Лабораторная работа №4

Выполнил работу

Гапанюк Антон Андреевич

Группа: 6204-010302D

Самара, 2025 г.

Цель работы: расширить возможности пакета для работы с функциями одной переменной добавив интерфейсы и классы для аналитически заданных функций, а также методы ввода и вывода табулированных функций.

Ход работы

Задание 1

В классах **ArrayTabulatedFunction** и **LinkedListTabulatedFunction** добавим конструкторы, получающие сразу все точки функции в виде массива объектов типа **FunctionPoint**. Добавим в них исключения: если точек задано меньше двух, или если точки в массиве не упорядочены по значению абсциссы, конструкторы должны выбрасывать исключение **IllegalArgumentException**.

Для конструктора **LinkedListTabulatedFunction** напишем вспомогательную функцию инициализации списка.

```
// Laba 4: Конструктор создания объекта таб. функции по массиву точек
public LinkedListTabulatedFunction(FunctionPoint[] points){
    // Проверка на кол-во точек
    if (points.length < 2){
        throw new IllegalArgumentException(s: "Кол-во точек должно быть не меньше двух");
    }
    // Проверка упорядоченности точек по x
    for (int i = 1; i < points.length; ++i){
        if (points[i].getX() <= points[i - 1].getX()){
            throw new IllegalArgumentException(s: "Абсциссы функции неупорядочены по возрастанию");
        }
    }

    // Инициализируем список
    initializeListFromPoints(points);
}

// Laba 4: Вспомогательный метод для инициализации списка из массива точек
private void initializeListFromPoints(FunctionPoint[] points){
    // Создаем голову двухсвязного цикклического списка
    head = new FunctionNode(point: null);
    head.setNext(head);
    head.setPrev(head);

    pointsCount = 0;
    lastAccessed = head;
    lastIndex = -1;

    // Добавляем точки в массив
    for (FunctionPoint point: points){
        addNodeToTail().setPoint(new FunctionPoint(point));
    }

}
```

Рис. 1 – Конструктор LinkedListTabulatedFunction

```java
// Lab 4: Конструктор, получающий сразу все точки в виде массива объектов типа FunctionPoint
public ArrayTabulatedFunction(FunctionPoint[] points){
    // Проверка на кол-во точек
    if (points.length < 2){
        throw new IllegalArgumentException(s: "Кол-во точек должно быть не меньше двух");
    }
    // Проверка упорядоченности точек по x
    for (int i = 1; i < points.length; ++i){
        if (points[i].getX() <= points[i - 1].getX()){
            throw new IllegalArgumentException(s: "Абсциссы функции неупорядочены по возрастанию");
        }
    }

    // Создаем копию массива для обеспечения инкапсуляции
    this.points = new FunctionPoint[points.length];
    for (int i = 0; i < points.length; ++i){
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

Рис. 2 – Конструктор ArrayTabulatedFunction

Задание 2

В пакете functions создадим интерфейс Function, описывающий функции одной переменной и содержащий следующие методы:

• **public double getLeftDomainBorder()** – возвращает значение левой границы области определения функции;

• **public double getRightDomainBorder()** – возвращает значение правой границы области определения функции;

• **public double getFunctionValue(double x)** – возвращает значение функции в заданной точке.

Исключим соответствующие методы из интерфейса **TabulatedFunction** и сделаем так, чтобы он расширял интерфейс **Function** (**public interface TabulatedFunction extends Function**).

Теперь табулированные функции буду частным случаем функций одной переменной.

```java
public interface Function {
    // Методы получения границ определения
    public double getLeftDomainBorder();
    public double getRightDomainBorder();
    // Метод возвращения значения функции в заданной точке
    public double getFunctionValue(double x);
}
```

Рис. 3 – Интерфейс **Function**

Задание 3

Создадим пакет **functions.basic**, в нём будут описаны классы ряда функций, заданных аналитически.

Создадим класс **Exp**, который будет вычислять значение экспоненты. Класс будет реализовывать интерфейс **Function**.

```
public class Exp implements Function {
    public double getLeftDomainBorder(){
        return Double.NEGATIVE_INFINITY;
    }
    public double getRightDomainBorder(){
        return Double.POSITIVE_INFINITY;
    }
    public double getFunctionValue(double x) {
        return Math.exp(x);
    }
}
```

Аналогично, создадим класс **Log**, объекты которого должны вычислять значение логарифма по заданному основанию.

```
public class Log implements Function {
    private double basis;
    // Проверка основания
    public Log(double basis){
        if (basis <= 0 || basis == 1){
            throw new IllegalArgumentException("Основание логарифма должно
быть положительным и не равным 1");
        }
        this.basis = basis;
    }
    public double getLeftDomainBorder(){
        return 0; // Логарифм определен для x > 0
    }
    public double getRightDomainBorder(){
        return Double.POSITIVE_INFINITY;
    }
    public double getFunctionValue(double x) {
        if (x <= 0){
            return Double.NaN; // Логарифм не определен для неположительных X
        }
```

```
        return Math.log(x)/Math.log(basis); // log - это ln. По формуле:
ln(x) / ln(a) = log_a(x)
    }
}
```

Создадим абстрактный класс для тригонометрических функций (синуса, косинуса и тангенса). Область определения этих функций совпадает, поэтому в **TrigonomerticFunction** опишем методы получения границ области определения, а в наследующих от него классах **Sin, Cos, Tan** опишем вычисление значения.

```
public abstract class TrigonometricFunction implements Function {
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }
    // Абстрактный метод, который должны реализовать наследники
    public abstract double getFunctionValue(double x);
}
```

Задание 4

Создадим пакет functions.meta, в нём будут описаны классы функций, позволяющие комбинировать функции.

Создадим класс Sum, объекты которого представляют собой функции, являющиеся суммой двух других функций. Класс будет реализовывать интерфейс Function. Конструктор класса будет получать ссылки типа Function на объекты суммируемых функций, а область определения функции будет получаться как пересечение областей определения исходных функций.

Аналогично, создадим класс Mult, объекты которого представляют собой функции, являющиеся произведением двух других функций.

```java
public class Mult implements Function{
    private Function f1;
    private Function f2;

    public Mult(Function f1, Function f2){
        this.f1 = f1;
        this.f2 = f2;
    }

    public double getLeftDomainBorder(){
        // Пересечение областей определения - берем макс. левую границу
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }
    public double getRightDomainBorder(){
        // Пересечение областей определения - берем мин. правую границу
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }

    public double getFunctionValue(double x){
        // Проверям, что x принадлежит пересечению областей определения
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
    }
}
```

```java
public class Sum implements Function{
    private Function f1;
    private Function f2;

    public Sum(Function f1, Function f2){
        this.f1 = f1;
        this.f2 = f2;
    }

    public double getLeftDomainBorder(){
        // Пересечение областей определения - берем макс. левую границу
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }
    public double getRightDomainBorder(){
        // Пересечение областей определения - берем мин. правую границу
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }

    public double getFunctionValue(double x){
        // Проверям, что x принадлежит пересечению областей определения
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
}
```

Рис. 4-5 Классы Sum и Mult

Создадим класс Power, объекты которого представляют собой функции, являющиеся степенью другой функции. Конструктор класса буддет получать ссылку на объекты базовой функции и степень, в которую будет возводиться её значения. Область определения функции будем считать совпадающей с областью определения исходной функции.

```
public class Power implements Function {
    private Function f;
    private double power; // степень

    public Power(double power, Function f){
        this.f = f;
        this.power = power;
    }
    public double getLeftDomainBorder() {
        return f.getLeftDomainBorder();
    }
    public double getRightDomainBorder() {
        return f.getRightDomainBorder();
    }
    public double getFunctionValue(double x){
        // Проверяем, что x принадлежит области определения
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }

        double baseValue = f.getFunctionValue(x);
        if (Double.isNaN(baseValue)) {
            return Double.NaN;
        }
        return Math.pow(baseValue, power);
    }
}
```

Рис. 6 – Класс **Power**

Создадим класс Scale, объекты которого описывают функции, полученные из исходных функций путём масштабирования вдоль осей координат. Конструктор класса будет получать ссылку на объект исходной функции, а также коэффициенты масштабирования вдоль оси абсцисс и оси ординат. Область определения функции будет получаться из области определения исходной функции масштабированием вдоль оси абсцисс, а значение функции – масштабированием значения исходной функции вдоль оси ординат. Коэффициенты масштабирования могут быть отрицательными.

Аналогично, создадим класс Shift, объекты которого описывают функции, полученные из исходных функций путём сдвига вдоль осей координат

```java
public class Scale implements Function{
    private Function f;
    private double scaleX;
    private double scaleY;
    public Scale(Function f, double scaleX, double scaleY){
        this.f = f;
        this.scaleX = scaleX;
        this.scaleY = scaleY;
    }

    public double getLeftDomainBorder() {
        if (scaleX > 0) {
            return f.getLeftDomainBorder() * scaleX;
        } else if (scaleX < 0) {
            return f.getRightDomainBorder() * scaleX;
        } else {
            return Double.NaN; // Масштаб 0 - функция не определена
        }
    }
    public double getRightDomainBorder() {
        if (scaleX > 0) {
            return f.getRightDomainBorder() * scaleX;
        } else if (scaleX < 0) {
            return f.getLeftDomainBorder() * scaleX;
        } else {
            return Double.NaN;
        }
    }
```

```java
    public double getFunctionValue(double x){
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        // Масштабируем x для исходной функции
        double scaledX = x / scaleX;
        double originalValue = f.getFunctionValue(scaledX);
        if (Double.isNaN(originalValue)) {
            return Double.NaN;
        }
        // Масштабируем результат по Y (умножение корректно)
        return originalValue * scaleY;
    }
}
```

```java
public class Shift implements Function {
    private Function f;
    private double shiftX;
    private double shiftY;

    public Shift(Function f, double shiftX, double shiftY){
        this.f = f;
        this.shiftX = shiftX;
        this.shiftY = shiftY;
    }

    public double getLeftDomainBorder() {
        // Сдвигаем область определения по X
        return f.getLeftDomainBorder() - shiftX;
    }
    public double getRightDomainBorder() {
        // Сдвигаем область определения по X
        return f.getRightDomainBorder() - shiftX;
    }

    public double getFunctionValue(double x){
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        // Масштабируем x для исходной функции
        double shiftedX = x - shiftX;
        double originalValue = f.getFunctionValue(shiftedX);
        if (Double.isNaN(originalValue)) {
            return Double.NaN;
        }
        // Масштабируем результат по Y (умножение корректно)
        return originalValue + shiftY;
    }
}
```

Рис. 7-9 – Классы **Scale** и **Shift**

И создадим класс Composition, объекты которого описывают композицию двух исходных функций. Конструктор класса будет получать ссылки на объекты первой и второй функции. Область определения функции будем считать совпадающей с областью определения исходной функции.

```java
public class Composition implements Function{
    private Function outFunc;
    private Function inFunc;

    public Composition(Function outFunc, Function inFunc){
        this.outFunc = outFunc;
        this.inFunc = inFunc;
    }

    // Область определения можно считать областью определения внутренней функции
    public double getLeftDomainBorder(){
        return inFunc.getLeftDomainBorder();
    }
    public double getRightDomainBorder(){
        return inFunc.getLeftDomainBorder();
    }

    public double getFunctionValue(double x){
        // Проверям, что x принадлежит пересечению областей определения
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        double inFuncValue = inFunc.getFunctionValue(x);
        if (Double.isNaN(inFuncValue)) {
            return Double.NaN;
        }

        // Проверяем, что результат внутренней функции принадлежит области определения внешней
        if (inFuncValue < outFunc.getLeftDomainBorder() || inFuncValue > outFunc.getRightDomainBorder()) {
            return Double.NaN;
        }
        // Затем передаем результат во внешнюю функцию
        return outFunc.getFunctionValue(inFuncValue);
    }
}
```

Рис. 10 – Класс Composition

Задание 5

В пакете functions создадим класс Functions, содержащий вспомогательные статические методы для работы с функциями. Сделаем приватный конструктор, чтобы нельзя было создать объект вне класса. Класс будет содержать следующие методы:

public static Function shift(Function f, double shiftX, double shiftY) – возвращает объект функции, полученной из исходной сдвигом вдоль осей;

public static Function scale(Function f, double scaleX, double scaleY) – возвращает объект функции, полученной из исходной масштабированием вдоль осей;

public static Function power(Function f, double power) – возвращает объект функции, являющейся заданной степенью исходной;

public static Function sum(Function f1, Function f2) – возвращает объект функции, являющейся суммой двух исходных;

public static Function mult(Function f1, Function f2) – возвращает объект функции, являющейся произведением двух исходных;

public static Function composition(Function f1, Function f2) – возвращает объект функции, являющейся композицией двух исходных.

```
public class Functions {
    // Приватный конструктор, чтобы нельзя было создать объект класса
    private Functions(){}

    // Сдвиг функции
    public static Function shift(Function f, double shiftX, double shiftY){
        return new Shift(f, shiftX, shiftY);
    }
    // Масштабирование функции
    public static Function scale(Function f, double scaleX, double scaleY){
        return new Scale(f, scaleX, scaleY);
    }
    // Возведение в степень функции
    public static Function power(Function f, double power){
        return new Power(f, power);
    }
    // Сумма функций
    public static Function sum(Function f1, Function f2){
        return new Sum(f1, f2);
    }
    // Произведение функций
    public static Function mult(Function f1, Function f2){
        return new Mult(f1, f2);
    }
    public static Function composition(Function f1, Function f2){
        return new Composition(f1, f2);
    }
```

Рис. 11 – Класс **Functions**

Задание 6

В пакете **functions** создадим класс **TabulatedFunctions**, содержащий вспомогательные статические методы для работы с табулированными функциями. Сделаем конструктор класса приватным, чтобы нельзя было создать его экземпляр

Опишим в классе метод **public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount)**, получающий функцию и возвращающий её табулированный аналог на заданном отрезке с заданным количеством точек.

```java
public class TabulatedFunctions {
    // Приватный конструктор
    private TabulatedFunctions(){}

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount){
        // Проверка, что отрезок табулирования находится в области определения функции
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()){
            throw new IllegalArgumentException(
                "Отрезок табулирования [" + leftX + ", " + rightX + "] " +
                "выходит за область определения функции [" +
                function.getLeftDomainBorder() + ", " + function.getRightDomainBorder() + "]");
        }

        // Проверка на область определения
        if (leftX >= rightX){
            throw new IllegalArgumentException("Левая граница (" + leftX + ") должна быть меньше правой (" + rightX + ")");
        }
        // Проверка на кол-во точек
        if (pointsCount < 2){
            throw new IllegalArgumentException(s: "Кол-во точек должно быть не меньше двух");
        }

        // Создаем массив значений функции
        double[] values = new double[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        // Заполняем массив значениями
        for (int i = 0; i < pointsCount; ++i){
            double x = leftX + i * step;
            values[i] = function.getFunctionValue(x);
        }

        // Возвращаем табулированную функцию (используем ArrayTabulatedFunction)
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }
}
```

Рис. 12 – Класс **TabulatedFunctions**

Задание 7

Добавим в **TabulatedFunctions** следующие методы.

Метод вывода табулированной функции в байтовый поток **public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out),** выводящий значения в указанный поток, по которым потом можно будет восстановить табулированную функцию, а именно количество точек в ней и значения координат точек.

Метод ввода табулированной функции из байтового потока **public static TabulatedFunction inputTabulatedFunction(InputStream in)**, который будет считывать из указанного потока данные о табулированной функции, создавать и настраивать её объект и возвращать его из метода.

Метод записи табулированной функции в символьный поток **public static void writeTabulatedFunction(TabulatedFunction function, Writer out)**, выводящий значения в указанный поток, по которым потом можно будет восстановить табулированную функцию, а именно количество точек в ней и значения координат точек.

Метод чтения табулированной функции из символьного **потока public static TabulatedFunction readTabulatedFunction(Reader in)** который будет считывать из указанного потока данные о табулированной функции, создавать и настраивать её объект и возвращать его из метода.

Заметим, что у нас возникают исключения **IOException**, и с ними нужно что-то сделать.

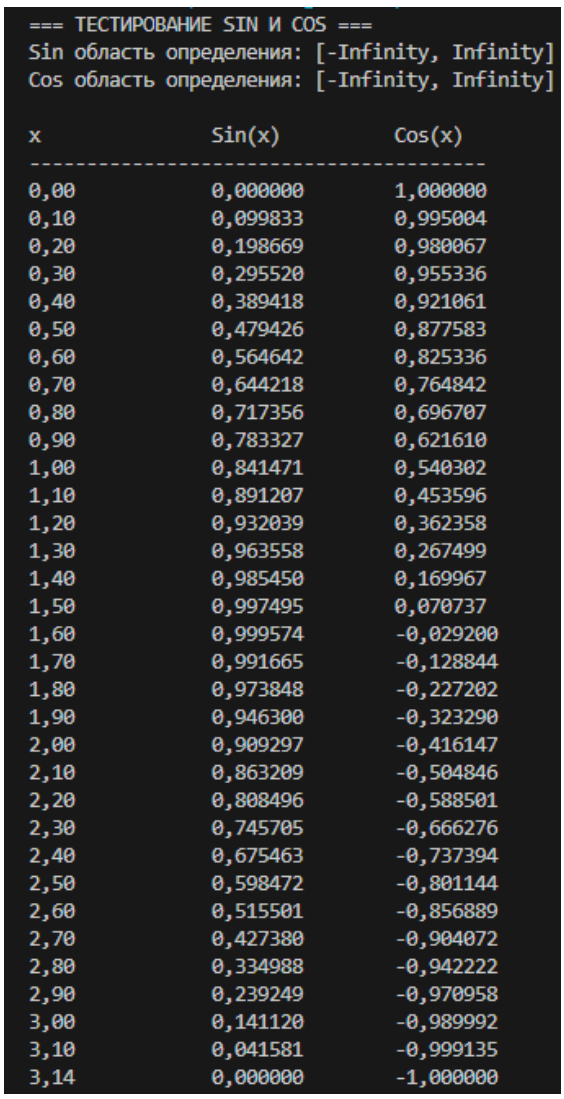
IOException — это системная ошибка, а не логическая. Ошибки ввода-вывода не могут быть обработаны на уровне логики табулированных функций. Это аппаратные/системные ошибки: диск полон, сеть недоступна, файл не существует и т.д.

Также, надо разобраться с тем, закрывать потоки внутри методов или нет. Ответ: нет. Есть несколько аргументов против закрытия потоков:

1. Тот, кто создал поток, должен его закрывать.
2. Наши методы только используют переданные потоки, но не владеют ими
3. Возможность повторного использования потоков
4. Вызывающий код может использовать разные типы потоков.

Задание 8

Проверим работу написанных классов:



```
=== ТЕСТИРОВАНИЕ SIN И COS ===
Sin область определения: [-Infinity, Infinity]
Cos область определения: [-Infinity, Infinity]

x               Sin(x)          Cos(x)
--------------------------------------
0,00            0,000000        1,000000
0,10            0,099833        0,995004
0,20            0,198669        0,980067
0,30            0,295520        0,955336
0,40            0,389418        0,921061
0,50            0,479426        0,877583
0,60            0,564642        0,825336
0,70            0,644218        0,764842
0,80            0,717356        0,696707
0,90            0,783327        0,621610
1,00            0,841471        0,540302
1,10            0,891207        0,453596
1,20            0,932039        0,362358
1,30            0,963558        0,267499
1,40            0,985450        0,169967
1,50            0,997495        0,070737
1,60            0,999574        -0,029200
1,70            0,991665        -0,128844
1,80            0,973848        -0,227202
1,90            0,946300        -0,323290
2,00            0,909297        -0,416147
2,10            0,863209        -0,504846
2,20            0,808496        -0,588501
2,30            0,745705        -0,666276
2,40            0,675463        -0,737394
2,50            0,598472        -0,801144
2,60            0,515501        -0,856889
2,70            0,427380        -0,904072
2,80            0,334988        -0,942222
2,90            0,239249        -0,970958
3,00            0,141120        -0,989992
3,10            0,041581        -0,999135
3,14            0,000000        -1,000000
```

```
=== ТАБУЛИРОВАННЫЕ АНАЛОГИ SIN И COS ===
Табулированный Sin:
Количество точек: 10
Область определения: [0.0, 3.141592653589793]

Табулированный Cos:
Количество точек: 10
Область определения: [0.0, 3.141592653589793]

=== ТОЧКИ ТАБУЛИРОВАННЫХ ФУНКЦИЙ ===
Sin точки:
  [0] x=0,0000, y=0,000000
  [1] x=0,3491, y=0,342020
  [2] x=0,6981, y=0,642788
  [3] x=1,0472, y=0,866025
  [4] x=1,3963, y=0,984808
  [5] x=1,7453, y=0,984808
  [6] x=2,0944, y=0,866025
  [7] x=2,4435, y=0,642788
  [8] x=2,7925, y=0,342020
  [9] x=3,1416, y=0,000000

Cos точки:
  [0] x=0,0000, y=1,000000
  [1] x=0,3491, y=0,939693
  [2] x=0,6981, y=0,766044
  [3] x=1,0472, y=0,500000
  [4] x=1,3963, y=0,173648
  [5] x=1,7453, y=-0,173648
  [6] x=2,0944, y=-0,500000
  [7] x=2,4435, y=-0,766044
  [8] x=2,7925, y=-0,939693
  [9] x=3,1416, y=-1,000000
```

Рис. 13 - Значение функций sin(x) и cos(x)

```
=== СРАВНЕНИЕ ИСХОДНЫХ И ТАБУЛИРОВАННЫХ ФУНКЦИЙ ===
x               Sin(x)          TabSin(x)       Разница         Cos(x)          TabCos(x)       Разница
-------------------------------------------------------------------------------------------------------
0,00            0,000000        0,000000        0,000000        1,000000        1,000000        0,000000
0,10            0,099833        0,097982        0,001852        0,995004        0,982723        0,012281
0,20            0,198669        0,195963        0,002706        0,980067        0,965446        0,014620
0,30            0,295520        0,293945        0,001576        0,955336        0,948170        0,007167
0,40            0,389418        0,385907        0,003512        0,921061        0,914355        0,006706
0,50            0,479426        0,472070        0,007355        0,877583        0,864608        0,012974
0,60            0,564642        0,558234        0,006409        0,825336        0,814862        0,010474
0,70            0,644218        0,643982        0,000235        0,764842        0,764620        0,000222
0,80            0,717356        0,707935        0,009421        0,696707        0,688404        0,008302
0,90            0,783327        0,771888        0,011439        0,621610        0,612188        0,009422
1,00            0,841471        0,835841        0,005630        0,540302        0,535972        0,004330
1,10            0,891207        0,883993        0,007214        0,453596        0,450633        0,002963
1,20            0,932039        0,918022        0,014017        0,362358        0,357141        0,005217
1,30            0,963558        0,952051        0,011508        0,267499        0,263648        0,003851
1,40            0,985450        0,984808        0,000642        0,169967        0,169931        0,000037
1,50            0,997495        0,984808        0,012687        0,070737        0,070437        0,000300
1,60            0,999574        0,984808        0,014766        -0,029200       -0,029056       0,000144
1,70            0,991665        0,984808        0,006857        -0,128844       -0,128549       0,000296
1,80            0,973848        0,966204        0,007644        -0,227202       -0,224761       0,002441
1,90            0,946300        0,932175        0,014125        -0,323290       -0,318254       0,005035
2,00            0,909297        0,898147        0,011151        -0,416147       -0,411747       0,004400
2,10            0,863209        0,862441        0,000768        -0,504846       -0,504272       0,000574
2,20            0,808496        0,798488        0,010008        -0,588501       -0,580488       0,008013
2,30            0,745705        0,734535        0,011170        -0,666276       -0,656704       0,009572
2,40            0,675463        0,670582        0,004881        -0,737394       -0,732920       0,004474
2,50            0,598472        0,594072        0,004401        -0,801144       -0,794171       0,006973
2,60            0,515501        0,507908        0,007593        -0,856889       -0,843917       0,012972
2,70            0,427380        0,421745        0,005635        -0,904072       -0,893664       0,010408
2,80            0,334988        0,334698        0,000290        -0,942222       -0,940984       0,001239
2,90            0,239249        0,236716        0,002533        -0,970958       -0,958261       0,012698
3,00            0,141120        0,138735        0,002385        -0,989992       -0,975537       0,014455
3,10            0,041581        0,040753        0,000828        -0,999135       -0,992814       0,006321

=== АНАЛИЗ ТОЧНОСТИ ===
Максимальная ошибка Sin: 0,01519193
Максимальная ошибка Cos: 0,01495977
Средняя ошибка Sin: 0,00646140
Средняя ошибка Cos: 0,00638619
```

Рис. 14 – Сравнение исходных тригонометрических функций и их табулированный вариант.

```
=== СУММА КВАДРАТОВ ТАБУЛИРОВАННЫХ SIN И COS ===

--- Количество точек: 5 ---
x               Sin^2+Cos^2     Ожидаемое       Ошибка
-------------------------------------------------
0,00            1,000000        1,0             0,000000
0,10            0,934912        1,0             0,065088
0,20            0,888816        1,0             0,111184
0,30            0,861714        1,0             0,138286
0,40            0,853604        1,0             0,146396
0,50            0,864487        1,0             0,135513
0,60            0,894363        1,0             0,105637
0,70            0,943232        1,0             0,056768
0,80            0,989312        1,0             0,010688
0,90            0,926997        1,0             0,073003
1,00            0,883675        1,0             0,116325
1,10            0,859345        1,0             0,140655
1,20            0,854009        1,0             0,145991
1,30            0,867665        1,0             0,132335
1,40            0,900315        1,0             0,099685
1,50            0,951957        1,0             0,048043
1,60            0,979028        1,0             0,020972
1,70            0,919487        1,0             0,080513
1,80            0,878938        1,0             0,121062
1,90            0,857382        1,0             0,142618
2,00            0,854819        1,0             0,145181
2,10            0,871249        1,0             0,128751
2,20            0,906671        1,0             0,093329
2,30            0,961086        1,0             0,038914
2,40            0,969150        1,0             0,030850
2,50            0,912382        1,0             0,087618
2,60            0,874606        1,0             0,125394
2,70            0,855824        1,0             0,144176
2,80            0,856034        1,0             0,143966
2,90            0,875237        1,0             0,124763
3,00            0,913432        1,0             0,086568
3,10            0,970621        1,0             0,029379

Статистика для 5 точек:
Максимальная ошибка: 0,14639599
Средняя ошибка: 0,09592664
```

```
--- Количество точек: 10 ---
x               Sin^2+Cos^2     Ожидаемое       Ошибка
-------------------------------------------------
0,00            1,000000        1,0             0,000000
0,10            0,975345        1,0             0,024655
0,20            0,970488        1,0             0,029512
0,30            0,985429        1,0             0,014571
0,40            0,984968        1,0             0,015032
0,50            0,970398        1,0             0,029602
0,60            0,975624        1,0             0,024376
0,70            0,999358        1,0             0,000642
0,80            0,975073        1,0             0,024927
0,90            0,970586        1,0             0,029414
1,00            0,985897        1,0             0,014103
1,10            0,984515        1,0             0,015485
1,20            0,970314        1,0             0,029686
1,30            0,975910        1,0             0,024090
1,40            0,998723        1,0             0,001277
1,50            0,974808        1,0             0,025192
1,60            0,970691        1,0             0,029309
1,70            0,986371        1,0             0,013629
1,80            0,984068        1,0             0,015932
1,90            0,970237        1,0             0,029763
2,00            0,976203        1,0             0,023797
2,10            0,998094        1,0             0,001906
2,20            0,974549        1,0             0,025451
2,30            0,970802        1,0             0,029198
2,40            0,986852        1,0             0,013148
2,50            0,983628        1,0             0,016372
2,60            0,970167        1,0             0,029833
2,70            0,976503        1,0             0,023497
2,80            0,997473        1,0             0,002527
2,90            0,974298        1,0             0,025702
3,00            0,970920        1,0             0,029080
3,10            0,987341        1,0             0,012659

Статистика для 10 точек:
Максимальная ошибка: 0,02983318
Средняя ошибка: 0,01951143
```

```
Статистика для 20 точек:
Максимальная ошибка: 0,00681713
Средняя ошибка: 0,00449588

--- Количество точек: 50 ---
x               Sin^2+Cos^2     Ожидаемое       Ошибка
-------------------------------------------------
0,00            1,000000        1,0             0,000000
0,10            0,998987        1,0             0,001013
0,20            0,999568        1,0             0,000432
0,30            0,999105        1,0             0,000895
0,40            0,999253        1,0             0,000747
0,50            0,999339        1,0             0,000661
0,60            0,999055        1,0             0,000945
0,70            0,999691        1,0             0,000309
0,80            0,998975        1,0             0,001025
0,90            0,999852        1,0             0,000148
1,00            0,999012        1,0             0,000988
1,10            0,999456        1,0             0,000544
1,20            0,999166        1,0             0,000834
1,30            0,999178        1,0             0,000822
1,40            0,999437        1,0             0,000563
1,50            0,999017        1,0             0,000983
1,60            0,999825        1,0             0,000175
1,70            0,998974        1,0             0,001026
1,80            0,999715        1,0             0,000285
1,90            0,999047        1,0             0,000953
2,00            0,999357        1,0             0,000643
2,10            0,999238        1,0             0,000762
2,20            0,999115        1,0             0,000885
2,30            0,999546        1,0             0,000454
2,40            0,998991        1,0             0,001009
2,50            0,999971        1,0             0,000029
2,60            0,998984        1,0             0,001016
2,70            0,999590        1,0             0,000410
2,80            0,999094        1,0             0,000906
2,90            0,999268        1,0             0,000732
3,00            0,999322        1,0             0,000678
3,10            0,999064        1,0             0,000936
```

```
--- Количество точек: 50 ---
x               Sin^2+Cos^2     Ожидаемое       Ошибка
-------------------------------------------------
0,00            1,000000        1,0             0,000000
0,10            0,998987        1,0             0,001013
0,20            0,999568        1,0             0,000432
0,30            0,999105        1,0             0,000895
0,40            0,999253        1,0             0,000747
0,50            0,999339        1,0             0,000661
0,60            0,999055        1,0             0,000945
0,70            0,999691        1,0             0,000309
0,80            0,998975        1,0             0,001025
0,90            0,999852        1,0             0,000148
1,00            0,999012        1,0             0,000988
1,10            0,999456        1,0             0,000544
1,20            0,999166        1,0             0,000834
1,30            0,999178        1,0             0,000822
1,40            0,999437        1,0             0,000563
1,50            0,999017        1,0             0,000983
1,60            0,999825        1,0             0,000175
1,70            0,998974        1,0             0,001026
1,80            0,999715        1,0             0,000285
1,90            0,999047        1,0             0,000953
2,00            0,999357        1,0             0,000643
2,10            0,999238        1,0             0,000762
2,20            0,999115        1,0             0,000885
2,30            0,999546        1,0             0,000454
2,40            0,998991        1,0             0,001009
2,50            0,999971        1,0             0,000029
2,60            0,998984        1,0             0,001016
2,70            0,999590        1,0             0,000410
2,80            0,999094        1,0             0,000906
2,90            0,999268        1,0             0,000732
3,00            0,999322        1,0             0,000678
3,10            0,999064        1,0             0,000936

Статистика для 50 точек:
Максимальная ошибка: 0,00102635
Средняя ошибка: 0,00068151
```

Рис. 15-18 – Сумма квадратов синуса и косинуса

Заметим, что чем больше мы берем, тем меньше ошибка.

```
=== ТЕСТИРОВАНИЕ ЭКСПОНЕНТЫ ===
Исходная табулированная экспонента:
Количество точек: 11
Область определения: [0.0, 10.0]

Точки исходной функции:
  [0] x=0,0, y=1,000000
  [1] x=1,0, y=2,718282
  [2] x=2,0, y=7,389056
  [3] x=3,0, y=20,085537
  [4] x=4,0, y=54,598150
  [5] x=5,0, y=148,413159
  [6] x=6,0, y=403,428793
  [7] x=7,0, y=1096,633158
  [8] x=8,0, y=2980,957987
  [9] x=9,0, y=8103,083928
  [10] x=10,0, y=22026,465795

Функция записана в файл: exp_function.txt

Функция прочитана из файла: exp_function.txt
Прочитанная табулированная экспонента:
Количество точек: 11
Область определения: [0.0, 10.0]

Точки прочитанной функции:
  [0] x=0,0, y=1,000000
  [1] x=1,0, y=2,718282
  [2] x=2,0, y=7,389056
  [3] x=3,0, y=20,085537
  [4] x=4,0, y=54,598150
  [5] x=5,0, y=148,413159
  [6] x=6,0, y=403,428793
  [7] x=7,0, y=1096,633158
  [8] x=8,0, y=2980,957987
  [9] x=9,0, y=8103,083928
  [10] x=10,0, y=22026,465795
```

```
=== СРАВНЕНИЕ ИСХОДНОЙ И ПРОЧИТАННОЙ ФУНКЦИЙ ===
x               Исходная        Прочитанная     Разница
-----------------------------------------------------------
0,0             1,000000        1,000000        0,0000000000
1,0             2,718282        2,718282        0,0000000000
2,0             7,389056        7,389056        0,0000000000
3,0             20,085537       20,085537       0,0000000000
4,0             54,598150       54,598150       0,0000000000
5,0             148,413159      148,413159      0,0000000000
6,0             403,428793      403,428793      0,0000000000
7,0             1096,633158     1096,633158     0,0000000000
8,0             2980,957987     2980,957987     0,0000000000
9,0             8103,083928     8103,083928     0,0000000000
10,0            22026,465795    22026,465795    0,0000000000

=== СТАТИСТИКА ТОЧНОСТИ ===
Максимальная ошибка: 0,0000000000
Средняя ошибка: 0,0000000000
```

Рис. 19-20 – Тестирование экспоненты

```
=== ТЕСТИРОВАНИЕ НАТУРАЛЬНОГО ЛОГАРИФМА ===
Исходная табулированная логарифмическая функция:
Количество точек: 11
Область определения: [0.1, 10.0]

Точки исходной функции:
  [0] x=0,1, y=-2,302585
  [1] x=1,1, y=0,086178
  [2] x=2,1, y=0,732368
  [3] x=3,1, y=1,121678
  [4] x=4,1, y=1,401183
  [5] x=5,1, y=1,619388
  [6] x=6,0, y=1,798404
  [7] x=7,0, y=1,950187
  [8] x=8,0, y=2,081938
  [9] x=9,0, y=2,198335
  [10] x=10,0, y=2,302585

Функция записана в бинарный файл: log_function.dat
Функция также записана в текстовый файл: log_function.txt

Функция прочитана из бинарного файла: log_function.dat
Прочитанная табулированная логарифмическая функция:
Количество точек: 11
Область определения: [0.1, 10.0]

Точки прочитанной функции:
  [0] x=0,1, y=-2,302585
  [1] x=1,1, y=0,086178
  [2] x=2,1, y=0,732368
  [3] x=3,1, y=1,121678
  [4] x=4,1, y=1,401183
  [5] x=5,1, y=1,619388
  [6] x=6,0, y=1,798404
  [7] x=7,0, y=1,950187
  [8] x=8,0, y=2,081938
  [9] x=9,0, y=2,198335
  [10] x=10,0, y=2,302585
```

```
=== СРАВНЕНИЕ ИСХОДНОЙ И ПРОЧИТАННОЙ ФУНКЦИЙ ===
x               Исходная        Прочитанная     Разница
-----------------------------------------------------------
0,1             -2,302585       -2,302585       0,0000000000
1,1             0,092705        0,092705        0,0000000000
2,1             0,740233        0,740233        0,0000000000
3,1             1,130147        1,130147        0,0000000000
4,1             1,409999        1,409999        0,0000000000
5,1             1,628429        1,628429        0,0000000000
6,1             1,807603        1,807603        0,0000000000
7,1             1,959502        1,959502        0,0000000000
8,1             2,091344        2,091344        0,0000000000
9,1             2,207812        2,207812        0,0000000000

=== СТАТИСТИКА ТОЧНОСТИ ===
Максимальная ошибка: 0,0000000000
Средняя ошибка: 0,0000000000
```

Рис. 21-22 Тестирование натурального логарифма

Сделаем некоторое заключение о форматах хранения:

Текстовый формат (writeTabulatedFunction/readTabulatedFunction)

Преимущества:

- **Человеко-читаемый** - можно открыть в любом текстовом редакторе
- **Легко отлаживать** - видно все данные и их структуру
- **Совместимость** - работает на любой платформе и архитектуре
- **Редактируемость** - можно вручную исправить данные
- **Простота** - понятный формат: количество точек, затем пары x y

Недостатки:

- **Большой размер** - 16 байт на точку против 8 в бинарном
- **Медленная обработка** - преобразование чисел в строки и обратно
- **Потери точности** - при преобразовании double → string → double
- **Нет типизации** - все данные хранятся как текст

Бинарный формат (outputTabulatedFunction/inputTabulatedFunction)

Преимущества:

- **Компактность** - в 2 раза меньше места (~8 байт на точку)
- **Высокая скорость** - прямое чтение/запись примитивных типов
- **Точность** - без потерь при преобразовании
- **Эффективность** - минимальные накладные расходы

Недостатки:

- **Нечитаемость** - нельзя открыть в текстовом редакторе
- **Сложность отладки** - непонятное содержимое файла
- **Нет редактирования** - нельзя исправить данные вручную

Задание 9

Сделаем так, чтобы объекты всех классов, реализующих интерфейс **TabulatedFunction**, были сериализуемыми.

Рассмотрим два случая:

1. с использованием интерфейса java.io.Serializable

2. с использованием интерфейса java.io.Externalizable

```
--- Тестирование Serializable ---
Оригинальная функция ln(exp(x)) = x (0 до 10, 11 точек):
f(0,0) = NaN
f(1,0) = 1,000000
f(2,0) = 2,000000
f(3,0) = 3,000000
f(4,0) = 4,000000
f(5,0) = 5,000000
f(6,0) = 6,000000
f(7,0) = 7,000000
f(8,0) = 8,000000
f(9,0) = 9,000000
f(10,0) = 10,000000
Десериализованная функция:
f(0,0) = NaN
f(1,0) = 1,000000
f(2,0) = 2,000000
f(3,0) = 3,000000
f(4,0) = 4,000000
f(5,0) = 5,000000
f(6,0) = 6,000000
f(7,0) = 7,000000
f(8,0) = 8,000000
f(9,0) = 9,000000
f(10,0) = 10,000000
```

Рис. 23 – Тестирование Serializable

```
=== ТЕСТИРОВАНИЕ СЕРИАЛИЗАЦИИ ===
Исходная функция: ln(exp(x))
Теоретически должна быть: f(x) = x

--- Тестирование Externalizable ---
Функция сериализована в: function_Externalizable.ser
Функция десериализована из: function_Externalizable.ser

Сравнение исходной и десериализованной функций:
x               Исходная        Десериализ.     Разница
-------------------------------------------------
0,0             0,000000        0,000000        0,0000000000
1,0             1,000000        1,000000        0,0000000000
2,0             2,000000        2,000000        0,0000000000
3,0             3,000000        3,000000        0,0000000000
4,0             4,000000        4,000000        0,0000000000
5,0             5,000000        5,000000        0,0000000000
6,0             6,000000        6,000000        0,0000000000
7,0             7,000000        7,000000        0,0000000000
8,0             8,000000        8,000000        0,0000000000
9,0             9,000000        9,000000        0,0000000000
10,0            10,000000       10,000000       0,0000000000
```

Рис. 24 – Тестирование Externalizable

Serializable подходит для:

- Быстрого прототипирования
- Простых объектов без требований к производительности
- Когда важна простота, а не эффективность

Externalizable подходит для:

- Высокопроизводительных приложений
- Больших объемов данных
- Когда важен размер файлов
- Для библиотечных классов