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 СЛУ методом Гаусса»**

**Выполнил**:

студент группы 3824Б1ПМ1-2

Ерин.Е.C

**Проверила**:

Бусько П.В.

Нижний Новгород

2025

**Содержание**

[Введение 3](#_Toc182419046)

[Постановка задачи 4](#_Toc182419047)

[Руководство пользователя 4](#_Toc182419048)

[Описание программной реализации 6](#_Toc182419049)

[Результаты экспериментов 7](#_Toc182419050)

[Заключение 8](#_Toc182419051)

[Литература 9](#_Toc182419052)

[Приложение 9](#_Toc182419053)

# Введение

Решение систем линейных алгебраических уравнений — это фундаментальная задача в линейной алгебре, имеющая широкое применение в различных научных и технических областях. Метод Гаусса с выбором главного элемента является эффективным подходом для решения таких систем. В рамках данной работы реализован метод Гаусса с выбором ведущего элемента для работы с действительными квадратными матрицами. Программа разработана с использованием объектно-ориентированного подхода, включая шаблонные классы для вектора и матрицы.

# Постановка задачи

Целью данной работы является разработка объектно-ориентированной реализации метода Гаусса для решения систем линейных алгебраических уравнений (СЛАУ) на C++. Для достижения поставленной цели необходимо выполнить следующие задачи:

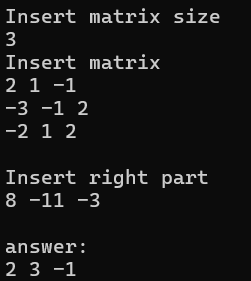
1. Разработка шаблонного класса**vector<T>:** Класс должен обеспечивать функциональность работы с векторами данных типа T.
2. Реализация класса**Matrix<T>:** Класс Matrix<T> должен представлять квадратную матрицу, наследуясь от класса Vector<Vector<T>>.
3. Реализация класса**SLU<T>:** Класс SLAY<T> должен представлять систему линейных алгебраических уравнений и наследоваться от класса Matrix<T>.
4. Реализация метод**а Gaus**в классе**SLU<T>:** Метод Gaus должен реализовывать метод Гаусса с выбором главного элемента. Метод должен принимать вектор правой части СЛАУ (вектор свободных членов) и возвращать вектор решений.
5. Использование вспомогательной функции (опционально**) swap:** Для упрощения реализации и повышения читаемости кода, рекомендуется использовать вспомогательную функцию swap\_v для обмена между строками матрицы и соответствующими элементами вектора правой части при выборе главного элемента.

# Руководство пользователя

**Запуск программы:**

Программа ожидает от пользователя ввода следующих данных через консоль:

1. Размер матрицы
2. Матрицу коэффициентов системы уравнений
3. Вектор свободных членов (правых частей уравнений)



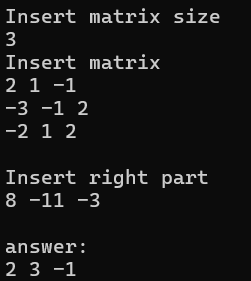
*Рис. 1. Пример входных данных*

**Выполнение:**

Программа автоматически выполняет прямой ход метода Гаусса и обратную подстановку.

**Результат:**

Вектор решений выводится в консоль:



*Рис. 2. Пример вывода решений*

# Описание программной реализации

**Структура проекта:**

1. vector.h - заголовочный файл с объявлением класса vector<T>
2. matix.h - заголовочный файл с объявлением класса Matrix<T>
3. SLU.h - заголовочный файл с объявлением класса SLU<T>
4. main.cpp – пример решения систем линейных алгебраических уравнений методом Гаусса

**Классы:**

1. vector<T> (Приложение 1) - шаблонный одномерный массив поддерживающий копирование, доступ к элементу по индексу.
2. Matrix<T> (Приложение 2) - двумерный шаблонный массив, наследуемый от vector<vector<T>>, представляющий собой матрицу с коэффициентами. Поддерживает копирование, доступ к элементу по индексу.
3. SLU<T> (Приложение 3) - наследник Matrix<T>, реализует метод Гаусса. Метод vector<T> Gaus возвращает вектор решений
4. Функция swap\_v(vector<T> a, vector<T> b) (Приложение 4) - меняет значения векторов для обмена между строками матрицы и соответствующими элементами вектора правой части при выборе главного элемента.

# Результаты экспериментов

**Тестовая система уравнений:**

**Результаты**: , ,

Решение совпадает с аналитическим.

# Заключение

В ходе выполнения лабораторной работы была разработана программа, решающая системы линейных уравнений (СЛУ) с квадратной матрицей методом Гаусса. Благодаря использованию объектно-ориентированного подхода и шаблонов проектирования, программа обладает гибкостью, модульностью и возможностью дальнейшего расширения. Тестирование подтвердило точность вычислений и корректность работы алгоритма.

# Литература

1. Кнут Д. Э. Искусство программирования – Издательский дом Вильямс, 2000. – Т. 3.
2. Керниган Б., Ритчи Д. Язык программирования СИ //М.: Финансы и статистика. – 1992.
3. Седжвик Р., Уэйн К. Алгоритмы на C++: краткий курс. 2013.
4. Лафоре Р. Объектно-ориентированное программирование в C++. — М.: Вильямс, 2009. — 864 с.
5. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. — М.: Вильямс, 2018.

# Приложение

**Приложение 1:**

template<typename T>

class vector

{

protected:

int size;

T\* array;

public:

vector() {

size = 0;

array = nullptr;

}

vector(vector& copy) {

this->size = copy.size;

this->array = new T[copy.size];

for (int i = 0; i < copy.size; i++) {

this->array[i] = copy.array[i];

}

}

vector(int \_size) {

size = \_size;

array = new T[\_size];

for (int i = 0; i < \_size; i++) { array[i] = 0; }

}

T& operator[](int \_num) {

return array[\_num];

}

int get\_size() { return size; };

~vector(){

delete[] array;

array = nullptr;

}

vector& operator=(const vector& other) {

size = other.size;

array = new T[size];

for (size\_t i = 0; i < size; ++i) {

array[i] = other.array[i];

}

return \*this;

}

};

**Приложение 2:**

template<typename T>

class Matrix :public vector<vector<T>>

{

public:

Matrix(int len) {

this->array = new vector<T>[len];

this->size = len;

for (int i = 0; i < len; i++)

{

this->array[i] = vector<T>(len);

}

};

friend std::istream& operator>>(std::istream& is, Matrix<T>& matrx) {

std::cout << "Insert matrix " << std::endl;

for (int i = 0; i < matrx.get\_size(); i++) {

for (int j = 0; j < matrx.get\_size(); j++) {

is >> matrx[i][j];

}

}

return is;

}

vector<T>& operator[](int index) { return this->array[index]; }

Matrix& operator=(const Matrix& other) {

for (size\_t i = 0; i < other.size; ++i) {

this[i] = other[i];

}

return \*this;

}

void print() {

for (int i = 0; i < this->size; i++) {

for (int j = 0; j < this->size; j++)

{

std::cout<< this->array[i][j]<<" ";

}

std::cout << "\n";

}

}

};

**Приложение 3:**

template<typename T>

class SLU : public Matrix<T>

{

private:

public:

SLU(Matrix<T>& r\_p) : Matrix<T>(r\_p) {};

vector<T> Gaus(vector<T>& Right\_P) {

int n = this->get\_size();

Matrix<T> newMatrix(\*this);

for (int i = 0; i < n; ++i) {

int maxRow = i;

for (int k = i + 1; k < n; ++k) {

if (fabs(newMatrix[k][i]) > fabs(newMatrix[maxRow][i])) {

maxRow = k;

}

}

if (maxRow != i) {

swap(newMatrix[i], newMatrix[maxRow]);

swap(Right\_P[i], Right\_P[maxRow]);

}

if (fabs(newMatrix[i][i]) = 0) {

return {};

}

for (int k = i + 1; k < n; ++k) {

double factor = newMatrix[k][i] / newMatrix[i][i];

for (int j = i; j < n; ++j) {

newMatrix[k][j] -= factor \* newMatrix[i][j];

}

Right\_P[k] -= factor \* Right\_P[i];

}

}

for (int i = n - 1; i >= 0; --i) {

for (int j = i + 1; j < n; ++j) {

Right\_P[i] -= newMatrix[i][j] \* Right\_P[j];

}

Right\_P[i] /= newMatrix[i][i];

}

return Right\_P;

}

};

**Приложение 4:**

template<typename T>

void swap\_v(vector<T> a, vector<T> b) {

for (int i = 0; i < a.get\_size(); i++)

{

b[i] = b[i] + a[i];

a[i] = b[i] - a[i];

b[i] = b[i] - a[i];

}

}