НИУ ИТМО
Факультет программной инженерии и компьютерных технологий

Отчет по лабораторной работе №1
по дисциплине Тестирование ПО

Студент группы № P33151 Шипулин Павел Андреевич

Преподаватель Харитонова Анастасия Евгеньевна

Санкт-Петербург
2024

Ход работы

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

Вариант 117319:

- Функция sec(x)
- Программный модуль для сортировки массива подсчетом (http://www.cs.usfca.edu/~galles/visualization/CountingSort.html)
- Описание предметной области:

  Много-много миллионов лет назад раса гиперразумных всемерных существ (чье физическое проявление в их всемерной вселенной практически не отличается от нашего) так устала от постоянных споров о смысле жизни, которые отвлекали их от их излюбленного времяпрепровождения -- брокианского ультра-крикета (забавная игра, заключающаяся в том, чтобы неожиданно ударить человека без видимой на то причины и убежать) -- что решила сесть и решить все вопросы раз и навсегда.

Вопросы к защите лабораторной работы:

1. Понятие тестирования ПО. Основные определения.
2. Цели тестирования. Классификация тестов.
3. Модульное тестирование. Понятие модуля.
4. V-образная модель. Статическое и динамическое тестирование.
5. Валидация и верификация. Тестирование методом "чёрного" и "белого" ящика.
6. Тестовый случай, тестовый сценарий и тестовое покрытие.
7. Анализ эквивалентности.
8. Таблицы решений и таблицы переходов.
9. Регрессионное тестирование.
10. Библиотека JUnit. Особенности API. Класс junit.framework.Assert.
11. Отличия JUnit 3 от JUnit 4.

Выполнение задания 1

Расчет значения функции

Функция по варианту:

$$\sec(x) = \frac{1}{\cos(x)}$$

MaclaurinSeries.java

```java
package Task1;

public class MaclaurinSeries {
    public static double calc(double x, double[] derivatives) {
        double partialSum = 0;
        double factorial = 1;
        double xPower = 1;

        for (int i = 0; i < derivatives.length; i++) {
            partialSum += derivatives[i] / factorial * xPower;
            xPower *= x;
            factorial *= i + 1;
        }

        return partialSum;
    }
}
```

Метод `calc` реализует подсчет частичной суммы ряда Макларена:

$$S_n = f(0) + f'(0) \cdot x + \frac{f''(0)}{2}x^2 + \cdots + \frac{f^{(n)}(0)}{n!}x^n = \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k!}x^k$$

Cos.java

```java
package Task1;

public class Cos {
    private static final double[] derivatives_at_zero_values =
            new double[] { 1.0, 0.0, -1.0, 0.0 };

    public static double[] derivativesAtZero(int order) {
        double[] derivatives = new double[order + 1];
        for (int i = 0; i < order; i++) {
            derivatives[i] = derivatives_at_zero_values[i & 3];
        }

        return derivatives;
    }
}
```

Метод `derivativesAtZero` реализует подсчет значений производных функции $cos(x)$ при $x = 0$.

Модульное тестирование

Анализ функции

Функция $\sec(x) = \frac{1}{\cos(x)}$, как и $\cos(x)$, имеет период $T = 2\pi$. Поэтому перед расчетом значения функции, будет происходить изменение аргумента на $\pm T$ так, чтобы в конце концов его значение было как можно ближе к 0. Это нужно для уменьшения ошибки при расчете частичной суммы ряда Макларена, так как эта частичная сумма обеспечивает сходимость только в небольшой окрестности 0.

Поиск области определения функции $\sec(x)$:

$$\cos(x) = 0 \Rightarrow x = \frac{\pi}{2} + \pi n, n \in Z$$

$$D\left(\frac{1}{\cos(x)}\right): x \in \left(-\frac{\pi}{2} + \pi n; \frac{\pi}{2} + \pi n\right), n \in Z$$

Таким образом, функция имеет точки разрыва при

$$x = \frac{\pi}{2} + \pi n, n \in Z$$

График функции

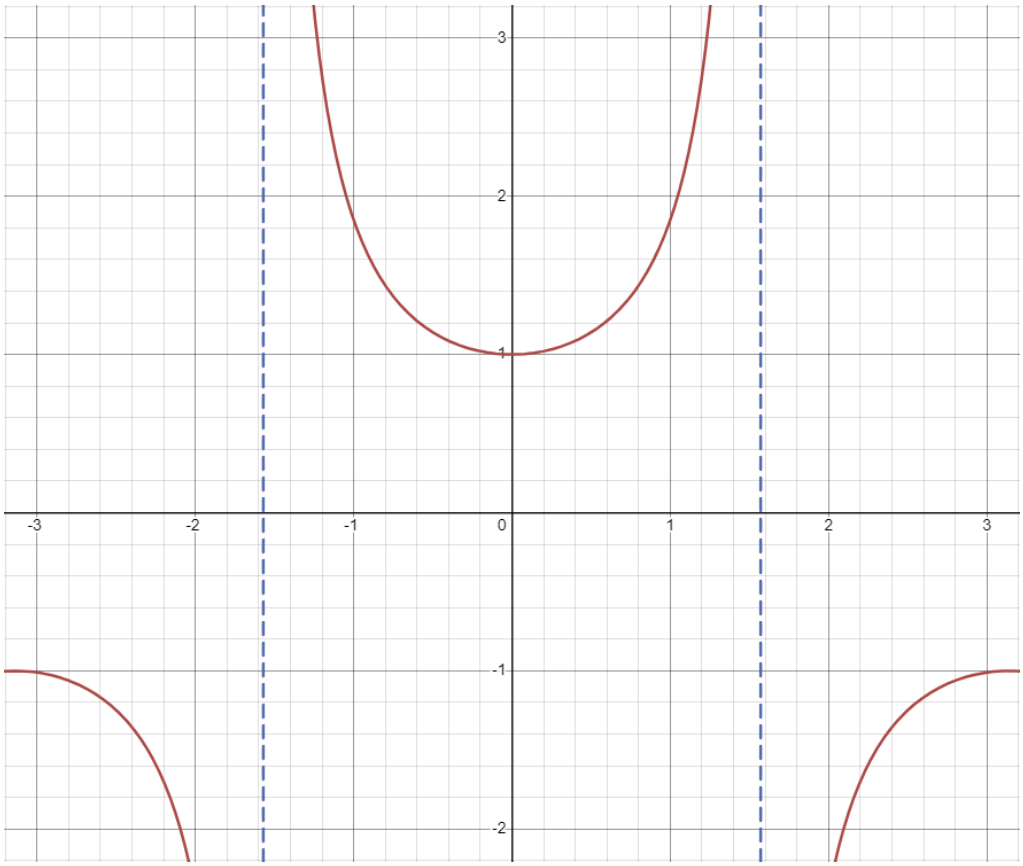


Рисунок 1. $y(x) = \sec(x)$. Синими пунктирными прямыми указаны точки разрыва.

test_values.csv

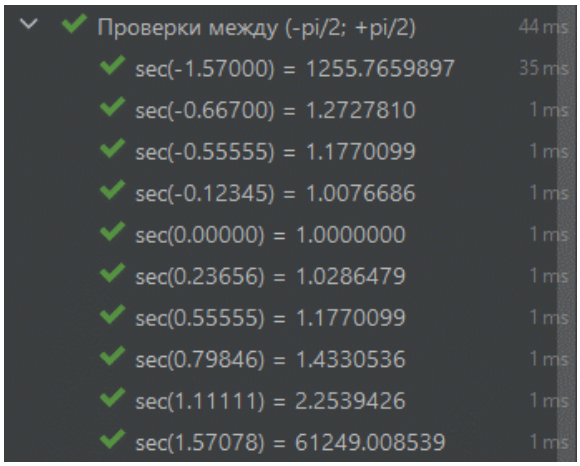
```
x,sec(x)
-1.57000,1255.7659897
-0.66700,1.2727810
-0.55555,1.1770099
-0.12345,1.0076686
0.00000,1.0000000
0.23656,1.0286479
0.55555,1.1770099
0.79846,1.4330536
1.11111,2.2539426
1.57078,61249.008539
```

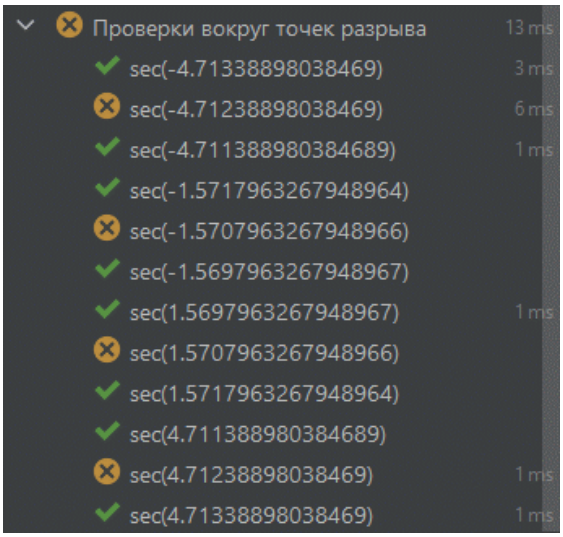
Файл содержит некоторые тестовые значения аргумента x и функции $\sec(x)$ при $x \in \left(-\frac{\pi}{2}; \frac{\pi}{2}\right)$.

SecByCosTest.java

```java
package Task1;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvFileSource;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class SecByCosTest {
    private static double changeArg(double x) {
        while (x > Math.PI) {
            x -= 2 * Math.PI;
        }

        while (x < -Math.PI) {
            x += 2 * Math.PI;
        }

        return x;
    }

    private static final double delta = 0.001;
    private static final int derivativesOrder = 100;
    private static final double accuracy = 0.001;

    @ParameterizedTest(name = "sec({0})")
    @DisplayName("Проверки вокруг точек разрыва")
    @ValueSource(doubles = {
            -3 * Math.PI / 2 - delta,
            -3 * Math.PI / 2,
            -3 * Math.PI / 2 + delta,
            -Math.PI / 2 - delta,
            -Math.PI / 2,
            -Math.PI / 2 + delta,
            Math.PI / 2 - delta,
            Math.PI / 2,
            Math.PI / 2 + delta,
            3 * Math.PI / 2 - delta,
            3 * Math.PI / 2,
            3 * Math.PI / 2 + delta
    })
    public void checkPI(double x) {
        final double corrected_x = changeArg(x);
        assertAll(() -> assertEquals(1 / Math.cos(corrected_x), 1 /
MaclaurinSeries.calc(corrected_x, Cos.derivativesAtZero(derivativesOrder)),
accuracy));
    }

    @ParameterizedTest(name = "sec({0}) = {1}")
    @DisplayName("Проверки между (-pi/2; +pi/2)")
    @CsvFileSource(resources = "/test_values.csv", numLinesToSkip = 1, delimiter =
',')
    public void checkBetweenPI(double x, double sec) {
        final double corrected_x = changeArg(x);
        assertAll(
                () -> assertEquals(sec, 1 / MaclaurinSeries.calc(corrected_x,
Cos.derivativesAtZero(derivativesOrder)), accuracy)
        );
    }
}
```

Результаты тестирования:





Тесты в интервале между $\pm\frac{\pi}{2}$:

- пройдены.

Тесты вблизи точек разрыва:

- провалены при значениях $x = \frac{\pi}{2} + \pi n, n \in Z$.

Сообщения об ошибках при $x = \frac{\pi}{2} + \pi n, n \in Z$:

x	**Expected**	**Actual**
$-\frac{3\pi}{2}$	$1.6331239353195 37E16$	$-1.0527591507754862E17$
$-\frac{\pi}{2}$	$1.6331239353195 37E16$	$-1.0527591507754862E17$
$\frac{\pi}{2}$	$1.6331239353195 37E16$	$-1.0527591507754862E17$
$\frac{3\pi}{2}$	$1.6331239353195 37E16$	$-1.0527591507754862E17$

Так как

$$\lim_{x\to\frac{\pi}{2}}\left|\frac{1}{\cos(x)}\right| = +\infty$$

и обе функции $\cos(x)$ (встроенная и тестовая) предоставляют такие схожие по модулю значения в точках разрыва $\sec(x)$, то можно считать, что тестовая функция работает как ожидается в точках разрыва.

Выполнение задания 2

Алгоритм

CountingSort.java

```java
package Task2;

import java.util.Arrays;

public class CountingSort {
    private static final int max_value = 30;

    public static int[] sort(int[] array) {
        if (array == null) {
            throw new NullPointerException("null указатель");
        }

        if (array.length <= 1) {
            return array;
        }

        int min = Arrays.stream(array).min().getAsInt();
        if (min < 0) {
            throw new IllegalArgumentException(String.format("min(array) = %d - поддерживаются
только натуральные числа и 0", min));
        }

        int max = Arrays.stream(array).max().getAsInt();
        if (max > max_value) {
            throw new IllegalArgumentException(String.format("max(array) = %d, максимальное
натуральное должно быть не больше %d", max, max_value));
        }

        int[] countingArray = new int[max + 1];
        Arrays.fill(countingArray, 0);

        for (int value : array) {
            countingArray[value]++;
        }

        for (int i = 0; i < max; i++) {
            countingArray[i + 1] += countingArray[i];
        }

        int[] sortedArray = new int[array.length];

        for (int j : array) {
            sortedArray[--countingArray[j]] = j;
        }

        return sortedArray;
    }
}
```

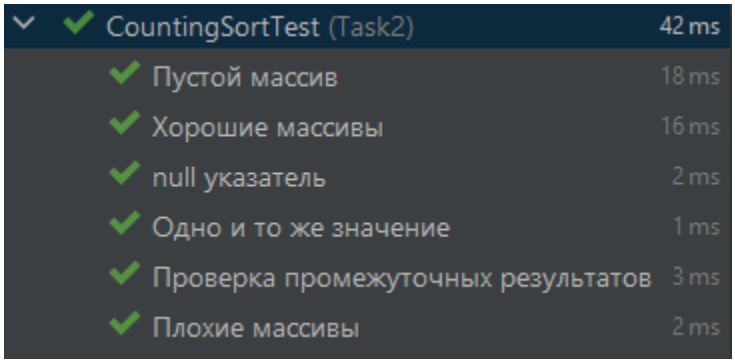
Модульное тестирование

CountingSortTest.java

```
package Task2;

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

import java.util.Arrays;

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

public class CountingSortTest {
    @Test
    @DisplayName("Хорошие массивы")
    void checkGood() {
        CountingSort counting_sort = new CountingSort();
        assertAll(
                () -> assertArrayEquals(new int[] { 1, 2, 3, 4 }, counting_sort.sort(new int[]
{ 4, 2, 3, 1 })),
                () -> assertArrayEquals(new int[] { 0, 1, 1, 2, 2, 3, 4, 5 },
counting_sort.sort(new int[] { 3, 5, 1, 2, 0, 1, 2, 4 })),
                () -> assertArrayEquals(new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 },
counting_sort.sort(new int[] { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 })),
                () -> assertArrayEquals(new int[] { 0, 1, 2, 3 }, counting_sort.sort(new int[]
{ 0, 1, 2, 3 })),
                () -> assertArrayEquals(
                        new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 30 },
                        counting_sort.sort(new int[] { 30, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 0, 1, 2, 3, 4, 5, 6, 7, 26, 27, 28, 29, 30 })
                )
        );
    }

    @Test
    @DisplayName("Пустой массив")
    void checkEmpty() {
        CountingSort counting_sort = new CountingSort();
        assertArrayEquals(new int[]{}, counting_sort.sort(new int[]{}));
    }

    @Test
    @DisplayName("Плохие массивы")
    void checkBad() {
        CountingSort counting_sort = new CountingSort();
        assertAll(
                () -> assertThrows(IllegalArgumentException.class, () ->
counting_sort.sort(new int[] { -1, 0, 1 })),
                () -> assertThrows(IllegalArgumentException.class, () ->
counting_sort.sort(new int[] { 1, 2, -30 })),
                () -> assertThrows(IllegalArgumentException.class, () ->
counting_sort.sort(new int[] { 0, 1, 31 }))
        );
    }

    @Test
    @DisplayName("Одно и то же значение")
    void checkSameNumbers() {
        CountingSort counting_sort = new CountingSort();
        assertAll(
                () -> assertArrayEquals(new int[] { 0, 0, 0 }, counting_sort.sort(new int[] {
0, 0, 0 })),
                () -> assertArrayEquals(new int[] { 30, 30, 30 }, counting_sort.sort(new int[]
{ 30, 30, 30 }))
        );
    }

    @Test
    @DisplayName("null указатель")
    void checkNull() {
        CountingSort counting_sort = new CountingSort();
        assertThrows(NullPointerException.class, () -> counting_sort.sort(null));
```

```
    }

    @Test
    @DisplayName("Проверка промежуточных результатов")
    void checkStdout() {
        CountingSort counting_sort = new CountingSort();
        String[] expected = new String[] {
                "Подсчитываем значения исходного массива",
                "countingArray[2] = 1",
                "countingArray[1] = 1",
                "countingArray[3] = 1",
                "Рассчитываем индексы в новом массиве для значений исходного",
                "countingArray[1] = 0",
                "countingArray[2] = 1",
                "countingArray[3] = 2",
                "Копируем элементы исходного массива на их места в отсортированном массиве",
                "countingArray[2] = 1",
                "countingArray[1] = 0",
                "countingArray[3] = 2"
        };

        assertArrayEquals(new int[] { 1, 2, 3 }, counting_sort.sort(new int[] { 2, 1, 3 }));
        assertLinesMatch(Arrays.asList(expected), counting_sort.getHistory());
    }
}
```

Результаты тестирования:



Выполнение задания 3

PLACEHOLDER.

Вывод

Научился создавать тесты с помощью JUnit. Провел модульное тестирование: два варианта поиска значения функции $\sec(x)$, алгоритм сортировки подсчетом, PLACEHOLDER.