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 № 2
Решение системы линейных уравнений итерационным методом и методом Гаусса-Зейделя
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

Закрепление знаний и умений по нахождению решений систем линейных уравнений различными способами.

2. Постановка задачи

Решить систему линейных уравнений методом Гаусса, итерационным методом и методом Гаусса-Зейделя. При необходимости преобразовать систему к диагонально преобладающему виду. Сделать оценку количества итераций для итерационных методов, сравнить. Задание по вариантам. Номер варианта – номер студента в списке группы. ε=0.001

Вариант 15:

$$\begin{cases} 1.6x_1 + 0.12x_2 + 0.57x_3 = 0.18 \\ 0.38x_1 + 0.25x_2 - 54x_3 = 0.63 \\ 0.28x_1 + 0.46 - 1.12x_2\,x_3 = 0.88 \end{cases}$$

3. Теоретические сведения

3.1. Критерий остановки

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

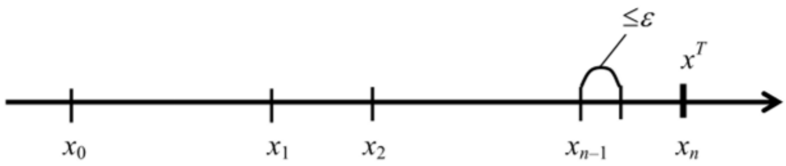


Рис. 1

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 при решении системы линейных уравнения:

1. $\left||x^{(k)} - \ x^{(k-1)}|)\right| = \sqrt{\sum_{i=1}^{n}(x_i^{(k)} - \ x_i^{(k-1)})^2} < \ \varepsilon$ – расстояние между последовательными приближениями меньше ε

2. $\max\limits_{1 \le j \le n} |x_i^{(k)} - \ x_i^{(k-1)}| < \ \varepsilon$ – максимум из поэлементных разностей двух последовательный приближений меньше ε

3.2. Метод Гаусса

Это метод последовательного исключения переменных, когда с помощью элементарных преобразований система уравнений приводится к равносильной системе треугольного вида, из которой последовательно, начиная с последних (по номеру), находятся все переменные системы.

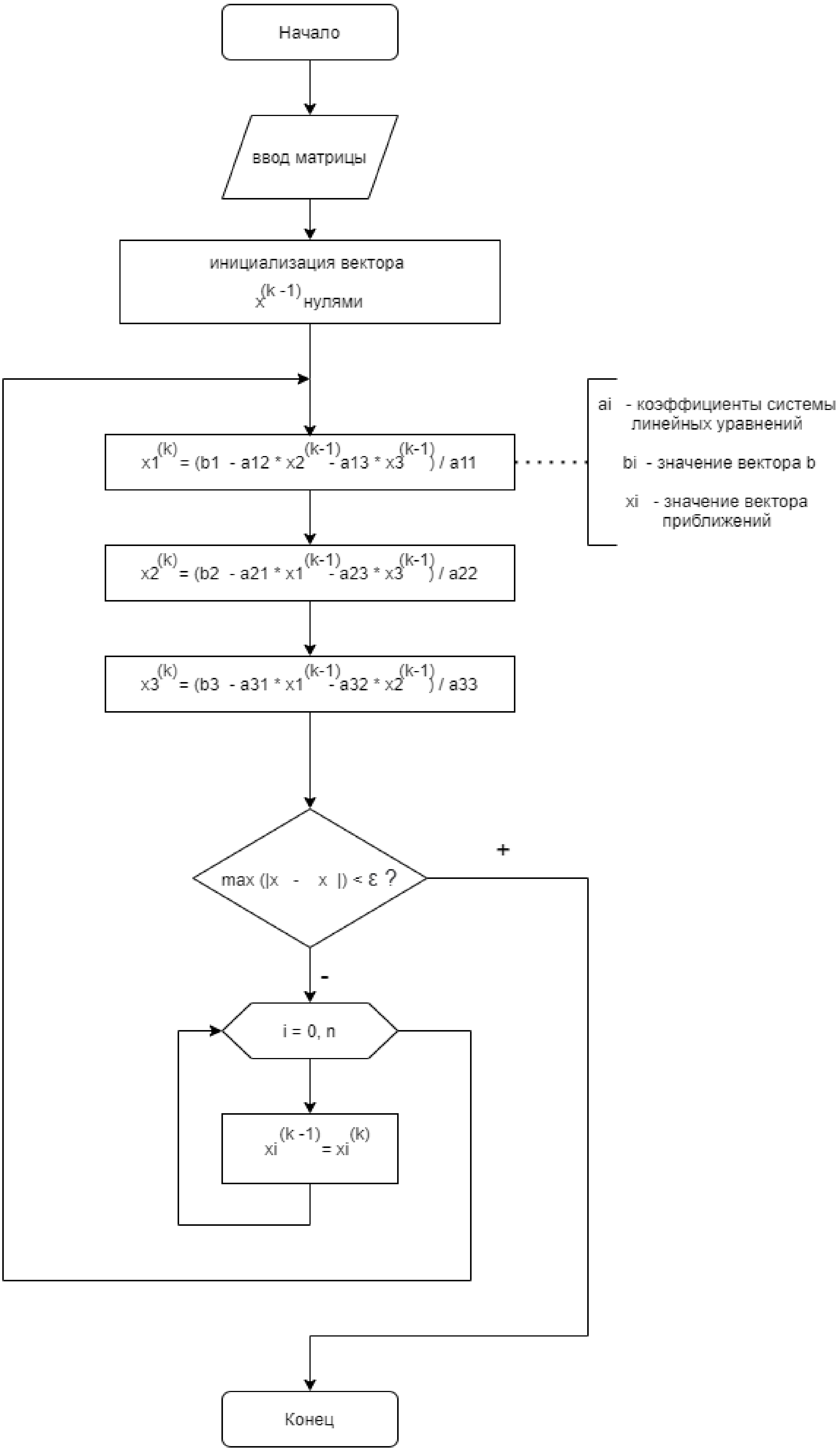
1. На первом этапе осуществляется так называемый прямой ход, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна. Для этого среди элементов первого столбца матрицы выбирают ненулевой, перемещают содержащую его строку в крайнее верхнее положение, делая эту строку первой. Далее ненулевые элементы первого столбца всех нижележащих строк обнуляются путём вычитания из каждой строки первой строки, домноженной на отношение первого элемента этих строк к первому элементу первой строки. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают, пока не останется матрица нулевого размера. Если на какой-то из итераций среди элементов первого столбца не нашёлся ненулевой, то переходят к следующему столбцу и проделывают аналогичную операцию.
2. На втором этапе осуществляется так называемый обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» наверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.

3.3. Метод простой итерации (Метод Якоби)

Метод Якоби — разновидность метода простой итерации для решения системы линейных алгебраических уравнений, которые обладают свойством строгого диагонального преобладания.

Для того, чтобы построить итеративную процедуру метода Якоби, необходимо провести предварительное преобразование системы уравнений $A\vec{x} = \vec{b}$ к итерационному виду $\vec{x} = B\vec{x} + \vec{g}$. Оно может быть осуществлено по следующему правилу:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{i \neq j} a_{ij} x_j^{(k)}\right), \qquad i = 1, 2, \dots, n.$$

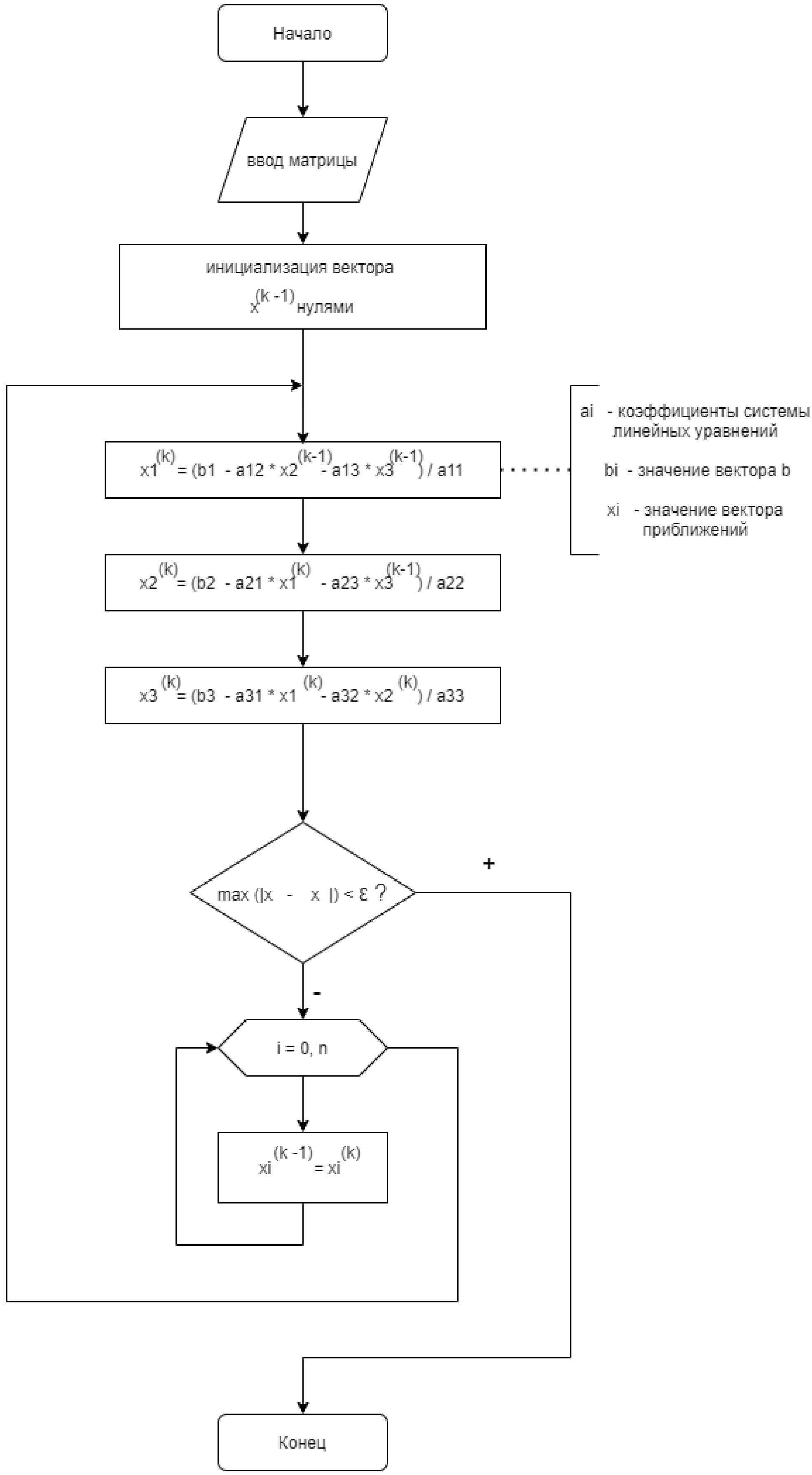

Начало
ввод матрицы
инициализация вектора
x(k -1) нулями
x1(k) = (b1 - a12 * x2(k-1) - a13 * x3(k-1)) / a11
x2(k) = (b2 - a21 * x1(k-1) - a23 * x3(k-1)) / a22
x3(k) = (b3 - a31 * x1(k-1) - a32 * x2(k-1)) / a33
ai - коэффициенты системы линейных уравнений
bi - значение вектора b
xi - значение вектора приближений
max (|x - x |) < ε ?
+
-
i = 0, n
xi(k -1) = xi(k)
Конец

3.4. Метод Гаусса - Зейделя

Метод Гаусса - Зейделя можно рассматривать как модификацию метода Якоби. Основная идея модификации состоит в том, что новые значения $\overrightarrow{x_i}$ используются здесь сразу же по мере получения, в то время как в методе Якоби они не используются до следующей итерации.

Для того, чтобы построить итеративную процедуру метода Гаусса - Зейделя, необходимо провести предварительное преобразование системы уравнений $A\vec{x} = \vec{b}$ к итерационному виду $\vec{x} = B\vec{x} + \vec{g}$. Оно может быть осуществлено по следующему правилу:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right), \qquad i = 1, 2, \ldots, n.$$


Начало
ввод матрицы
инициализация вектора
x(k -1) нулями
x1(k) = (b1 - a12 * x2(k-1) - a13 * x3(k-1)) / a11
ai - коэффициенты системы линейных уравнений
bi - значение вектора b
xi - значение вектора приближений
x2(k) = (b2 - a21 * x1(k) - a23 * x3(k-1)) / a22
x3(k) = (b3 - a31 * x1(k) - a32 * x2(k)) / a33
max (|x - x |) < ε ?
+
-
i = 0, n
xi(k -1) = xi(k)
Конец

4. Расчетные данные

Исходная система:

$$\begin{cases} 1.6x_1 + 0.12x_2 + 0.57x_3 = 0.18 \\ 0.38x_1 + 0.25x_2 - 54x_3 = 0.63 \\ 0.28x_1 + 0.46 - 1.12x_2\, x_3 = 0.88 \end{cases}$$

x_1	x_2	x_3	e_1	e_2	e_3	
0	0	0	0			
1	0.113	1.913	-0.0117	0.113	1.913	0.0117
2	-0.0268	1.816	-0.00202	-0.0857	-0.0969	-0.00965
3	-0.023	1.924	-0.00345	-0.00383	0.108	0.00143
4	-0.0306	1.919	-0.00292	0.00761	-0.00581	-0.000528
5	-0.0304	1.925	-0.003	-0.000248	0.00592	8.0E-5
6	-0.0308	1.924	-0.00297	0.000415	-0.000347	-2.9E-5

Таблица 1. Метод простой итерации
(метод Якоби)

N	x_1	x_2	x_3	e_1	e_2	e_3
0	0	0	0			
1	0.113	1.845	-0.00234	0.113	1.845	0.00234
2	-0.025	1.923	-0.00294	-0.0875	0.078	0.000606
3	-0.0306	1.925	-0.00297	0.00564	0.00195	3.1E-5
4	-0.0308	1.925	-0.00297	0.000136	8.0E-6	1.0E-6

Таблица 2. Метод Гаусса - Зейделя

x_1	-82653 / 2685172 = -0.3078126838
x_2	9003 / 4678 = 1.92454040188
x_3	-1996 / 671293 = -0.00297336632

Таблица 3. Метод Гаусса

Метод	**Полученный вектор X**
Метод Гаусса	{-0.03078, 1.92454, -0.00297}
Метод простой итераций	{-0.03077, 1.92422, -0.00297}
Метод Гаусса - Зейделя	{-0.03078, 1.92454, -0.00297}

Таблица 4. Значения полученные при помощи разработанной программы

5. Листинг разработанной программы

Main.java

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


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


/**
 * Класс, содержащий точку входа в программу - метод main.
 * Язык: java
 *
 * Реализация второй лабораторной работы по дисциплине: Вычислительная математика
 *
 * Текст задания:
 *  Решить систему линейных уравнений методом Гаусса, итерационным методом и методом Гаусса-Зейделя.
 *  При необходимости преобразовать систему к диагонально преобладающему виду. Сделать оценку количества итераций
 *  для итерационных методов, сравнить. Задание по вариантам. Номер варианта – номер студента в списке группы.
 *  ε=0.001
 *
 * Ур-ие:
 *  1.6x1  + 0.12x2 + 0.57x3 = 0.18
 *  0.38x1 + 0.25x2 - 54x3   = 0.63
 *  0.28x1 + 0.46x2 - 1.12x3 = 0.88
 *
 * WARNING!!!
 *   Программа не имеет системы ввода коэффициентов системы ур-ий, т.к.
 *   предназначена для решения только 15 Варианта ЛР №2
 *
 * @release: -
 * @last_update: -
 *
 * @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\лабораторная работа №2 <<" + PURPLE + "Решение системы +
      + линейных уравнений итерационным методом и методом Гаусса-Зейделя" +
      + RESET + ">>");

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


    System.out.println("Программа для решения системы 3ёх нелинейных уравнений.");
    System.out.println("\tОбщий вид системы таких уравнений: ");
    System.out.println("\t\ta11 * x1 + a12 * x2 + a13 * x3 = b1");
    System.out.println("\t\ta21 * x1 + a22 * x2 + a23 * x3 = b2");
    System.out.println("\t\ta31 * x1 + a32 * x2 + a33 * x3 = b3");
    System.out.println();


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


    //Изначально ур-ие имеет вид:
    //  1.6 * x1  + 0.12 * x2 + 0.57 * x3 = 0.18
    //  0.38 * x1 + 0.25 * x2 - 54.0 * x3 = 0.63
    //  0.28 * x1 + 0.46 * x2 - 1.12 * x3 = 0.88
    //
    //Однако при записи в систему строки 2 и 3 переставлены
    //В противном случае из-за сильно выделяющегося коэффициента -54.0
    //система не сходится
    double[][] coefficients = {{1.6,  0.12,  0.57},
                               {0.28, 0.46, -1.12},
                               {0.38, 0.25, -54.0}};
    double[]  vectorB        = {0.18, 0.88, 0.63};
```

```
//Запись введенных коэффициентов в систему
system.setCoefficients(coefficients);


//запись вектора B в систему
system.setVectorB(vectorB);


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


//Переменная для хранения результата ввода
String ch = "";


//Выбор стратегии решения
while (!ch.equals("q"))
{
    System.out.println("Выберите метод для решения уранвения:");
    System.out.println("\t1. Метод Гаусса");
    System.out.println("\t2. Метод Простой итерации");
    System.out.println("\t3. Метод Гаусса-Зейделся");
    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 GaussSolution();
        case ("2") -> strategy = new SimpleIterationSolution();
        case ("3") -> strategy = new GaussSeidelSolution();
        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
            double[] resArr = strategy.getSolution(system, validator);


            //Выводим получившиеся ответы
            System.out.print(RED + "Ответ: " + RESET);
            for (int i = 0; i < resArr.length - 1; i++)
            {
                System.out.printf("x" + (i + 1) + ": %.5f", resArr[i]);
                System.out.print("; ");
            }
            System.out.println();


            //Выводим затраченное время для данного решения
            System.out.println("Затраченное время: " + CYAN + (end - start)/1000.0 +
                                                                    + RESET + " секунд");
            System.out.println("Кол-во итераций:   " + CYAN + (int)resArr[3] + RESET +
                                                                        + " итераций\n");
        }
    }
}
```

equation/SystemOfEquations.java

```
 package
 equation;

/**
 * Интерфейс реализующий основные методы системы ур-ий.
 *
 * Общий вид системы:
 *      a11 * x1 + a12 * x2 + ... + a1n * xn = b1
 *      a21 * x1 + a22 * x2 + ... + a2n * xn = b2
 *                        ...
 *      an1 * x1 + an2 * x2 + ... + ann * x2 = bn
```

```
 *
 * Содержит 1 метод:
 *    setCoefficients - для задания коэффициентов при x
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SystemOfThreeEquations
 * */
public interface SystemOfEquations
{
    /**
     * Метод для задания коэфициентов при x
     *
     * @param coefficients - массив коэфициентов при членах уравнения.
     * */
    void setCoefficients(double[][] coefficients);


    /**
     * Метод для получения коэффициентов при x
     *
     * @return массив коэффициентов при членах уравнения.
     * */
    double[][] getCoefficients();


    /**
     * Метод для задания вектора b
     *
     * @param vectorB - массив значений вектора b.
     * */
    void setVectorB(double[] vectorB);


    /**
     * Метод для получения вектора b
     *
     * @return массив значений вектора b.
     * */
    double[] getVectorB();
}
```

equation/SystemOfThreeEquations.java

```
package equation;


import java.util.Arrays;
```

```
/**
 * Класс системы из 3ёх нелинйеных уравнений
 * Реализует интерфейс SystemOfEquations
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SystemOfEquations
 * */
public class SystemOfThreeEquations implements SystemOfEquations
{
    public double[][] coefficients = new double[3][3];    //массив коэффициентов перед
                                                          //иксами

    public double[]   vectorB      = new double[3];       //массив значений вектора b


    /**
     * Метод для задания коэффициентов при x
     *
     * @param coefficients - массив коэффициентов при членах уравнения.
     * */
    @Override
    public void setCoefficients(double[][] coefficients)
    {
        for (int i = 0; i < coefficients.length; i++)
        {
            System.arraycopy(coefficients[i], 0, this.coefficients[i], 0,
                                                 coefficients[0].length);
        }
    }


    /**
     * Метод для получения коэффициентов при x
     *
     * @return  массив коэффициентов при членах уравнения.
     * */
    @Override
    public double[][] getCoefficients()
    {
        return coefficients;
    }


    /**
     * Метод для задания вектора b
     *
     * @param vectorB - массив значений вектора b.
```

```
     * */
    @Override
    public void setVectorB(double[] vectorB)
    {
        System.arraycopy(vectorB, 0, this.vectorB, 0, vectorB.length);


    }


    @Override
    public double[] getVectorB() {
        return vectorB;
    }


    /**
     * Конуструктор без параметров
     * */
    public SystemOfThreeEquations()
    {
    }
}
```

validator/Validator.java

```
package validator;


import java.util.List;


/**
 * Интерфейс реализующий метод проверки
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see ResponseValidator
 * */
public interface Validator
{
    /**
     * Метод для проверки правильности значения
     *
     * @param prevValues    - вектор значений, полученных про прошлой итерации
     * @param presentValues - вектор значений, полученный при текущей итерации
     * @return true - если найдено подходящее решение
     * */
    boolean isValid(double[] presentValues, double[] prevValues);
```

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

validator/ResponceValidator.java

```java
package validator;


import java.util.Arrays;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;


/**
 * Singleton класс реализующий интерфейс проверки Validator.
 * Проверяет решение на соответствие критериям остановки.
 *
 * WARNING!!!
 *    Критерий остановки: найдено точное значение f(Xn) = 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:
     *  - Максимум из поэлементных разностей двух векторов решений < ε
     *    max(|Xi^(k) - Xi^(k - 1)|) < ε
     *
```

```
 * @param prevValues    - предыдущее значение функции - f(Xn-1)
 * @param presentValues - текущее значение функции - f(Xn)
 * @return true - если найдено подхоядщее решение
 * */
@Override
public boolean isValid(double[] presentValues, double[] prevValues)
{
    Double[] tempArr = new Double[presentValues.length - 1];


    for (int i = 0; i < presentValues.length - 1; i++)
    {
        tempArr[i] = Math.abs((presentValues[i] - prevValues[i]));
    }


    return Collections.max(Arrays.asList(tempArr)) < 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;
```

```java
    }


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

equation_solution_strategy/SolutionStrategy.java

```java
package equation_solution_strategy;


import equation.SystemOfEquations;
import validator.Validator;


/**
 * Общий интерфейс всех стратегий решения.
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see GaussSolution
 * @see GaussSeidelSolution
 * @see SimpleIterationSolution
 * */
public interface SolutionStrategy
{
    /**
     * Метод для вызова той или иной стратегии решения.
     *
     * @param system    - система, которую необходимо решить
     * @param validator - валидатор, с заданным параметром проверки
     * @return список значений, являющимися решениями данной системы уравнений.
     * */
    double[] getSolution(SystemOfEquations system, Validator validator);
}
```

equation_solution_strategy/GaussSolution.java

```java
package equation_solution_strategy;


import equation.SystemOfEquations;
import validator.Validator;
```

```java
/**
 * Класс, реализующий решение методом Гаусса.
 * Реализует интерфейс SolutionStrategy
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class GaussSolution implements SolutionStrategy
{
    /**
     * Метод для получения решений методом Гаусса.
     *   Все шаги данного метода прописаны только Варианта №15
     *   ЛР №2
     *
     * @param system    - система, которую необходимо решить
     * @param validator - валидатор, с заданным параметром проверки
     * @return список значений, являющимися решениями данной системы уравнений.
     * */
    @Override
    public double[] getSolution(SystemOfEquations system, Validator validator)
    {
        double[][] coefficients     = new double[3][3];            //перезаписываем
                                                                   //значения из массива

        double[][] tempCoefficients = system.getCoefficients();    //коэффициентов системы
                                                                   //в промежуточный массив

        for (int i = 0; i < coefficients.length; i++)              //в противном случае при
        {                                                          //работе с массивом
                                                                   //меняются коэф-ты в
                                                                   //самой системе
            System.arraycopy(tempCoefficients[i], 0, coefficients[i], 0,
                                                 coefficients[0].length);
        }


        double[] vectorB = new double[3];               //аналогичную перезапись производим
                                                        //для вектора b
        System.arraycopy(system.getVectorB(), 0, vectorB, 0, vectorB.length);


        double[]   currentApproximation = new double[4];       //массив для хранения
                                                               //вектора текущих

        currentApproximation[3] = 0;
```

```
//1-ую строку делим на 1.6
for (int i = 0; i < coefficients.length; i++)
{
    coefficients[0][i] /= 1.6;
}
vectorB[0] /= 1.6;


//от 2 строки отнимаем 1 строку, умноженную на 0.28;
//от 3 строки отнимаем 1 строку, умноженную на 0.38
for (int i = 0; i < coefficients.length; i++)
{
    coefficients[1][i] = coefficients[1][i] - 0.28 * coefficients[0][i];
    coefficients[2][i] = coefficients[2][i] - 0.38 * coefficients[0][i];
}
vectorB[1] = vectorB[1] - vectorB[0] * 0.28;
vectorB[2] = vectorB[2] - vectorB[0] * 0.38;


//2-ую строку делим на 0.439
for (int i = 0; i < coefficients.length; i++)
{
    coefficients[1][i] /= 0.439;
}
vectorB[1] /= 0.439;


//от 1 строки отнимаем 2 строку, умноженную на 0.075;
//от 3 строки отнимаем 2 строку, умноженную на 0.2215
for (int i = 0; i < coefficients.length; i++)
{
    coefficients[0][i] = coefficients[0][i] - 0.075  * coefficients[1][i];
    coefficients[2][i] = coefficients[2][i] - 0.2215 * coefficients[1][i];
}
vectorB[0] = vectorB[0] - 0.075  * vectorB[1];
vectorB[2] = vectorB[2] - 0.2215 * vectorB[1];


//3-ую строку делим на -53.5199430523918
for (int i = 0; i < coefficients.length; i++)
{
    coefficients[2][i] /= -53.5199430523918;
}
vectorB[2] /= -53.5199430523918;


//от 1 строки отнимаем 3 строку, умноженную на 0.564636;
//к 2 строке добавляем 3 строку, умноженную на 2.7784738041
```

```
        for (int i = 0; i < coefficients.length; i++)
        {
            coefficients[0][i] = coefficients[0][i] - 0.564636     * coefficients[2][i];
            coefficients[1][i] = coefficients[1][i] + 2.7784738041 * coefficients[2][i];
        }
        vectorB[0] = vectorB[0] - 0.564636 * vectorB[2];
        vectorB[1] = vectorB[1] + 2.7784738041 * vectorB[2];


        //После последнего шага система принимает вид:
        //   1 0 0 | -0.03078
        //   0 1 0 | 1.92454
        //   0 0 1 | -0.00297
        //
        //Дальше сопоставляем иксы и их значения из вектора b
        for (double[] coefficient : coefficients)
        {
            for (int j = 0; j < coefficients[0].length; j++)
            {
                if (coefficient[j] == 1.0) {
                    currentApproximation[j] = vectorB[j];
                }
            }
        }


        return currentApproximation;
    }


    /**
     * Конструктор без параметров.
     * */
    public GaussSolution()
    {
    }
}
```

equation solution strategy/GaussSeidelSolution.java

```
package equation_solution_strategy;


import equation.SystemOfEquations;
import validator.Validator;


/**
```

```
 * Класс реализующий решение методом Гаусса-Зейделя
 * Реализует интерфейс SolutionStrategy
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class GaussSeidelSolution implements SolutionStrategy
{
    /**
     * Метод для получения решений методом Гаусса-Зейделя
     *
     * @param system    - система, которую необходимо решить
     * @param validator - валидатор, с заданным параметром проверки
     * @return список значений, являющимися решениями данной системы уравнений.
     * */
    @Override
    public double[] getSolution(SystemOfEquations system, Validator validator)
    {
        double[][] coefficients     = new double[3][3];            //перезаписываем
                                                                   //значения из массива

        double[][] tempCoefficients = system.getCoefficients();    //коэффициентов системы
                                                                   //в промежуточный массив

        for (int i = 0; i < coefficients.length; i++)              //в противном случае при
                                                                   //работе с массивом
        {                                                          //меняются коэф-ты в
                                                                   //самой системе
            System.arraycopy(tempCoefficients[i], 0, coefficients[i], 0,
                                                     coefficients[0].length);
        }


        double[] vectorB = new double[3];              //аналогичную перезапись производим
                                                       //для вектора b

        System.arraycopy(system.getVectorB(), 0, vectorB, 0, vectorB.length);


        double[]   prevApproximation    = {0.0, 0.0, 0.0};     //инициализируем значения
                                                               //вектора
                                                               //прошлых приближений нулями


        double[]   currentApproximation = new double[4];       //массив для хранения
                                                               //вектора текущих
                                                               //приближений. В последний
                                                               //элемент записывается
```

```
                                                            //кол-во итераций

    int count = 0;                                                  //счетчик итераций

    //В цикле вычисляем значения вектора приближений
    for (;;)
    {
        //значение X1
        currentApproximation[0] = (vectorB[0]-coefficients[0][1]*prevApproximation[1]+
                coefficients[0][2] * prevApproximation[2])) / coefficients[0][0];

        //значение X2

        currentApproximation[1]= (vectorB[1]-coefficients[1][0]*currentApproximation[0]
          + coefficients[1][2] * prevApproximation[2])) / coefficients[1][1];

        //значение X3

        currentApproximation[2]=(vectorB[2]-(coefficients[2][0]*currentApproximation[0]
            + coefficients[2][1] * currentApproximation[1])) / coefficients[2][2];




        //При каждой итерации увеличиваем счетчик
        count++;


        //Проверка на соответствие условия валидатора
        if (validator.isValid(currentApproximation, prevApproximation)) break;


        //В вектор прошлый значений заносим текущие
        System.arraycopy(currentApproximation, 0, prevApproximation, 0,
                                             prevApproximation.length);
    }


    //Заносим кол-во итераций в конец вектора приближений
    currentApproximation[3] = count;


    return currentApproximation;
}
```

```
    /**
     * Конструктор без параметров.
     * */
    public GaussSeidelSolution()
    {
    }
}
```

equation_solution_strategy/SimpleIterationSolution.java

```
package equation_solution_strategy;


import equation.SystemOfEquations;
import validator.Validator;


/**
 * Класс, реализующий решение методом Простой итерации (метод Якоби).
 * Реализует интерфейс SolutionStrategy
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 * @see SolutionStrategy
 * */
public class SimpleIterationSolution implements SolutionStrategy
{
    /**
     * Метод для получения решений методом Простой итерации (метод Якоби).
     *
     * @param system    - система, которую необходимо решить
     * @param validator - валидатор, с заданным параметром проверки
     * @return список значений, являющимися решениями данной системы уравнений.
     * */
    @Override
    public double[] getSolution(SystemOfEquations system, Validator validator)
    {
        double[][] coefficients     = new double[3][3];            //перезаписываем
                                                                   //значения из массива

        double[][] tempCoefficients = system.getCoefficients();    //коэффициентов системы
                                                                   //в промежуточный массив

        for (int i = 0; i < coefficients.length; i++)              //в противном случае при
                                                                   //работе с массивом
        {                                                          //меняются коэф-ты в
                                                                   //самой системе
            System.arraycopy(tempCoefficients[i], 0, coefficients[i], 0,
                                                  coefficients[0].length);
```

```
        }

        double[] vectorB = new double[3];               //аналогичную перезапись производим
                                                        //для вектора b

        System.arraycopy(system.getVectorB(), 0, vectorB, 0, vectorB.length);

        double[]   prevApproximation    = {0.0, 0.0, 0.0};

        double[]   currentApproximation = new double[4];

        int count = 0;                                                      //счетчик итераций

        //В цикле вычисляем значения вектора приближений
        for (;;)
        {
            currentApproximation[0] = (vectorB[0]- (coefficients[0][1]*prevApproximation[1]
               + coefficients[0][2] * prevApproximation[2])) / coefficients[0][0];

            currentApproximation[1] = (vectorB[1]- (coefficients[1][0]*prevApproximation[0]
               + coefficients[1][2] * prevApproximation[2])) / coefficients[1][1];

            currentApproximation[2] = (vectorB[2] -(coefficients[2][0]*prevApproximation[0]
               + coefficients[2][1] * prevApproximation[1])) / coefficients[2][2];

            //При каждой итерации увеличиваем счетчик
            count++;

            //Проверка на соответствие условия валидатора
            if (validator.isValid(currentApproximation, prevApproximation)) break;

            //В вектор прошлый значений заносим текущие
            System.arraycopy(currentApproximation, 0, prevApproximation, 0,
prevApproximation.length);
```

```
        }


        //Заносим кол-во итераций в конец вектора приближений
        currentApproximation[3] = count;


        return currentApproximation;
    }


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

6. Результаты работы программы

```
				Лабораторная работа №2 <<Решение системы линейных уравнений итерационным методом и методом Гаусса-Зейделя>>
Программа для решения системы 3ёх нелинейных уравнений.
	Общий вид системы таких уравнений:
		a11 * x1 + a12 * x2 + a13 * x3 = b1
		a21 * x1 + a22 * x2 + a23 * x3 = b2
		a31 * x1 + a32 * x2 + a33 * x3 = b3

Введите точность ответа (epsilon): 0,001

Выберите метод для решения уравнения:
	1. Метод Гаусса
	2. Метод Простой итерации
	3. Метод Гаусса-Зейделся

	Введите q для выхода
Ввод: 1

Ответ: x1: -0,03078; x2: 1,92454; x3: -0,00297;
Затраченное время: 0.0 секунд
Кол-во итераций:   0 итераций
```

```
	1. Метод Гаусса
	2. Метод Простой итерации
	3. Метод Гаусса-Зейделся

	Введите q для выхода
Ввод: 2

Ответ: x1: -0,03077; x2: 1,92422; x3: -0,00297;
Затраченное время: 0.0 секунд
Кол-во итераций:   6 итераций

Выберите метод для решения уравнения:
	1. Метод Гаусса
	2. Метод Простой итерации
	3. Метод Гаусса-Зейделся

	Введите q для выхода
Ввод: 3

Ответ: x1: -0,03078; x2: 1,92454; x3: -0,00297;
Затраченное время: 0.0 секунд
Кол-во итераций:   4 итераций

Выберите метод для решения уравнения:
	1. Метод Гаусса
	2. Метод Простой итерации
	3. Метод Гаусса-Зейделся

	Введите q для выхода
Ввод: q

Завершение работы...
```

7. Вывод

Итерационные методы решения системы линейных уравнений удобно использовать при большом кол-ве неизвестных, т.к. сложность прямых вычислений вырастает при каждом новом неизвестном.

Метод Гаусса имеет высокую сложность вычислений и сложную реализацию для решения уравнений с любыми коэффициентами.

Методы Якоби и Гаусса-Зейделя имеют простую программную реализацию для решений системы линейных уравнений общего вида при любом кол-ве неизвестных (в ходе лабораторной реализованы частные случаи для 3ех неизвестных).

Метод Гаусса-Зейделя является более быст родейственным, чем метод Якоби, что было подтверждено практически в ходе выполнения работы.