Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Математический анализ и основы вычислений

Лабораторная работа №1

Студент: Вдовин Герман Евгеньевич

Группа: P3122

Преподаватель: Табиева Арина Вадимовна

САНКТ-ПЕТЕРБУРГ
2025

Задание

Реализовать несколько алгоритмов.
Часть 1 - Алгоритмы нахождения минимумов функций на отрезке. Метод дихотомии(он же половинное деление) или метод золотого сечения.

В своей лабораторной работе я реализовал сразу оба варианта.

Часть 2 - Алгоритм нахождения корня функции на отрезке. Реализуем тот же алгоритм половинного деления, но будем искать значения, когда `f(x)=0`.

Ход решения

писал на java. но не стал сильно запариваться, реализовал все требования в одном классе

задача 1 - прочитал условие, вспомнил, что я раньше уже это делал, и выдал вот такое:

```java
static IFuncMinimumFinder dichotomy = new IFuncMinimumFinder() {
    public ArrayList<Double> run(Function<Double, Double> func, double eps, double a, double b) {

        ArrayList<Double> answer = new ArrayList<Double>();

        double mid, y, z, d = eps / 2;

        while ((b - a) > eps) {
            mid = (a + b) / 2;
            y = mid - d;
            z = mid + d;

            if (func.apply(y) <= func.apply(z)) {
                b = z;
                answer.add((a + b) / 2);
            } else {
                a = y;
                answer.add((a + b) / 2);
            }
        }

        return answer;
    };
};
```

задача 2 - да, по условию достаточно сделать что-то одно, но почему бы и нет? метод золотого сечения - что-то интересное, дополнительно копался в гугле. понял, что метод мало чем отличается от предыдущего и выдал вот такое:

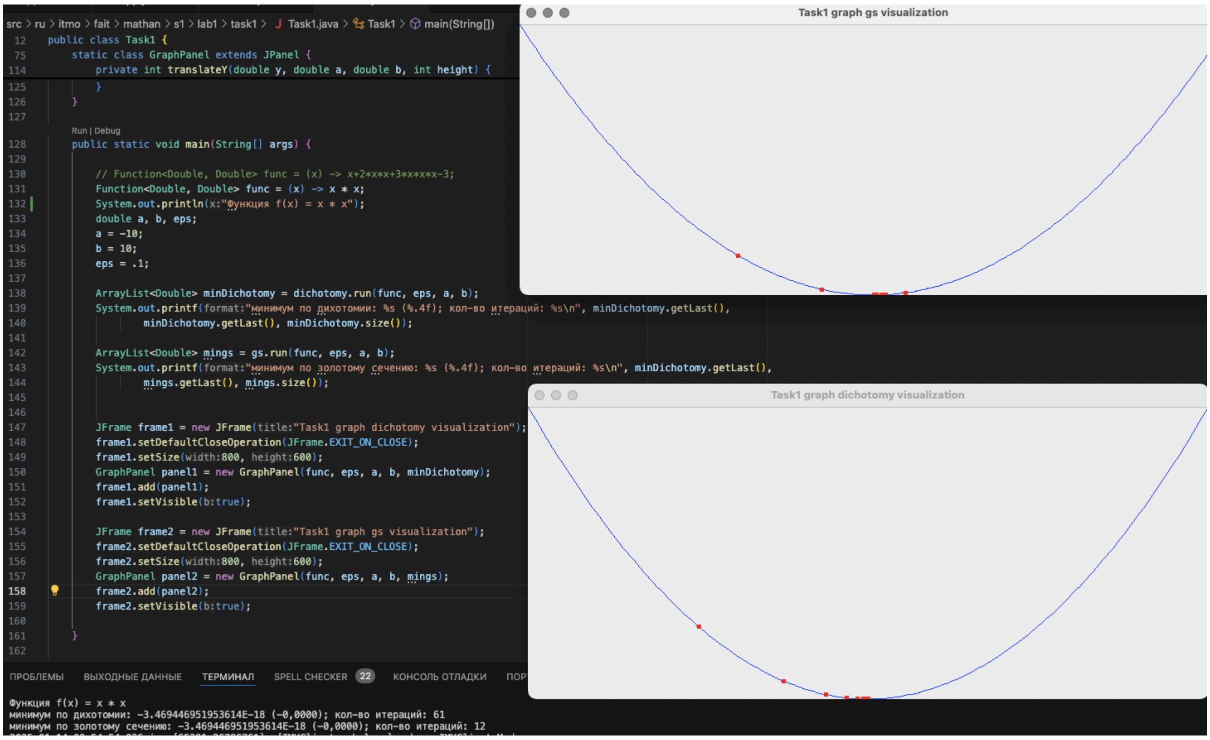
```java
static IFuncMinimumFinder gs = new IFuncMinimumFinder() {
    private static final double PHI = 1.618033988749894848820d;

    @Override
    public ArrayList<Double> run(Function<Double, Double> func, double eps, double a, double b) {
        ArrayList<Double> answer = new ArrayList<>();

        double y, z;
        y = b - (b - a) / PHI;
        z = a + (b - a) / PHI;

        while ((b - a) > eps) {
            if (func.apply(y) <= func.apply(z)) {
                b = z;
                z = y;
                y = b - (b - a) / PHI;
                answer.add((a + b) / 2);
            } else {
                a = y;
                y = z;
                z = a + (b - a) / PHI;
                answer.add((a + b) / 2);
            }
        }

        return answer;
    }

};
```

потыкав несколько функций - золотое сечение всегда работало быстрее, чем половинное деление. 12 итераций против 61

далее самое веселое - отрисовать графики. не без помощи, но расписал вот такую штуку. был вариант использовать готовые библиотеки, но отрисовывать графику самому - круто!

```java
static class GraphPanel extends JPanel {
    private final Function<Double, Double> func;
    private final double a, b, eps;
    private final ArrayList<Double> minima;

    public GraphPanel(Function<Double, Double> func, double eps, double a, double b, ArrayList<Double> minima) {
        this.func = func;
        this.eps = eps;
        this.a = a;
        this.b = b;
        this.minima = minima;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;

        g2.setColor(Color.BLUE);
        for (double x = a; x < b; x += eps) {
            int x1 = translateX(x, a, b, getWidth());
            int y1 = translateY(func.apply(x), a, b, getHeight());
            int x2 = translateX(x + eps, a, b, getWidth());
            int y2 = translateY(func.apply(x + eps), a, b, getHeight());
            g2.drawLine(x1, y1, x2, y2);
        }

        g2.setColor(Color.RED);
        for (double x : minima) {
            int xCoord = translateX(x, a, b, getWidth());
            int yCoord = translateY(func.apply(x), a, b, getHeight());
            g2.fillOval(xCoord - 3, yCoord - 3, width:6, height:6);
        }
    }

    private int translateX(double x, double a, double b, int width) {
        return (int) ((x - a) / (b - a) * (width));
    }

    private int translateY(double y, double a, double b, int height) {
        double maxY = Double.NEGATIVE_INFINITY;
        double minY = Double.POSITIVE_INFINITY;

        for (double x = a; x <= b; x += eps) {
            double currentY = func.apply(x);
            maxY = Math.max(maxY, currentY);
            minY = Math.min(minY, currentY);
        }

        return (int) ((maxY - y) / (maxY - minY) * height);
    }
}
```

по итогам получаем вот такой вывод для функции `f(x) = x^2`



Задача 2:

алгоритм по нахождению корня функци.

возможно не лучшее решение, но я решил использовать все тот же половинчатый метод, но для нахождения одного корня.

```java
static RootFinder bisection = new RootFinder() {

    @Override
    public ArrayList<Double> findRoot(Function<Double, Double> func, double eps, double a, double b) {
        ArrayList<Double> answer = new ArrayList<>();

        double c;

        while ((b - a) / 2 > eps) {
            c = (a + b) / 2;

            if (Math.abs(func.apply(c)) < eps) {
                answer.add(c);
                return answer;
            } else if (func.apply(a) * func.apply(c) < 0) {
                b = c;
                answer.add(b);
            } else {
                a = c;
                answer.add(a);
            }
        }
        return answer;
    }

};
```

вывод для `f(x) = x - 4` eps=.001, [a,b] = [-2,2] :

Функция f(x) = x - 4
Корень: 1.998046875 (1,9980) за 11 итераций, rds от ожидаемого: 0,6963

Исходный код: [ссылка]()