(euclidDist, 10);

}

/\*\*

\* переопределенная функция от Comparable.

\* нужна для того, чтобы сравнивать две особи между собой. Сравниваем особи по значению эвклидова расстояния

\* @param ind - особь, с которой сравниваем

\* @return 1 - больше,

\* -1 - меньше,

\* 0 - равны

\*/

@Override

public int compareTo(Individidual ind) {

BigDecimal bd1 = BigDecimal.valueOf(this.getEuclidDist());

BigDecimal bd2 = BigDecimal.valueOf(ind.getEuclidDist());

return bd1.compareTo(bd2);

}

}

5.3 Класс Population

package com.company;

import java.util.\*;

public class Population {

List<Individidual> population = new ArrayList<>(); // список особей в популяции

int numberOfIndividual; // количество особей

int numberOfSigns; // количество признаков

private double[] requiredFunction; // искомая функция (Y)

private double[] xForRequiredFunc; // x для искомой функции

public Population(int numberOfIndividual, int numberOfSigns) {

this.numberOfIndividual = numberOfIndividual;

this.numberOfSigns = numberOfSigns;

// Находим искомую функцию

xForRequiredFunc = WorkWithNumbers.getArrayFromTo(0.1, 0.1, 100);

requiredFunction = WorkWithNumbers.getFunction(xForRequiredFunc);

}

public Population() {

}

/\*\*

\* Запускает генетический алгоритм. Здесь в цикле размножение, мутация и удаление особей.

\* Остановка цикла в случае, если эвклидово рассояние <= 0.5

\*/

public void startGeneticAlg(){

// Здесь в цикле будем размножать, уничтожать особи до тех пор, пока эвклидово расстояние не будет равно 0,5.

final double EUCLIDDIST = 0.01; // Остановить мутационный процесс при достижении эвклидова расстояния = 0,5.

double minEuclid; // посчитанное эвклидово расстояние для особей

Individidual indWithMinEuclidDist; // особь с минимальным эвклидовым расстоянием

int count = 0; // кол-во популяций, прошедших цикл

do {

// 3. Создать алгоритм, размножения и мутации, способный на каждом шаге удваивать популяцию.

crossoverByTwoPointOperator(); // двухточечный кроссовер

mutatePopulation();

// 4. Уничтожать половину удвоенной популяции, эвклидово расстояние признаков

// которых максимально удалено от искомой функции.

delete();

// тут просто выводим, чтобы отслеживать набор особей

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\* Популяция после удаления \*\*\*\*\*\*\*\*\*\*\*\*\*");

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

Individidual ind = population.get(i);

System.out.println(i + ") " + ind.toString() + " euclid=" + ind.getEuclidDist());

}

// Ищем минимальное эвклидовое расстояние

indWithMinEuclidDist = getIndWithMinEuclidDist(); // особь с min dist

minEuclid = indWithMinEuclidDist.getEuclidDist();

System.out.println("МИНИМАЛЬНОЕ ЭВКЛИДОВОЕ РАССТОЯНИЕ = " + minEuclid);

count++;

} while (minEuclid > EUCLIDDIST); // 5. Остановить мутационный процесс при достижении эвклидова расстояния = 0,5.

System.out.println("Количество поколений, прошедших через цикл: " + count);

}

/\*\*

\* Создаем начальную популяцию. Создаем 20 особей и заполняем признаки особей рандомными числами при помощи getRandomArray()

\*/

public void createPopulation(){

// создаем 20 особей, у каждой 100 признаков

for (int i = 0; i < numberOfIndividual; i++){

double[] signs = WorkWithNumbers.getRandomArray(numberOfSigns, 0.0, 2.0); // массив признаков для текущей особи

Individidual individidual = new Individidual(numberOfSigns); // создаем особь

individidual.setSigns(signs); // добавляем особи признаки

population.add(individidual); // добавляем особь в популяцию

}

}

/\*\*

\* Размножение (Двухточечный ОПЕРАТОР КРОССИНГОВЕРА)

\* В каждой хромосоме определяются две точки оператора кроссинговера,

\* и хромосомы обмениваются участками, расположенными между двумя точками оператора кроссинговера.

\* Точки оператора кроссинговера в двухточечном операторе кроссинговера также определяются случайно.

\*/

public void crossoverByTwoPointOperator(){

// 1. выбираются 2 особи случайно из текущей популяции

// 1.1 для этого перемешаем arraуlist

Collections.shuffle(population);

// 1.2 и будем брать особи по порядку и попарно скрещивать

Population resOfCrossover = new Population(); // здесь храним результат скрещивания, новые сооби

for (int i = 0; i < population.size() - 1; i += 2){

// берем родителей

Individidual parent1 = population.get(i);

Individidual parent2 = population.get(i + 1);

// скрещиваем и получаем еще две особи

Individidual[] children = Individidual.crossoverByTwoPoints(parent1, parent2);

// вносим особи в популяцию

resOfCrossover.addInd(children[0]);

resOfCrossover.addInd(children[1]);

}

// после того как посчитали новые особи добавляем их в общую популяцию:

for (Individidual ind:

resOfCrossover.getPopulation()) {

population.add(ind);

}

}

/\*\*

\* Метод для добавления особи в популяцию

\* @param ind - особь, которую нужно добавить

\*/

public void addInd(Individidual ind) {

population.add(ind);

}

/\*\*

\* Проводим мутацию для каждой особи

\*/

public void mutatePopulation(){

for (Individidual el:

population) {

el.mutateInStepIncrease(requiredFunction);

}

}

/\*\*

\* Метод для поиска и удаления максимально отдаленных особей. Сначала сортируем по эвклидову

\* расстоянию, а затем удаляем вторую часть с наибольшими значениями.

\*/

public void delete() {

// здесь сортируем популяцию по возрастанию эвклидового расстояния

sortByEuclid();

// удаляем половину (элементы, которые идут после среднего)

int mid = population.size() / 2; // индекс среднего элемента

System.out.println("mid=" + mid);

// формируем новую популяцию

// в нее включаем половину особей (тех, у кого эвклидово расстояние минимально)

ArrayList<Individidual> newPopulation = new ArrayList<>();

for (int i = 0; i < mid; i++){

newPopulation.add(population.get(i));

}

// теперь новая популяция заняла место старой:

population = newPopulation;

}

/\*\*

\* Функция для сортировки особей по возрастанию по эвклидовому расстоянию между

\* признаками особи и искомой функцией

\*/

public void sortByEuclid(){

// считаем эвклидово расстояние для всех особей

for (Individidual ind:

population) {

ind.countEuclidDistanceForSignsWithRequiredFunc(requiredFunction); // сравниваем с искомой функцией

}

// здесь сортируем популяцию по возрастанию

Collections.sort(population, new Comparator<Individidual>() {

@Override

public int compare(Individidual o1, Individidual o2) {

return o1.compareTo(o2);

}

});

}

@Override

public String toString() {

String formatted = "";

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

formatted += i + ") ";

formatted += population.get(i).toString();

formatted += "\n";

}

return formatted;

}

/\*\*

\* Выводит особь с минимальным эвклидовым расстоянием.

\* @return особь с наименьшим значением эвклидового расстояния среди всей популяции

\*/

public Individidual getIndWithMinEuclidDist(){

// минимум = первому элементу

Individidual indWithMinEuclid = population.get(0);

// смотрим по всей популяции

for (Individidual ind:

population) {

double minEuclidDist = indWithMinEuclid.getEuclidDist(); // смотрим минимум на данный момент

if (ind.getEuclidDist() < minEuclidDist){ // если он меньше то это новый минимум

indWithMinEuclid = ind;

}

}

return indWithMinEuclid;

}

public List<Individidual> getPopulation() {

return population;

}

public double[] getRequiredFunction() {

return requiredFunction;

}

public double[] getxForRequiredFunc() {

return xForRequiredFunc;

}

}

5.4 Класс WorkWithNumbers

package com.company;

import java.util.Random;

public class WorkWithNumbers {

/\*\*

\* Получить рандомное число от a до b

\* @param a Начальное значение диапазона - "от"

\* @param b Конечное значение диапазона - "до"

\* @return рандомное число от a до b

\*/

// static Random random = new Random();

public static int getRandomFromTo(int a, int b){

// return a + (random.nextInt(1) \* b);

return (int) (a + (Math.random() \* b));

}

/\*\*

\* Общая формула евклидова расстояния для n-мерного случая (n переменных)

\* @param x - первая точка

\* @param y - вторая точка

\*/

public static double getEuclideanDistance(double[] x, double[] y){

double d = 0;

for (int i = 0; i < x.length; i++){

d += Math.pow( ( x[i] - y[i] ), 2 );

}

d = Math.sqrt(d);

return d;

}

/\*\*

\* Общая формула евклидова расстояния для 2-мерного случая

\* @param x - первая точка

\* @param y - вторая точка

\*/

public static double getEuclideanDistance(double x, double y){

double d;

d = Math.pow( (x - y), 2 );

d = Math.sqrt(d);

return d;

}

/\*\*

\* Создание массива double со случайными числами

\* @param countOfElements - количество элементов, которые нужно сгенерировать

\* @param rangeMin - от этого числа начнется генериция

\* @param rangeMax - до этого числа будем генерировать

\* @return массив чисел

\*/

public static double[] getRandomArray(int countOfElements, double rangeMin, double rangeMax) {

double[] randomArray = new double[countOfElements];

Random r = new Random();

for(int i = 0; i < countOfElements; ++i) {

randomArray[i] = rangeMin + (rangeMax - rangeMin) \* r.nextDouble();

randomArray[i] = round(randomArray[i], 10);

}

return randomArray;

}

/\*\*

\* Создание массива с заданным шагом от точки start до точки end

\* @param step - шаг

\* @param start - начальная точка

\* @param countOfElements - кол-во элементов, которое нужно сгенерить

\*/

public static double[] getArrayFromTo(double step, double start, int countOfElements){

double[] result = new double[countOfElements];

result[0] = start;

for (int k = 1; k < countOfElements; k++){

result[k] = result[k - 1] + step; // считаем значение прибавив к нему шаг

result[k] = round(result[k], 10); // округляем

}

return result;

}

/\*\*

\* создаем искомую функцию

\* @param x - массив иксов

\* @return массив Y = искомая функция для ГА

\*/

public static double[] getFunction(double[] x){

// задаем размер массива

int countOfElements = x.length;

double[] y = new double[countOfElements];

// здесь заполняем массив.

for (int i = 0; i < countOfElements; i++){

y[i] = Math.cos(2\*x[i]);

y[i] = round(y[i], 10);

}

return y;

}

/\*\*

\* Округление

\* @param value число, которое нужно округлить

\* @param precision - до какого знака

\* @return округленное число

\*/

static double round(double value, int precision) {

int scale = (int) Math.pow(10, precision);

return (double) Math.round(value \* scale) / scale;

}

}