

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА



Институт радиоэлектроники и информационных технологий  
Кафедра вычислительные системы и технологии

Лабораторная работа № 1  
Решение нелинейного уравнения  
Вариант №15

## ОТЧЕТ

по лабораторной работе

по дисциплине

Вычислительная математика

РУКОВОДИТЕЛЬ:

\_\_\_\_\_

Суркова А.С.

СТУДЕНТ:

\_\_\_\_\_

Сапожников В.О.

19-ИВТ-3

Работа защищена «\_\_» \_\_\_\_\_

С оценкой \_\_\_\_\_

Нижний Новгород 2021

## Содержание

1. Цель.....	3
2. Постановка задачи.....	4
3. Теоретические сведения.....	5
3.1. Критерий остановки.....	5
3.2. Метод бисекции.....	6
3.3. Метод хорд.....	7
3.4. Метод Ньютона.....	9
3.5. Метод простых итераций.....	11
4. Расчётные данные.....	13
5. Листинг разработанной программы.....	14
6. Результаты работы программы.....	33
7. Вывод.....	35

## **1. Цель**

Закрепление знаний и умений по нахождению решений нелинейных уравнений различными способами.

## 2. Постановка задачи

Решить нелинейное уравнение с одним неизвестным с использованием четырех методов (метод биссекции, метод хорд, метод Ньютона, метод простой итерации). Задание по вариантам. Номер варианта – номер студента в списке группы.  $\varepsilon=0.001$

Вариант 15:

$$x^3 + 0.1x^2 + 0.4x - 1.2 = 0$$

### 3. Теоретические сведения

#### 3.1. Критерий остановки

Процесс нахождения оптимального решения чаще всего имеет итерационный характер, т.е. последовательность  $\{x_0, x_1, \dots, x_n \rightarrow x\}$  стремится к точному решению при увеличении кол-ва итераций  $n$ .

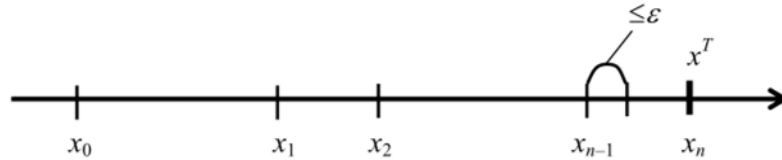


Рис. 1

Весьма важным элементом всех итерационных методов является критерий (правило) остановки итерационного процесса. Именно критерий определяет точность достижения решения, а соответственно и эффективность метода.

Наиболее распространённые критерии остановки:

1.  $f(x) = 0$  – найдено точное решение
2.  $|f(x_n)| < \epsilon$  – найдена заданная точность функции
3.  $|f(x_n - x_{n+1})| < \epsilon$  – значение двух последовательных приближений

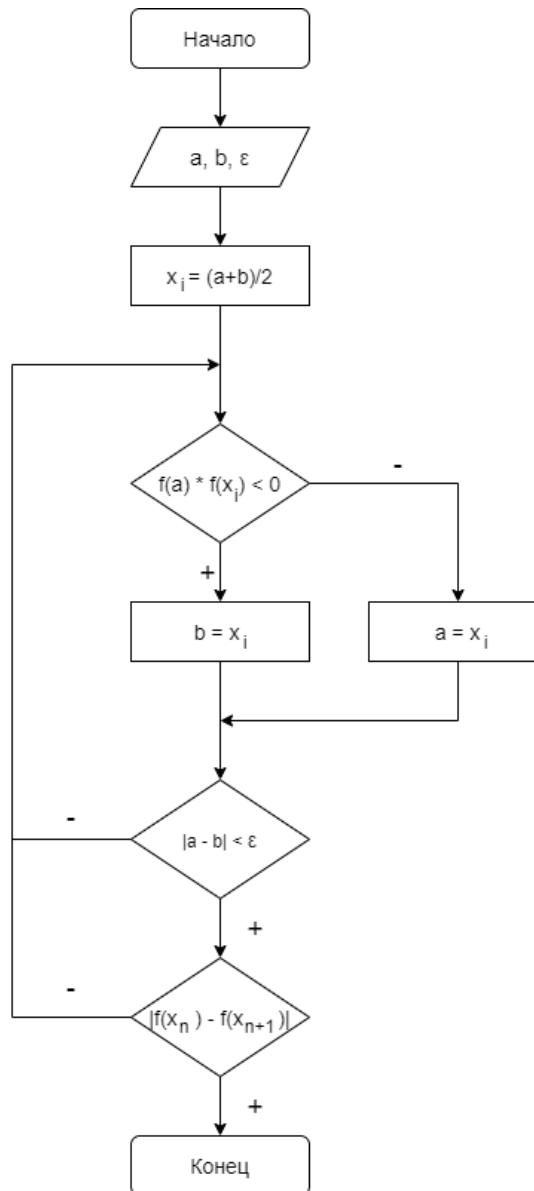
отличаются меньше, чем на  $\epsilon$

### 3.2. Метод биссекции

Пусть был выбран интервал изоляции  $[a; b]$ . Примем за первое приближение корня точку  $c$ , которая является серединой отрезка  $[a; b]$ . Далее будем действовать по следующему алгоритму:

1. Находим точку  $x_i = \frac{a_i + b_i}{2}$ ;
2. Находим значение  $f(x_i)$ ;
3. Если  $f(a) * f(x_i) < 0$ , то корень лежит на интервале  $[a; x_i]$ , иначе корень лежит на интервале  $[x_i; b]$ ;
4. Если величина интервала меньше либо равна  $\varepsilon$ , либо разность двух последовательных приближений меньше либо равна  $\varepsilon$ , то найдено решение с точностью до  $\varepsilon$  иначе возвращаемся к п.1.

Итак, для вычисления одного из корней уравнения методом половинного деления достаточно знать интервал изоляции корня  $[a; b]$  и точность вычисления  $\varepsilon$ .



### 3.3. Метод хорд

Этот метод отличается от метода бисекции тем, что очередное приближение берём не в середине отрезка, а в точке пересечения с осью X прямой, соединяющей точки  $(a, f(a))$  и  $(b, f(b))$ .

Запишем уравнение прямой, проходящей через точки с координатами точки  $(a, f(a))$  и  $(b, f(b))$ :

$$\frac{y - f(a)}{f(b) - f(a)} = \frac{x - a}{b - a}, \quad y = \frac{f(b) - f(a)}{b - a} (x - a) + f(a)$$

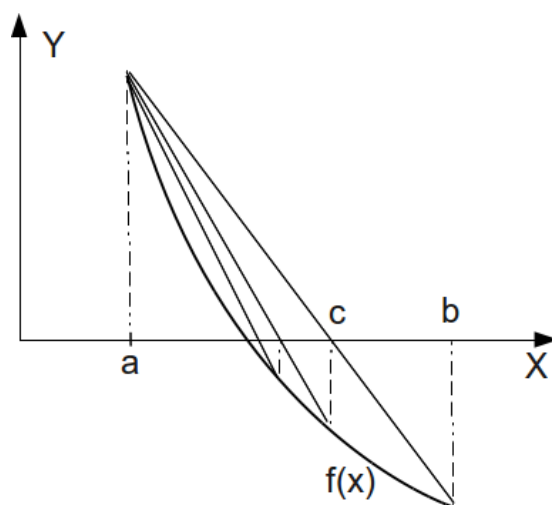
Прямая, заданная уравнением, пересекает ось X при условии  $y = 0$ .

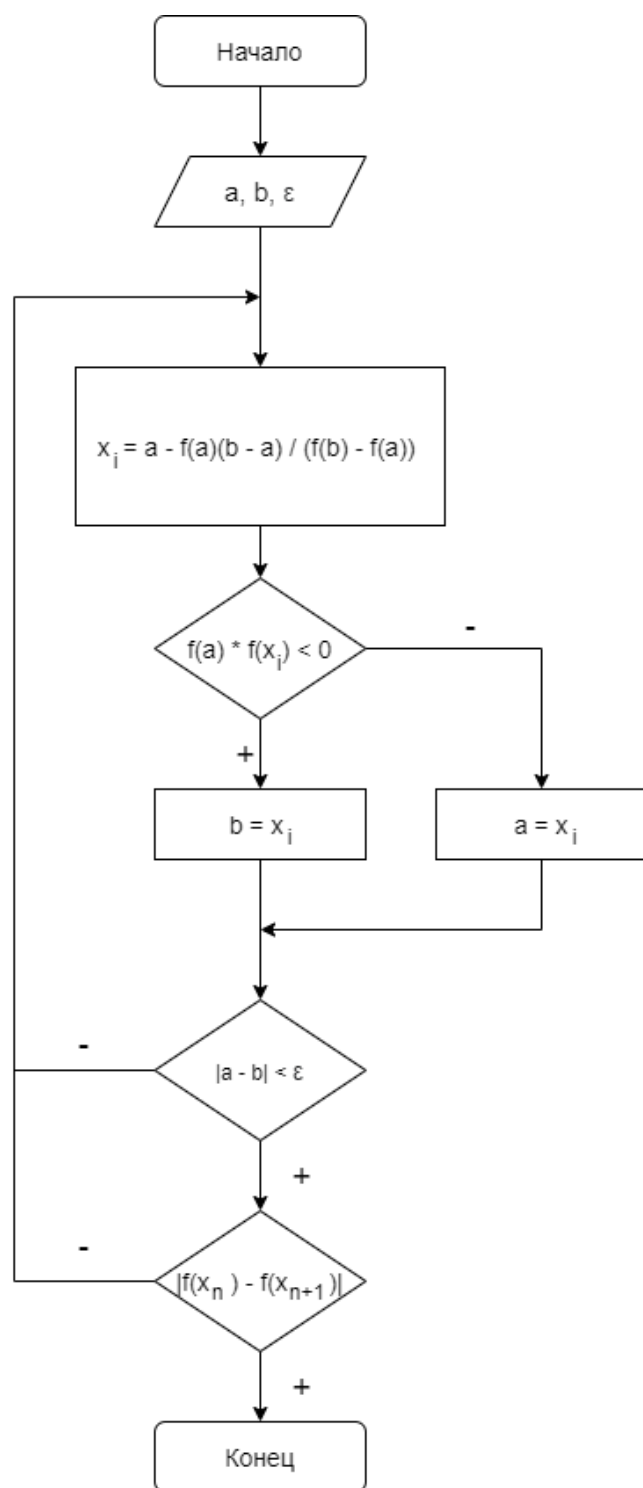
Найдём точку пересечения хорды с осью X:

$$y = \frac{f(b) - f(a)}{b - a} (x - a) + f(a), \quad x = a - \frac{f(a)(b - a)}{f(b) - f(a)} \text{ итак, } x_i = a - \frac{f(a)}{f(b) - f(a)} (b - a).$$

Далее необходимо вычислить значение функции в точке  $X_i$ . Это и будет приближённое значение корня уравнения.

Для вычисления одного из корней уравнения методом хорд достаточно знать интервал изоляции корня и точность вычисления  $\varepsilon$ .







### 3.4. Метод Ньютона

В одной из точек интервала  $[a; b]$ , пусть это будет точка  $a$ , проведём касательную. Запишем уравнение этой прямой:

$$y = kx + m$$

Так как эта прямая является касательной, и она проходит через точку  $(x_i, f(x_i))$ , то  $k = f'(x_i)$ .

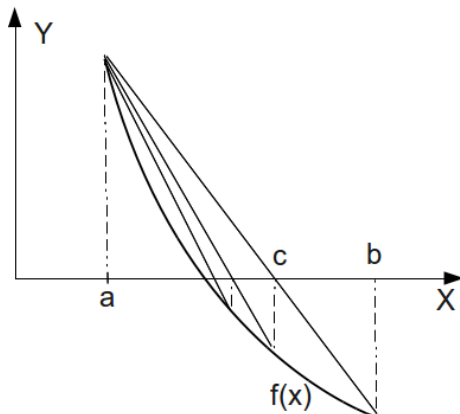
Следовательно,

$$y = f'(x_i)x + m, \quad f(x_i) = f'(x_i)x_i + m, \quad m = f(x_i) - x_i f'(x_i),$$

$$y = f'(x_i)x + f(x_i) - x_i f'(x_i), \quad y = f'(x_i)(x - x_i) + f(x_i)$$

Найдём точку пересечения касательной с осью X:

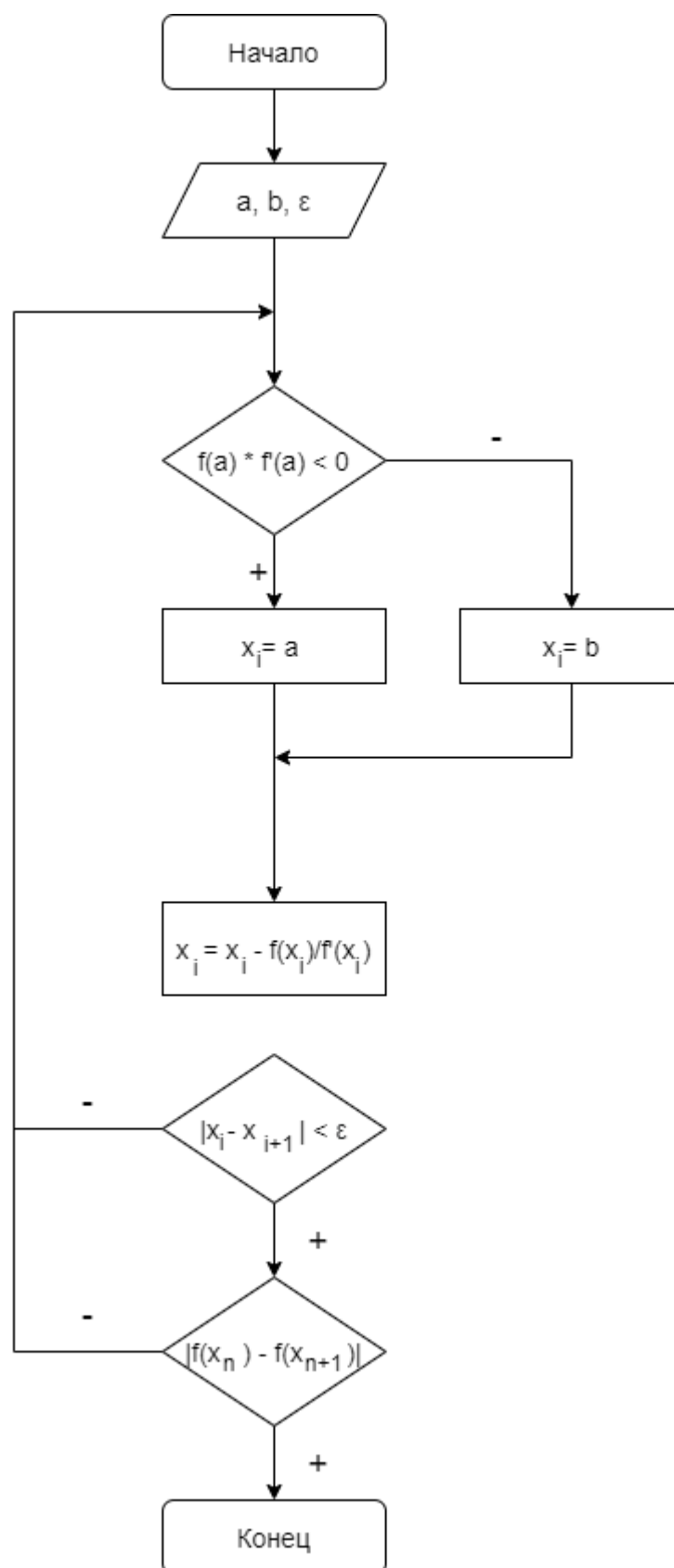
$$f'(x_i)(x - x_i) + f(x_i) = 0, \quad x = x_i - \frac{f(x_i)}{f'(x_i)}$$



Если  $|f(x)| < \varepsilon$ , то точность достигнута, и точка  $X$  — решение; иначе необходимо переменной  $x$  присвоить значение  $X$  и провести касательную через новую точку  $X_i$ ; так продолжать до тех пор, пока  $|f(x)|$  не станет меньше  $\varepsilon$ . Осталось решить вопрос, что выбрать в качестве точки начального приближения  $X_i$ .

В этой точке должны совпадать знаки функции и её второй производной. А так как нами было сделано допущение, что вторая и первая производные не меняют знак, то можно проверить условие  $f(x)f''(x) > 0$  на обоих концах интервала, и в качестве начального приближения взять ту точку, где это условие выполняется.

Здесь, как и в предыдущих методах, для вычисления одного из корней уравнения достаточно знать интервал изоляции корня и точность вычисления  $\varepsilon$ .

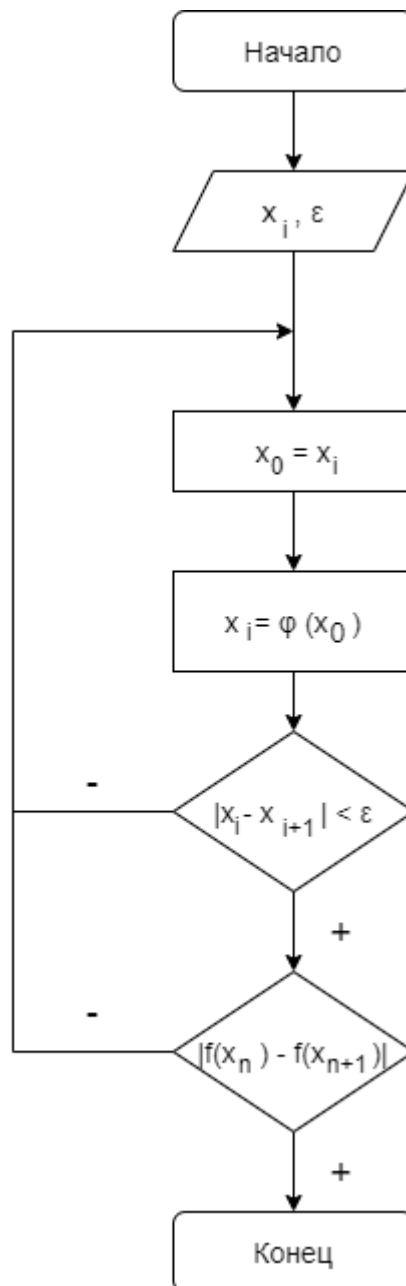


### 3.5. Метод простых итераций

Для решения уравнения этим методом необходимо записать уравнение в виде  $x = \varphi(x)$ , задать начальное приближение  $x_0 \in [a; b]$  и организовать следующий итерационный вычислительный процесс:  $x_i = \varphi(x)$ ,  $i = 0, 1, 2, \dots, n$  сходится к решению  $X^*$

Если неравенство  $|\varphi'(x)| < 1$  выполняется на всём интервале  $[a; b]$ , то последовательность  $x_0, x_1, x_2, \dots, x_n$  сходится к решению.

Уравнение можно привести к виду  $x = \varphi(x)$  следующим образом. Умножить обе части уравнения  $f(x) = 0$  на число  $\beta$ . К обеим частям уравнения  $\beta f(x) = 0$  добавить число  $x$ . Получим  $x = x + \beta f(x)$ .



#### 4. Расчетные данные

Исходное уравнение:

$$x^3 + 0.1x^2 + 0.4x - 1.2 = 0$$

Сжимающее уравнение:

$$x = x - 0.1(x^3 + 0.1x^2 + 0.4x - 1.2)$$

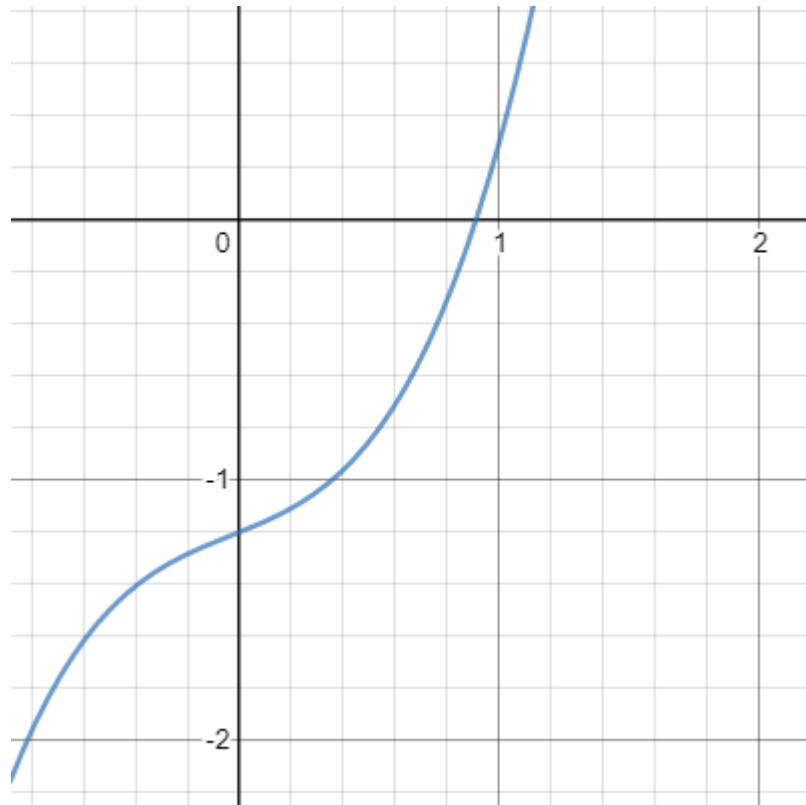


Рис.2 График функции

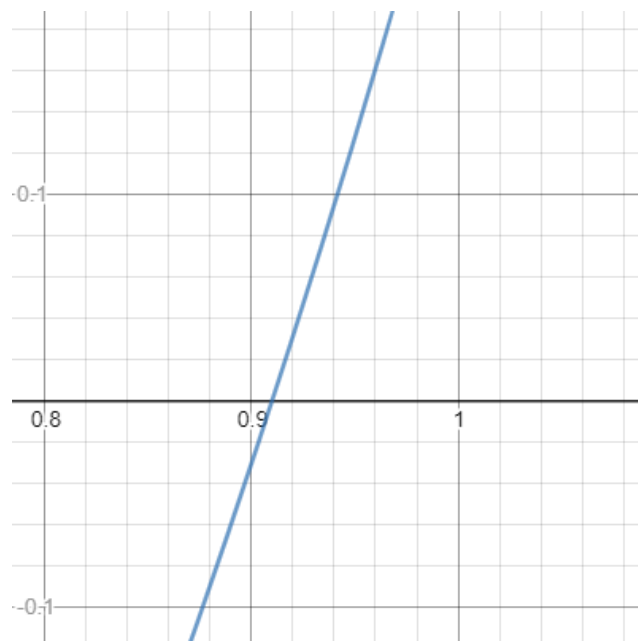


Рис.3 Увеличенное изображение  
пересечения графика с осью OX

<b>Метод</b>	<b>Полученный X</b>
Метод бисекции	0,91016
Метод хорд	0,90982
Метод Ньютона	0,90992
Метод простой итерации	0,91049

## 5. Листинг разработанной программы

### Main.java

```
import equation.NoLinearThirdDegreeEquation;
import equation_solution_strategy.*;
import validator.ResponseValidator;
import validator.Validator;

import java.util.List;
import java.util.Scanner;

/**
 * Класс, содержащий точку входа в программу - метод main.
 * Язык: java
 *
 * Реализация первой лабораторной работы по дисциплине: Вычислительная математика
 *
 * Текст задания:
 * Решить нелинейное уравнение с одним неизвестным с использованием четырех
 * методов (метод биссекции, метод хорд, метод Ньютона, метод простой итерации).
 *
 * Задание по вариантам.
 * Номер варианта – номер студента в списке группы.  $\varepsilon=0.001$ 
 *
 * @release: 27.02.21
 * @last_update: 27.02.21
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 */
public class Main
{
    //Константы для хранения последовательностей для
    //изменения цвета текста в консоли
    public static final String RESET = "\u001B[0m";
    public static final String RED = "\u001B[31m";
    public static final String PURPLE = "\u001B[35m";
    public static final String CYAN = "\u001B[36m";

    /**
     * Точка входа в программу
     */
    public static void main(String[] args)
    {
        System.out.println("\t\t\t\tЛабораторная работа №1 <<" + PURPLE + "Решение
                               нелинейного уравнения" + RESET + ">>");
    }
}
```

```

//Создаем переменную для хранения ур-ия
//Открываем поток ввода
NoLinearThirdDegreeEquation equation = new NoLinearThirdDegreeEquation();
Scanner scanner = new Scanner(System.in);

System.out.println("Программа для решения нелинейных уравнений Зей +
                                                                + степени.");

System.out.println("\tОбщий вид таких уравнений: ");
System.out.println("\t\t $a*x^3 + b*x^2 + c*x + d = 0$ ");
System.out.println();

System.out.print("Введите точность ответа (epsilon): ");
double epsilon = scanner.nextDouble();
System.out.println();

//Ввод коэффициентов уравнения
double[] coefficients = new double[4];
System.out.print("Введите коэффициент a: ");
coefficients[0] = scanner.nextDouble();
System.out.print("Введите коэффициент b: ");
coefficients[1] = scanner.nextDouble();
System.out.print("Введите коэффициент c: ");
coefficients[2] = scanner.nextDouble();
System.out.print("Введите коэффициент d: ");
coefficients[3] = scanner.nextDouble();
System.out.println();

//Запись введенных коэффициентов у равнение
equation.setCoefficients(coefficients);

//Создание ссылки на объект, реализующий интерфейс
//SolutionStrategy
SolutionStrategy strategy = null;

//Переменная для хранения результата ввода
String ch;

//Сброс потока ввода
ch = scanner.nextLine();

//Выбор стратегии решения
while (!ch.equals("q"))
{
    System.out.println("Выберите метод для решения уравнения:");
    System.out.println("\t1. Метод биссекций");
    System.out.println("\t2. Метод хорд");
    System.out.println("\t3. Метод Ньютона");
    System.out.println("\t4. Метод простых итераций");
}

```

```

System.out.println();
System.out.println("\tВведите q для выхода");
System.out.print("Ввод: ");
ch = scanner.nextLine();
System.out.println();

switch (ch)
{
    case ("1") -> strategy = new BisectionSolution();
    case ("2") -> strategy = new ChordSolution();
    case ("3") -> strategy = new NewtonSolution();
    case ("4") -> strategy = new SimpleIterationSolution();
    case ("q") -> {
        System.out.println(RED + "Завершение работы..." +
                           + RESET);

        System.exit(0);
    }
    default -> System.out.println(RED + "Неверный ввод!" + RESET);
}

//Создание объекта класса валидатор и установка
//значения для сравнения (эпсилон)
Validator validator = ResponseValidator.getInstance();
validator.setParameter(epsilon);

//Засекаем время до начала решения
double start = System.currentTimeMillis();

assert strategy != null;

//Засекаем время после конца решения
double end = System.currentTimeMillis();
//Получаем список реше
List<Double> resultLst = strategy.getSolution(equation, validator);

System.out.print(CYAN+ "Значения функции при найденных ответах:\n" +
                  + RESET);

for (int i = 0; i < resultLst.size(); i++)
{
    System.out.printf("y" + (i + 1) + ": %.5f",
                      equation.getValueAtX(resultLst.get(i)));
    System.out.print("; ");
}
System.out.println();

//Выводим получившиеся ответы
System.out.print(RED + "Ответ: " + RESET);
for (int i = 0; i < resultLst.size(); i++)

```



```

        {
            System.out.printf("x" + (i + 1) + ": %.5f", resultLst.get(i));
            System.out.print("; ");
        }
        System.out.println();

        //Выводим затраченное время для данного решения
        System.out.println("Затраченное время: " + (end - start)/1000.0 + " +
                                + секунд\n");
    }
}
}

```

### equation/Equation.java

```

package equation;

import java.util.List;

/**
 * Интерфейс, реализующий основные методы уравнений любого вида.
 *
 * Общий вид уравнения:
 * 
$$a \cdot x^n + b \cdot x^{(n-1)} + c \cdot x^{(n-1)} + \dots + d \cdot x^0 = 0$$

 *
 * Содержит 4 метода необходимых для данной лабораторной работы:
 * - задание коэффициентов уравнения
 * - получение значения функции в точке
 * - получение значения первой производной в точке
 * - получение значения второй производной в точке
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see NoLinearThirdDegreeEquation
 * */
public interface Equation
{
    /**
     * Метод для задания коэффициентов
     *
     * @param coefficients - массив коэффициентов при членах уравнения.
     * */
    void setCoefficients(double[] coefficients);

    /**
     * Метод для получения значения уравнения в заданной точке.
     *
     * @param x - точка, в которой необходимо получить значение функции.
     * @return значение функции в данной точке
     */
}

```

```

    * */
double getValueAtX(double x);

/**
 * Метод для получения значения первой производной при заданном x.
 *
 * @param x - точка, в которой необходимо получить значение первой производной.
 * @return значение первой производной в данной точке
 * */
double getFstDerivativeAtX(double x);

/**
 * Метод для получения значения второй производной в при заданном x.
 *
 * @param x - точка, в которой необходимо получить значение второй производной.
 * @return значение второй производной в данной точке
 * */
double getSecDerivativeAtX(double x);

/**
 * Метод для получения списка интервалов монотонности.
 *
 * @return список из чисел, которые составляют отрезки монотонности
 *         типа [i; i+1]
 * */
List<List<Double>> getIntervalsOfMonotony();
}

```

### **equation/NoLinearThirdDegreeEquation.java**

```

package equation;

import java.util.*;

/**
 * Класс Нелинейных уравнений третьей степени.
 * Реализует интерфейс Equation
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see Equation
 * */
public class NoLinearThirdDegreeEquation implements Equation
{

    private double a;    //коэффициент при x^3
    private double b;    //коэффициент при x^2
    private double c;    //коэффициент при x^1
    private double d;    //коэффициент при x^0

```

```

/**
 * Метод для задания коэффициента при  $x^3$ .
 * Скрыт, т.к. используется в общем методе задания коэффициентов.
 *
 * @param a - значения коэффициента, которое необходимо установить.
 * */
private void setA(double a)
{
    this.a = a;
}

/**
 * Метод для задания коэффициента при  $x^2$ .
 * Скрыт, т.к. используется в общем методе задания коэффициентов.
 *
 * @param b - значения коэффициента, которое необходимо установить.
 * */
private void setB(double b)
{
    this.b = b;
}

/**
 * Метод для задания коэффициента при  $x^1$ .
 * Скрыт, т.к. используется в общем методе задания коэффициентов.
 *
 * @param c - значения коэффициента, которое необходимо установить.
 * */
private void setC(double c)
{
    this.c = c;
}

/**
 * Метод для задания коэффициента при  $x^0$ .
 * Скрыт, т.к. используется в общем методе задания коэффициентов.
 *
 * @param d - значения коэффициента, которое необходимо установить.
 * */
private void setD(double d)
{
    this.d = d;
}

/**
 * Метод для получения коэффициента при  $x^3$ .
 * */

```

```

private double getA()
{
    return a;
}

/**
 * Метод для получения коэффициента при  $x^2$ .
 * */
private double getB()
{
    return b;
}

/**
 * Метод для получения коэффициента при  $x^1$ .
 * */
private double getC()
{
    return c;
}

/**
 * Метод для получения коэффициента при  $x^0$ .
 * */
private double getD()
{
    return d;
}

/**
 * Общий метод для задания сразу всех коэффициентов уравнения.
 *
 * @param coefficients - массив коэффициентов, который необходимо установить
 * */
@Override
public void setCoefficients(double[] coefficients)
{
    setA(coefficients[0]);
    setB(coefficients[1]);
    setC(coefficients[2]);
    setD(coefficients[3]);
}

/**
 * Метод для получения значения функции в заданной точке
 *
 * @param x - точка, в которой необходимо получить значение функции
 * @return значение функции в заданной точке

```

```

    * */
@Override
public double getValueAtX(double x)
{
    return getA()*Math.pow(x, 3) + getB()*Math.pow(x, 2) + getC()*x + getD();
}

/**
 * Метод для получения значения первой производной при заданном x.
 *
 * @param x - точка, в которой необходимо получить значение первой производной
 * @return значение первой производной в заданной точке
 * */
public double getFstDerivativeAtX(double x)
{
    return 3*getA()*Math.pow(x,2) + 2.0*getB()*x + getC();
}

/**
 * Метод для получения значения первой производной при заданном x.
 *
 * @param x - точка, в которой необходимо получить значение второй производной
 * @return значение второй производной в заданной точке
 * */
public double getSecDerivativeAtX(double x)
{
    return 6.0*getA()*x + 2.0*getB();
}

/**
 * Метод для получения списка интервалов монотонности:
 * 1. Получение значений функции в промежутке от -100 до 100
 * 2. Выделение промежутков монотонности
 *
 * (Как альтернатива получения промежутка графическим способом)
 *
 * @return список интервалов монотонности.
 * */
public List<List<Double>> getIntervalsOfMonotony()
{
    List<Double> xList = new LinkedList<>();
    List<Double> yList = new LinkedList<>();

    //Задаем интервал иксов от -100 до 100
    for (double i = -100.0; i <= 100; i++)
    {
        xList.add(i);
    }
}

```

```

//Получаем значения функции в каждой точке интервала
//от -100 до 100
for (Double aDouble : xList)
{
    yList.add(getValueAtX(aDouble));
}

//В коллекцию, которая может содержать только уникальные элементы
//вносим значения, при которых функция меняет знак
Set<Double> uniqSet = new HashSet<>();
for (int i = 0; i < yList.size() - 1; i++)
{
    if (yList.get(i) * yList.get(i+1) < 0)
    {
        uniqSet.add(xList.get(i));
        uniqSet.add(xList.get(i+1));
    }
}

//Дублируем значения в промежуточный список и сортируем по возрастанию
List<Double> newList = new LinkedList<>(uniqSet);
Collections.sort(newList);

//Попарно объединяем значения в интервалы
List<List<Double>> cordList = new LinkedList<>();
for (int i = 0; i < newList.size() - 1; i++)
{
    List<Double> tempList = new LinkedList<>();
    if (Math.abs(newList.get(i) - newList.get(i + 1)) <= 1)
    {
        tempList.add(newList.get(i));
        tempList.add(newList.get(i + 1));
    }
    if (!tempList.isEmpty()) cordList.add(tempList);
}
return cordList;
}

/**
 * Конструктор без параметров
 * */
public NoLinearThirdDegreeEquation()
{
}
}

```

### validator/Validator.java

```
package validator;

/**
 * Интерфейс реализующий метод проверки
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see ResponseValidator
 * */
public interface Validator
{
    /**
     * Метод для проверки правильности значения
     *
     * @param prevValue    - предыдущее значение функции -  $f(X_{n-1})$ 
     * @param presentValue - текущее значение функции -  $f(X_n)$ 
     * @return true - если найдено подходящее решение
     * */
    boolean isValid(double presentValue, double prevValue);

    /**
     * Метод задания параметра для сравнения.
     *
     * @param parameter - значение, с которым будет происходить сравнение
     * */
    void setParameter(double parameter);
}
```

### validator/ResponseValidator.java

```
package validator;

/**
 * Singleton класс реализующий интерфейс проверки Validator.
 * Проверяет решение на соответствие критериям остановки.
 *
 * WARNING!!!
 * Критерий остановки: найдено точное значение  $f(X_n) = 0$  не используется
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see Validator
 * */
public class ResponseValidator implements Validator
{
    private static ResponseValidator instance;    //поля для хранения ссылки на единственный
                                                //экземпляр класса
}
```

```

private double epsilon;                                //поля для хранения значения с которым
                                                        //будет проводится сравнение

/**
 * Проверка решение на соответствие критериям остановки:
 * - Достигнута заданная точность  $|f(X_n)| < \epsilon$ 
 * - Значение двух последних приближений отличается меньше, чем на  $\epsilon$ 
 *    $|X(n-1) - X(n)| < \epsilon$ 
 *
 * @param prevValue    - предыдущее значение функции -  $f(X_{n-1})$ 
 * @param presentValue - текущее значение функции -  $f(X_n)$ 
 * @return true - если найдено подходящее решение
 * */
@Override
public boolean isValid(double prevValue, double presentValue)
{
    if (Math.abs(prevValue - presentValue) < epsilon) return true;
    return Math.abs(presentValue) < epsilon;
}

/**
 * Метод для получения единственного экземпляра класса.
 * Нам достаточно лишь одного экземпляра класса ResponseValidator
 * для вызова методов проверки
 *
 * @return ссылку на один единственный экземпляр класса
 * */
public static ResponseValidator getInstance()
{
    if (instance == null)
    {
        instance = new ResponseValidator();
    }

    return instance;
}

/**
 * Метод для установки параметра сравнений.
 *
 * @param epsilon - значения для сравнений
 * */
@Override
public void setParameter(double epsilon)
{
    this.epsilon = epsilon;
}

```



```

/**
 * Приватный конструктор запрещает создание объекта из вне.
 * */
private ResponseValidator()
{
}
}

```

### **equation solution strategy/SolutionStrategy.java**

```

package equation_solution_strategy;

import java.util.List;

/**
 * Общий интерфейс всех стратегий решения.
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see BisectionSolution
 * @see ChordSolution
 * @see NewtonSolution
 * @see SimpleIterationSolution
 * */
public interface SolutionStrategy
{
    /**
     * Метод для вызова той или иной стратегии решения.
     *
     * @param equation - ур-ие, которое необходимо решить
     * @param validator - валидатор с заданным условием проверки
     * @return список значений, которые являются решениями данного
     *         ур-ия
     * */
    List<Double> getSolution(Equation equation, Validator validator);
}

```

### **equation solution strategy/BisectionSolution.java**

```

package equation_solution_strategy;

/**
 * Класс, реализующий решение методом биссекций.
 * Реализует интерфейс SolutionStrategy
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class BisectionSolution implements SolutionStrategy

```

```

{
    /**
     * Метод для получения решений методом бисекции.
     *
     * @param equation - ур-ие, которое необходимо решить
     * @param validator - валидатор, с заданным параметром проверки
     * @return список значений, являющимися решениями данного уравнения.
     */
    @Override
    public List<Double> getSolution(Equation equation, Validator validator)
    {
        //список, в который будут заноситься ответы
        List<Double> resList = new LinkedList<>();

        //Значение, для хранения X(i-1) - ответ полученный на пред. итерации
        //Изначально задается такое значение, которое не пройдет условие валидатора
        double prevValue = (-Double.MAX_VALUE);

        //Переменная для хранения значения Xi
        double xI;

        //Получение списка интервалов монотонности, которые содержат
        //решения
        List<List<Double>> intervals = equation.getIntervalsOfMonotony();

        //Проход по каждому интервалу [a;b]
        for (List<Double> interval: intervals)
        {
            //Изначально задается такое значение, которое не пройдет условие валидатора
            xI = Double.MAX_VALUE;

            //Пока не сработает условие валидатора
            //В предыдущее значение записываем текущее значение Xi
            //Получаем новое значение Xi = (a+b)/2
            while (!validator.isValid(equation.getValueAtX(prevValue),
                                     equation.getValueAtX((xI))))
            {
                prevValue = xI;
                xI = ((interval.get(0)) + interval.get(1)) / 2.0;

                //Если значение функции при Xi и при X=a имеет разные знаки,
                //то меняем b из промежутка [a;b] на Xi
                //иначе меняем a на Xi
                if (equation.getValueAtX(xI) * equation.getValueAtX(interval.get(0)) < 0)
                {
                    interval.set(1, xI);
                }
            }
            else

```

```

        {
            interval.set(0, xI);
        }
    }

    //Записываем полученный ответ в результирующий список
    resList.add(xI);
}
return resList;
}

/**
 * Конструктор без параметров.
 * */
public BisectionSolution()
{
}
}

```

### **equation solution strategy/ChordSolution.java**

```

package equation_solution_strategy;

import equation.Equation;
import validator.Validator;

import java.util.LinkedList;
import java.util.List;

/**
 * Класс, реализующий решение методом хорд.
 * Реализует интерфейс SolutionStrategy
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class ChordSolution implements SolutionStrategy
{
    /**
     * Метод для получения решений методом хорд.
     *
     * @param equation - ур-ие, которое необходимо решить
     * @param validator - валидатор, с заданным параметром проверки
     * @return список значений, являющимися решениями данного уравнения.
     * */
    @Override
    public List<Double> getSolution(Equation equation, Validator validator)
    {

```

```

//список, в который будут заноситься ответы
List<Double> resList = new LinkedList<>();

//Значение, для хранения X(i-1) - ответ полученный на пред. итерации
//Изначально задается такое значение, которое не пройдет условие валидатора
double prevValue;

//Переменная для хранения значения Xi
double xI;

//Получение списка интервалов монотонности, которые содержат
//решения
List<List<Double>> intervals = equation.getIntervalsOfMonotony();

//Проход по каждому интервалу [a;b]
for (List<Double> interval: intervals)
{
    //Отдельно запоминаем изначальные границы интервала,
    //т.к. в самом списке они будут сужаться в ходе решения
    //и тогда программа может поменять статичную точку.
    double A = interval.get(0);
    double B = interval.get(1);

    //Конец a интервала [a;b] неподвижен
    if (equation.getValueAtX(A) * equation.getSecDerivativeAtX(B) > 0)
    {
        //Поскольку a не подвижен, то изменять будет b
        xI = interval.get(1);

        //Изначально задается такое значение, которое не пройдет условие валидатора
        prevValue = (-Double.MAX_VALUE);

        //Пока не сработает условие валидатора
        //В предыдущее значение записываем текущее значение Xi
        //Получаем новое значение  $X_{i+1} = x_i - \frac{f(x_i) \cdot (x_i - a)}{f(x_i) - f(a)}$ 
        while (!validator.isValid(prevValue, equation.getValueAtX(xI))) {
            prevValue = xI;
            xI = prevValue - ((equation.getValueAtX(prevValue) * (prevValue -
                interval.get(0))) / (equation.getValueAtX(prevValue) -
                equation.getValueAtX(interval.get(0))));
        }
    }
    //Конец b интервала [a;b] неподвижен
    else
    {
        //Поскольку b не подвижен, то изменять будет a
        xI = interval.get(0);
    }
}

```

```

        //Изначально задается такое значение, которое не пройдет условие валидатора
        prevValue = (-Double.MAX_VALUE);

        //Пока не сработает условие валидатора
        //В предыдущее значение записываем текущее значение Xi
        //Получаем новое значение  $X_{i+1} = x_i - (f(x_i) * (x_i - b) / (f(x_i) - f(b)))$ 
        while (!validator.isValid(equation.getValueAtX(prevValue),
                                   equation.getValueAtX((xI))))
        {
            prevValue = xI;
            xI = prevValue - ((equation.getValueAtX(prevValue) * (prevValue -
                                                                    interval.get(1))) / (equation.getValueAtX(prevValue) -
                                                                    equation.getValueAtX(interval.get(1))));
        }
    }

    //Записываем полученный ответ в результирующий список
    resList.add(xI);
}
return resList;
}

/**
 * Конструктор без параметров.
 * */
public ChordSolution()
{
}
}

```

### **equation solution strategy/NewtonSolution.java**

```

package equation_solution_strategy;

import equation.Equation;
import validator.Validator;

import java.util.LinkedList;
import java.util.List;

/**
 * Класс, реализующий решение методом Ньютона (касательных).
 * Реализует интерфейс SolutionStrategy
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class NewtonSolution implements SolutionStrategy
{

```

```

/**
 * Метод для получения решений методом Ньютона.
 *
 * @param equation - ур-ие, которое необходимо решить
 * @param validator - валидатор, с заданным параметром проверки
 * @return список значений, являющимися решениями данного уравнения.
 * */
@Override
public List<Double> getSolution(Equation equation, Validator validator)
{
    //список, в который будут занойтятся ответы
    List<Double> resList = new LinkedList<>();

    //Значение, для хранения X(i-1) - ответ полученный на пред. итерации
    //Изначально задается такое значение, которое не пройдет условие валидатора
    double prevValue;

    //Переменная для хранения значения Xi
    double xI;

    //Получение списка интервалов монотонности, которые содержат
    //решения
    List<List<Double>> intervals = equation.getIntervalsOfMonotony();

    //Проход по каждому интервалу [a;b]
    for (List<Double> interval: intervals)
    {
        //Изначально задается такое значение, которое не пройдет условие валидатора
        prevValue = 0;
        xI = interval.get(1) - 0.001;

        //Пока не сработает условие валидатора
        //В предыдущее значение записываем текущее значение Xi
        //Получаем новое значение Xi+1 = f(xi)/f'(xi)
        while (!validator.isValid(equation.getValueAtX(prevValue),
                                equation.getValueAtX((xI))))
        {
            prevValue = xI;
            xI -= (equation.getValueAtX(prevValue)/
                  equation.getFstDerivativeAtX(prevValue));
        }

        //Записываем полученный ответ в результирующий список
        resList.add(xI);
    }
    return resList;
}

```

```

    /**
     * Конструктор без параметров.
     * */
    public NewtonSolution()
    {
    }
}

```

**equation solution strategy/SimpleIterationSolution.java**

```
package equation_solution_strategy;
```

```
import equation.Equation;
import validator.Validator;
```

```
import java.util.LinkedList;
import java.util.List;
```

```

/**
 * Класс, реализующий решение методом Простой итерации.
 * Реализует интерфейс SolutionStrategy
 *
 * WARNING! WARNING! WARNING! WARNING! WARNING! WARNING! WARNING!
 * Подобранная функция:  $x - 0.1 * (x^3 + 0.1x^2 + 0.4x + 1.2)$ 
 * подходит только для ур-ия Варианта 15
 * WARNING! WARNING! WARNING! WARNING! WARNING! WARNING! WARNING!
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */

```

```
public class SimpleIterationSolution implements SolutionStrategy
```

```

{
    /**
     * Метод для получения решений методом хорд.
     *
     * @param equation - ур-ие, которое необходимо решить
     * @param validator - валидатор, с заданным параметром проверки
     * @return список значений, являющимися решениями данного уравнения.
     * */
    @Override
    public List<Double> getSolution(Equation equation, Validator validator)
    {
        //список, в который будут заноситься ответы
        List<Double> resList = new LinkedList<>();

        //Значение, для хранения X(i-1) - ответ полученный на пред. итерации
        //Изначально задается такое значение, которое не пройдет условие валидатора
        double prevValue;
    }
}

```

```

//Переменная для хранения значения Xi
double xI;

//Получение списка интервалов монотонности, которые содержат
//решения
List<List<Double>> intervals = equation.getIntervalsOfMonotony();

//Проход по каждому интервалу [a;b]
for (List<Double> interval: intervals)
{
    //Инициализируем x0 значением из промежутка [a;b]
    //в данном случае [0;1]
    //Math.random() возвращает числа от 0 до 1, что
    //соответствует данному промежутку
    prevValue = (interval.get(0) + interval.get(1)) / Math.random();

    for (;;)
    {
        //Данная сжимающая функция подобрана только для варианта 15
        xI = prevValue - 0.1*(Math.pow(prevValue, 3) + 0.1*Math.pow(prevValue, 2) +
                                + 0.4*prevValue -1.2);

        //Проверка критерием остановки
        if (validator.isValid(equation.getValueAtX(prevValue),
                                equation.getValueAtX((xI)))) break;

        prevValue = xI;
    }
    resList.add(xI);
}
return resList;
}

/**
 * Конструктор без параметров.
 * */
public SimpleIterationSolution()
{
}
}

```



## 6. Результаты работы программы

```
————— Лабораторная работа №1 <<Решение нелинейного уравнения>>
Программа для решения нелинейных уравнений 3ей степени.
—— Общий вид таких уравнений:
——  $a \cdot x^3 + b \cdot x^2 + c \cdot x + d = 0$ 

Введите точность ответа (epsilon): 0,001

Введите коэффициент a: 1,0
Введите коэффициент b: 0,1
Введите коэффициент c: 0,4
Введите коэффициент d: -1,2

Выберите метод для решения уравнения:
—— 1. Метод биссекций
—— 2. Метод хорд
—— 3. Метод Ньютона
—— 4. Метод простых итераций

—— Введите q для выхода
Ввод: 1

Значения функции при найденных ответах:
y1: 0,00086;
Ответ: x1: 0,91016;
Затраченное время: 0.0 секунд
```

```
Выберите метод для решения уравнения:
—— 1. Метод биссекций
—— 2. Метод хорд
—— 3. Метод Ньютона
—— 4. Метод простых итераций

—— Введите q для выхода
Ввод: 2

Значения функции при найденных ответах:
y1: -0,00018;
Ответ: x1: 0,90982;
Затраченное время: 0.0 секунд

Выберите метод для решения уравнения:
—— 1. Метод биссекций
—— 2. Метод хорд
—— 3. Метод Ньютона
—— 4. Метод простых итераций

—— Введите q для выхода
Ввод: 3

Значения функции при найденных ответах:
y1: 0,00012;
Ответ: x1: 0,90992;
```

Выберите метод для решения уравнения:

- 1. Метод биссекций
- 2. Метод хорд
- 3. Метод Ньютона
- 4. Метод простых итераций

— Введите q для выхода

Ввод: 4

Значения функции при найденных ответах:

y1: 0,00189;

Ответ: x1: 0,91049;

Затраченное время: 0.0 секунд

Выберите метод для решения уравнения:

- 1. Метод биссекций
- 2. Метод хорд
- 3. Метод Ньютона
- 4. Метод простых итераций

— Введите q для выхода

Ввод: q

Завершение работы...

## 7. Вывод

Итерационные методы решения нелинейных уравнений удобно применять, когда удастся выделить промежуток, на котором находится ровно один корень (значения функции на концах отрезка имеют разные знаки) и нас интересует его значение с некоторой заданной точностью  $\varepsilon$ .

В ходе данной лабораторной работы были рассмотрены четыре итерационных метода решений уравнений: метод бисекции, метод хорд, метод Ньютона и метод простых итераций.

Самым простым с точки зрения вычислений является метод бисекции, но приближение в данном метода происходит медленнее других способов.

Метод хорд работает быстрее метода бисекции, но его алгоритм усложняется условием выбора неподвижного конца.

Самым простым с точки зрения реализации алгоритма является метод Ньютона (метод касательных), и он же является самым быстрым.

Сложнейшим является метод простых итераций, т.к. для его работы необходимо вывести доп. уравнение  $x = \varphi(x)$ . От того насколько “качественным” окажется выведенное уравнение зависит точность полученного результат и кол-во итераций необходимых для его получения.