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