Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Информатики»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

Основы функционального программирования

Вариант 11

Преподаватель _____ _____

подпись, дата инициалы, фамилия

Студент ЗКИ21-16БВВ_031625881 20.03.2025 Е.М.Хорошко

подпись, дата инициалы, фамилия

Красноярск 2025

1. Задание

Вычислить и вывести на экран 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 и шапкой. Каждая строка таблицы должна содержать значение аргумента, значение функции и количество просуммированных членов ряда.

$$11.\quad \operatorname{arth} x = \sum_{n=0}^{\infty} \frac{1}{(2n+1)x^{2n+1}} = \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \ldots, \quad |x| > 1.$$

2. Исходный код основного алгоритма

```
import scala.annotation.tailrec
import scala.util.{Try, Success, Failure}

object Main extends App {
  if (args.length != 4) {
    println("Использование: program Хнач Хкон dx e")
    sys.exit(1)
  }

  private case class Params(
    xStart: Double,
    xEnd: Double,
    dx: Double,
    epsilon: Double
  )

  private val paramsTry = Try {
    Params(
      args(0).toDouble,
      args(1).toDouble,
      args(2).toDouble,
      args(3).toDouble
    )
  }

  paramsTry match {
    case Failure(_) =>
```

```
      println("Ошибка: все параметры должны быть числами с плавающей
точкой")
      sys.exit(1)

    case Success(params) =>
      validateParams(params) match {
        case Left(errors) =>
          println("Ошибки валидации:\n" + errors.mkString("\n"))
          sys.exit(1)

        case Right(_) =>
          val xs = generateXValues(params.xStart, params.xEnd,
params.dx)
          printResults(xs, params.epsilon)
      }
  }

  private def validateParams(p: Params): Either[List[String], Unit] =
{
    val errors = List.newBuilder[String]

    // Проверка модулей X
    if (math.abs(p.xStart) <= 1) errors += "|Хнач| должен быть > 1"
    if (math.abs(p.xEnd) <= 1) errors += "|Хкон| должен быть > 1"

    // Проверка точности
    if (p.epsilon <= 0) errors += "Точность e должна быть > 0"

    // Проверка шага
    if (p.dx == 0) {
      errors += "Шаг dx не может быть нулевым"
    } else {
      val expectedDirection = math.signum(p.xEnd - p.xStart)
      if (math.signum(p.dx) != expectedDirection && expectedDirection
!= 0) {
        errors += "Шаг dx имеет неверное направление"
      }
    }

    val errorList = errors.result()
    if (errorList.isEmpty) Right(()) else Left(errorList)
  }

  private def generateXValues(start: Double, end: Double, step:
Double): LazyList[Double] = {
    LazyList.iterate(start)(_ + step).takeWhile { x =>
```

```
      (step > 0 && x <= end) || (step < 0 && x >= end)
    }
  }

  private def calculateArth(x: Double, epsilon: Double): (Double,
Int) = {
    @tailrec
    def loop(n: Int, sum: Double): (Double, Int) = {
      val exponent = 2 * n + 1
      val term = 1.0 / (exponent * math.pow(x, exponent))
      val newSum = sum + term
      if (math.abs(term) < epsilon) (newSum, n + 1)
      else loop(n + 1, newSum)
    }
    loop(0, 0.0)
  }

  private def printResults(xs: LazyList[Double], epsilon: Double):
Unit = {
    val header = "|      x      |   arth(x)    | Итерации |"
    val separator = "-" * header.length

    println("\n" + header)
    println(separator)

    xs.foreach { x =>
      val (arth, iterations) = calculateArth(x, epsilon)
      println(f"| ${x}%9.4f | ${arth}%10.6f | ${iterations}%8d |")
    }
  }
}
```

3. Результат

scala Main01.scala 1.5 3.0 0.1 0.0001

| x | arth(x) | Итерации |

| 1.5000 | 0.804679 | 9 |

| 1.6000 | 0.733138 | 8 |

1.7000	0.674930	7
1.8000	0.626368	7
1.9000	0.585012	6
2.0000	0.549294	6
2.1000	0.518040	6
2.2000	0.490396	5
2.3000	0.465768	5
2.4000	0.443645	5
2.5000	0.423645	5
2.6000	0.405462	5
2.7000	0.388850	5
2.8000	0.373606	5
2.9000	0.359553	4