Федеральное государственное автономное

образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | Институт космических и информационных технологий | | институт | | Кафедра программной инженерии | | кафедра | |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ**

|  |
| --- |
| Основы функционального программирования в Scala |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | К. В. Богданов |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ21-17/1Б, 032161381 |  |  |  | Л. М. Соколов |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2024

**СОДЕРЖАНИЕ**

[1 Цели 3](#_Toc126862301)

[2 Задачи 3](#_Toc126862302)

[3 Описание варианта задания 3](#_Toc126862303)

[4 Ход выполнения 3](#_Toc126862304)

[5 Выводы 8](#_Toc126862305)

[Список использованных источников 9](#_Toc126862306)

# Цели

# Написать скрипт, вычисляющий и выводящий в консоль в виде таблицы значения функции, заданной с помощью ряда Тейлора на интервале от Хнач до Хкон с шагом dx с точностью e. Таблицу снабдить заголовком.

# Задачи

Необходимо выполнить следующие задачи:

* Реализовать задание;
* Составить отчет о выполнении задания.

# Описание варианта задания

* Входные параметры скрипта: Хнач, Хкон, dx, e - числа с плавающей запятой.
* Для форматирования вывода использовать метод formatted или аналогичные способы.
* Запрещается использовать var-переменные в явном или неявном виде и циклы со счетчиком (придерживаться функционального стиля разработки).
* Необходимо осуществить проверку на адекватность значений перед расчетом (конечное значение должно быть достижимо при заданном шаге и начальном значении, ошибка больше нуля, конечные и начальные значения укладываются в условия по варианту).
* Не забыть, что ошибка (отклонение) рассчитывается как абсолютное значение.

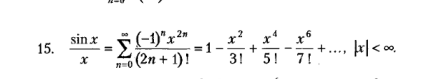


Рисунок 1 – вариант задания

# Ход выполнения

На языке Scala был реализован класс Taylor, принимающий в себя необходимые по заданию параметры, а также исходную функцию для аппроксимации и функцию суммы ряда, в который разлагается исходная функция. Класс реализует вычисление заданной функции для определенного x с точностью e, а также позволяет выводить таблицу со значениями функции в заданном диапазоне.

Листинг 1- Class Taylor

/\*\*

\* Класс для вычисления ряда Тейлора.

\*

\* @param xS Начальное значение x.

\* @param xF Конечное значение x.

\* @param dx Шаг изменения x.

\* @param e Ошибка.

\* @param func Функция, для которой вычисляется ряд Тейлора.

\* @param taylorSumBody Функция, вычисляющая сумму ряда Тейлора.

\*/

class Taylor(

xS: BigDecimal = -0.9,

xF: BigDecimal = -0.4,

dx: BigDecimal = 0.1,

e: BigDecimal = 0.00000001,

func: BigDecimal => BigDecimal,

taylorSumBody: (BigDecimal, Int) => BigDecimal

) {

// Проверяем адекватность параметров

if (dx <= 0 || e <= 0 || xS >= xF) {

throw new IllegalArgumentException("Некорректные параметры")

}

// Проверяем достижимость xF от xS с заданным шагом

if ((xF - xS) % dx != 0) {

throw new IllegalArgumentException("Невозможно достичь xF от xS с заданным шагом dx.")

}

/\*\*

\* Запускает вычисления ряда Тейлора и выводит результаты.

\*/

def run(): Unit = {

println("╔" + "═" \* 12 + "╤" + "═" \* 22 + "╤" + "═" \* 22 + "╤" + "═" \* 7 + "╗")

println(f"║ X │ f(x) │ Taylor(x) │ TI ║")

forIFin(xS, xF, dx, printTaylorResult)

println("╚" + "═" \* 12 + "╧" + "═" \* 22 + "╧" + "═" \* 22 + "╧" + "═" \* 7 + "╝")

}

/\*\*

\* Выводит результаты вычисления ряда Тейлора для заданного x.

\*

\* @param x Значение x.

\*/

private def printTaylorResult(x: BigDecimal): Unit = {

/\*\*

\* Вычисляет ряд Тейлора для заданного x.

\*

\* @return Пара значений: сумма ряда Тейлора и количество членов ряда.

\*/

def taylorFunc(): (BigDecimal, Int) = {

val terms = LazyList.from(0).map {

n => taylorSumBody(x, n)

}.takeWhile(\_.abs >= e)

(terms.sum, terms.size)

}

val funcRes = taylorFunc()

val taylorFuncResult = funcRes.\_1.setScale(15 min e.scale, BigDecimal.RoundingMode.HALF\_UP)

val termsCount = funcRes.\_2

val builtItFuncResult = func(x).setScale(15 min e.scale, BigDecimal.RoundingMode.HALF\_UP)

println("╟" + "─" \* 12 + "┼" + "─" \* 22 + "┼" + "─" \* 22 + "┼" + "─" \* 7 + "╢")

println(f"║ $x%-10s │ $builtItFuncResult%-20s │ $taylorFuncResult%-20s │ $termsCount%-5s ║")

}

}

Функция реализации цикла for i, в стиле ФП, а также функция факториала реализованы в Singleton классе Utils.

Листинг 2 – Object Utils

/\*\*

\* Объект, содержащий утилитарные методы для работы с числами.

\*/

object Utils {

/\*\*

\* Выполняет итерацию от начального значения до конечного с заданным шагом и применяет к каждому значению

\* указанное тело.

\*

\* @param start Начальное значение.

\* @param stop Конечное значение.

\* @param step Шаг итерации.

\* @param body Функция, применяемая к каждому значению.

\*/

def forIFin(

start: BigDecimal,

stop: BigDecimal,

step: BigDecimal,

body: BigDecimal => Unit

): Unit = {

@tailrec

def iteration(i: BigDecimal): Unit = {

body(i)

if (i < stop) iteration(i + step)

}

iteration(start)

}

/\*\*

\* Вычисляет факториал числа n.

\*

\* @param n Число, для которого вычисляется факториал.

\* @return Факториал числа n.

\*/

def factorial(n: BigDecimal): BigDecimal = {

if (n <= 1) 1 else n \* factorial(n - 1)

}

}

Ввод чисел и проверка их на возможность преобразование в BigDeciml осуществляется в основном классе программы – Main

Листинг 3 – Object Main

/\*\*

\* Объект, содержащий точку входа в приложение.

\*/

object Main {

/\*\*

\* Точка входа в приложение.

\*

\* @param args Аргументы командной строки.

\*/

def main(args: Array[String]): Unit = {

val fx = (x: BigDecimal) => math.sin(x.toDouble) / x

val taylorSeq = (x: BigDecimal, n: Int) => (BigDecimal(-1).pow(n) \* x.pow(2 \* n)) / factorial(2 \* n + 1)

println("Введите через пробел: x\_начальное, x\_конечное, dx - шаг, e - точность")

val numbersStr = readLine().replace(",", ".").split("\\s+")

if (numbersStr.length == 4) {

try {

val numbers = numbersStr.map(BigDecimal.apply)

val taylor = new Taylor(

xS = numbers(0),

xF = numbers(1),

dx = numbers(2),

e = numbers(3),

func = fx,

taylorSumBody = taylorSeq

)

taylor.run()

} catch {

case e: NumberFormatException =>

System.err.println("Ошибка при преобразовании строки в BigDecimal. Некорректный формат числа.")

case e: IllegalArgumentException =>

System.err.println("Ошибка в значениях: " + e.getMessage)

}

} else {

System.err.println("Ошибка: ввод должен содержать ровно 4 числа.")

}

}

}

На рисунках 1-3 приведены примеры работы программы и обработки ошибок

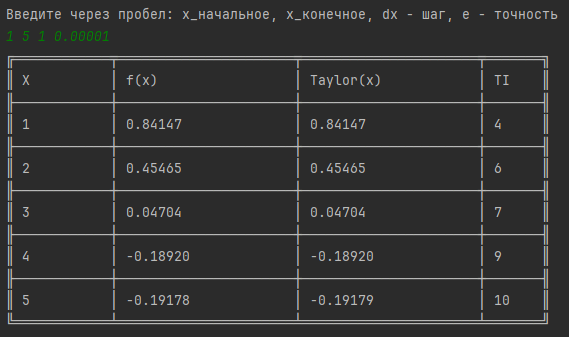


Рисунок 2 – корректная работа программы

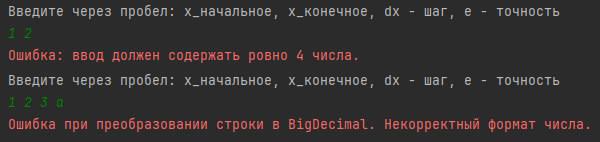


Рисунок 3 – обработка некорректного ввода

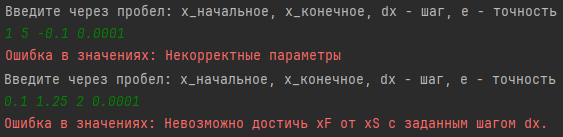


Рисунок 4 – обработка некорректных значений (по смыслу)

# Выводы

В ходе выполнения практической работы:

* был изучен теоретический материал по основам языка Scala и парадигме ФП;
* были получены навыки работы с базовым синтаксисом языка Scala;
* были выполнены действия в соответствии с заданием.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СТО 7.5-07-2021 Система менеджмента качества. Общие требования построению, изложению и оформлению документов учебной деятельности. – Взамен СТО 4.2-07-2014 ; введ. 07.12.2021. – Красноярск : СФУ 2021. – 61 с.