

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчет по лабораторной работе

«Решение системы линейных алгебраических уравнений методом Гаусса»

Выполнил:

Студент группы 3824Б1ПМ1
Ушаков С. Д.

Проверила:

Бусько П.В.

Содержание

Введение	3
Постановка задачи.....	4
Руководство пользователя	5
Описание программной реализации	7
Результаты экспериментов	20
Заключение.....	23
Литература	24
Приложение	25

Введение

Системы линейных алгебраических уравнений (СЛАУ) занимают центральное место в прикладной математике и программной инженерии. Они возникают при решении множества практических задач, связанных с моделированием физических процессов, анализом электрических цепей, оптимизацией, обработкой изображений, машинным обучением и другими направлениями науки и техники.

Одним из наиболее универсальных и широко применяемых методов решения СЛАУ является метод Гаусса. Этот метод основан на последовательном выполнении элементарных преобразований строк расширенной матрицы системы, что позволяет свести её к верхнетреугольному виду, из которого решение может быть найдено обратным ходом. Благодаря своей наглядности, эффективности и относительной простоте реализации, метод Гаусса часто используется как в учебных курсах, так и в инженерной практике.

Задача реализации алгоритма метода Гаусса в виде программного кода имеет важное образовательное значение. Она позволяет закрепить навыки программирования, научиться работать с многомерными структурами данных, обрабатывать ввод и вывод, а также реализовать численные алгоритмы с контролем точности и устойчивости. Кроме того, проверка систем на наличие несовместности или бесконечного числа решений позволяет лучше понять математическую природу линейных уравнений и особенности их решений.

Таким образом, выполнение данной лабораторной работы способствует формированию у студентов прочного базового понимания как численных методов, так и принципов разработки программ, способных решать прикладные задачи анализа и моделирования.

Постановка задачи

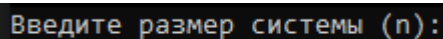
Цель данной работы: реализовать метод Гаусса для решения системы линейных алгебраических уравнений (СЛАУ).

Для достижения данной цели были поставлены следующие задачи:

1. Реализовать шаблонный класс вектор.
2. Реализовать класс квадратная матрица, который является шаблоном класса вектор от вектора.
3. Реализовать класс СЛАУ, который является наследником квадратной матрицы и у него должна быть метод Гаусса. Метод Гаусса принимает правую часть в качестве аргумента и возвращает ответ в виде вектора.
4. Реализовать функцию `swap`, меняющая строки и элементы местами.

Руководство пользователя

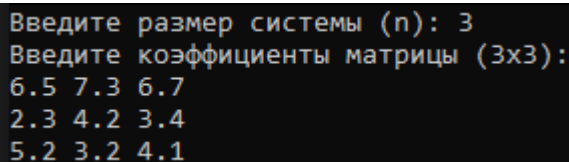
При запуске кода, пользователю предлагают выбрать размерность системы. Программа заведомо предполагает, что система будет квадратная, поэтому пользователю достаточно ввести одно число (если будет введено число 3, то будет создаваться матрица 3x3).



```
Введите размер системы (n):
```

Рис. 1 – Первая строчка программы.

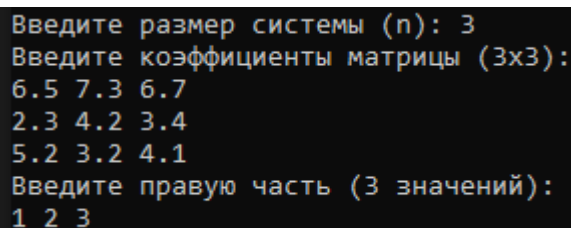
После того, как пользователь введет размер системы, необходимо заполнить матрицу самостоятельно. Для этого с клавиатуры вводятся значения переменных. Таблица заполняется построчно, для каждой новой переменной нужно вводить пробел.



```
Введите размер системы (n): 3
Введите коэффициенты матрицы (3x3):
6.5 7.3 6.7
2.3 4.2 3.4
5.2 3.2 4.1
```

Рис. 2 – введенные коэффициенты матрицы.

Затем пользователь должен указать правую часть созданной матрицы.



```
Введите размер системы (n): 3
Введите коэффициенты матрицы (3x3):
6.5 7.3 6.7
2.3 4.2 3.4
5.2 3.2 4.1
Введите правую часть (3 значений):
1 2 3
```

Рис. 3 – введенная правая часть.

В конце программа выводит решение системы и завершается.

```
Введите размер системы (n): 3
Введите коэффициенты матрицы (3x3):
6.5 7.3 6.7
2.3 4.2 3.4
5.2 3.2 4.1
Введите правую часть (3 значений):
1 2 3
Решение:
x[0] = -4.66169
x[1] = -6.38153
x[2] = 11.6248
```

Рис. 4 – конец программы.

Описание программной реализации

Код состоит из 4 файлов: vector.h, matrix.h, SLAU.h, GaussinElim.cpp.

Vector.h

Файл: Vector.h

Назначение: шаблонный класс Vector, который реализует простой динамический массив (аналог std::vector, но написан вручную).

Шаблон: template<typename T> — универсальный, работает с любым типом T.

Содержит:

Закрытые поля для хранения данных и размера

Методы для доступа, добавления элементов и управления памятью

Поля класса

```
#pragma once

#include <iostream>

template<typename T>
class Vector {
private:
    T* data;           // Указатель на массив
    int size;          // Количество элементов
    int capacity;       // Выделенная память
};
```

Рис. 5 – начало программы и поля класса.

T* data - Указатель на динамический массив элементов типа T

int size - Текущее количество элементов

int capacity - Текущая выделенная вместимость (максимум, что можно без перераспределения)

Метод `resize`.

```
void resize() {  
    capacity *= 2;  
    T* newData = new T[capacity];  
    for (int i = 0; i < size; ++i)  
        newData[i] = data[i];  
    delete[] data;  
    data = newData;  
}
```

Рис. 6 – метод `resize`, внутри `private`.

Назначение: Увеличивает вместимость массива, если она исчерпана

Алгоритм:

- Удваивает `capacity`
- Создаёт новый массив с новой ёмкостью
- Копирует туда старые данные
- Удаляет старый массив
- Переназначает `data` на новый

Конструкторы.

1. Конструктор по умолчанию

```
Vector() {  
    size = 0;  
    capacity = 4;  
    data = new T[capacity];  
}
```

Рис. 7 - Конструктор по умолчанию.

- Устанавливает размер 0
- Начальная вместимость — 4
- Выделяет память под 4 элемента

2. Конструктор с параметрами

```
Vector(int n, const T& value = T()) {  
    size = n;  
    capacity = n;  
    data = new T[capacity];  
    for (int i = 0; i < size; ++i)  
        data[i] = value;  
}
```

Рис. 8 - Конструктор с параметрами.

- Создаёт вектор из n элементов
- Все элементы инициализируются значением value
- Если value не передан, используется значение по умолчанию для T

3. Конструктор копирования

```
Vector(const Vector& other) {  
    size = other.size;  
    capacity = other.capacity;  
    data = new T[capacity];  
    for (int i = 0; i < size; ++i)  
        data[i] = other.data[i];  
}
```

Рис. 10 - Конструктор копирования.

- Копирует данные из другого объекта Vector
- Выделяет новую память, копирует каждый элемент

Оператор присваивания

```
Vector& operator=(const Vector& other) {  
    if (this != &other) {  
        delete[] data;  
        size = other.size;  
        capacity = other.capacity;  
        data = new T[capacity];  
        for (int i = 0; i < size; ++i)  
            data[i] = other.data[i];  
    }  
    return *this;  
}
```

Рис. 9 – Оператор присваивания.

- Проверка на самоприсваивание
- Удаляет старые данные
- Копирует размер, ёмкость, значения
- Возвращает *this для возможности цепочки присваиваний

Деструктор

```
~Vector() {  
    delete[] data;  
}
```

Рис. 11 – Деструктор.

Освобождает память, выделенную под массив data

Публичные методы

1. void push_back(const T& value)

```
void push_back(const T& value) {  
    if (size >= capacity)  
        resize();  
    data[size++] = value;  
}
```

Рис. 12 – push_back

- Добавляет элемент в конец
- Если достигнута ёмкость — вызывает resize()
- Увеличивает size

2. T& operator[](int index)

```
T& operator[](int index) {  
    return data[index];  
}
```

Рис. 13 – Перегрузка оператора [].

Позволяет обращаться к элементам вектора **по индексу** и **изменять** их. Это перегрузка оператора [].

3. const T& operator[](int index) const

```
const T& operator[](int index) const {  
    return data[index];  
}
```

Рис. 14 – Перегруженный вариант.

- Возвращают ссылку на элемент по индексу
- Перегрузка: одна для изменения, вторая для чтения из const объекта

4. int getSize() const

```
int getSize() const {  
    return size;  
}
```

Рис. 15 – Геттер.

- Возвращает текущий размер (кол-во элементов)

5. void swap(int i, int j)

```
void swap(int i, int j) {  
    T temp = data[i];  
    data[i] = data[j];  
    data[j] = temp;  
}
```

Рис. 16 - swap

- Меняет местами два элемента по индексам *i* и *j*

2. Matrix.h

1. Подключение заголовка

```
#include "Vector.h"
```

Рис. 17 – заголовок в matrix.h.

Подключает твой самописный динамический массив **Vector<T>**, используется внутри матрицы.

2. Объявление шаблонного класса

```
template<typename T>  
class Matrix {
```

Рис. 18 – класс Matrix.

Матрица любого типа *T*. Используется **вложенный вектор векторов**:
Vector<Vector<T>> — строки из элементов.

3. Поля

```
protected:
    int size;
    Vector<Vector<T>> data;
```

Рис. 19 – поля в Matrix.

size — размер квадратной матрицы (n x n)

data — вектор строк, каждая строка — Vector<T>, то есть data[i][j] — доступ к элементу (i, j)

4. Конструктор

```
Matrix(int n) : size(n), data(n) {
    for (int i = 0; i < size; ++i) {
        data[i] = Vector<T>(size);
    }
}
```

Рис. 20 – Конструктор в Matrix.

- Создаёт квадратную матрицу n x n
- Сначала создаётся n пустых строк (внутренний Vector<T> не инициализирован)
- Затем каждая строка инициализируется вектором длины n — это и есть строка матрицы

5. int getSize() const

```
int getSize() const {
    return size;
}
```

Рис. 21 – Геттер в Matrix.

Возвращает размер матрицы (число строк/столбцов)

6. Перегрузка operator[]

```
Vector<T>& operator[](int i) {  
    return data[i];  
}
```

Рис. 22 – operator[].

Позволяет обращаться к строке по индексу:

- matrix[i][j] работает: сначала строка matrix[i], потом элемент matrix[i][j]
- Первый вариант для изменения
- Второй — для чтения из const объекта

7. void swapRows(int i, int j)

```
void swapRows(int i, int j) {  
    data.swap(i, j); // использование swap() из vector.h  
}
```

Рис. 23 – swapRows.

Меняет местами строки i и j. Использует swap из Vector.h.

8. void print() const

```
void print() const {  
    for (int i = 0; i < size; ++i) {  
        for (int j = 0; j < size; ++j) {  
            std::cout << data[i][j] << "\t";  
        }  
        std::cout << "\n";  
    }  
}
```

Рис. 24 – print(), Matrix.

Печатает матрицу в консоль в табличной форме

Краткий итог

Класс Matrix<T>:

- Представляет квадратную матрицу
- Использует вектор векторов
 - Позволяет:
 - Получить размер
 - Менять строки
 - Получать доступ к элементам
 - Печатать содержимое

3. Matrix.h

Общая структура.

```
#include "Matrix.h"
#include "Vector.h"

class SLAU : public Matrix<double> {
|
```

Класс SLAU наследует Matrix<double> — то есть он уже матрица коэффициентов. Это удобно, так как доступ к элементам — через (*this)[i][j].

using Matrix::Matrix.

```
using Matrix::Matrix;
```

Наследует конструкторы из Matrix, чтобы создавать SLAU как квадратную матрицу.

Метод: Vector<double> gauss(Vector<double> rhs)¹

Решает СЛАУ методом Гаусса с выбором главного элемента.

Параметр:

rhs — правая часть

Возвращает:

Вектор решений Vector<double>

Прямой ход:

Поиск главного элемента по модулю в столбце i

Перестановка строк (матрицы и rhs)

Проверка на вырождение

Вся строка = 0, rhs $\neq 0 \rightarrow$ нет решений

Вся строка = 0, rhs = 0 $\rightarrow \infty$ решений

Деление строки на ведущий элемент

Обнуление нижних строк

Обратный ход:

Идём снизу вверх

Выражаем переменные одну за другой

Результат:

Возвращает `Vector<double>` — вектор решений системы.

¹Часть кода слишком большая. Смотреть в самом коде или в конце отчета в приложении.

4. GaussinElim_LW3.cpp

Заголовки.

```
#include "Vector.h"  
#include "Matrix.h"  
#include "SLAU.h"
```

Рис. 25 – Заголовки в GaussinElim_LW3.cpp.

Подключаются все классы: вектор, матрица, СЛАУ.

setlocale(LC_ALL, "Ru").

```
setlocale(LC_ALL, "Ru");
```

Рис. 26 – Функция setlocale.

Позволяет выводить русские символы в консоль (на Windows).

Ввод размерности

```
int n;  
std::cout << "Введите размер системы (n): ";  
std::cin >> n;
```

Рис. 27 – Ввод размерности.

Пользователь вводит размер системы $n \times n$.

Создание объектов

```
SLAU system(n);  
Vector<double> rhs(n);
```

Рис. 28 – Создание объектов.

Ввод матрицы

```
std::cout << "Введите коэффициенты матрицы (" << n << "x" << n << "):\n";
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        double value;
        std::cin >> value;
        system[i][j] = value;
    }
}
```

Рис. 29 – Ввод и создание матрицы.

Пользователь вводит $n \times n$ коэффициентов.

Ввод правой части

```
std::cout << "Введите правую часть (" << n << " значений):\n";
for (int i = 0; i < n; ++i) {
    std::cin >> rhs[i];
}
```

Рис. 30 – Ввод правой части.

Ввод n значений правой части.

Решение

```
Vector<double> solution = system.gauss(rhs);
```

Рис. 31 – Решение созданной матрицы.

Вызывается метод `gauss`, возвращает вектор решений.

Вывод результата

```
std::cout << "Решение:\n";
for (int i = 0; i < n; ++i) {
    std::cout << "x[" << i << "] = " << solution[i] << "\n";
}
```

Рис. 32 – Вывод итога.

Печать всех найденных $x[i]$.

Результаты экспериментов

В рамках лабораторной работы была разработана и протестирована программа для решения систем линейных алгебраических уравнений методом Гаусса. Для проверки корректности работы алгоритма были проведены следующие эксперименты:

Эксперимент 1. Система с единственным решением

Рассмотрим систему уравнений:

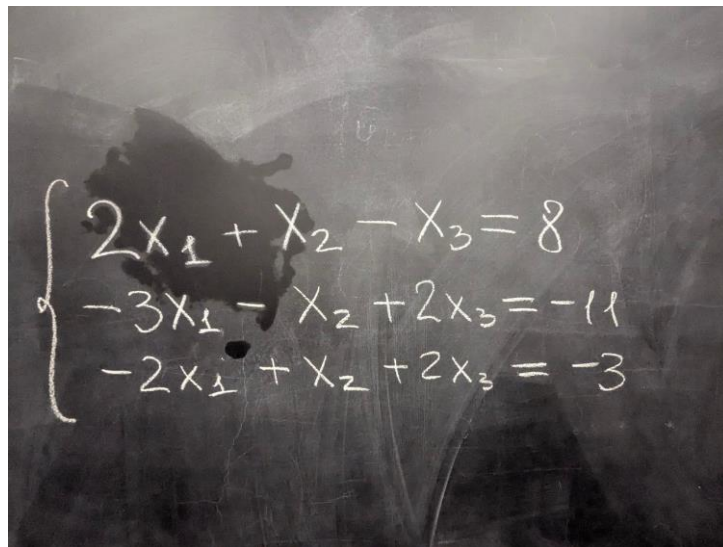

$$\begin{cases} 2x_1 + x_2 - x_3 = 8 \\ -3x_1 - x_2 + 2x_3 = -11 \\ -2x_1 + x_2 + 2x_3 = -3 \end{cases}$$

Рис. 33 – Эксперимент 1, задание.

Ввод данной системы в программу дал следующий результат:

```
Введите размер системы (n): 3
Введите коэффициенты матрицы (3x3):
2 1 -1
-3 -1 2
-2 1 2
Введите правую часть (3 значений):
8 -11 -3
Решение:
x[0] = 2
x[1] = 3
x[2] = -1
```

Рис. 34 – Эксперимент 1, что вывелось.

Вот правильный ответ:

Ответ:

$$\begin{cases} x_1 = 2 \\ x_2 = 3 \\ x_3 = -1 \end{cases}$$

Рис. 35 – Эксперимент 1, правильный ответ.

Сравнение с аналитическим решением показывает полное совпадение, что подтверждает корректность работы алгоритма на типичном примере.

Эксперимент 2. Бесконечно много решений

Система:

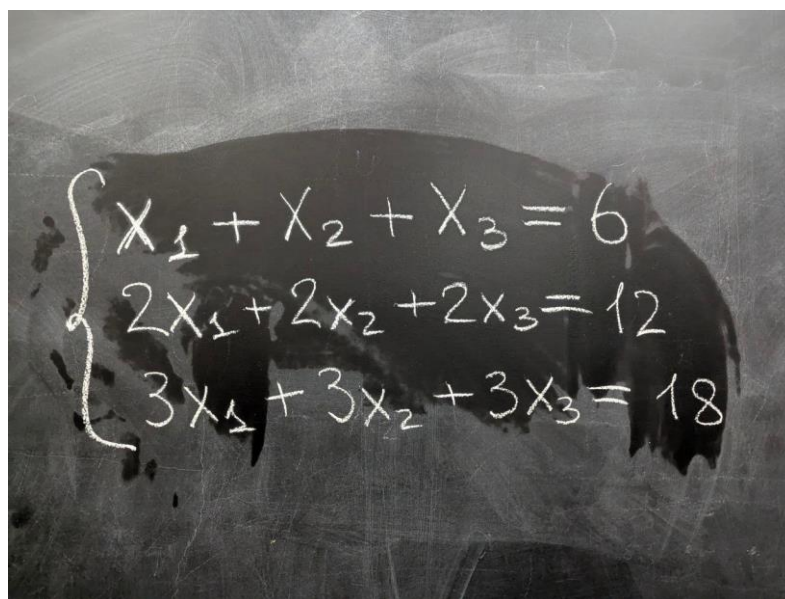

$$\begin{cases} x_1 + x_2 + x_3 = 6 \\ 2x_1 + 2x_2 + 2x_3 = 12 \\ 3x_1 + 3x_2 + 3x_3 = 18 \end{cases}$$

Рис. 36 – Эксперимент 2, задание.

Ввод данной системы в программу дал следующий результат:

```
Введите размер системы (n): 3
Введите коэффициенты матрицы (3x3):
1 1 1
2 2 2
3 3 3
Введите правую часть (3 значений):
6 12 18
Система имеет бесконечно много решений (недоопределена) .
```

Рис. 37 - Эксперимент 1, что вывелось.

Вот правильный ответ:

Ответ:
Система имеет множество решений:
 $\{ x_1 + x_2 + x_3 = 6$

Рис. 38 – Эксперимент 2, правильный ответ.

Программа корректно определила, что система имеет бесконечно много решений (все уравнения линейно зависимы). В ходе вычислений обнаруживаются строки, состоящие только из нулей.

Заключение

В ходе выполнения лабораторной работы была разработана и реализована программа для решения систем линейных алгебраических уравнений методом Гаусса.

Реализация включала выбор главного элемента по столбцу для повышения устойчивости вычислений, а также проверку корректности системы: наличие единственного решения, несовместность или бесконечное множество решений. Были рассмотрены все возможные ситуации при решении системы, включая вырожденные и недоопределённые случаи.

Также в процессе выполнения работы были получены практические навыки построения модульной структуры программы, разделения кода на заголовочные и исходные файлы, использования шаблонов, работы с памятью, тестирования алгоритма на различных примерах и отладки.

Таким образом, поставленные задачи лабораторной работы были успешно решены, а её выполнение способствовало закреплению теоретических знаний и развитию практических навыков программирования на языке C++.

Литература

1. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования СИ //М.: Финансы и статистика. – 1992.
2. Кнут Д. Э. Искусство программирования: Сортировка и поиск. – Издательский дом Вильямс, 2000. – Т. 3.

Приложение

```
#include <iostream>
#include "Vector.h"
#include "Matrix.h"
#include "SLAU.h"

int main() {
    setlocale(LC_ALL, "Ru");

    int n;
    std::cout << "Введите размер системы (n): ";
    std::cin >> n;

    SLAU system(n);
    Vector<double> rhs(n); // правая часть уравнений

    std::cout << "Введите коэффициенты матрицы (" << n << "x" << n <<
");\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            double value;
            std::cin >> value;
            system[i][j] = value;
        }
    }

    std::cout << "Введите правую часть (" << n << " значений):\n";
    for (int i = 0; i < n; ++i) {
        std::cin >> rhs[i];
    }

    Vector<double> solution = system.gauss(rhs);

    std::cout << "Решение:\n";
    for (int i = 0; i < n; ++i) {
        std::cout << "x[" << i << "] = " << solution[i] << "\n";
    }

    return 0;
}
```

```
Vector<double> gauss(Vector<double> rhs) {
    int n = getSize();

    // Прямой ход
    for (int i = 0; i < n; ++i) {
        // Выбор главного элемента по модулю в i-м столбце
        int maxRow = i;
        for (int k = i + 1; k < n; ++k) {
```

```

        if (std::abs((*this)[k][i]) >
std::abs((*this)[maxRow][i])) {
            maxRow = k;
        }
    }

    // Обмен строк
    swapRows(i, maxRow);
    rhs.swap(i, maxRow);

    double pivot = (*this)[i][i];
    if (pivot == 0.0) {
        // Проверим: вся строка – нули, а правая часть не ноль ?
несовместна
        bool allZero = true;
        for (int j = 0; j < n; ++j) {
            if ((*this)[i][j] != 0.0) {
                allZero = false;
                break;
            }
        }

        if (allZero && rhs[i] != 0.0) {
            std::cerr << "Система несовместна: строка " << i << "
содержит только нули, но правая часть = " << rhs[i] << "\n";
            exit(1);
        }
        else if (allZero && rhs[i] == 0.0) {
            // Бесконечно много решений – строка вида 0 = 0
            std::cerr << "Система имеет бесконечно много решений
(недоопределена).\n";
            exit(1);
        }
        else {
            std::cerr << "Система вырождена или плохо обусловлена
(pivot = 0).\n";
            exit(1);
        }
    }

    // Нормализация строки
    for (int j = i; j < n; ++j) {
        (*this)[i][j] /= pivot;
    }
    rhs[i] /= pivot;

    // Обнуляем нижние строки
    for (int k = i + 1; k < n; ++k) {
        double factor = (*this)[k][i];
        for (int j = i; j < n; ++j) {
            (*this)[k][j] -= factor * (*this)[i][j];
        }
    }

```

```

        rhs[k] -= factor * rhs[i];
    }
}

// Обратный ход
Vector<double> result(n);
for (int i = n - 1; i >= 0; --i) {
    result[i] = rhs[i];
    for (int j = i + 1; j < n; ++j) {
        result[i] -= (*this)[i][j] * result[j];
    }
}

return result;
}

```