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**

**«Метод Гаусса»**

**Выполнила**:

студентка группы 3824Б1ПМ1

Сайгушева В.И.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2025

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Задача: реализовать универсальную программу метода Гаусса-Жордана для решения систем линейных алгебраических уравнений. СЛАУ может быть случайной размерности, а также содержать в себе элементы различных типов данных. В моём случае – «int», «float», «double», «char». Кроме того, было необходимо реализовать класс матрицы как наследуемый класс класса вектора и создать класс своего исключения. Пользователь может выбрать тип данных, ввести элементы матрицы и правого вектора.

# Метод решения

Для решения СЛАУ в моей работе используется алгоритм метода Гаусса-Жордана с выбором ведущего элемента, наибольшего по модулю в данном столбце, что позволяет уменьшить ошибку округления и позволяет добиться большей точности в вычислениях и избежать деления на ноль.

Метод Гаусса-Жордана был реализован с помощью шаблонных классов «Vector» и «Matrix», которые работают с разными типами данных (T).

Сам алгоритм метода основан на приведении матрицы к ступенчатому виду, что позволяет либо сразу найти решения системы, либо выразить решение как пространство линейных векторов.

Элемент класса «Vector» представляет собой массив координат, которые могут быть произвольного типа данных T. Спецификатор конструктора «explicit» у конструктора «Vector» при создании матрицы как вектора векторов через «initializer\_list» не позволяет вызвать конструктор по размеру, чтобы избежать ошибок и утечек памяти. Класс «Matrix» наследует класс «Vector», все его конструкторы. В классе «SLAU» матрица – тоже вектор векторов, но эта матрица уже расширенная.

В каждом столбце выбирается ведущий элемент среди ненулевых строк – максимальный по модулю элемент. Если он меньше погрешности eps, то столбец считается свободным, что возникает при наличии бесконечного множества решений. При необходимости строки меняются местами, чтобы ведущий элемент был на диагонали, что обеспечивает корректность следующих преобразований. Ведущая строка делится на значение ведущего элемента, а затем все строки обнуляются так, чтобы в текущем столбце все элементы, кроме ведущего, были нулями.

Проверка корректности реализована в методе «check\_solution», который для каждой строки матрицы пересчитывает левую часть, перемножая элементы строки на соответствующие значения вектора решений и складывает результат. Полученная сумма сравнивает со значением из правой части, но не напрямую, а с учетом погрешности eps. Если все уравнения выполняются с заданной точностью, функция выводит сообщение, что решение правильно.

# Руководство пользователя

Пользователю при запуске программы предлагается выбрать тип данных: int, float, double или char:



Рисунок . Ввод типа данных

Затем нужно ввести количество уравнений, то есть, строк матрицы:



Рисунок . Ввод количества строк матрицы

Далее ввести количество переменных, то есть, столбцов матрицы:

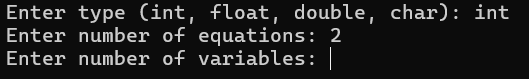


Рисунок . Ввод количества переменных

Ввести элементы матрицы и правого вектора, после чего на экран выводится ответ:

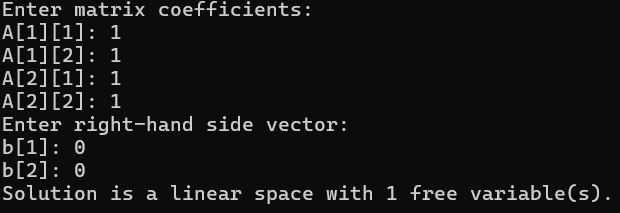


Рисунок . Ввод матрицы

В конце также проводится проверка на правильность решения СЛАУ, время выполнения и количество операций:

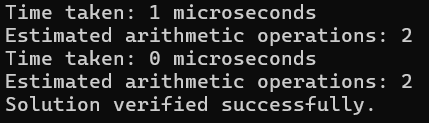


Рисунок . Вывод проверки, времени выполнения и количества операций

# Описание программной реализации

Проект состоит из следующих файлов:

Заголовочные файлы (\*.h):

1. Exception.h — определение собственного класса исключений.
2. Vector.h — шаблонный класс динамического вектора.
3. Matrix.h — шаблонный класс матрицы, унаследованный от вектора векторов.
4. SLAU.h — реализация метода Гаусса-Жордана и проверки решения.
5. Исходный файл (main.cpp):

Точка входа в программу: пользователь выбирает тип данных, вводит систему уравнений, запускает алгоритм, и получает результат с проверкой, временем выполнения и количеством операций.

Описание файлов:

1. Exception.h

Класс Exception - собственный класс исключений с полем message.

Метод what() возвращает текст ошибки.

1. Vector.h

Класс Vector<T>:

Универсальный вектор с поддержкой шаблонов типов.

Методы:

1. Доступ по индексу с проверкой границ.
2. Конструкторы копирования, перемещения, resize, печать.
3. Matrix.h

Класс Matrix<T> наследуется от Vector<Vector<T>>.

Методы:

1. get\_num\_of\_lines() — возвращает количество строк.
2. print\_matrix() — выводит матрицу.
3. swap() — обмен строк (для Гаусса).
4. SLAU.h

Класс SLAU<T> наследуется от Matrix<T>.

Методы:

1. gauss\_jordan:

* Реализует метод Гаусса-Жордана с выбором ведущего элемента.
* Преобразует матрицу к редуцированному ступенчатому виду.
* Определяет совместность системы.
* Вычисляет частное или общее решение.
* Подсчитывает количество арифметических операций.
* Измеряет время выполнения (в микросекундах).

1. check\_solution:

* Проверяет корректность полученного решения: перемножает матрицу на вектор и сравнивает с правой частью.
* Учитывает заданную погрешность eps.

1. main.cpp

Функции:

1. Выбор типа данных (int, float, double, char).
2. Ввод матрицы и вектора пользователем.
3. Вызов метода gauss\_jordan, вывод решения.
4. Проверка решения с помощью check\_solution.
5. Вывод статистики: время выполнения и количество операций.

# Подтверждение корректности

Программа предусматривает несколько механизмов, позволяющих убедиться в правильности найденного решения системы линейных уравнений методом Гаусса–Жордана. Ключевую роль в этом играет функция check\_solution, которая вызывается после завершения основного алгоритма. Она заново пересчитывает левую часть каждого уравнения, умножая исходную матрицу коэффициентов на полученный вектор решения. Затем результат сравнивается с вектором свободных членов. Если хотя бы одно уравнение не выполняется с учётом заданной погрешности epsilon, пользователю сообщается об этом.

Этот подход позволяет автоматически проверить корректность вычислений без необходимости ручной проверки. Он особенно полезен при работе с вещественными числами, где возможны погрешности округления, или при решении плохо обусловленных систем.

Для надёжной обработки ошибок в программе реализован собственный класс исключений Exception. Он используется для генерации ошибок в ситуациях, когда обнаруживаются логические противоречия в системе, нарушается размерность входных данных или результат не соответствует ожиданиям. Все исключения обрабатываются на уровне main, где выводятся понятные сообщения, объясняющие пользователю причину сбоя или отклонения.

В совокупности эти проверки делают программу устойчивой к ошибкам ввода и повышают доверие к полученному результату. Они также позволяют упростить отладку и повысить точность при использовании метода в практических задачах.

# Результаты экспериментов

Алгоритмы, основанные на методах Гаусса и Гаусса–Жордана, требуют значительных вычислительных ресурсов при увеличении размера системы. С ростом числа уравнений и переменных время выполнения возрастает довольно быстро. С точки зрения теории, оба метода относятся к алгоритмам с кубической сложностью — , что означает трёхкратную зависимость от размера входных данных, то есть, при увеличении количества переменных вычислительная нагрузка возрастает в разы, что делает такие методы менее эффективными для решения очень больших систем.

Эксперимент заключался в том, что я запустила программу для матриц размером , , по 10 раз со случайными элементами матрицы. Затем нашла среднее арифметическое времени, количества операций для каждого размера матрицы, а результаты записала в таблицу. После этого построила график зависимости этих средних арифметических и размеров матриц. Таким образом, можно заметить, что при увеличении размера матрицы в 10 раз количество операций изменилось в среднем в 1000 раз, а время – примерно в 900 раз. Так, измерения демонстрируют кубическую зависимость, что соответствует теоретической сложности.

|  |  |
| --- | --- |
| Размер матрицы | Среднее количество операций |
| Строки – 10  Столбцы - 10 | 1359 |
| Строки – 100  Столбцы - 100 | 1039485 |
| Строки – 1000  Столбцы - 1000 | 1003992193 |

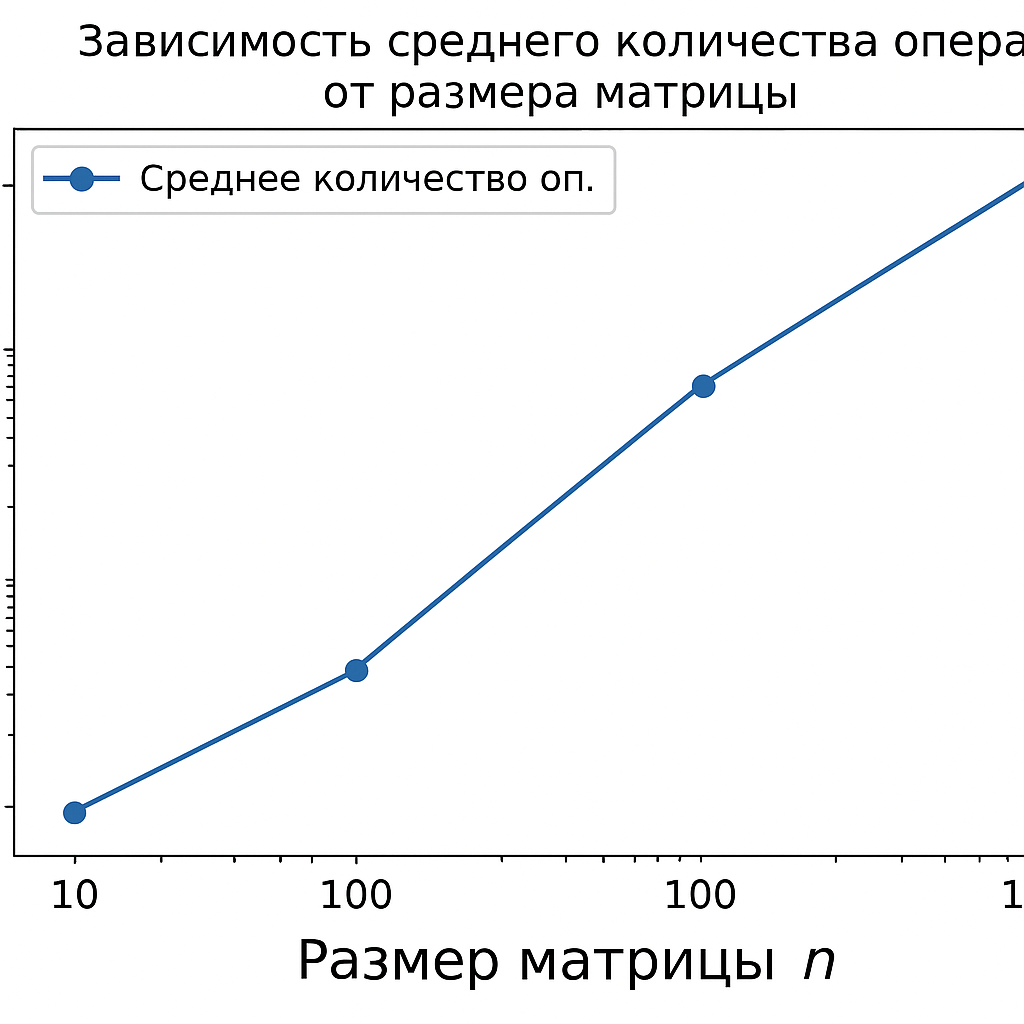


Рисунок . График зависимости среднего количества операций от размера матрицы

|  |  |
| --- | --- |
| Размер матрицы | Среднее время выполнения |
| Строки – 10  Столбцы - 10 | 598 |
| Строки – 100  Столбцы - 100 | 11482 |
| Строки – 1000  Столбцы - 1000 | 11713468 |

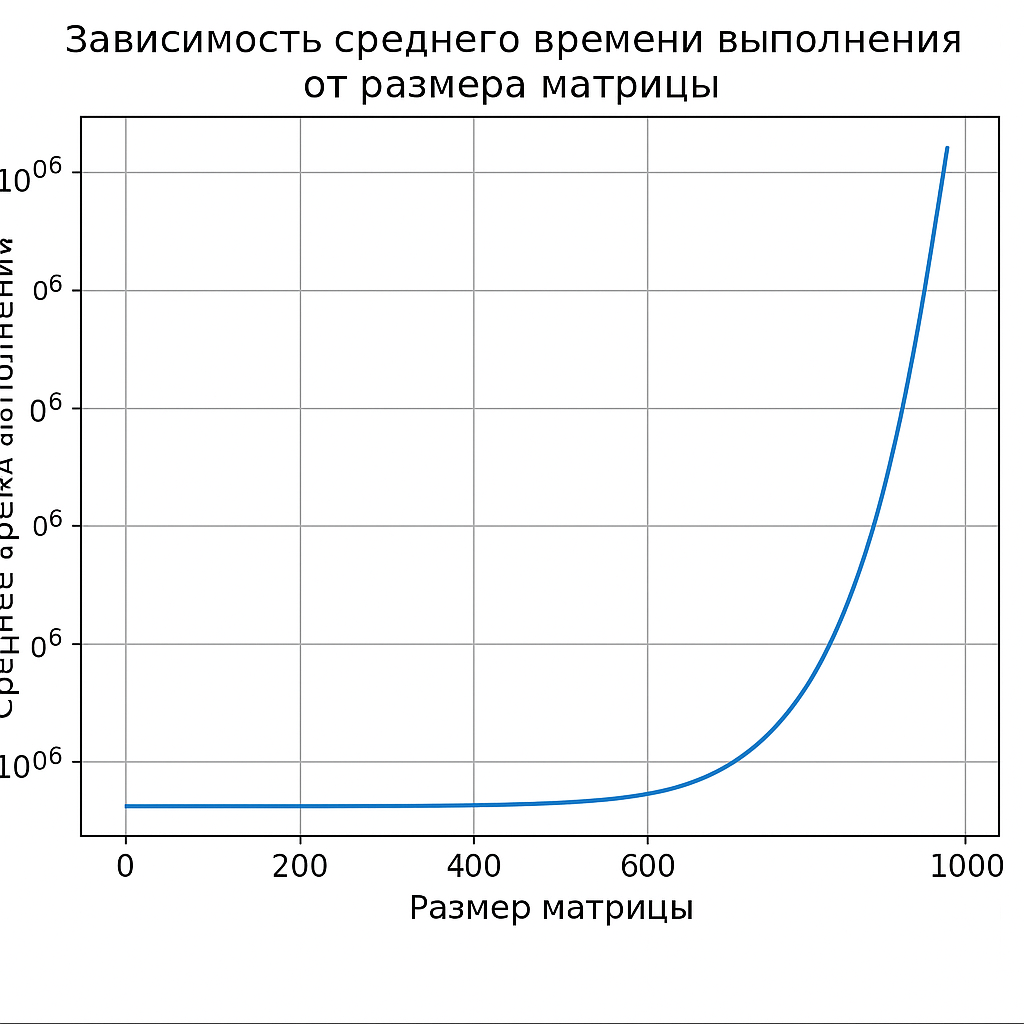


Рисунок .График зависимости среднего времени выполнения от размера матрицы

# Заключение

При выполнении данной лабораторной работы я реализовала метод Гаусса-Жордана для решения систем линейных алгебраических уравнений, создала метод проверки правильности решения, шаблонные классы, матрицу – как вектор векторов, а для повышения надежности работы программы создала собственное исключения. Программа корректно приводит матрицу к ступенчатому виду, выбирая больший по модулю вектор для минимизации ошибок округления и исключения деления на ноль, результат выводит на экран.

Результаты экспериментов совпали с теоретической сложностью, что подтверждает правильность написанного кода.

# Приложение

#pragma once

#include <string>

class MyException {

std::string message;

public:

MyException(const std::string& msg) : message(msg) {}

const std::string& what() const { return message; }

};

#include "Matrix.h"

#include "oper.h"

#include <chrono>

#include <iostream>

#define eps 1e-10

template <typename T>

class SLAU : public Matrix<T> {

public:

using Matrix<T>::Matrix;

Vector<T> gauss\_jordan(const Vector<T>& right\_side) {

int lines = this->get\_num\_of\_lines();

int columns = (\*this)[0].get\_vector\_size();

if (right\_side.get\_vector\_size() != lines)

throw std::invalid\_argument("Right side size does not match number of lines");

Vector<Vector<T>> augmented\_matrix(lines);

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

augmented\_matrix[i] = Vector<T>(columns + 1);

for (int j = 0; j < columns; j++) {

augmented\_matrix[i][j] = (\*this)[i][j];

}

augmented\_matrix[i][columns] = right\_side[i];

}

auto start\_time = std::chrono::high\_resolution\_clock::now();

for (int col = 0, row = 0; col < columns && row < lines; col++) {

int max\_row = row;

T max\_val = std::abs(augmented\_matrix[row][col]);

for (int i = row + 1; i < lines; i++) {

operation\_count++;

if (std::abs(augmented\_matrix[i][col]) > max\_val) {

max\_val = std::abs(augmented\_matrix[i][col]);

max\_row = i;

}

}

if (max\_val < eps) continue;

if (max\_row != row) {

std::swap(augmented\_matrix[row], augmented\_matrix[max\_row]);

operation\_count += columns + 1;

}

T pivot = augmented\_matrix[row][col];

for (int j = col; j <= columns; j++) {

augmented\_matrix[row][j] /= pivot;

operation\_count++;

}

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

if (i != row && std::abs(augmented\_matrix[i][col]) > eps) {

T factor = augmented\_matrix[i][col];

for (int j = col; j <= columns; j++) {

augmented\_matrix[i][j] -= factor \* augmented\_matrix[row][j];

operation\_count += 2;

}

}

}

++row;

}

auto end\_time = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end\_time - start\_time);

std::cout << "Time taken: " << duration.count() << " microseconds\n";

std::cout << "Estimated arithmetic operations: " << operations\_count << "\n";

Vector<T> solution(lines);

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

solution[i] = augmented\_matrix[i][columns];

}

return solution;

}

void print\_solution(const Vector<T>& solution) const {

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

std::cout << 'x' << (i + 1) << " = " << solution[i] << ";\n";

}

}

bool check\_solution(const Vector<T>& solution, const Vector<T>& right\_side) const {

int n = this->get\_num\_of\_lines();

int m = solution.get\_vector\_size();

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

T sum = 0;

for (int j = 0; j < m; ++j) {

sum += (\*this)[i][j] \* solution[j];

}

if (std::abs(sum - right\_side[i]) > eps) {

std::cout << "Check failed at row " << i + 1 << ": got " << sum << ", expected " << right\_side[i] << "\n";

return false;

}

}

std::cout << "Solution verified successfully.\n";

return true;

}

};

#include <iostream>

#include <initializer\_list>

#include <stdexcept>

#include <locale.h>

#include "Exception.h"

using namespace std;

template <typename T>

class Vector {

private:

T\* cords;

int size;

public:

Vector() : cords(nullptr), size(0) {}

explicit Vector(int \_size) : size(\_size) {

if (size > 0) {

cords = new T[size]();

}

else {

cords = nullptr;

}

}

Vector(initializer\_list<T> list) : size(static\_cast<int>(list.size())) {

if (size > 0) {

cords = new T[size]();

}

else {

cords = nullptr;

}

int i = 0;

for (const T& val : list) { cords[i++] = val; }

}

Vector(int \_size, const T& value) : size(\_size) {

if (size > 0) {

cords = new T[size]();

}

else {

cords = nullptr;

}

for (int i = 0; i < size; ++i) { cords[i] = value; }

}

Vector(const Vector& other) : size(other.size) {

if (size > 0) {

cords = new T[size]();

}

else {

cords = nullptr;

}

for (int i = 0; i < size; ++i) { cords[i] = other.cords[i]; }

}

Vector& operator=(const Vector& other) {

if (this != &other) {

delete[] cords;

size = other.size;

cords = (size > 0) ? new T[size] : nullptr;

for (int i = 0; i < size; ++i) cords[i] = other.cords[i];

}

return \*this;

}

Vector(Vector&& other) noexcept : cords(other.cords), size(other.size) {

other.cords = nullptr;

other.size = 0;

}

Vector& operator=(Vector&& other) noexcept {

if (this != &other) {

delete[] cords;

cords = other.cords;

size = other.size;

other.cords = nullptr;

other.size = 0;

}

return \*this;

}

~Vector() { delete[] cords; }

T& operator[](int index) {

if (index < 0 || index >= size)

throw MyException("Vector index out of range");

return cords[index];

}

const T& operator[](int index) const {

if (index < 0 || index >= size)

throw MyException("Vector index out of range");

return cords[index];

}

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

void print\_vector() const {

cout << "[";

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

cout << cords[i] << (i != size - 1 ? ", " : "");

}

cout << "]";

}

};

#include "Vector.h"

#include "Exception.h"

template <typename T>

class Matrix : public Vector<Vector<T>>{

private:

Vector<Vector<T>> matrix;

int size;

public:

Matrix() : size(0) {}

Matrix(std::initializer\_list<Vector<T>> vecs) : size(vecs.size()) {

matrix = Vector<Vector<T>>(size);

int i = 0;

for (const auto& vec : vecs) {

matrix[i++] = vec;

}

}

Vector<T>& operator[](int index) {

if (index < 0 || index >= size)

throw MyException("Matrix index out of range");

return matrix[index];

}

const Vector<T>& operator[](int index) const {

if (index < 0 || index >= size)

throw MyException("Matrix index out of range");

return matrix[index];

}

int get\_num\_of\_lines() const { return size; }

void resize(int \_size) {

size = \_size;

matrix = Vector<Vector<T>>(size);

}

void print\_matrix() const {

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

matrix[i].print\_vector();

cout << endl;

}

}

};

#include <iostream>

#include "SLAU.h"

#include "Exception.h"

using namespace std;

template <typename T>

void run\_solver() {

int n, m;

cout << "Enter number of equations: ";

cin >> n;

cout << "Enter number of variables: ";

cin >> m;

SLAU<T> matrix;

matrix.resize(n);

cout << "Enter matrix coefficients:\n";

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

matrix[i] = Vector<T>(m);

for (int j = 0; j < m; j++) {

cout << "A[" << i + 1 << "][" << j + 1 << "]: ";

cin >> matrix[i][j];

}

}

Vector<T> right(n);

cout << "Enter right-hand side vector:\n";

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

cout << "b[" << i + 1 << "]: ";

cin >> right[i];

}

try {

matrix.gauss\_jordan(right);

auto solution = matrix.gauss\_jordan(right);

matrix.check\_solution(solution, right);

}

catch (MyException& e) {

cout << "Error: " << e.what() << endl;

}

}

int main() {

string type;

cout << "Enter type (int, float, double, char): ";

cin >> type;

if (type == "int") run\_solver<int>();

else if (type == "float") run\_solver<float>();

else if (type == "double") run\_solver<double>();

else if (type == "char") run\_solver<char>();

else cout << "Unsupported type\n";

return 0;

}