**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

**Факультет Программной инженерии и компьютерной техники**

Тестирование программного обеспечения

Лабораторная работа №1

Вариант 2287

Группа: P32312

Выполнил:

Романов А. И.

Преподаватель:

Гаврилов А. В.

Санкт-Петербург

2024

# Задание

1. Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.
2. Провести модульное тестирование указанного алгоритма. Для этого выбрать характерные точки внутри алгоритма, и для предложенных самостоятельно наборов исходных данных записать последовательность попадания в характерные точки. Сравнить последовательность попадания с эталонной.
3. Сформировать доменную модель для заданного текста.  Разработать тестовое покрытие для данной доменной модели

# Условия задания

1. Функция arctg(x)
2. Программный модуль для работы с хеш-таблицей с открытой адресацией (Hash Integer, <http://www.cs.usfca.edu/~galles/visualization/ClosedHash.html>)
3. Описание предметной области:

Под куполом был огромный космический корабль, длиной около полутораста метров, имевший форму кроссовки, совершенно белый и умопомрачительно красивый. В самом сердце его, невидимая, лежала небольшая золотая коробочка, внутри которой находился самый непостижимый уму прибор из всех когда-либо придуманных: прибор, который делал космический корабль уникальным в истории Галактики, и по имени которого корабль был назван -- Золотое Сердце.

# Выполнение задания

## Arctan(x)

Изображение выглядит как линия, График, диаграмма, Параллельный

Автоматически созданное описание

Рис. 1 График функции *arctan(x)*

Код модуля, выполняющего вычисление функции:

**package** edu.testing.part1\_arctg;

**import** **static** java.lang.Math.\*;

**public** **class** **ArctgRow** {

/\*\* Function returns result of function arctg(X) with the given X.

Max value = Pi/2, min value = -Pi/2

@param x value of X

@param n amount of row elements

\*/

**public** **static** **double** **getRowElement**(**double** x, **int** n) {

**double** sum = **0**d;

**if** (abs(x) <= **1**) {

**for** (**int** i = **0**; i <= n; i++) {

sum += pow(-**1**, i) \* (pow(x, **2** \* i + **1**) / (**2** \* i + **1**));

}

}

**else** {

**double** f1 = PI \* signum(x) / **2**;

**double** f2 = **0**d;

**for** (**int** i = **0**; i <= n; i++) {

f2 += pow(-**1**, i) \* pow(x, -**1** - **2** \* i) / (**1** + **2** \* i);

}

sum += f1 - f2;

}

**return** sum;

}

}

Код тестирования модуля:

**package** part1\_arctg;

**import** **edu.testing.part1\_arctg.ArctgRow**;

**import** **org.junit.jupiter.api.Assertions**;

**import** **org.junit.jupiter.api.Test**;

**import** **static** java.lang.Double.NaN;

**public** **class** **ArctgRowTest** {

**private** **static** **final** **double** DELTA = **0.0001d**;

**private** **static** **final** **int** N = **100000**;

**protected** **static** String **getErrorMessage**(**double** xGiven, **double** yExpected, **double** yActual, **int** n, **double** delta) {

**return** String.format("Arctan from x=%f is not equal to expected %f, but is equal to %f.\n Delta = %f, row members = %d,", xGiven, yExpected, yActual, delta, n);

}

**@Test**

**public** **void** **checkArctgAtNegativeInfinity**() {

**double** x = Double.NEGATIVE\_INFINITY;

**final** **double** Y\_EXPECTED = - Math.PI / **2**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowVeryLongBeforeMinusOne**() {

**double** x = -**5000452**;

**final** **double** Y\_EXPECTED = -**1.57079613**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowLongBeforeMinusOne**() {

**double** x = -**6489**;

**final** **double** Y\_EXPECTED = -**1.57064222**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowAfterMinusOne**() {

**double** x = -**3.6**;

**final** **double** Y\_EXPECTED = -**1.29984948**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowAtMinusOne**() {

**double** x = -**1**;

**final** **double** Y\_EXPECTED = - Math.PI / **4**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowBeforeZero**() {

**double** x = -**0.5**;

**final** **double** Y\_EXPECTED = -**0.46364**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgAtZero**() {

**double** x = **0**;

**final** **double** Y\_EXPECTED = **0**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgAtMinusZero**() {

**double** x = -**0**;

**final** **double** Y\_EXPECTED = **0**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowBeforeOne**() {

**double** x = **0.5**;

**final** **double** Y\_EXPECTED = **0.46364**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowAtOne**() {

**double** x = **1**;

**final** **double** Y\_EXPECTED = Math.PI / **4**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowAfterOne**() {

**double** x = **3**;

**final** **double** Y\_EXPECTED = **1.24904577**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowLongAfterOne**() {

**double** x = **4242**;

**final** **double** Y\_EXPECTED = **1.57056059**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgRowVeryLongAfterOne**() {

**double** x = **7000592**;

**final** **double** Y\_EXPECTED = **1.57079618**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgAtPositiveInfinity**() {

**double** x = Double.POSITIVE\_INFINITY;

**final** **double** Y\_EXPECTED = Math.PI / **2**;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(result, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

**@Test**

**public** **void** **checkArctgSymmetricalSmall**() {

**double** x1 = Math.random();

**double** x2 = -x1;

**double** result1 = ArctgRow.getRowElement(x1, N);

**double** result2 = ArctgRow.getRowElement(x2, N);

Assertions.assertEquals(result1, -result2, DELTA, "Function should return symmetrical results.");

}

**@Test**

**public** **void** **checkArctgSymmetricalBig**() {

**double** x1 = Math.random() \* **1000000**;

**double** x2 = -x1;

**double** result1 = ArctgRow.getRowElement(x1, N);

**double** result2 = ArctgRow.getRowElement(x2, N);

Assertions.assertEquals(result1, -result2, DELTA, "Function should return symmetrical results.");

}

**@Test**

**public** **void** **checkArctgRowAtNaN**() {

**double** x = NaN;

**final** **double** Y\_EXPECTED = NaN;

**double** result = ArctgRow.getRowElement(x, N);

Assertions.assertEquals(x, Y\_EXPECTED, DELTA, getErrorMessage(x, Y\_EXPECTED, result, N, DELTA));

}

}

## Closed hashing algorithm

Код реализации закрытого алгоритма хэширования:

**package** edu.testing.part2\_hashing;

**import** **edu.testing.part2\_hashing.utils.FullTableException**;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**import** **java.util.Objects**;

**import** **java.util.function.BinaryOperator**;

**import** **java.util.function.Function**;

**public** **class** **ClosedHashingTable** **implements** Table<Integer> {

**private** **final** List<Node<Integer>> table;

**private** **int** elementsAmount;

**private** **final** Integer tableSize;

**private** **final** Function<Integer, Integer> hashingAlgorithm;

**private** **final** BinaryOperator<Integer> probingAlgorithm;

**public** **static** BinaryOperator<Integer> **getLinearProbing**() {

**return** (i, x) -> i;

}

**public** **static** BinaryOperator<Integer> **getQuadraticProbing**() {

**return** (i, x) -> i \* i ;

}

**public** **static** BinaryOperator<Integer> **getDoubleProbing**(Function<Integer, Integer> hash2) {

**return** (i, x) -> i \* hash2.apply(x);

}

**public** **static** Function<Integer, Integer> sampleMainHash(**int** tableSize) {

**return** k -> k % tableSize;

}

**public** **static** Function<Integer, Integer> sampleHash2() {

**return** x -> **7** - x % **7**;

}

**public** **ClosedHashingTable**(**int** tableSize, Function<Integer, Integer> mainAlgorithm, BinaryOperator<Integer> probingAlgorithm) {

**this**.table = **new** ArrayList<>();

**for** (**int** i = **0**; i < tableSize; i++) {

table.add(**new** Node<>());

}

**this**.tableSize = tableSize;

**this**.elementsAmount = **0**;

**this**.probingAlgorithm = probingAlgorithm;

**this**.hashingAlgorithm = mainAlgorithm;

}

**@Override**

**public** **int** **insert**(Integer key) **throws** FullTableException {

**if** (elementsAmount == tableSize) {

**throw** **new** **FullTableException**("Table is full.");

}

**int** i = **0**;

**int** hashingResult = hashingAlgorithm.apply(key);

**int** position = hashingResult;

**do** {

var element = table.get(position);

**if** (element.getNodesState() != Node.NodesState.PRESENT) {

table.get(position).setValue(key);

elementsAmount++;

**return** position;

}

i++;

position = hashingResult + probingAlgorithm.apply(i, key);

} **while** (**true**);

}

**@Override**

**public** **int** **find**(Integer key) {

**int** i = **0**;

**int** hashingResult = hashingAlgorithm.apply(key);

**int** position = hashingResult;

**do** {

var element = table.get(position);

**if** (element.getNodesState() == Node.NodesState.PRESENT && Objects.equals(element.getValue(), key)) {

**return** position;

}

i++;

position = hashingResult + probingAlgorithm.apply(i, key);

} **while** (**true**);

}

**@Override**

**public** **int** **delete**(Integer key) {

**int** i = **0**;

**int** hashingResult = hashingAlgorithm.apply(key);

**int** position = hashingResult;

**do** {

var element = table.get(position);

**if** (element.getNodesState() == Node.NodesState.PRESENT && Objects.equals(element.getValue(), key)) {

element.setValue(**null**);

**return** position;

}

i++;

position = hashingResult + probingAlgorithm.apply(i, key);

} **while** (**true**);

}

// @Override

// public void clear() {

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

//

// }

// }

**public** List<Node<Integer>> **getTable**() {

**return** table;

}

**public** **int** **getElementsAmount**() {

**return** elementsAmount;

}

**public** Integer **getTableSize**() {

**return** tableSize;

}

**@Override**

**public** String **toString**() {

StringBuilder s = **new** StringBuilder();

**for** (**int** i = **0**; i < tableSize; i++) {

s.append(i).append(", ").append(table.get(i)).append(";\n");

}

**return** "table=[" + s + "]";

}

}

Код тестирования работы алгоритма закрытого хэширования:

<https://github.com/artemiyjjj/TPO_labs>

## Предметная область

Код программ, содержащих описание доменной области и её тестирование:

<https://github.com/artemiyjjj/TPO_labs>

# Вывод

В ходе выполнения были изучены подходы к тестированию ПО и фреймворк junit, который был задействован при тестировании разработанных в соответствии с выданным вариантом модулей.