Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Санкт-Петербургский Государственный Электротехнический
Университет «ЛЭТИ» им. В.И. Ульянова (Ленина)»

Факультет компьютерных технологий и информатики
Кафедра автоматики и процессов управления

ОТЧЕТ

по лабораторной работе №1
по дисциплине «СМиСПИС»
Вариант №3

Студент гр. 5371	__________________	Мартынов М.
Студентка гр. 5371	__________________	Козлова С.
Студент гр. 5371	__________________	Аверкиев В.
Преподаватель	__________________	Кораблев Ю.А.

Санкт-Петербург
2019

1. Цель работы

Изучить основы разработки *Java*-программ. Изучить основные типы *Java,* создание и работу с массивами.

2. Задание на лабораторную работу №1. Вариант №3.

Написать программу, которая осуществляет расчет значения функции, используя рекуррентную формулу. Для расчета значений ряда предусмотреть возможность ввода значения x и точности вычисления.

Вариант	Функция	Ряд	Реккурентная формула
3	cos(x)	$1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots=\sum_{i=0}^{\infty}u_i$	$u_i=-u_{i-1}\cdot\frac{x^2}{(2\cdot i-1)\cdot 2\cdot i}$

3. Разработка математического и алгоритмического обеспечения

Числовым рядом называется бесконечная сумма *S* некоторой последовательности.

$$S = a_1+a_2+a_3+\ldots+a_i+\ldots = \sum_{i=1}^{\infty} a_i$$

С помощью разложения в числовой ряд можно вычислить многие из элементарных функций, например

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots,$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \ldots \quad \text{при} \quad 0 \le x \le \frac{\pi}{4};$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} - \ldots \quad \text{при} \quad |x| < 1.$$

Поскольку ряд имеет бесконечное число членов, вычисления производят с определенной точностью, т.е. суммирование прекращают, когда очередной член ряда оказывается меньше *допустимой погрешности вычислений*. Допустимую погрешность вычислений называют иначе *точностью вычислений*. Она задается малым числом ε, где 0<ε<1, ($\varepsilon=10^{-2}$, 10^{3},…), чем меньше ε, тем точнее решение. При $\varepsilon=10^{i}$ решение будет точным до *i*-го знака после запятой.

Основная проблема при вычислении суммы числового ряда состоит в вычислении очередного члена последовательности. Можно выделить три подхода к решению этой проблемы:

1) использование формулы общего члена ряда;

2) использование рекуррентного соотношения;

3) смешанный подход, основанный на двух предыдущих.

Использование рекуррентного соотношения

Понятие ***рекуррентной последовательности*** в курсе математики вводится так: пусть известно k чисел: $a_1, \ldots, a_k$. Эти числа являются началом числовой последовательности. Следующие элементы этой последовательности вычисляются так:

$$a_{k+1} = F(a_1, \ldots, a_k);\ a_{k+2} = F(a_2, \ldots, a_{k+1});\ a_{k+3} = F(a_3, \ldots, a_{k+2})\ \ldots$$

Здесь $F(\ldots)$ – функция от k аргументов. Формула вида

$$a_i = F(a_{i-k}, \ldots, a_{i-1})$$

называется ***рекуррентной формулой***. Другими словами, *рекуррентная последовательность* – это бесконечный ряд чисел, каждое из которых, за исключением k начальных, вычисляется через предыдущие члены ряда.

Например, арифметическая и геометрическая прогрессии:

$a_1=1$, $a_2=3$, $a_3=5$,… Рекуррентная формула: $a_i = \begin{cases} 1, \text{если } i = 1, \\ a_{i-1} + 2, \text{если } i > 1 \end{cases}$.

a_1=1, a_2=2, a_3=4, … Рекуррентная формула: $a_i = \begin{cases} 1, \text{если } i = 1, \\ a_{i-1} \cdot 2, \text{если } i > 1 \end{cases}$.

Глубина рекурсии в обоих случаях равна 1 (такую зависимость называют одношаговой рекурсией).

Числа Фибоначчи: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, … Начиная с третьего элемента, каждое число равно сумме двух предыдущих, т.е. это рекуррентная последовательность с глубиной, равной 2. Рекуррентная формула для чисел последовательности Фибоначчи:

$$f_i = \begin{cases} 1, \text{если } i = 1, i = 2, \\ f_{i-1} + f_{i-2}, \text{если } i > 2 \end{cases}.$$

Для вывода рекуррентной формулы часто можно воспользоваться соотношением вида:

$$\frac{a_i}{a_{i-1}} = f(i).$$

Получив *f(i)*, мы по сути найдем основную часть рекуррентного соотношения, так как

$$a_i = a_{i-1} \cdot f(i).$$

Правда, при этом нельзя забывать о начальном условии, без которого рекуррентная формула не будет полной.

Определим рекуррентное соотношение для ряда: $S = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$.

Положим $a_0 = \frac{x^0}{0!} = 1; \quad a_1 = \frac{x^1}{1!} = x; \quad a_i = \frac{x^i}{i!}; \quad a_{i-1} = \frac{x^{i-1}}{(i-1)!}$.

Тогда

$$f(i) = \frac{a_i}{a_{i-1}} = \frac{x^i \cdot (i-1)!}{x^{i-1} \cdot i!} = \frac{x}{i}.$$

Получаем рекуррентное соотношение: $a_i = \begin{cases} 1, & если\ i = 0 \\ a_{i-1} \cdot \frac{x}{i}, & если\ i > 0 \end{cases}$.

Рекуррентное соотношение используется в тех случаях, когда следующий член ряда можно выразить через предыдущий (или предыдущие), а использование формулы общего члена приведет к появлению вложенных циклов, что сделает программу неэффективной.

Рекуррентная формула для вычисления функции cos(x)

Выпишем члены ряда

$$a_0 = 1, \; a_1 = -\frac{x^2}{2!}, \; a_2 = \frac{x^4}{4!}, \; a_i = (-1)^i \frac{x^{2i}}{(2i)!}, \; a_{i-1} = (-1)^{i-1} \frac{x^{2(i-1)}}{(2 \cdot (i-1))!}$$

.

Поскольку каждый член ряда может быть получен из предыдущего домножением на определенный множитель, а вычисление «в лоб» каждого члена ряда приведет к появлению вложенных циклов, эффективнее будет использовать рекуррентное соотношение. Выведем его

$$\frac{a_i}{a_{i-1}} = \frac{x^{2i} \cdot (-1)^i \cdot (2 \cdot (i-1))!}{x^{2(i-1)} \cdot (-1)^{i-1} \cdot (2i)!} = -\frac{x^2}{(2i-1) \cdot (2i)}$$

Полученное рекуррентное соотношение:

$$a_i = \begin{cases} 1, & если\ i = 0 \\ a_{i-1} \cdot \left(-\frac{x^2}{(2i-1) \cdot (2i)} \right), & если\ i > 0 \end{cases}$$

4. Разработка программного обеспечения

Для разработки программы для выполнения данной лабораторной работы была разработана программа на языке Kotlin под JVM.

Kotlin (Ко́тлин) — статически типизированный язык программирования, разрабатываемый компанией JetBrains.

Описание функций программы

Разработанная программа состоит из 7 функций.

Название функции	Описание функции
main(args: Array<String>)	Функция с которой начинается выполнение программы, в которой содержится логика верхнего уровня по управлению программой.
getPrecisionValue(): Int	Считывает с командной строки точность вычисления, которую ввел пользователь и проверяет введенное значение на валидность.
getAngleValue(): Double	Считывает с командной строки угол в радианах для вычисления cos(x), которую ввел пользователь и проверяет введенное значение на валидность.
cos(x: Double, precision: Int): Double	Вычисляет значение функции cos(x) с заданной точностью precision.
nextSequenceElement(i: Int, x: Double, prevElement: Double): Double	Возвращает i-ый рекуррентный элемент последовательности при входном i-1-ом элементе.

precisionIsChanging(sum: Double, precision: Int, nextElement: Double): Boolean	Возвращает значение типа Boolean, сообщающее о том, достигнута ли заданная точность вычисления функции cos(x)
Double.round(decimals: Int): Double	Extension функция для типа Double, округляющее значение типа Double до decimals знаков после запятой.

Описание работы программы

Написанная программа исполняется на JVM, поэтому для запуска необходимо в командной строке ввести команду:

```
java -jar lab1.jar
```

После чего в командной строке отобразится сообщение о том, что необходимо ввести угол в радианах:

```
Enter the angle value in radians (Double value) X =
```

После ввода угла необходимо ввести точность вычисления:

```
Enter precision (Integer value):
```

После ввода точности программа отобразит результат с заданной точностью:

```
Enter the angle value in radians (Double value) X = 2.13
Enter precision (Integer value): 5
cos(2.13)=-0.53051
```

При вводе некорректных данных программа напечатает в консоль сообщение об ошибке и предложит ввести данные заново.

Чтобы выйти из программы необходимо ввести стандартное сочетание клавиш «Control + C».

Исходный код программы

```
package com.github.sindicat

import java.lang.Exception
import kotlin.math.pow
import kotlin.math.roundToLong

fun main(args: Array<String>) {
    while (true) {
        try {
            val x: Double = getAngleValue()
            val precision: Int = getPrecisionValue()
            println("cos($x)=${cos(x, precision)}")
        } catch (e: Exception) {
            println(e.message)
        }
    }
}

internal fun getPrecisionValue(): Int {
    print("Enter precision (Integer value): ")
    try {
        return readLine()!!.toInt().also {
            if (it > 308) throw IllegalArgumentException("Too large value for
precision. Max precision 308 decimal places")
            if (it < 0) throw IllegalArgumentException("Precision must be greater
zero.")
        }
    } catch (e: NumberFormatException) {
        throw IllegalArgumentException("Incorrect input. Please, enter Integer
value.")
    }
}

internal fun getAngleValue(): Double {
    print("Enter the angle value in radians (Double value) X = ")
    try {
        return readLine()!!.toDouble()
    } catch (e: NumberFormatException) {
        throw IllegalArgumentException("Incorrect input. Please, enter Double value.")
    }
}

internal fun cos(x: Double, precision: Int): Double {
    var angleInRadians = x % (2 * Math.PI)
    var currentElement = 1.0
    var sum = 0.0
    var index = 1
    while (precisionIsChanging(sum, precision, currentElement)) {
        sum += currentElement
        currentElement = nextSequenceElement(index++, angleInRadians, currentElement)
    }
    return sum.round(precision)
}

private fun nextSequenceElement(i: Int, x: Double, prevElement: Double): Double =
    -1 * prevElement * ((x.pow(2.0)) / ((2 * i - 1) * 2 * i))

private fun precisionIsChanging(sum: Double, precision: Int, nextElement: Double):
Boolean =
    sum.round(precision).compareTo((sum + nextElement).round(precision)) != 0

fun Double.round(decimals: Int): Double {
    var multiplier = 1.0
    repeat(decimals) { multiplier *= 10 }
    return (this * multiplier).roundToLong() / multiplier
}
```

Все данные по выполненной работе можно найти в [репозитории на Github](https://github.com/Sindicat/krlabs/tree/master/lab1/) (https://github.com/Sindicat/krlabs/tree/master/lab1/): исходный код в директории src, скомпилированный .jar файл в директории target, отчет в директории report.

Выводы

В лабораторной работе была разработана программа для вычисления функции cox(x) через рекуррентную формулу на JVM языке Kotlin.