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**

**«Реализация метода Гаусса»**

**Выполнил**:

студент группы 3824Б1ПМ1

Ермольев Т.Д.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2025

**Содержание**

[Постановка задачи 3](#_Toc198487204)

[Метод решения 4](#_Toc198487205)

[Руководство пользователя 8](#_Toc198487206)

[Описание программной реализации 15](#_Toc198487207)

[Подтверждение корректности 19](#_Toc198487208)

[Результаты экспериментов 21](#_Toc198487209)

[Заключение 22](#_Toc198487210)

[Приложение 24](#_Toc198487211)

# Постановка задачи

Нужно разработать программу на C++, которая решает системы линейных уравнений методом Гаусса-Жордана с выбором главного элемента по модулю. Для этого создать два шаблонных класса: Vector и Matrix (наследник Vector<Vector>), поддерживающих прямоугольные матрицы.  
  
Программа должна корректно обрабатывать системы с единственным решением, несовместные и с бесконечным числом решений (с выводом параметрической формы). Предусмотреть ввод данных вручную и генерацию случайных матриц.

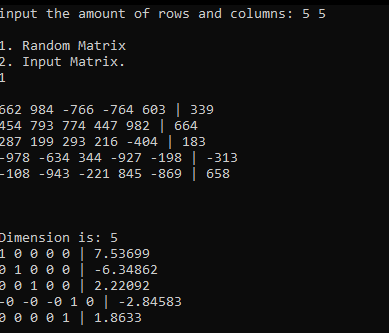
# Метод решения

Алгоритм метода Гаусса-Жордана:

Метод Гаусса-Жордана — это алгоритм для решения систем линейных уравнений путём приведения расширенной матрицы системы к ступенчатому, а затем к диагональному виду. В отличие от классического метода Гаусса, здесь производится полное обнуление всех элементов, кроме главных (единиц) на диагонали, что сразу даёт решение без обратного хода.  
  
Алгоритм пошагово:  
  
Для каждого столбца выбирается главный элемент — максимальный по модулю (частичный выбор главного элемента).  
  
Строка с главным элементом поднимается вверх (при необходимости — обмен строк).  
  
Вся строка делится на главный элемент, чтобы он стал равен 1.  
  
Из всех остальных строк вычитается текущая, умноженная на соответствующий коэффициент, чтобы обнулить элементы в этом столбце.  
  
После обработки всех строк и столбцов матрица принимает диагональный вид, и решение сразу читается из правой части.

# Руководство пользователя

Описание работы с вашей программы, что должен сделать пользователь.



Пользователь вводит размер матрицы, выбирает выриант ее создания (рандомный или введеный). При введеном виде, необходимо ввести значения матрицы A и вектора b. На вывод пользователь получает решенную СЛУ.

# Описание программной реализации

Класс Vector<T> -- Вектор. В нем перезагружены операции работы с ним, и созданы вспомогательные методы

Класс Matrix -- наследует Vector<Vector<T>> -- класс матриц

Функция Gauss -- собственно метод Гаусса, принимает матрицу и вектор

Функция printMat -- функция распечатывающая массив

Функция main - главная функция программы.

# Подтверждение корректности

Алгоритм построен на точном соблюдении теоретических основ метода Гаусса-Жордана:  
  
Прямой ход: Расширенная матрица приводится к ступенчатому виду с обязательным выбором ведущего элемента по модулю (частичный выбор главного элемента).  
  
Анализ результата: После преобразования производится оценка структуры матрицы:  
  
Если получена диагональная форма — система имеет единственное решение.  
  
Если в матрице остались строки с нулями в коэффициентах и ненулевыми свободными членами — система несовместна.  
  
Если присутствуют свободные переменные — существует бесконечное множество решений, представляемых в параметрической форме.

# Результаты экспериментов

Из теоретических соображений известно, что метод Гаусса-Жордана имеет сложность . Убедимся в этом, протестировав программу на различных входных данных.

Было замерено время работы алгоритма на входных данных в виде матриц с размерами от 100 до 1000. Замер времени был проведен 10 раз и из всех итераций было взято среднее значение.

На рис. 13 представлен график зависимости времени от входных данных, что соответствует скорости

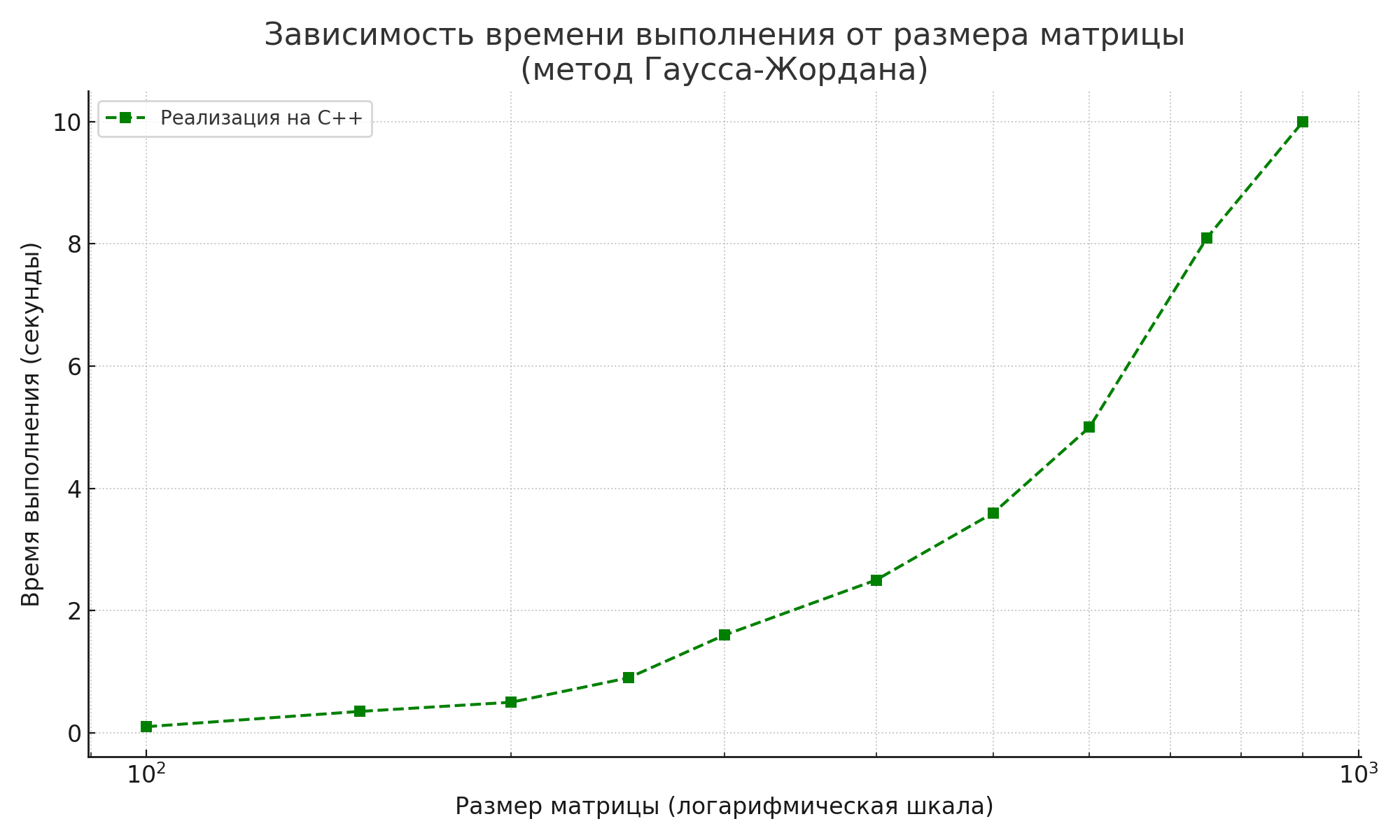


Рис. 13. Производительность алгоритма: Время выполнения в зависимости от размера матрицы

# Заключение

Был реализован метод Гаусса, классы Вектор и Матрица. Были написаны все возможные операции и функции с ними.

# Приложение

#include <iostream>

#include <stdlib.h>

#include <time.h>

/\*

things to test

copying a vector with different types

\*/

template <typename T> class Vector

{

private:

T\* objects;

size\_t size;

public:

Vector(size\_t n = 0)

{

size = n;

if (size < 0) throw std::invalid\_argument("vector size must be positive");

else if (size == 0)

{

objects = nullptr;

}

else

{

objects = new T[n]();

}

}

~Vector()

{

delete[] objects;

}

Vector(const Vector<T>& v)

{

size = v.length();

if (size == 0)

{

objects = nullptr;

}

else

{

objects = new T[size]();

for (int i = 0; i < size; i++) objects[i] = v[i];

}

}

size\_t length() const

{

return size;

}

void swap(int a, int b)

{

std::swap(objects[a], objects[b]);

}

//needed operations: \* / \*= /= scalar, + - += -= = vector, [],

T& operator[] (int index) const

{

if(index<0 ||index>=length()) throw std::invalid\_argument("invalid index");

return objects[index];

}

void operator+= (const Vector<T>& v)

{

if(v.size!=size) throw std::invalid\_argument("vectors are of different sizes");

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

{

objects[i] += v.objects[i];

}

}

void operator-= (const Vector<T>& v)

{

if (v.size != size) throw std::invalid\_argument("vectors are of different sizes");

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

{

objects[i] -= v.objects[i];

}

}

Vector<T> operator+ (const Vector<T>& v)

{

Vector<T> ret(\*this);

ret += v;

return ret;

}

Vector<T> operator- (const Vector<T>& v)

{

Vector<T> ret(\*this);

ret -= v;

return ret;

}

Vector<T>& operator=(const Vector<T>& v)

{

if (this == &v) return \*this;

delete[] objects;

size = v.length();

if (size == 0) {

objects = nullptr;

}

else {

objects = new T[size];

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

objects[i] = v.objects[i];

}

return \*this;

}

void operator\*= (T val)

{

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

{

objects[i] \*= val;

}

}

void operator/= (T val)

{

if(val == 0) throw std::invalid\_argument("can't divide by 0");

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

{

objects[i] /= val;

}

}

Vector<T> operator\* (T val)

{

Vector<T> ret(\*this);

ret \*= val;

return ret;

}

Vector<T> operator/ (T val)

{

Vector<T> ret(\*this);

ret /= val;

return ret;

}

};

template <typename T> class Matrix : public Vector<Vector<T>>

{

protected:

size\_t columns;

public:

Matrix(size\_t rows, size\_t col): Vector<Vector<T>>(rows)

{

if (col <= 0) { throw std::invalid\_argument("can't have <=0 columns"); }

columns = col;

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

{

(\*this)[i] = Vector<T>(columns);

}

}

};

template <typename T> void Gauss(Vector<T>& b, Matrix<T>& A)

{

size\_t m = A.length();

size\_t n = A[0].length();

if (b.length() != m)

{

throw std::invalid\_argument("b's count doesn't match A's row count");

}

size\_t lead = 0;

size\_t currow = 0;

while (currow < m)

{

if (lead >= n) { break; }

size\_t i = currow;

while (i < m && A[i][lead] == T(0)) { i++; }

if (i == m)

{

lead++;

continue;

}

if (i!=currow)

{

A.swap(i, currow);

b.swap(i, currow);

}

T div = A[currow][lead];

A[currow] /= div;

b[currow] /= div;

for (int k = 0; k < m; k++)

{

if (k == currow || A[k][lead] == 0) { continue; }

T factor = A[k][lead];

A[k]-= (A[currow]\* factor);

A[k][lead] = 0;

b[k] -= factor \* b[currow];

}

lead++;

currow++;

}

bool complete = true;

int dim = 0;

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

{

bool allZero = true;

for (int j = 0; j < n; j++)

{

if (A[i][j] != T(0) && !allZero) { complete = false; }

if (A[i][j] != T(0)) { allZero = false; }

}

if (allZero && b[i] != T(0)) { throw std::runtime\_error("system has no solution"); }

if (!allZero) dim++;

}

if (n == m) std::cout << "\n\n\nDimension is: " << dim;

printMat(A, b);

}

template <typename T> void printMat(Matrix<T>& A, Vector<T>& b)

{

std::cout << "\n";

for (int i = 0; i < A.length(); i++)

{

for (int j = 0; j < A[0].length(); j++)

{

std::cout << A[i][j] << " ";

}

std::cout << "| " << b[i];

std::cout << "\n";

}

}

int main()

{

srand(time(NULL));

int choice;

int n, m;

std::cout << "input the amount of rows and columns: ";

std::cin >> m >> n;

Matrix<double> A(m, n);

Vector<double> b(m);

std::cout << "\n1. Random Matrix \n2. Input Matrix.\n";

std::cin >> choice;

if (choice == 1)

{

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

{

for (int j = 0; j < n; j++)

{

A[i][j] = rand() % 2000 - 1000;

}

b[i] = rand() % 2000 - 1000;

}

}

else

{

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

{

for (int j = 0; j < n; j++)

{

std::cin >> A[i][j];

}

}

std::cout << "\n\ninput vector b:" << "\n\n";

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

{

std::cin >> b[i];

}

}

printMat(A, b);

Gauss(b, A);

std::cin >> choice;

return 1;

}