Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Лабораторная работа №1
по дисциплине
“Математический анализ и основы исчислений”

Семестр I

Выполнили:
студенты

Тиганов Вадим Игоревич
гр. J3112
ИСУ 467701

Вагин Арсений Антонович
гр. J3112
ИСУ 465339

Волков Владимир Алексеевич
гр. J3112
ИСУ 465430

Отчет сдан:
xx.01.2025

Санкт-Петербург
2025

Цель лабораторной работы

Цель работы заключалась в следующем: ознакомиться с критериями унимодальности функции, оценкой экстремума без определения производной функции и решением задачи нахождения корней на заданном отрезке, применить полученные знания на практике для выполнения заданий лабораторной работы.

Задачи лабораторной работы

С помощью выбранного языка программирования реализовать функции для двух заданий:

1. Реализовать поиск экстремума функции, имея только метод определения $f(x)$ при заданном x. Был выбрал метод Дихотомии. Про него с теоретической и практической стороны см. далее.

2. Реализовать программу, которая решит уравнение $f(x) = 0$, $x \in [a,b]$, $f(x)$ определена и непрерывна на $[a,b]$. Проанализировать отклонения и обработать исключения.

3. Провести ручное и автоматическое тестирование программы, убедиться в корректности работы.

Теоретическая подготовка

Использованные средства и инструменты для выполнения лабораторной работы:

- Использовали язык программирования Java.
- Использованы стандартные Java библиотеки для ввода математической функции от пользователя (для тестов и прочего) и построения графиков.
- Использованы стандартные библиотеки для написания *Unit-тестов.*
- Работа проводилась совместно с помощью *Github*, задачи были распределены между участниками.
- Весь проект был собран с помощью сборщика $Maven$ для удобного запуска и тестирования на любом устройстве с JVM (виртуальная машина $Java$, для запуска скомпилированного $.jar$ архива).

Теоретическая подготовка, метод Дихотомии

Метод Дихотомии для определения экстремума функции (по совместительству и для нахождения единственного корня) на заданном промежутке, если функция в конечных точках этого промежутка имеет разные знаки, определена на всем промежутке и непрерывна на нем. (согласно теореме о промежуточном значении: если на отрезке функция имеет разные знаки, то где-то между этими двумя точками существует хотя бы один корень функции)

Метод чем-то напоминает бинарный поиск. Суть его работы алгоритмически:

1. Определяется промежуток, на котором требуется найти корень. Предполагается, что условия смены знака и непрерывности выполнены.
2. Вычисляется середина отрезка с помощью формулы $m = \frac{a+b}{2}$.
3. Если $f(m) = 0$, то корень найден.
4. Если $f(m) \cdot f(a) < 0$, то корень лежит в интервале $[a, m]$, и обновляем правую границу: $b = m$.
5. Если $f(m) \cdot f(b) < 0$, то корень лежит в интервале $[m, b]$, и обновляем левую границу: $a = m$.
6. Процесс повторяется, пока длина интервала не станет достаточно малой, что означает приближённое значение корня.

В программе реализован ручной ввод пользователем математической функции и границ, в которых требуется изучить функцию и найти приблизительное значение экстремума.

Согласно техническому заданию, предполагаемый экстремум на каждой итерации добавляется на график, строится график сужения интервала поиска после каждой итерации, выводится график со всеми точками предполагаемых экстремумов и динамики уменьшения зоны поиска.

Теоретическая подготовка, решение уравнения вида *f(x) = 0*

Требовалось реализовать программу, которая решит следующую задачу: Для заданной функции $f(x)$, гарантированно непрерывной и определенной на интервале $[a, b]$ найти решение уравнения $f(x) = 0$.
При этом реализован функционал: ввод пользователем интересующих границ для изучения функции, ввод заведомо верного корня, подсчет среднеквадратичного отклонения найденного решения по отношению к верному. (Предполагается, что оно известно заранее и введено корректно)

По большому счету, данная программа представляет собой доработанную версию предыдущей, в которой был использован принцип дихотомии.

Произведенные улучшения:

- Обработка случаев с несколькими корнями.
- Вывод нескольких корней и расчет отклонения для каждого из них.
- Согласно техническому заданию, обработка случая, когда корней нет. (реализовано и протестировано также в первом)

Суть работы алгоритмически:

1. Имеем зону поиска и требуемую точность.
2. Определяем, есть ли корни вообще:
3. Итерируемся через всю зону изначального поиска, проверяя концы на различие в знаках.
4. Если знаки различаются, корень есть, ищем его и добавляем в список корней.

$$f(x) \cdot f(x + \epsilon) < 0,\ x \in [a, b],\ \epsilon > 0$$

5. Повторяем итерирование до конца отрезка.

$$x + \epsilon \leq b, x \in [a, b]$$

Этап реализации алгоритмов

Задание 1, метод Дихотомии

Коротко про реализацию на выбранном языке программирования:

Главная функция, поиск экстремума:

```java
public static float dichotomyMethod(String expression, float a, float b
    , float tolerance) {
    if (func(expression, a) * func(expression, b) > 0) {
        throw new IllegalArgumentException("f(a) and f(b) must have
            opposite signs!");
    }

    float c;
    XYSeries series = new XYSeries("Iterations");
    ArrayList<Float> intervalLengths = new ArrayList<>();

    while ((b - a) / 2.0 > tolerance) {
        c = (a + b) / 2;
        series.add(c, func(expression, c));

        intervalLengths.add((b - a));

        if (func(expression, c) == 0) {
            return c;
        } else if (func(expression, c) * func(expression, a) < 0) {
            b = c;
        } else {
            a = c;
        }
    }

    series.add((a + b) / 2, func(expression, (a + b) / 2));

    showGraph(series);

    plotIntervalLengths(intervalLengths);

    return (a + b) / 2;
}
```

Listing 1: Java Code for Dichotomy Method

Определяется функция dichotomyMethod, которая принимает в себя следующие аргументы: введенную пользователем функцию, правую и левую границы интересующей области, (float для большей точности, чтобы не ограничиваться целыми числами) точность работы.

Согласно алгоритму, концы интересующего отрезка должны иметь противоположные знаки, поэтому обрабатываем исключение вв противном случае.

XY серии и список с длинами интервала понадобится далее для построения графиков.

Тело функции - цикл. Пока находимся в рамках погрешности, введенной пользователем, смотрим в середину отрезка, добавляем ее в список возможных экстремумов, также добавляем в свой список длину текущего интервала.
Далее все по алгоритму, если попали в ноль, то экстремум найден. Если нет, то оцениваем знаки и сдвигаем границы области поиска.
В конце, когда вышли за пределы точности, добавляем последнюю найденную точку для отображения на графике, возвращаем ее же как ответ и рисуем график со всеми промежуточными значениями, а также длинами интервалов при итерациях.

Полный код с пояснениями см. в приложениях.

Задание 2, решение уравнения

```
public static List<Double> findRoots(String function, double a, double
    b, double epsilon) {
     Expression expression = new ExpressionBuilder(function)
             .variables("x")
             .build();

     List<Double> roots = new ArrayList<>();
     double step = epsilon;

     for (double x = a; x <= b; x += step) {
         if (f(expression, x) * f(expression, x + step) < 0) {
             double root = findRootInInterval(expression, x, x + step,
                epsilon);
             roots.add(root);
         }
     }

     return roots;
}

private static double findRootInInterval(Expression expression, double
    a, double b, double epsilon) {
     if (f(expression, a) * f(expression, b) >= 0) {
         return Double.NaN;
     }

     double c = a;
     int iterations = 0;

     while ((b - a) >= epsilon) {
         c = (a + b) / 2;
         if (f(expression, c) == 0.0) {
             break;
         } else if (f(expression, c) * f(expression, a) < 0) {
             b = c;
         } else {
             a = c;
```

```
35          }
36          iterations++;
37      }
38
39      System.out.println("Number of iterations: " + iterations);
40      return c;
41  }
```

Listing 2: Java Code for Finding Multiple Roots

Для данного задания пришлось реализовать сразу две функции. Главную, которая будет выводить и собирать все найденные корни и функцию поиска корня на каждом из отрезков. (вторая функция действует по принципу дихотомии, пояснено в предыдущем задании)

В первой функции поиска всех корней инициализируем математическую функцию с помощью библиотеки-билдера специально для этой задачи. (*ExpressionBuilder*)

Инициализируем список корней, инициализируем шаг размера ϵ для *"передвижения"* по интервалу.

```
1 for (double x = a; x <= b; x += step) {
2         if (f(expression, x) * f(expression, x + step) < 0) {
3             double root = findRootInInterval(expression, x, x + step,
                  epsilon);
4             roots.add(root);
5         }
6     }
```

Listing 3: Java Code, Main cycle

Далее в цикле: если знаки на концах различны, следовательно, есть корень, вызываем функцию поиска корня на отрезке и заносим найденный корень в список корней. Возвращаем список корней.
Задача выполнена.

Реализован вывод всех найденных корней и отклонения для каждого из них, полный код см. в приложении

Выводы

1. В ходе лабораторной работы мы изучили основные методы нахождения корней функций, такие как метод дихотомии.

2. Метод дихотомии (или бисекции) — это простой и надежный способ найти корень функции. Он основан на теореме, которая говорит, что если непрерывная функция принимает значения разных знаков на концах интервала, то в этом интервале есть корень.

3. Мы реализовали алгоритмы для нахождения корней функций на языке программирования Java. В частности, мы реализовали метод дихотомии, который постепенно сужает интервал, содержащий корень, до тех пор, пока его длина не станет меньше заданной точности.

4. Мы провели эксперименты с различными функциями и интервалами, чтобы оценить точность и эффективность наших методов. Например, для функции $f(x) = x^3 - 4x - 9$ на интервале [2, 3] мы нашли корень с точностью до шести знаков после запятой.

5. Мы провели тесты, чтобы проверить правильность наших методов. Мы проверяли случаи, когда функция не имеет корней в заданном интервале, а также случаи, когда функция имеет несколько корней.

6. Результаты показали, что метод дихотомии надежен и прост в реализации, но может требовать больше итераций по сравнению с другими методами. Например, для функции $f(x) = \sin(x)$ на интервале [3, 4] потребовалось около 20 итераций для достижения заданной точности.

7. Важно правильно выбирать начальные условия и точность при использовании метода для нахождения корней, так как это может сильно влиять на результаты. Например, если начальный интервал выбран неправильно, метод может не сходиться к корню.

8. В ходе лабораторной работы мы изучили и применили основные принципы численных методов, такие как итеративное приближение и оценка ошибки. Это помогло нам лучше понять, как работают численные методы и как их можно применять на практике.

Рефлексия, заключение

В ходе выполнения лабораторной работы мы получили ценный опыт в области численных методов и программирования. Этот опыт помог нам лучше понять, как важны точность и эффективность алгоритмов в решении реальных задач. Мы осознали, что математические методы, такие как метод дихотомии, являются фундаментальными инструментами, которые можно применять в различных областях, включая машинное обучение, оптимизацию и анализ данных.

Работа над этой лабораторной работой также помогла нам развить навыки программирования на языке Java. Мы научились более эффективно использовать библиотеки для работы с математическими выражениями и поняли, как важно тестирование и проверка корректности реализации алгоритмов. Этот опыт будет полезен в нашей будущей работе, так как навыки программирования и понимание численных методов являются ключевыми для успешной карьеры в области машинного обучения и инженерии данных.

В будущем мы планируем продолжать изучать и применять численные методы в более сложных задачах. Мы хотим углубить свои знания в области машинного обучения и искусственного интеллекта, чтобы иметь возможность работать над интересными и значимыми проектами. Мы также планируем улучшать свои навыки программирования и осваивать новые языки и технологии, чтобы быть более конкурентоспособными на рынке труда.

Мы понимаем, что успех в области машинного обучения требует не только технических навыков, но и умения работать в команде, решать сложные задачи и постоянно учиться. Мы готовы к этим вызовам и уверены, что полученные знания и опыт помогут нам достичь наших целей и внести свой вклад в развитие технологий и науки.

Все еще не научился писать в листинге на русском, переведено с помощью ChatGPT

```java
package com.example;

public class Task1 {

    // Calculates the value of the given expression for a specific x
    public static float func(String expression, float x) {
        Expression e = new ExpressionBuilder(expression) // Build the mathematical expression using exp4j
                .variable("x")       // Define the variable "x"
                .build()             // Compile the expression
                .setVariable("x", x); // Set the value of "x"
        return (float) e.evaluate();   // Evaluate and return the result
    }

    // Implements the dichotomy method to find the root of the equation
    public static float dichotomyMethod(String expression, float a, float b, float tolerance) {
        if (func(expression, a) * func(expression, b) > 0) {
            // Check if the root exists within the interval [a, b]
            throw new IllegalArgumentException("f(a) and f(b) must have opposite signs!");
        }

        float c; // Midpoint of the interval
        XYSeries series = new XYSeries("Iterations"); // Create a series to store iteration data
        ArrayList<Float> intervalLengths = new ArrayList<>(); // List to store interval lengths

        // Continue iterating until the interval size is less than the tolerance
        while ((b - a) / 2.0 > tolerance) {
            c = (a + b) / 2;  // Compute the midpoint
            series.add(c, func(expression, c));  // Add the midpoint and its function value to the series

            intervalLengths.add((b - a));   // Record the current interval length

            if (func(expression, c) == 0) {   // Check if an exact root is found
                return c;
            } else if (func(expression, c) * func(expression, a) < 0) {
                b = c;   // Update the upper bound
            } else {
                a = c;  // Update the lower bound
            }
        }

```

```
41          series.add((a + b) / 2, func(expression, (a + b) / 2)); // Add
                the final midpoint
42
43          showGraph(series); // Display the graph of the function values
44          plotIntervalLengths(intervalLengths); // Display the interval
                length convergence graph
45
46          return (a + b) / 2; // Return the approximate root
47      }
48
49      // Displays a graph of the function values across iterations
50      public static void showGraph(XYSeries series) {
51          SwingUtilities.invokeLater(() -> {
52              XYSeriesCollection dataset = new XYSeriesCollection(series)
                    ; // Create a dataset for the series
53
54              JFreeChart chart = ChartFactory.createXYLineChart(
55                      "Estimated position of extremum", // Title of the
                            chart
56                      "x",   // Label for the X-axis
57                      "f(x)",   // Label for the Y-axis
58                      dataset,   // Data for the chart
59                      PlotOrientation.VERTICAL,   // Chart orientation
60                      true,   // Show legend
61                      true,   // Enable tooltips
62                      false
63              );
64
65              XYPlot plot = chart.getXYPlot();
66              XYLineAndShapeRenderer renderer = new
                    XYLineAndShapeRenderer(true, true); // Configure
                    renderer for lines and shapes
67
68              renderer.setSeriesShape(0, new java.awt.geom.Ellipse2D.
                    Double(-6, -6, 10, 10)); // Set point shapes
69              renderer.setSeriesPaint(0, Color.BLACK); // Set line color
70              renderer.setSeriesStroke(0, new BasicStroke(1.0f)); // Set
                    line thickness
71
72              plot.setRenderer(renderer);
73
74              ChartPanel chartPanel = new ChartPanel(chart); // Create a
                    chart panel
75              chartPanel.setPreferredSize(new java.awt.Dimension(800,
                    600)); // Set panel size
76
77              JFrame frame = new JFrame("Graph"); // Create a JFrame to
                    hold the chart
78              frame.getContentPane().add(chartPanel, BorderLayout.CENTER)
                    ; // Add chart panel to frame
79              frame.pack(); // Adjust frame size to fit content
80              frame.setVisible(true); // Display the frame
81          });
```

```
    }

    // Plots the interval lengths across iterations
    public static void plotIntervalLengths(ArrayList<Float>
        intervalLengths) {
         SwingUtilities.invokeLater(() -> {
             XYSeries intervalSeries = new XYSeries("Interval Lengths");
                 // Create a series for interval lengths
             for (int i = 0; i < intervalLengths.size(); i++) {
                 intervalSeries.add(i + 1, intervalLengths.get(i)); //
                    Add each interval length with its iteration index
             }

             XYSeriesCollection dataset = new XYSeriesCollection(
                intervalSeries); // Create a dataset for the series
             JFreeChart chart = ChartFactory.createXYLineChart(
                     "Interval Length vs Iteration",
                     "Iteration",
                     "Interval Length",
                     dataset,
                     PlotOrientation.VERTICAL,
                     true,
                     true,
                     false
             );

             XYPlot plot = chart.getXYPlot();
             XYLineAndShapeRenderer renderer = new
                XYLineAndShapeRenderer(true, false);
             renderer.setSeriesStroke(0, new BasicStroke(5.0f));

             plot.setRenderer(renderer);

             ChartPanel chartPanel = new ChartPanel(chart);
             chartPanel.setPreferredSize(new java.awt.Dimension(800,
                600));

             JFrame frame = new JFrame("Interval Lengths");
             frame.getContentPane().add(chartPanel, BorderLayout.CENTER)
                ;
             frame.pack();
             frame.setVisible(true);
         });
    }
}
```

Listing 4: Java Code for Task1

```
package com.example;

public class Task2 {

    public static List<Double> findRoots(String function, double a,
       double b, double epsilon) {
```

```
6           Expression expression = new ExpressionBuilder(function)
7                   .variables("x")
8                   .build();
9
10          List<Double> roots = new ArrayList<>();
11          double step = epsilon; // Step size for checking intervals
12
13          for (double x = a; x <= b; x += step) {
14              if (f(expression, x) * f(expression, x + step) < 0) {
15                  double root = findRootInInterval(expression, x, x +
                        step, epsilon);
16                  roots.add(root);
17              }
18          }
19
20          return roots;
21      }
22
23      private static double findRootInInterval(Expression expression,
            double a, double b, double epsilon) {
24          if (f(expression, a) * f(expression, b) >= 0) {
25              return Double.NaN;
26          }
27
28          double c = a;
29          int iterations = 0;
30
31          while ((b - a) >= epsilon) {
32              c = (a + b) / 2;
33              if (f(expression, c) == 0.0) {
34                  break;
35              } else if (f(expression, c) * f(expression, a) < 0) {
36                  b = c;
37              } else {
38                  a = c;
39              }
40              iterations++;
41          }
42
43          System.out.println("Number of iterations: " + iterations); //
                Print iteration count
44          return c;
45      }
46
47      private static double f(Expression expression, double x) {
48          expression.setVariable("x", x); // Set the variable 'x' in the
                expression
49          return expression.evaluate(); // Evaluate the expression and
                return the result
50      }
51
52      public static double calculateRMS(double foundRoot, double trueRoot
            ) {
```

```
        return Math.sqrt((foundRoot - trueRoot) * (foundRoot - trueRoot
            )); // Calculate RMS error
    }
}
```

Listing 5: Java Code for Task2

```
import net.objecthunter.exp4j.Expression; // For evaluating
    mathematical expressions
import net.objecthunter.exp4j.ExpressionBuilder; // For building
    expressions with variables
import org.jfree.chart.ChartFactory; // For creating charts
import org.jfree.chart.ChartPanel; // For embedding charts in Swing
    applications
import org.jfree.chart.JFreeChart; // Represents a chart in JFreeChart
    library
import org.jfree.chart.plot.PlotOrientation; // Defines the orientation
     of a plot (horizontal or vertical)
import org.jfree.data.xy.XYSeries; // For storing series of XY data for
     plotting
import org.jfree.data.xy.XYSeriesCollection; // A collection of
    XYSeries for plotting

import javax.swing.*; // For GUI components like JFrame, JPanel, etc.
import java.awt.*; // For handling graphics and layout management
import org.jfree.chart.plot.XYPlot; // For managing the plotting of XY
    data
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer; // For
    customizing the appearance of XY plots
import java.awt.BasicStroke; // For setting line styles in graphics
import java.util.ArrayList; // For creating dynamic arrays (lists)
```

Listing 6: Used Libraries