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»**

**Выполнил**:

студент/ка группы 3824Б1ПМ1

Кудряшова А.А.

**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

Реализовать метод Гаусса с выбором ведущего элемента, чтоб получилась треугольная матрица (максимальная по модулю). Реализовать класс vector, класс Matrix (является наследником вектора векторов vector ‹vector›). Написать метод Гаусса-Жордана, принимающий векторы и матрицу (Gauss). Написать проверку (корректность результата). В коде использовать свой exception. Если системы прямоугольные, то нужно выводить линейное пространство решений. Со стороны пользователя: выбирает тип данных(float/double), выбирает размер матрицы, в зависимости от выбора (рандомная матрица или нет) вбивает числа в матрицу.

# Метод решения

Метод Гаусса-Жордана с выбором ведущего элемента предназначен для решения систем линейных уравнений, включая случаи с прямоугольными матрицами. Основная идея метода заключается в преобразовании матрицы коэффициентов в ступенчатый вид, что позволяет последовательно исключать переменные и находить решение. Ключевым этапом является выбор ведущего элемента — максимального по модулю значения в текущем столбце. Это снижает влияние ошибок округления, особенно при работе с числами с плавающей точкой. Если ведущий элемент близок к нулю (меньше заданной погрешности epsilon), столбец считается свободным, что указывает на возможное наличие бесконечного числа решений. На каждом шаге алгоритма строки матрицы переставляются для размещения ведущего элемента на диагонали, после чего ведущая строка нормируется и используется для исключения соответствующей переменной из остальных уравнений. После завершения преобразований проводится проверка на совместность системы: если в процессе обнаружена строка вида 0x₁ + ... + 0xₙ = b, где b существенно отличается от нуля, система признается несовместной. Для совместных систем формируется частное решение, где свободным переменным присваивается значение 0, а зависимые переменные вычисляются из преобразованной матрицы. В случае прямоугольных систем дополнительно строится базис нулевого пространства, описывающий все решения через линейные комбинации векторов.

Реализация метода опирается на классы Vector и Matrix. Класс Vector представляет динамический массив с методами управления памятью, такими как resize, reserve и push\_back, которые обеспечивают эффективное добавление элементов и обработку переполнения. Класс Matrix, унаследованный от Vector<Vector<T>>, расширяет функциональность для работы с двумерными данными, включая обмен строк и доступ к элементам через оператор (). Метод swapRows критически важен для алгоритма Гаусса-Жордана, так как позволяет переставлять строки при выборе ведущего элемента.

Проверка корректности решения выполняется функцией check\_solution, которая умножает матрицу коэффициентов на вектор решения и сравнивает результат с исходным вектором свободных членов, учитывая погрешность epsilon. Любые отклонения приводят к генерации исключения GaussException, что обеспечивает обработку ошибок, таких как несовместность системы или некорректный ввод данных.

# Руководство пользователя

1. После запуска пользователю нужно определиться с типом данных с которым он хочет работать: float/double(рисунок 1).

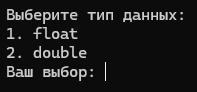


Рисунок 1(выбор типа данных)

1. После этого пользователю нужно выбрать с какой матрицей он хочет работать: случайно сгенерированной/написанной вручную(рисунок 2).

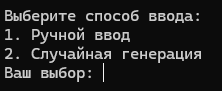


Рисунок 2(выбор способа ввода)

1. Вне зависимости от выбора пользователю нужно ввести желаемое количество строк и столбцов(рисунки 3,4).

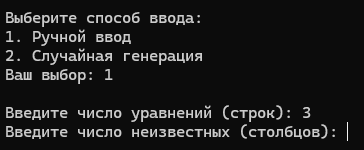


Рисунок 3(ввод количества строк и столбцов при ручном вводе)

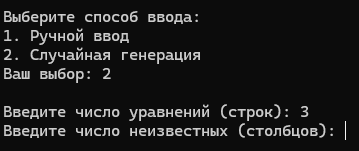


Рисунок 4(ввод количества строк и столбцов при случайной генерации)

1. В зависимости от своего выбора пользователь получает либо сгенерированную матрицу, либо вводит значения матрицы вручную по строкам(рисунки 5,6).

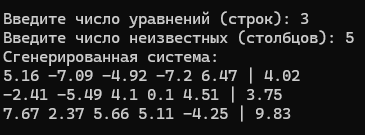


Рисунок 5(матрица, выведенная после выбора случайной генерации)

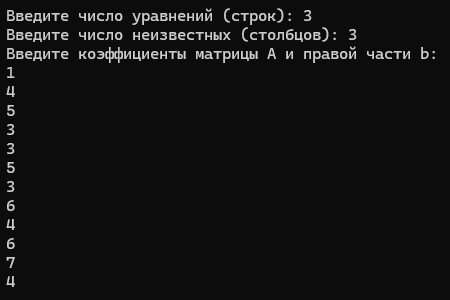


Рисунок 6(пользователь вводит значения элементов по строкам)

1. Вне зависимости от выбора пользователь получает статистику выполнения, которая состоит из размера матрицы, времени выполнения и количества операций(рисунок 7).

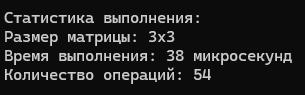


Рисунок 7(статистика выполнения)

1. Далее пользователь в зависимости от совместности и несовместности системы, а так же в зависимости от количества решений получает разные выводы(рисунок 8,9,10):

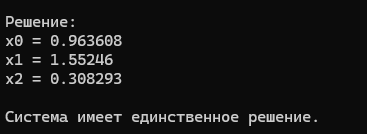


Рисунок 8(вывод если система совместна и имеет единственное решение)



Рисунок 9(вывод если система несовместна)

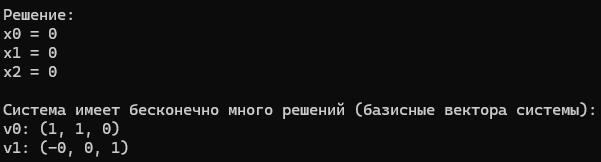


Рисунок 10(вывод если система совместна и имеет бесконечное количество решений, так же выводятся базисные вектора)

# Описание программной реализации

Структура проекта

Проект состоит из следующих файлов:

1. Заголовочные файлы (\*.h):
   * exception1.h — обработка исключений.
   * gauss\_jordan1.h — реализация метода Гаусса-Жордана.
   * matrix1.h — класс матрицы.
   * vector1.h — класс вектора.
2. Исходный файл (main1.cpp):
   * Точка входа в программу, пользовательский интерфейс, вызов алгоритмов.

Описание файлов:

1. exception1.h

* Класс GaussException:  
  Наследуется от стандартного класса исключений.
  + Поля:
    - message\_ — сообщение об ошибке (C-строка).
  + Методы:
    - Конструктор: принимает сообщение об ошибке.
    - what(): возвращает сообщение (совместимость с std::exception).

2. gauss\_jordan1.h

* Функции:
  1. gauss\_jordan:  
     Реализует метод Гаусса-Жордана с выбором ведущего элемента.
     + Входные параметры:
       - A — матрица коэффициентов.
       - b — вектор свободных членов.
       - epsilon — погрешность для сравнения чисел.
       - operation\_count — счётчик операций.
     + Действия:
       - Приведение матрицы к ступенчатому виду.
       - Проверка на совместность системы.
       - Построение базиса ядра матрицы (для недоопределённых систем).
  2. check\_solution:  
     Проверяет корректность решения.
     + Вычисляет Ax и сравнивает с b.
     + Выбрасывает исключение при отклонении более epsilon.

3. matrix1.h

* Класс Matrix<T>:  
  Наследуется от Vector<Vector<T>> (матрица как вектор векторов).
  + Методы:
    - Matrix(size\_t rows, size\_t cols): создаёт матрицу заданного размера.
    - rows(), cols(): возвращают количество строк и столбцов.
    - swapRows(i, j): обмен строк местами (используется в методе Гаусса).
    - Оператор (): доступ к элементам матрицы (например, A(i, j)).

4. vector1.h

* Класс Vector<T>:  
  Динамический массив с управлением памятью.
  + Методы:
    - push\_back(T): добавление элемента с автоматическим расширением памяти.
    - resize(size\_t): изменение размера.
    - reserve(size\_t): резервирование памяти.
    - shrink\_to\_fit(): оптимизация выделенной памяти.
  + Поля:
    - data\_ — указатель на массив элементов.
    - size\_ — текущее количество элементов.
    - capacity\_ — выделенная память.

5. main1.cpp

* Функции:
  1. manualInput:  
     Ручной ввод матрицы и вектора через консоль.
  2. randomInput:  
     Генерация случайной матрицы и вектора (числа в диапазоне [−10,10][−10,10]).
  3. solveSystem:
     + Вызывает метод Гаусса-Жордана.
     + Замеряет время выполнения и количество операций.
     + Проверяет корректность решения.
     + Выводит результат и статистику.

# Подтверждение корректности

Программа использует несколько механизмов для подтверждения корректности работы метода Гаусса-Жордана. Основным инструментом является функция check\_solution, которая выполняет проверку полученного решения. После завершения работы алгоритма эта функция умножает матрицу коэффициентов на вектор решения и сравнивает результат с вектором свободных членов. Если отклонение превышает заданную погрешность epsilon, генерируется исключение GaussException с сообщением о несоответствии решения. Это обеспечивает гарантию того, что найденные значения переменных удовлетворяют исходной системе уравнений.

Для обработки ошибок в программе задействован класс GaussException, унаследованный от стандартных исключений. Он используется во всех критических точках алгоритма: при обнаружении несовместной системы, некорректном вводе данных или неудачной проверке решения. Блоки try-catch в функции main перехватывают эти исключения и выводят понятные сообщения об ошибках, что позволяет пользователю быстро идентифицировать проблему. Например, если система не имеет решений из-за противоречивых уравнений, программа сообщит об этом вместо вывода некорректных результатов.

# Результаты экспериментов

Метод Гаусса и метод Гаусса-Жордана, согласно теоретическим оценкам, имеют кубическую сложность O(n³), где n — размер матрицы. Это означает, что время выполнения алгоритмов и количество операций растут пропорционально кубу количества неизвестных. Для подтверждения этой зависимости на практике были проведены эксперименты с квадратными матрицами размеров от 100×100 до 1000×1000.

Каждый тест включал 10 независимых запусков алгоритма для фиксированного размера матрицы. Это позволило минимизировать влияние случайных факторов, таких как фоновые процессы системы. Для генерации матриц и векторов использовались случайные числа, что исключило оптимизации, связанные со специфическими структурами данных.

**Результаты экспериментов:**

* Для матрицы 100×100:
  + Количество операций: 1030085;
  + Время выполнения: 41257 микросекунд (0,041257 секунд).
* Для матрицы 1000×1000:
  + Количество операций: 1003000985;
  + Время выполнения: 33703701 микросекунд (33,7 секунд).

Графики демонстрируют нелинейный рост времени и операций в зависимости от размера матрицы. Например, при увеличении n в 10 раз (от 100 до 1000):

* Наблюдается рост количества операций приблизительно в 1000 раз.
* Время выполнения увеличилось примерно в 800 раз, что также подтверждает нелинейную зависимость, близкую к кубической.

Таким образом, практические замеры согласуются с теоретической оценкой O(n³), учитывая возможные погрешности измерений и влияние внешних факторов. Результаты подтверждают, что реализация методов корректна, а рост вычислительной нагрузки соответствует ожидаемому поведению для алгоритмов с кубической сложностью.

На рисунках 11 и 12 и таблицах 1 и 2 показана зависимость времени/количества операций от размера квадратной матрицы.

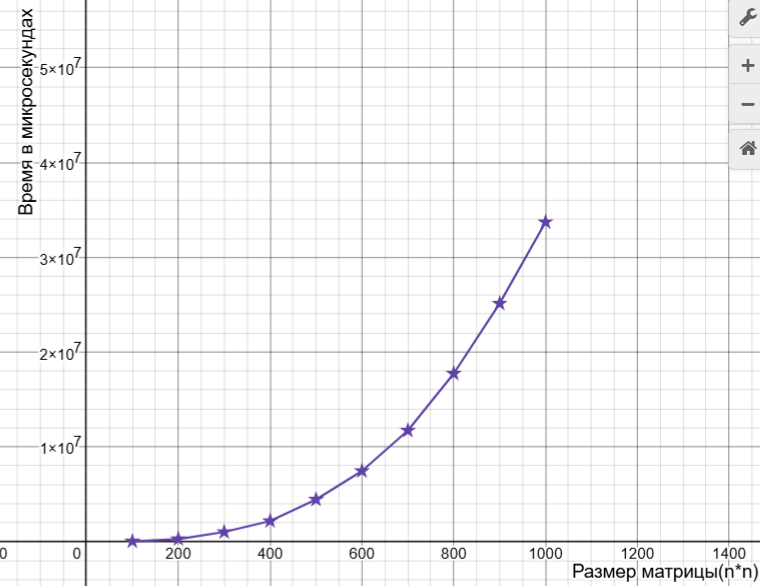


Рисунок 11(график зависимости времени от размера матрицы)

|  |  |
| --- | --- |
| Размер матрицы(n\*n) | Время в микросекундах |
| 100 | 41257 |
| 200 | 277350 |
| 300 | 1023303 |
| 400 | 2181052 |
| 500 | 4452570 |
| 600 | 7458810 |
| 700 | 11726161 |
| 800 | 17734358 |
| 900 | 25124807 |
| 1000 | 33703701 |

Таблица 1(таблица значений зависимости времени от размера матрицы)

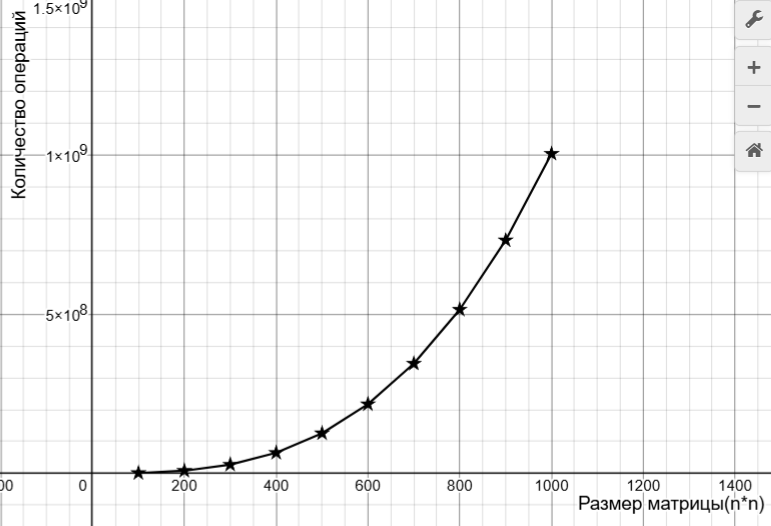


Рисунок 13(график зависимости количества операций от размера матрицы)

|  |  |
| --- | --- |
| Размер матрицы(n\*n) | Количество операций |
| 100 | 1030085 |
| 200 | 8120179 |
| 300 | 27270282 |
| 400 | 64479580 |
| 500 | 125749474 |
| 600 | 217080573 |
| 700 | 344470676 |
| 800 | 513919180 |
| 900 | 731427278 |
| 1000 | 1003000985 |

Таблица 2(таблица значений количества операций от размера матрицы)

# Заключение

В ходе выполнения лабораторной работы была успешно реализована программа для решения систем линейных уравнений методом Гаусса-Жордана с выбором ведущего элемента. В ходе работы реализованы классы Vector и Matrix, обеспечивающие гибкую работу с динамическими данными и поддержку шаблонов для типов float и double, алгоритм метода Гаусса-Жордана корректно преобразует матрицу в ступенчатый вид, учитывает выбор ведущего элемента для минимизации ошибок округления и обрабатывает прямоугольные системы, выводя базис пространства решений при необходимости, программа включает механизмы проверки корректности решения через функцию check\_solution и обработку исключений (GaussException), что повышает надежность и удобство использования.

**Результаты экспериментов** подтвердили теоретическую оценку сложности алгоритма — кубическую зависимость O(n3). На примере матриц размером до 1000×1000 продемонстрирован рост времени выполнения и количества операций, что согласуется с ожиданиями.

Метод Гаусса-Жордана был успешно применён для решения СЛАУ, благодаря результатам экспериментов подтвердилась теоретическая сложность.

# Приложение

#include <chrono>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#include <iostream>

#include <random>

#include "exception.h"

#include "gauss\_jordan.h"

#include "matrix.h"

#include "vector.h"

template <typename T>

void manualInput(Matrix<T>& A, Vector<T>& b, size\_t n, size\_t m) {

std::cout << "Введите коэффициенты матрицы A и правой части b:\n";

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

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

std::cin >> A(i, j);

}

std::cin >> b[i];

}

}

template <typename T>

void randomInput(Matrix<T>& A, Vector<T>& b, size\_t n, size\_t m) {

srand(static\_cast<unsigned int>(time(nullptr)));

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

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

A(i, j) = static\_cast<T>((rand() % 2000 - 1000) / 100.0);

}

b[i] = static\_cast<T>((rand() % 2000 - 1000) / 100.0);

}

std::cout << "Сгенерированная система:\n";

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

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

std::cout << A(i, j) << " ";

}

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

}

}

template <typename T>

void solveSystem(size\_t n, size\_t m, int inputChoice) {

Matrix<T> A(n, m);

Vector<T> b(n);

if (inputChoice == 1) {

manualInput(A, b, n, m);

}

else {

randomInput(A, b, n, m);

}

Vector<T> x;

Vector<Vector<T>> null\_space\_basis;

size\_t operation\_count = 0;

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

gauss\_jordan(A, b, x, null\_space\_basis, static\_cast<T>(1e-12),

&operation\_count);

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

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(

end\_time - start\_time);

check\_solution(A, x, b);

std::cout << "\nСтатистика выполнения:\n";

std::cout << "Размер матрицы: " << n << "x" << m << "\n";

std::cout << "Время выполнения: " << duration.count() << " микросекунд\n";

std::cout << "Количество операций: " << operation\_count << "\n";

std::cout << "\nРешение:\n";

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

std::cout << "x" << i << " = " << x[i] << "\n";

}

if (!null\_space\_basis.empty()) {

std::cout << "\nСистема имеет бесконечно много решений (базисные вектора системы):\n";

for (size\_t k = 0; k < null\_space\_basis.size(); ++k) {

std::cout << "v" << k << ": (";

for (size\_t j = 0; j < null\_space\_basis[k].size(); ++j) {

std::cout << null\_space\_basis[k][j];

if (j + 1 < null\_space\_basis[k].size()) std::cout << ", ";

}

std::cout << ")\n";

}

}

else {

std::cout << "\nСистема имеет единственное решение.\n";

}

}

int main() {

try {

setlocale(LC\_ALL, "Russian");

int dataTypeChoice;

std::cout << "Выберите тип данных:\n";

std::cout << "1. float\n";

std::cout << "2. double\n";

std::cout << "Ваш выбор: ";

std::cin >> dataTypeChoice;

int inputChoice;

std::cout << "\nВыберите способ ввода:\n";

std::cout << "1. Ручной ввод\n";

std::cout << "2. Случайная генерация\n";

std::cout << "Ваш выбор: ";

std::cin >> inputChoice;

size\_t n, m;

std::cout << "\nВведите число уравнений (строк): ";

std::cin >> n;

std::cout << "Введите число неизвестных (столбцов): ";

std::cin >> m;

if (dataTypeChoice == 1) {

solveSystem<float>(n, m, inputChoice);

}

else {

solveSystem<double>(n, m, inputChoice);

}

}

catch (const GaussException& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

}

catch (...) {

std::cerr << "Неизвестная ошибка" << std::endl;

}

return 0;

}