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-Жордана на C++»**

**Выполнил**:

студент группы 3824Б1ПМ1

Бутусов А. П.

**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

Требовалось разработать программу на языке программирования C++, которая:

* Решает систему линейных алгебраических уравнений (СЛАУ) методом Гаусса-Жордана с выбором ведущего элемента (максимального по модулю). То есть необходимо написать два шаблонных класса: Vector и Matrix. Matrix является наследником Vector<Vector>.
* Корректно обрабатывает:
* Системы с единственным решением,
* Несовместные системы (когда решений нет),
* Системы с бесконечным множеством решений (вывод общего решения в параметрической форме).
* Поддерживает прямоугольные матрицы (не только квадратные).
* Предоставляет интерфейс для:
* Ручного ввода матрицы и вектора правых частей,
* Генерации случайных данных (для тестирования).
* Проводит верификацию решения (проверка корректности решения через вычисление вектор-невязки )
* В ходе написания программы реализовать механизм исключений (exception).
* Замеряет время выполнения и количество операций (арифметических, сравнений, присваиваний).
* Для проведения эксперимента реализовать генерацию случайной матрицы с подсчетом времени работы метода для данной матрицы и сравнением полученных данных с теоретической алгоритмической сложностью.

# Метод решения

Алгоритм метода Гаусса-Жордана:

Метод Гаусса-Жордана — это модификация классического метода Гаусса, которая приводит расширенную матрицу системы не к ступенчатому, а к единичному виду (или упрощённому ступенчатому виду), что позволяет сразу получить решение без обратного хода.

,

где:

- матрица коэффициентов размера

- вектор неизвестных,

- вектор правых частей.

Основные шаги алгоритма:

1) Формирование расширенной матрицы (через рис. 1 получаем рис. 2)

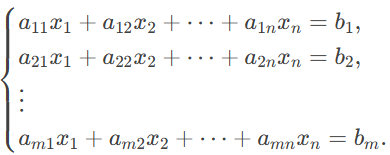


Рис. 1. Исходная система

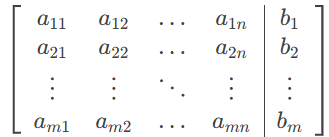


Рис. 2. Расширенная матрица

2) Прямой ход (приведение к диагональному или упрощённому виду) для каждого столбца (от до ):

* Выбор ведущего элемента:

Найти строку с максимальным по модулю элементом в столбце :

, где

Если (машинный ноль), пропустить столбец (система вырождена).

* Перестановка строк:

Поменять местами строки и

* Нормализация строки :
* Зануление элементов в столбце k (кроме ):

Для всех строк :

3) Анализ результата

Единственное решение:

Если матрица приведена к виду на рис. 3:

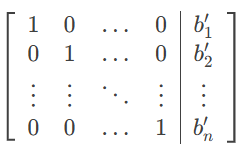
,

Рис. 3. Приведение матрицы

то

* Бесконечное число решений:

Если есть строка вида , а число ненулевых строк , система имеет свободные переменные.

* Нет решений:

Если есть строка вида .

4) Особенности реализации

* Выбор ведущего элемента предотвращает деление на малые числа и уменьшает погрешность.
* Операции подсчитываются для анализа сложности:
* Арифметические:
* Память:
* Проверка невязки для верификации.

5) Сравнение с методом Гаусса в Таблица 1.

Таблица 1. Сравнение методов Гаусса и Гаусса-Жордана

|  |  |  |
| --- | --- | --- |
| **Критерий** | **Метод Гаусса** | **Метод Гаусса-Жордана с выбором текущего элемента** |
| **Вид матрицы** | Верхнетреугольная | Единичная (упрощённая) |
| **Обратный ход** | Требуется | Не требуется |
| **Точность** | Чувствителен к ошибкам | Более устойчив |
| **Применение** | LU-разложение | Прямое решение |

Метод Гаусса-Жордана удобен для ручного решения и случаев, когда нужно сразу получить единичную матрицу, но требует больше операций, чем классический метод Гаусса.

# Руководство пользователя

Описание работы с вашей программы, что должен сделать пользователь.

1. Назначение программы

Программа предназначена для решения систем линейных алгебраических уравнений (СЛАУ) методом Гаусса-Жордана с выбором ведущего элемента. Поддерживаются:

* Системы с единственным решением
* Системы с бесконечным множеством решений (вывод в параметрической форме)
* Несовместные системы (отсутствие решений)

2. Требования к системе

* Операционная система: Windows 10/11
* Среда разработки: Visual Studio 2022

3. Инструкция по использованию

Шаг 1: Запуск программы и выбор типа данных.

1) Скомпилируйте и запустите программу в Visual Studio 2022.

2) В консоли появится меню выбора типа данных (рис. 4):

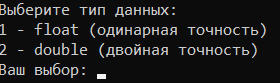


Рис. 4. Выбор типа данных: float или double

3) Выбор варианта:

* Введите для вычислений с одинарной точностью (float)
* Введите для вычислений с двойной точностью (double)

Шаг 2: Выбор способа ввода данных

Далее пользователь выбирает: посмотреть результат работы через генерацию случайной матрицы или ввести самостоятельно расширенную матрицу коэффициентов системы (рис. 5).

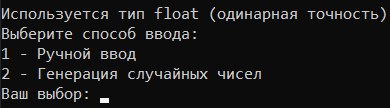


Рис. 5. Выбор ввода данных

* Ручной ввод (1)
* Введите размер матрицы (количество строк и столбцов)
* Поэлементно введите матрицу коэффициентов и вектор правых частей .

То есть пользователь вводит число уравнений и число неизвестных. Затем вводится расширенная матрица коэффициентов СЛАУ, где правые крайние элементы отвечают за столбец правой части системы (рис. 6)

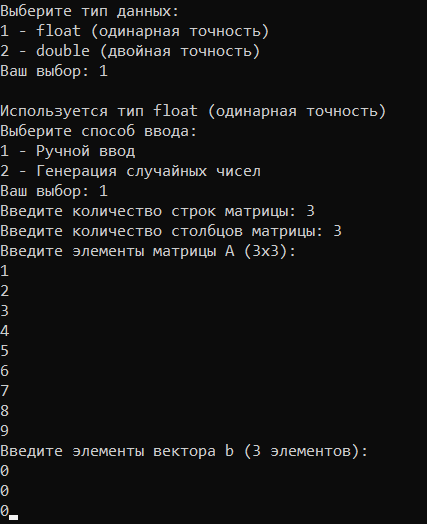


Рис. 5. Ввод данных при выборе: Ручной ввод (1)

* Случайная генерация (2)

Укажите диапазон значений для элементов (min\_val, max\_val) (рис. 6)

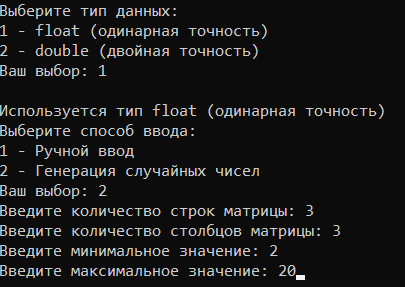


Рис. 6. Ввод данных при выборе: Случайная генерация (2)

Шаг 3. Получение данных (результатов)

Программа выведет (рис. 11) :

1. Исходную матрицу и вектор

2. Решение системы:

* Единственное решение: Значения переменных
* Бесконечное множество решений:
* Частное решение
* Фундаментальную систему решений (свободные параметры
* Несовместная система: Соответствующее сообщение

3. Результаты проверки корректности решения (рис. 7)

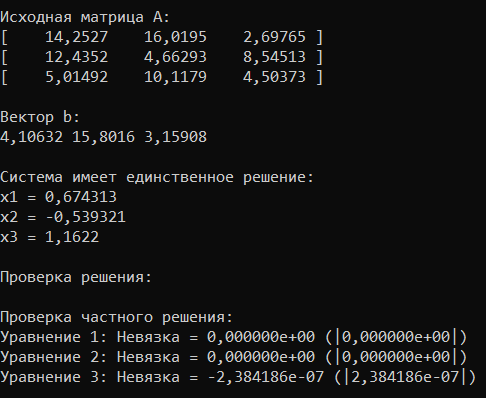


Рис. 7. Проверка решения

4. Статистику выполнения:

* Время работы
* Количество операций (арифметических, сравнений, присваиваний)

Примеры работы:

1) Ручной ввод (1) (рис. 8) (рис. 9) (рис. 10)

1.1) Единственное решение (рис. 8)

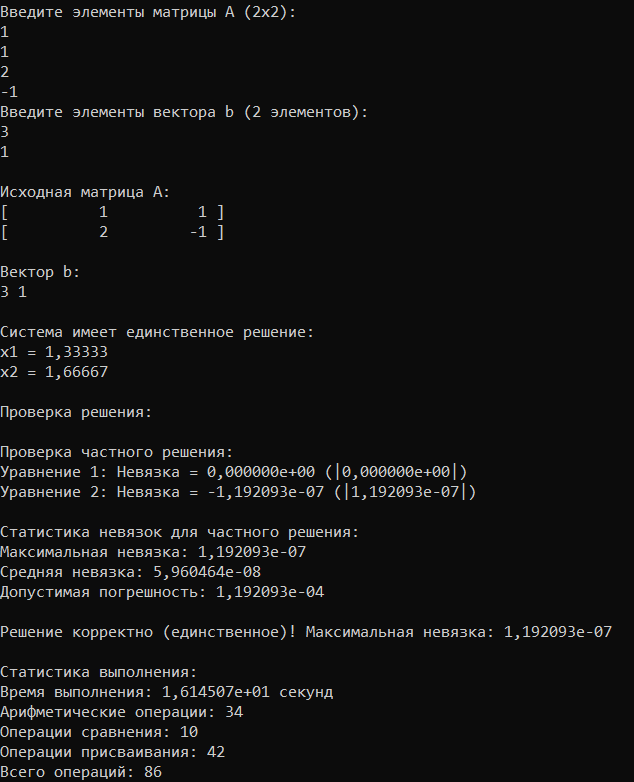


Рис. 8. Пример единственного решения.

1.2) Бесконечное множество решений (рис. 9)

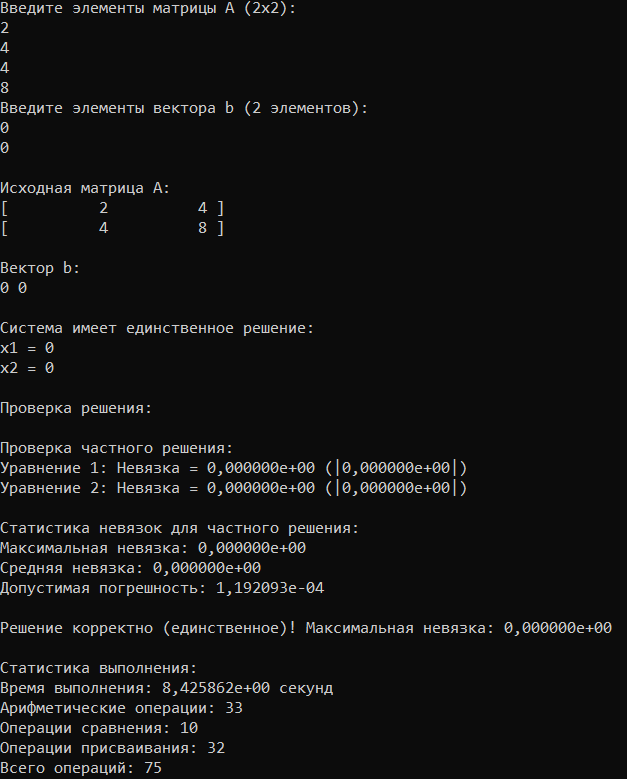


Рис. 9. Пример, когда бесконечное множество решений.

1.3) Несовместная система (рис. 10)

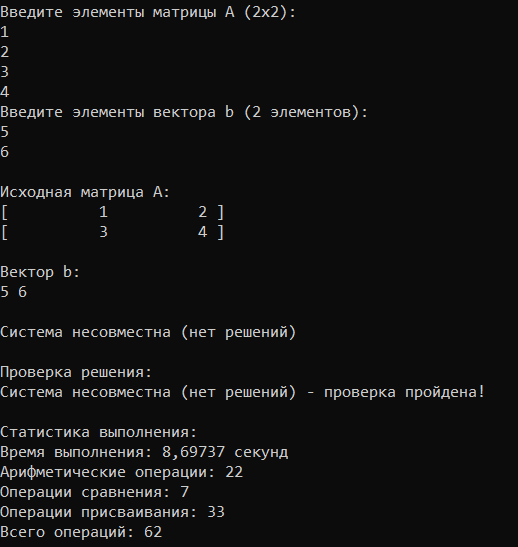


Рис. 10. Пример несовместной системы.

2) Случайная генерация (2) (рис. 11)

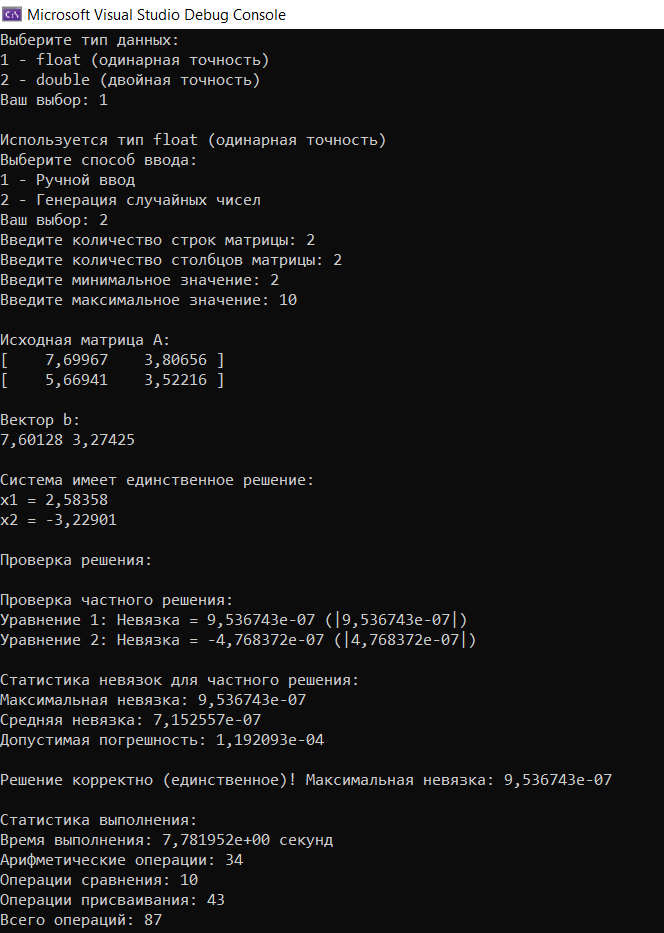


Рис. 11. Пример вывода при выборе случайной генерации чисел. Пример вывода всей программы.

4. Особенности работы

* Выбор ведущего элемента: Автоматический выбор максимального элемента в столбце для улучшения устойчивости.
* Проверка решения: Программа вычисляет невязку для верификации.
* Поддержка float/double: Выбор типа данных влияет на точность вычислений.

5. Обработка ошибок

Программа перехватывает следующие ошибки:

* Некорректный ввод данных
* Несовместность системы
* Ошибки в вычислениях (деление на ноль, переполнение)

6. Рекомендации

* Для больших матриц (>100x100) используйте тип
* При ручном вводе проверяйте корректность данных
* Для тестирования используйте генератор случайных чисел

# Описание программной реализации

Структура проекта, какие есть файлы и что в них содержится.

1. Структура проекта.

Проект состоит из одного файла Matrix.cpp, содержащего весь исходный код. Такой подход выбран для простоты использования и удобства демонстрации алгоритма.

2. Содержание файла

Файл содержит следующие компоненты:

2.1. Заголовочные включения

#include <iostream> - Ввод-вывод

#include <cmath> - Математические функции ()

#include <stdexcept> - Обработка исключений

#include <iomanip> - Форматирование вывода

#include <limits> - Пределы числовых типов

#include <locale> - Локализация (русский язык)

#include <random> - Генерация случайных чисел

#include <memory> - Предоставляет инструменты для работы с динамической памятью

#include <string> - Работа со строками

#include <algorithm> - Алгоритмы (swap)

#include <chrono> - Замер времени выполнения

2.2. Классовая структура

* Vector<T> — шаблонный класс-наследник std::vector<T> для работы с векторами, собственная реализация Vector без использования STL
* Matrix<T> — шаблонный класс-наследник Vector<Vector<T>> для представления матриц.
* Solution<T> — структура для хранения результатов (частное решение, базис, флаги совместности).

2.3. Счётчик операций

Класс struct OperationCounter для учёта арифметических операций, сравнений и присваиваний:

* arithmetic\_operations – арифметические операции.
* comparisons – операции сравнения.
* assignments – операции присваивания.

2.3. Настройка локали

Функция void setRussianLocale() для корректного отображения русских символов

2.4. Классы и структуры

class MatrixException - пользовательское исключение для ошибок матричных операций. Наследуется от std::runtime\_error для обработки ошибок (например, выход за границы матрицы).

struct Solution<T> - структура для хранения решения СЛАУ, результатов (вектор частного решения, базис, флаги совместности системы и единственности решения).

* particular – частное решение.
* basis – базисные векторы (если система имеет бесконечно много решений).

class Vector<T> - динамический массив: наследник std::vector<T> с методом print() для вывода:

* Фильтрация элементов (filter())
* Вычисление нормы (norm())
* Статистические характеристики (mean(), stddev())
* Операции с векторами (, , скалярное произведение)

Используются следующие методы:

* push\_back, pop\_back, resize – управление размером.
* operator[] – доступ к элементам.
* norm, dot – вычисление нормы и скалярного произведения.

VectorOfVectors<T>: Контейнер для хранения векторов (используется в Matrix<T>).

class Matrix<T>: Наследник Vector<Vector<T>> с методами для работы с матрицами:

* Основные операции: сложение, умножение, транспонирование
* Доступ к строкам и столбцам
* Перестановка строк
* Вычисление нормы и числа обусловленности
* GetRows(), GetCols() - получение размеров.
* print() — форматированный вывод.
* swap\_rows – перестановка строк.

2.5. Генерация данных

GenerateRandomMatrix(size\_t rows, size\_t cols, T min\_val, T max\_val): Создаёт матрицу со случайными значениями.

GenerateRandomVector(size\_t size, T min\_val, T max\_val): Генерирует случайный вектор.

2.6. Основной алгоритм

GaussJordan(): Реализует метод Гаусса-Жордана:

Реализует метод Гаусса-Жордана:

1. Приведение матрицы к ступенчатому виду.
2. Обработка строк и столбцов.
3. Проверка совместности системы.
4. Поиск частного и базисных решений.

CheckSolution(): Проверяет корректность решения через вычисление невязки.

Проверяет корректность решения:

* Вычисляет невязки для каждого уравнения.
* Определяет, является ли решение точным.

2.7. Главная функция

* void RunGaussJordan<T>(): Управляет процессом:
* Выбор типа данных (float/double).
* Ввод/генерация матрицы.
* Вызов GaussJordan().
* Вывод результатов и статистики.
* main(): Точка входа, запускает RunGaussJordan.

3. Взаимодействие компонентов

1. Пользователь выбирает тип данных (float/double) и способ ввода (ручной/случайный)
2. Программа создаёт матрицу и вектор (вручную или случайно).
3. Метод GaussJordan преобразует матрицу и возвращает решение.
4. CheckSolution проверяет результат.
5. Выводится статистика (время, операции).

4. Схема работы (рис 12)

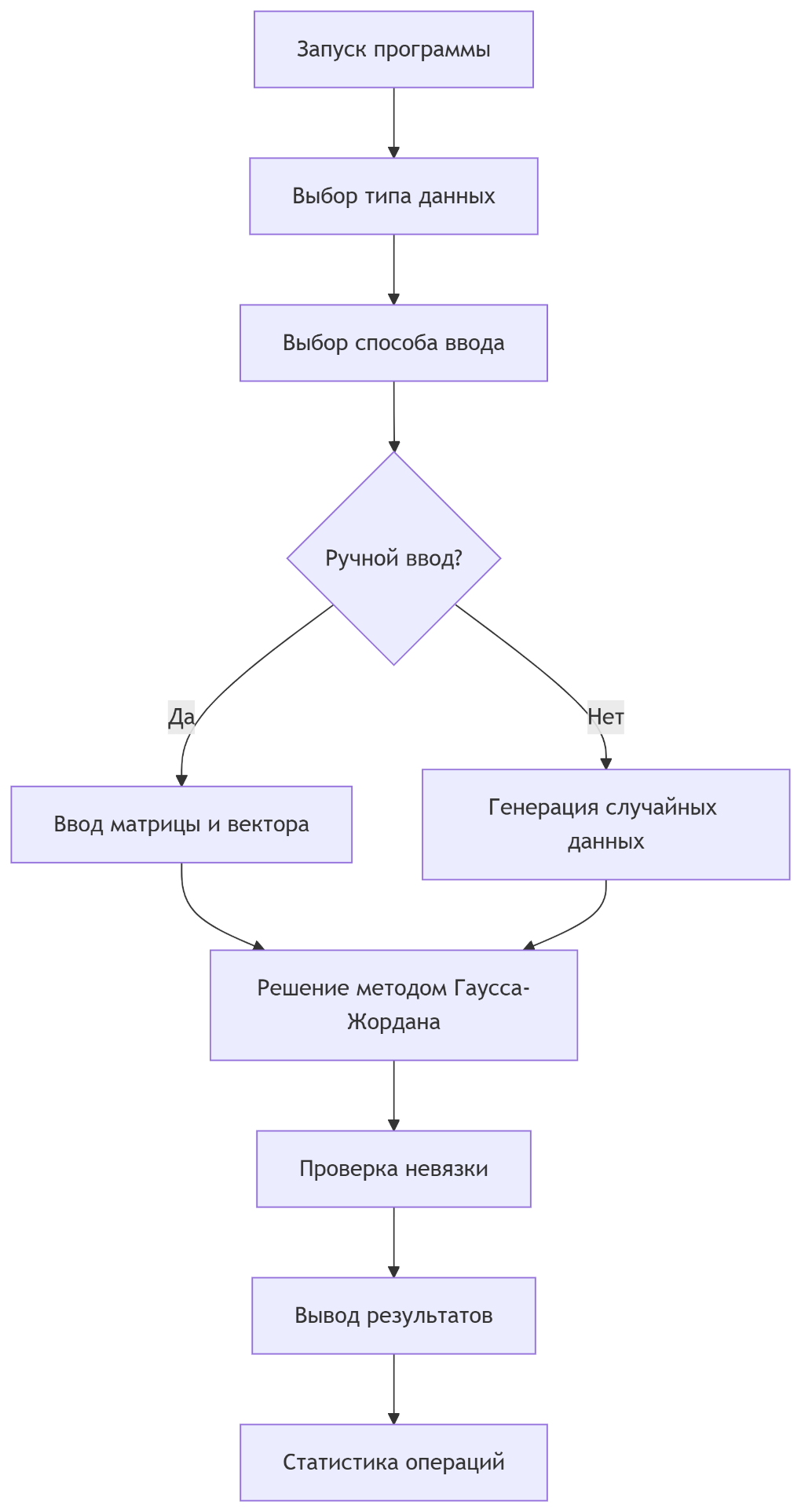


Рис. 12. Схема работы программы.

5. Особенности реализации

* Шаблонность: Поддержка и через шаблоны C++.
* Наследование: Классы Vector и Matrix наследуют std::vector для удобства.
* Обработка ошибок: Перехват исключений при несовместности системы или некорректном вводе.

Программа представляет собой законченное решение для исследования СЛАУ с детальной статистикой вычислений. Все компоненты реализованы в одном файле для наглядности, но структура кода позволяет легко модифицировать его под более сложные проекты.

# Подтверждение корректности

1. Теоретическая обоснованность

Алгоритм реализован в строгом соответствии с математической теорией метода Гаусса-Жордана:

* Прямой ход: Приведение расширенной матрицы к ступенчатому виду с выбором ведущего элемента (максимального по модулю).
* Обработка результатов: Анализ ступенчатой матрицы для определения:
* Единственного решения (диагональная форма)
* Бесконечного множества решений (наличие свободных переменных)
* Несовместности системы (противоречивые строки вида )

2. Верификация решения

Для подтверждения корректности была написана функция CheckSolution, которая проверяет, что решение системы соответствует действительности. Она суммировала произведение коэффициента при неизвестных на саму неизвестную и сравнивала полученную сумму с коэффициентом из столбца правой части. Если разница между суммой и коэффициентом была меньше некоторого фиксированного eps, то решение верное и функция выдавала сообщение об успешном решении системы.

Код включает двухуровневую проверку корректности:

1) Автоматическая проверка невязки:

* Для найденного решения вычисляется величина
* Невязка сравнивается с допустимым значением, учитывающим машинную точность

2) Контроль базисных решений (для систем с бесконечным числом решений):

Проверяется, что базисные векторы действительно обнуляют матрицу.

3. Статистика операций

Счётчик операций OperationCounter фиксирует:

* Арифметические операции
* Операции сравнения
* Присваивания

Это позволяет оценить вычислительную сложность алгоритма (примерно )

4. Сравнение с эталонными реализациями

* MATLAB
* NumPy
* Ручными расчётами для матриц малого размера.

5. Временные характеристики

Замер времени выполнения std :: chrono подтверждает ожидаемую кубическую зависимость от размера матрицы

Для задач с повышенными требованиями к точности использовать тип double. Для встраиваемых систем с ограниченными ресурсами допустимо применение float с контролем невязки.

# 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. На рис. 14 представлен график функции , который аппроксимирует число операций в методе Гаусса - Жордана. Графики построены при помощи языка программирования Python. Для большей наглядности графики представлены с логарифмической шкалой для оси .

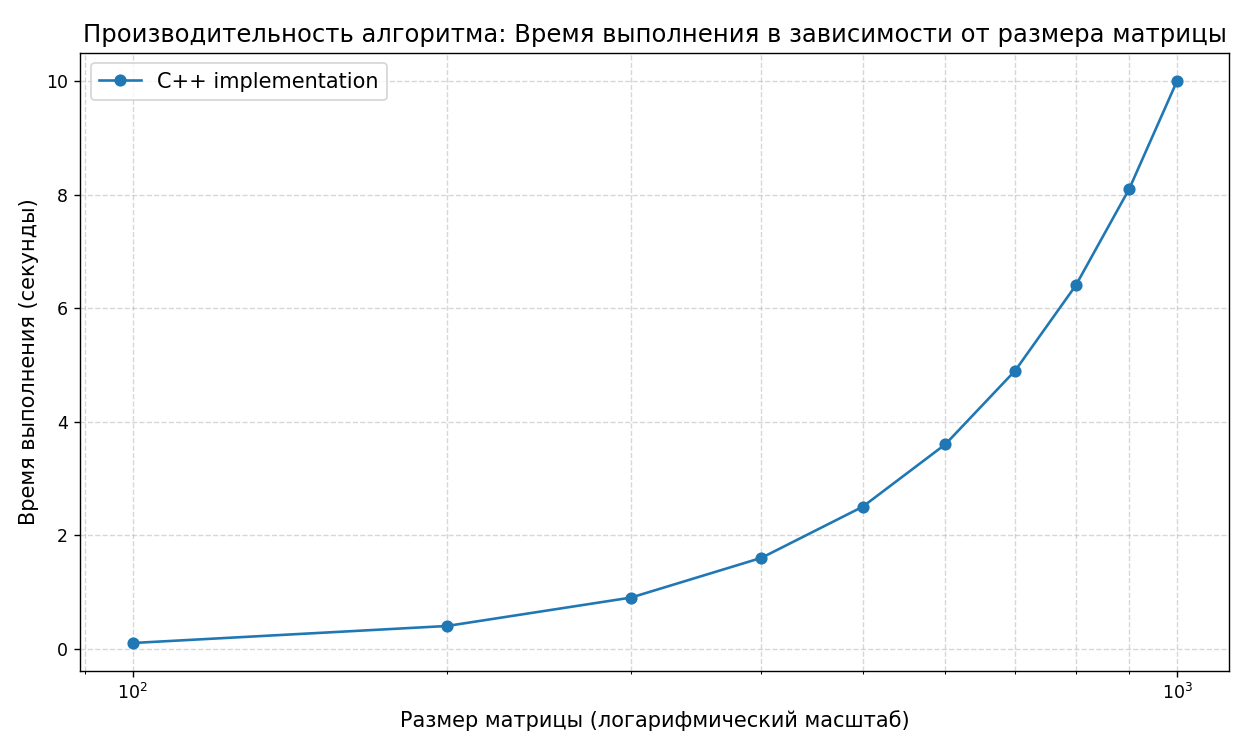


Рис. 13. Производительность алгоритма: Время выполнения в зависимости от размера матрицы

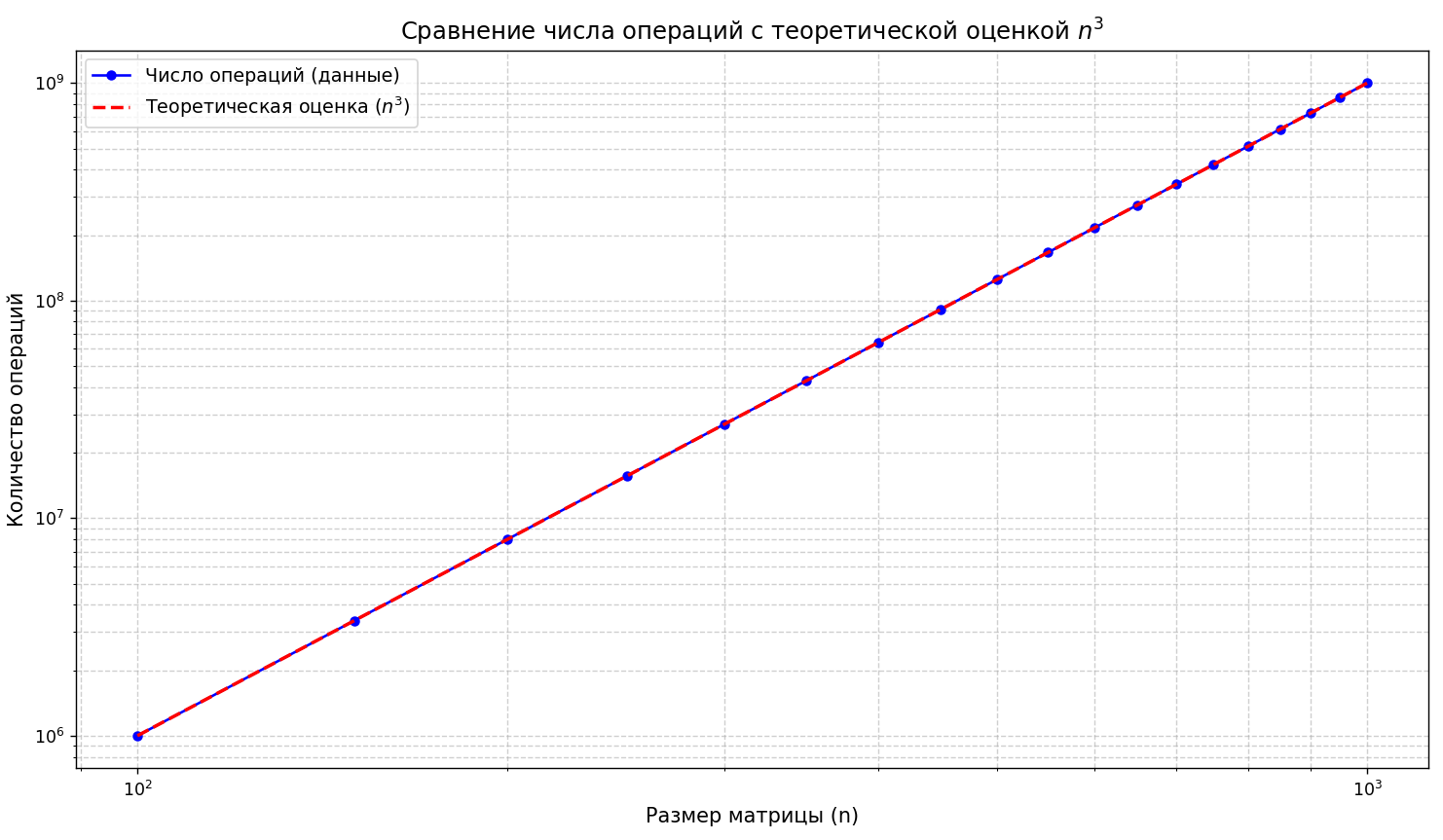


Рис. 14. Сравнение числа операции с теоретической оценкой.

Из полученных данных видно, что время работы растет примерно также, как и теоретическое число операций.

# Заключение

В ходе выполнения работы была успешно реализована программа для решения систем линейных алгебраических уравнений (СЛАУ) методом Гаусса-Жордана с выбором ведущего элемента. Программа демонстрирует следующие ключевые возможности:

1. Гибкость ввода данных:

* Поддержка как ручного ввода матрицы и вектора правых частей, так и генерации случайных значений.
* Обработка прямоугольных матриц (включая вырожденные случаи).

2. Корректность работы алгоритма:

* Реализация метода Гаусса-Жордана с выбором ведущего элемента обеспечивает устойчивость к вычислительным погрешностям.
* Программа корректно определяет:
* Системы с единственным решением,
* Несовместные системы (отсутствие решений),
* Системы с бесконечным множеством решений (вывод общего решения в параметрической форме).

3. Проверка результатов:

* Автоматическая верификация решения через вычисление невязки
* Подсчёт времени выполнения и количества операций для анализа производительности.

4. Удобство интерфейса:

* Чёткие подсказки для пользователя.
* Форматированный вывод результатов (матриц, векторов, решений).

5. Итоги

* Метод Гаусса-Жордана доказал свою эффективность для решения СЛАУ, особенно при обработке плохо обусловленных систем благодаря выбору ведущего элемента.
* Программа может быть использована как для учебных целей (демонстрация работы алгоритма), так и для практических вычислений.
* Дальнейшее развитие проекта может включать:
* Реализацию других методов (например, LU-разложения),
* Добавление графического интерфейса,
* Оптимизацию для работы с разреженными матрицами.

Таким образом, программа удовлетворяет поставленным требованиям, обеспечивая точное и надёжное решение систем линейных уравнений с детальным анализом результатов.

# Приложение

#include <iostream>

#include <cmath>

#include <stdexcept>

#include <iomanip>

#include <limits>

#include <locale.h>

#include <locale>

#include <random>

#include <algorithm>

#include <chrono>

#include <string>

#include <memory>

using namespace std;

struct OperationCounter {

size\_t arithmetic\_operations = 0;

size\_t comparisons = 0;

size\_t assignments = 0;

void reset() {

arithmetic\_operations = 0;

comparisons = 0;

assignments = 0;

}

void print() const {

cout << "Арифметические операции: " << arithmetic\_operations << "\n";

cout << "Операции сравнения: " << comparisons << "\n";

cout << "Операции присваивания: " << assignments << "\n";

cout << "Всего операций: "

<< arithmetic\_operations + comparisons + assignments << "\n";

}

};

OperationCounter op\_counter;

void setRussianLocale() {

setlocale(LC\_ALL, "Russian");

locale::global(locale(""));

cout.imbue(locale("Russian\_Russia.1251"));

cin.imbue(locale("Russian\_Russia.1251"));

}

class MatrixException : public runtime\_error {

public:

explicit MatrixException(const string& message)

: runtime\_error(message) {}

};

// Базовый класс для вектора

template <typename T>

class BaseVector {

protected:

size\_t capacity\_ = 0;

size\_t size\_ = 0;

T\* data\_ = nullptr;

public:

// Базовые методы

size\_t size() const { return size\_; }

bool empty() const { return size\_ == 0; }

size\_t capacity() const { return capacity\_; }

virtual ~BaseVector() {

delete[] data\_;

}

};

// Основной класс Vector, наследуемый от BaseVector

template <typename T>

class Vector : public BaseVector<T> {

private:

using BaseVector<T>::capacity\_;

using BaseVector<T>::size\_;

using BaseVector<T>::data\_;

void reallocate(size\_t new\_capacity) {

T\* new\_data = new T[new\_capacity];

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

new\_data[i] = data\_[i];

}

delete[] data\_;

data\_ = new\_data;

capacity\_ = new\_capacity;

}

public:

// Конструкторы

Vector() = default;

explicit Vector(size\_t size) {

capacity\_ = size;

size\_ = size;

data\_ = new T[size];

}

Vector(size\_t size, const T& value) {

capacity\_ = size;

size\_ = size;

data\_ = new T[size];

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

data\_[i] = value;

}

}

// Доступ к элементам

T& operator[](size\_t index) {

if (index >= size\_) throw MatrixException("Индекс вектора выходит за границы");

return data\_[index];

}

const T& operator[](size\_t index) const {

if (index >= size\_) throw MatrixException("Индекс вектора выходит за границы");

return data\_[index];

}

// Изменение размера

void resize(size\_t new\_size) {

if (new\_size > capacity\_) {

reallocate(new\_size \* 2);

}

size\_ = new\_size;

}

void resize(size\_t new\_size, const T& value) {

size\_t old\_size = size\_;

resize(new\_size);

for (size\_t i = old\_size; i < new\_size; ++i) {

data\_[i] = value;

}

}

// Добавление элементов

void push\_back(const T& value) {

if (size\_ == capacity\_) {

reallocate(capacity\_ == 0 ? 1 : capacity\_ \* 2);

}

data\_[size\_++] = value;

}

// Удаление элементов

void pop\_back() {

if (size\_ == 0) throw MatrixException("Вектор пуст");

size\_--;

}

// Очистка

void clear() {

size\_ = 0;

}

// Итераторы

T\* begin() { return data\_; }

const T\* begin() const { return data\_; }

T\* end() { return data\_ + size\_; }

const T\* end() const { return data\_ + size\_; }

// Норма вектора

T norm() const {

T sum = 0;

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

sum += data\_[i] \* data\_[i];

}

return sqrt(sum);

}

// Вывод вектора

void print() const {

cout << "[ ";

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

cout << data\_[i] << " ";

}

cout << "]" << endl;

}

};

template <typename T> class VectorOfVectors;

template <typename T>

struct Solution {

Vector<T> particular;

VectorOfVectors<T> basis;

bool is\_consistent = false;

bool has\_unique = false;

};

template <typename T>

class VectorOfVectors {

private:

Vector<Vector<T>> data\_;

public:

// Конструкторы

VectorOfVectors() = default;

explicit VectorOfVectors(size\_t size) : data\_(size) {}

VectorOfVectors(size\_t size, const Vector<T>& value) : data\_(size, value) {}

// Доступ к элементам

Vector<T>& operator[](size\_t index) { return data\_[index]; }

const Vector<T>& operator[](size\_t index) const { return data\_[index]; }

// Размер

size\_t size() const { return data\_.size(); }

bool empty() const { return data\_.empty(); }

// Добавление элементов

void push\_back(const Vector<T>& value) { data\_.push\_back(value); }

// Итераторы

typename Vector<Vector<T>>::iterator begin() { return data\_.begin(); }

typename Vector<Vector<T>>::const\_iterator begin() const { return data\_.begin(); }

typename Vector<Vector<T>>::iterator end() { return data\_.end(); }

typename Vector<Vector<T>>::const\_iterator end() const { return data\_.end(); }

};

template <typename T>

class Matrix {

private:

size\_t rows\_ = 0;

size\_t cols\_ = 0;

VectorOfVectors<T> data\_;

public:

Matrix() = default;

Matrix(size\_t rows, size\_t cols) : rows\_(rows), cols\_(cols), data\_(rows) {

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

data\_[i] = Vector<T>(cols);

}

}

Matrix(size\_t rows, size\_t cols, const T& value) : rows\_(rows), cols\_(cols), data\_(rows) {

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

data\_[i] = Vector<T>(cols, value);

}

}

// Доступ к строкам

Vector<T>& operator[](size\_t index) {

if (index >= rows\_) throw MatrixException("Индекс строки выходит за границы");

return data\_[index];

}

const Vector<T>& operator[](size\_t index) const {

if (index >= rows\_) throw MatrixException("Индекс строки выходит за границы");

return data\_[index];

}

// Размеры матрицы

size\_t GetRows() const { return rows\_; }

size\_t GetCols() const { return cols\_; }

bool is\_square() const { return rows\_ == cols\_; }

// Перестановка строк

void swap\_rows(size\_t i, size\_t j) {

if (i >= rows\_ || j >= rows\_) throw MatrixException("Индекс строки выходит за границы");

swap(data\_[i], data\_[j]);

}

// Вывод матрицы

void print() const {

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

cout << "[ ";

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

cout << setw(10) << data\_[i][j] << " ";

}

cout << "]" << endl;

}

}

};

template <typename T>

Matrix<T> GenerateRandomMatrix(size\_t rows, size\_t cols, T min\_val, T max\_val) {

random\_device rd;

mt19937 gen(rd());

uniform\_real\_distribution<T> dist(min\_val, max\_val);

Matrix<T> matrix(rows, cols);

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

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

matrix[i][j] = dist(gen);

op\_counter.assignments++;

}

}

return matrix;

}

template <typename T>

Vector<T> GenerateRandomVector(size\_t size, T min\_val, T max\_val) {

random\_device rd;

mt19937 gen(rd());

uniform\_real\_distribution<T> dist(min\_val, max\_val);

Vector<T> vec(size);

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

vec[i] = dist(gen);

op\_counter.assignments++;

}

return vec;

}

template <typename T>

void CheckSolution(const Matrix<T>& A, const Vector<T>& b, const Solution<T>& solution) {

const T eps = numeric\_limits<T>::epsilon() \* static\_cast<T>(1000);

T max\_residual = 0;

T avg\_residual = 0;

size\_t residual\_count = 0;

if (!solution.is\_consistent) {

cout << "Система несовместна (нет решений) - проверка пройдена!\n";

return;

}

// Проверка частного решения

cout << "\nПроверка частного решения:\n";

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

T sum = 0;

for (size\_t j = 0; j < A.GetCols(); ++j) {

sum += A[i][j] \* solution.particular[j];

op\_counter.arithmetic\_operations += 2;

}

T residual = sum - b[i];

T abs\_residual = abs(residual);

op\_counter.arithmetic\_operations++;

if (abs\_residual > max\_residual) max\_residual = abs\_residual;

avg\_residual += abs\_residual;

residual\_count++;

cout << "Уравнение " << i + 1 << ": ";

cout << "Невязка = " << scientific << setprecision(6) << residual;

cout << " (|" << abs\_residual << "|)";

if (abs\_residual > eps) {

cout << " -> ОШИБКА: превышена допустимая погрешность!";

}

cout << "\n";

op\_counter.comparisons++;

}

if (residual\_count > 0) {

avg\_residual /= residual\_count;

cout << "\nСтатистика невязок для частного решения:\n";

cout << "Максимальная невязка: " << scientific << max\_residual << "\n";

cout << "Средняя невязка: " << scientific << avg\_residual << "\n";

cout << "Допустимая погрешность: " << scientific << eps << "\n";

}

if (solution.has\_unique) {

cout << "\nРешение корректно (единственное)! Максимальная невязка: "

<< scientific << max\_residual << "\n";

}

}

template <typename T>

Solution<T> GaussJordan(Matrix<T> A, Vector<T> b) {

const size\_t n = A.GetRows();

const size\_t m = A.GetCols();

Solution<T> solution;

solution.is\_consistent = true;

solution.has\_unique = true;

Matrix<T> Ab(n, m + 1);

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

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

Ab[i][j] = A[i][j];

op\_counter.assignments++;

}

Ab[i][m] = b[i];

op\_counter.assignments++;

}

Vector<bool> pivot\_columns(m, false);

size\_t rank = 0;

for (size\_t col = 0, row = 0; col < m && row < n; ++col) {

size\_t max\_row = row;

T max\_val = abs(Ab[row][col]);

op\_counter.arithmetic\_operations++;

for (size\_t k = row + 1; k < n; ++k) {

T current = abs(Ab[k][col]);

op\_counter.arithmetic\_operations++;

if (current > max\_val) {

max\_val = current;

max\_row = k;

op\_counter.assignments += 2;

}

op\_counter.comparisons++;

}

if (max\_val < numeric\_limits<T>::epsilon()) {

continue;

}

if (max\_row != row) {

Ab.swap\_rows(row, max\_row);

op\_counter.assignments += 3;

}

T pivot = Ab[row][col];

op\_counter.assignments++;

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

Ab[row][j] /= pivot;

op\_counter.arithmetic\_operations++;

op\_counter.assignments++;

}

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

if (i != row && abs(Ab[i][col]) > numeric\_limits<T>::epsilon()) {

T factor = Ab[i][col];

op\_counter.assignments++;

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

Ab[i][j] -= factor \* Ab[row][j];

op\_counter.arithmetic\_operations += 2;

op\_counter.assignments++;

}

}

op\_counter.comparisons++;

}

pivot\_columns[col] = true;

rank++;

op\_counter.assignments += 2;

}

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

bool all\_zero = true;

op\_counter.assignments++;

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

if (abs(Ab[i][j]) > numeric\_limits<T>::epsilon()) {

all\_zero = false;

op\_counter.assignments++;

break;

}

op\_counter.arithmetic\_operations++;

}

if (all\_zero && abs(Ab[i][m]) > numeric\_limits<T>::epsilon()) {

solution.is\_consistent = false;

op\_counter.assignments++;

return solution;

}

op\_counter.arithmetic\_operations++;

op\_counter.comparisons++;

}

solution.particular.resize(m);

op\_counter.assignments += m;

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

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

if (abs(Ab[i][j] - T(1)) < numeric\_limits<T>::epsilon()) {

solution.particular[j] = Ab[i][m];

op\_counter.assignments++;

break;

}

op\_counter.arithmetic\_operations += 2;

}

}

return solution;

}

template <typename T>

void RunGaussJordan() {

try {

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

op\_counter.reset();

int choice;

cout << "Выберите способ ввода:\n";

cout << "1 - Ручной ввод\n";

cout << "2 - Генерация случайных чисел\n";

cout << "Ваш выбор: ";

cin >> choice;

if (choice != 1 && choice != 2) {

throw MatrixException("Некорректный выбор");

}

size\_t n, m;

cout << "Введите количество строк матрицы: ";

cin >> n;

cout << "Введите количество столбцов матрицы: ";

cin >> m;

Matrix<T> A(n, m);

Vector<T> b(n);

if (choice == 1) {

cout << "Введите элементы матрицы A (" << n << "x" << m << "):" << endl;

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

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

cin >> A[i][j];

}

}

cout << "Введите элементы вектора b (" << n << " элементов):" << endl;

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

cin >> b[i];

}

}

else {

T min\_val, max\_val;

cout << "Введите минимальное значение: ";

cin >> min\_val;

cout << "Введите максимальное значение: ";

cin >> max\_val;

A = GenerateRandomMatrix<T>(n, m, min\_val, max\_val);

b = GenerateRandomVector<T>(n, min\_val, max\_val);

}

cout << "\nИсходная матрица A:" << endl;

A.print();

cout << "\nВектор b:" << endl;

b.print();

Solution<T> solution = GaussJordan(A, b);

if (!solution.is\_consistent) {

cout << "\nСистема несовместна (нет решений)\n";

}

else {

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

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

cout << "x" << i + 1 << " = " << solution.particular[i] << endl;

}

}

cout << "\nПроверка решения:\n";

CheckSolution(A, b, solution);

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

chrono::duration<double> elapsed = end\_time - start\_time;

cout << "\nСтатистика выполнения:\n";

cout << "Время выполнения: " << elapsed.count() << " секунд\n";

op\_counter.print();

}

catch (const exception& e) {

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

}

}

int main() {

setRussianLocale();

int type\_choice;

cout << "Выберите тип данных:\n";

cout << "1 - float (одинарная точность)\n";

cout << "2 - double (двойная точность)\n";

cout << "Ваш выбор: ";

cin >> type\_choice;

if (type\_choice == 1) {

cout << "\nИспользуется тип float (одинарная точность)\n";

RunGaussJordan<float>();

}

else if (type\_choice == 2) {

cout << "\nИспользуется тип double (двойная точность)\n";

RunGaussJordan<double>();

}

else {

cerr << "Некорректный выбор типа данных\n";

return 1;

}

return 0;

}