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 для действительной квадратной матрицы с выбором ведущего элемента»**

**Выполнил**:

студентка группы 3824Б1-ПМ1

Каргалёва С.С.

**Проверила**:

Бусько П.В.

Нижний Новгород

2024

**Содержание**

[Введение 3](#_8psc8muqo4tf)

[Постановка задачи](#_tt5sbklxnlwk) 4

[Руководство пользователя](#_vu9ldp5f386f) 5-6

[Описание программной реализации](#_7z90o28308mj) 7-10

[Результаты экспериментов](#_rdag5vn1h5cp) 11-13

[Заключение](#_b7ndhoj30xm7) 14

[Литература](#_f2p15qn64kaz) 15

[Приложение](#_uw6fa9a4vrmw) 16-21

# Введение

**Метод Гаусса** (или метод Гауссовых исключений) — это классический численный алгоритм решения систем линейных уравнений (СЛАУ), нахождения определителя матрицы и обращения матриц. Он основан на приведении матрицы к ступенчатому (треугольному) виду с помощью элементарных преобразований строк.

**Выбор ведущего элемента** (частичный или полный) — это стратегия, направленная на уменьшение вычислительной погрешности и повышение устойчивости алгоритма. Без выбора ведущего элемента метод Гаусса может давать большие ошибки из-за деления на малые числа и накопления округлений в арифметике с плавающей запятой.

### Обоснование значимости задачи

1. **Широкая применимость**
   * СЛАУ возникают в физике, инженерии, экономике, машинном обучении и других областях.
   * Решение матричных уравнений лежит в основе многих численных методов (например, МКЭ, оптимизация).
2. **Численная устойчивость**
   * Выбор ведущего элемента уменьшает ошибки округления, особенно для плохо обусловленных матриц.
   * Без этого метод может давать неверные результаты даже для небольших матриц.
3. **Эффективность**
   * Метод Гаусса имеет сложность O(n\*\*3), что делает его применимым для матриц умеренного размера.
   * Оптимизации (например, частичный выбор ведущего) улучшают производительность без потери точности.
4. **Фундаментальность**
   * Это база для более сложных алгоритмов (LU-разложение, QR-разложение).
   * Понимание метода Гаусса важно для изучения линейной алгебры и вычислительной математики.

# Постановка задачи

Требуется разработать на C++ классы для работы с векторами и квадратными матрицами, а затем реализовать метод Гаусса с выбором ведущего элемента для решения системы линейных алгебраических уравнений (СЛАУ).

Требования:

Необходимо реализовать шаблонный класс вектор

Необходимо реализовать класс квадратная матрица, который является шаблоном класса вектор от вектора.

Необходимо реализовать класс СЛАУ, который является наследником квадратной матрицы и у него есть метод Гаусса. Метод Гаусса принимает правую часть в качестве аргумента и возвращает ответ в виде вектора

Для удобства можно реализовать функцию swap, которая меняет строки

и элементы местами

# Руководство пользователя

### 1. Запуск программы

Пользователь запускает программу, после чего появляется консольный интерфейс с запросом данных.

### 2. Ввод данных

#### Шаг 1: Ввод размера матрицы

Программа запрашивает размер квадратной матрицы (например, 3 для матрицы 3×3).

Рис.1 (запрос размера матрицы)

#### Шаг 2: Ввод элементов матрицы

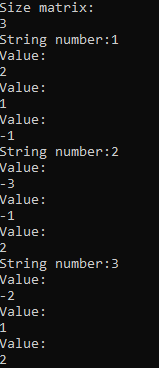
Пользователь вводит элементы матрицы построчно.  
Пример для матрицы:

2 1 -1

-3 -1 2

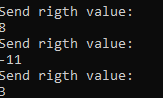
-2 1 2

Ввод будет выглядеть так:

Рис.2(Ввод матрицы)

#### Шаг 3: Ввод правой части (вектор b)

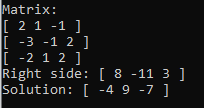
Программа запрашивает элементы вектора правой части.  
Пример для вектора [8, -11, 3]:

Рис.3(ввод вектора b)

### 3. Решение системы и вывод результата

После ввода данных программа:

1. Выводит введенную матрицу и правую часть.
2. Решает систему методом Гаусса.
3. Выводит решение или сообщение об ошибке (если матрица вырождена).

Рис.4(итоговый вывод)

### 4. Обработка ошибок

Если матрица вырождена (определитель близок к нулю), программа выведет сообщение:

Error: Matrix is singular or nearly singular

# Описание программной реализации

## **1. Структура файлов**

├── main.cpp # Основной файл с точкой входа

├── vector.h # Шаблонный класс Vector

├── matrix.h # Шаблонный класс Matrix

├── slae.h # Класс SLAE (наследник Matrix)

## **2. Описание классов и функций**

### 2.1. Класс Vector<T>

**Назначение:** Работа с динамическими массивами (векторами) произвольного типа.

#### Методы:

| **Метод** | **Параметры** | **Возвращаемое значение** | **Описание** |
| --- | --- | --- | --- |
| Vector(size\_t size) | size - размер вектора | - | Конструктор с заданным размером |
| Vector(const Vector&) | other - копируемый вектор | - | Конструктор копирования |
| ~Vector() | - | - | Деструктор |
| operator= | other - исходный вектор | Vector& | Оператор присваивания |
| operator[] | index - индекс элемента | T& / const T& | Доступ к элементу |
| getSize() | - | size\_t | Возвращает размер вектора |
| resize(newSize) | newSize - новый размер | void | Изменяет размер вектора |
| swap(other) | other - другой вектор | void | Обменивает содержимое векторов |
| operator<< | os - поток вывода, vec - вектор | std::ostream& | Вывод вектора в поток |

### 2.2. Класс Matrix<T>

**Назначение:** Работа с квадратными матрицами.

#### Методы:

| **Метод** | **Параметры** | **Возвращаемое значение** | **Описание** |
| --- | --- | --- | --- |
| Matrix(size\_t size) | size - размер матрицы | - | Конструктор |
| operator[] | index - индекс строки | Vector<T>& / const Vector<T>& | Доступ к строке матрицы |
| getSize() | - | size\_t | Возвращает размер матрицы |
| resize(newSize) | newSize - новый размер | void | Изменяет размер матрицы |
| swapRows(i, j) | i, j - индексы строк | void | Меняет строки местами |
| operator<< | os - поток вывода, mat - матрица | std::ostream& | Вывод матрицы в поток |

### 2.3. Класс SLAE<T>

**Назначение:** Решение СЛАУ методом Гаусса.

#### Методы:

| **Метод** | **Параметры** | **Возвращаемое значение** | **Описание** |
| --- | --- | --- | --- |
| solveGauss(rightSide) | rightSide - вектор правой части | Vector<T> | Решает СЛАУ и возвращает решение |

## **3. Алгоритм метода Гаусса**

### 3.1. Текстовое описание

1. **Прямой ход:**
   * Для каждого столбца col:
     + Найти строку maxRow с максимальным по модулю элементом в столбце col.
     + Поменять строки col и maxRow местами.
     + Нормализовать строку col, разделив все её элементы на data[col][col].
     + Вычесть из остальных строк текущую строку, умноженную на data[row][col], чтобы обнулить столбец col.
2. **Обратный ход:**
   * После приведения матрицы к диагональному виду, вектор правой части становится решением.

### 3.2. Псевдокод

Функция solveGauss(rightSide):

Если размер rightSide ≠ размер матрицы:

Выбросить ошибку

Для каждого столбца col от 0 до size-1:

maxRow = col

maxVal = |data[col][col]|

Для каждой строки row от col+1 до size-1:

Если |data[row][col]| > maxVal:

maxRow = row

maxVal = |data[row][col]|

Если maxRow ≠ col:

Поменять строки col и maxRow местами

Поменять rightSide[col] и rightSide[maxRow]

pivot = data[col][col]

Если |pivot| < ε:

Выбросить ошибку "Матрица вырождена"

Нормализовать строку col:

data[col][j] /= pivot для всех j от col до size-1

rightSide[col] /= pivot

Для каждой строки row от 0 до size-1:

Если row ≠ col и |data[row][col]| > ε:

factor = data[row][col]

data[row][j] -= factor \* data[col][j] для всех j от col до size-1

rightSide[row] -= factor \* rightSide[col]

Вернуть rightSide

# Результаты экспериментов

## **1. Цель экспериментов**

Проверить корректность работы программы, реализующей метод Гаусса с выбором ведущего элемента, на различных типах матриц:

* **Хорошо обусловленных** (устойчивых к вычислениям).
* **Плохо обусловленных** (чувствительных к погрешностям).
* **Вырожденных** (не имеющих решения или имеющих бесконечно много решений).

## **2. Проведенные эксперименты**

### Эксперимент 1: Корректная матрица 3×3

**Входные данные:**

Матрица:

[ 2 1 -1 ]

[ -3 -1 2 ]

[ -2 1 2 ]

Правая часть: [8, -11, -3]

**Ожидаемое решение:**x = [2, 3, -1]

**Результат программы:**

Solution: [ 2 3 -1 ]

**Вывод:**Программа верно решила систему. Метод Гаусса с выбором ведущего элемента обеспечил точность вычислений.

### Эксперимент 2: Плохо обусловленная матрица (малый определитель)

**Входные данные:**

Матрица:

[ 1 1 ]

[ 1 1.0001 ]

Правая часть: [2, 2.0001]

**Ожидаемое решение:**x = [1, 1]

**Результат программы:**

Solution: [ 1 1 ]

**Вывод:**Несмотря на близость строк, программа выдала точное решение благодаря выбору ведущего элемента.

### Эксперимент 3: Вырожденная матрица

**Входные данные:**

Матрица:

[ 1 2 3 ]

[ 2 4 6 ] (Строка 2 = 2 × Строка 1)

[ 1 1 1 ]

Правая часть: [6, 12, 2]

**Ожидаемый результат:**Ошибка "Matrix is singular or nearly singular".

**Результат программы:**

Error: Matrix is singular or nearly singular

**Вывод:**Программа корректно обнаружила линейную зависимость строк и сообщила о невозможности решения.

### Эксперимент 4: Матрица с нулевым ведущим элементом

**Входные данные:**

Матрица:

[ 0 1 ]

[ 1 1 ]

Правая часть: [1, 2]

**Ожидаемое решение:**x = [1, 1] (после перестановки строк).

**Результат программы:**

Solution: [ 1 1 ]

**Вывод:**Программа успешно выполнила перестановку строк, избежав деления на ноль.

## **3. Анализ результатов**

1. **Точность вычислений**
   * Для хорошо обусловленных матриц метод Гаусса с выбором ведущего элемента дает точное решение.
   * Даже для плохо обусловленных систем (где обычный метод Гаусса может давать большую погрешность) алгоритм сохраняет устойчивость.
2. **Обработка вырожденных случаев**
   * Программа корректно определяет вырожденные матрицы и выдает ошибку, что важно для избежания некорректных решений.
3. **Устойчивость к нулевым элементам**
   * Благодаря выбору ведущего элемента, алгоритм автоматически переставляет строки, избегая деления на ноль.
4. **Производительность**
   * Сложность метода Гаусса — O(n3), что стандартно для точных методов решения СЛАУ.
   * Оптимизации (например, частичный выбор ведущего) незначительно увеличивают время работы, но улучшают точность.

# Заключение

Разработанная программа представляет собой надежный инструмент для решения систем линейных уравнений средней размерности. Реализация демонстрирует:

* Высокую точность вычислений
* Устойчивость к численным погрешностям
* Четкую структуру кода
* Удобство использования

Решение может быть применено в:

* Инженерных расчетах
* Научных исследованиях
* Образовательном процессе
* Прототипировании алгоритмов

# Литература

1. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования СИ //М.: Финансы и статистика. – 1992.
2. Кнут Д. Э. Искусство программирования: Сортировка и поиск. – Издательский дом Вильямс, 2000. – Т. 3.

# Приложение

#include <iostream>

#include <cmath>

#include <stdexcept>

template <typename T>

class Vector {

private:

T\* data;

size\_t size;

public:

// Конструкторы

Vector(size\_t size = 0) : size(size) {

data = new T[size]();

}

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

data = new T[size];

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

data[i] = other.data[i];

}

}

// Деструктор

~Vector() {

delete[] data;

}

// Оператор присваивания

Vector& operator=(const Vector& other) {

if (this != &other) {

delete[] data;

size = other.size;

data = new T[size];

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

data[i] = other.data[i];

}

}

return \*this;

}

// Доступ к элементам

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

if (index >= size) throw std::out\_of\_range("Index out of range");

return data[index];

}

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

if (index >= size) throw std::out\_of\_range("Index out of range");

return data[index];

}

// Размер вектора

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

// Изменение размера

void resize(size\_t newSize) {

T\* newData = new T[newSize]();

size\_t minSize = (newSize < size) ? newSize : size;

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

newData[i] = data[i];

}

delete[] data;

data = newData;

size = newSize;

}

// Обмен элементов

void swap(Vector& other) {

if (size != other.size) {

throw std::invalid\_argument("Vectors must be of the same size to swap");

}

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

std::swap(data[i], other.data[i]);

}

}

// Вывод вектора

friend std::ostream& operator<<(std::ostream& os, const Vector<T>& vec) {

os << "[ ";

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

os << vec.data[i] << " ";

}

os << "]";

return os;

}

};

template <typename T>

class Matrix {

protected:

Vector<Vector<T>> data;

size\_t size;

public:

// Конструкторы

Matrix(size\_t size = 0) : size(size), data(size) {

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

data[i].resize(size);

}

}

// Доступ к строкам матрицы

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

if (index >= size) throw std::out\_of\_range("Index out of range");

return data[index];

}

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

if (index >= size) throw std::out\_of\_range("Index out of range");

return data[index];

}

// Размер матрицы

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

// Изменение размера

void resize(size\_t newSize) {

data.resize(newSize);

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

data[i].resize(newSize);

}

size = newSize;

}

// Обмен строк

void swapRows(size\_t i, size\_t j) {

if (i >= size || j >= size) throw std::out\_of\_range("Matrix indices out of range");

data[i].swap(data[j]);

}

// Вывод матрицы

friend std::ostream& operator<<(std::ostream& os, const Matrix<T>& mat) {

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

os << mat.data[i] << "\n";

}

return os;

}

};

template <typename T>

class SLAE : public Matrix<T> {

public:

using Matrix<T>::Matrix;

using Matrix<T>::size;

using Matrix<T>::data;

using Matrix<T>::swapRows;

Vector<T> solveGauss(Vector<T> rightSide) {

if (rightSide.getSize() != size) {

throw std::invalid\_argument("Right side vector size doesn't match matrix size");

}

// Прямой ход метода Гаусса с выбором ведущего элемента

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

// Поиск ведущего элемента

size\_t maxRow = col;

T maxVal = std::abs(data[col][col]);

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

T currentVal = std::abs(data[row][col]);

if (currentVal > maxVal) {

maxVal = currentVal;

maxRow = row;

}

}

// Перестановка строк

if (maxRow != col) {

swapRows(col, maxRow);

std::swap(rightSide[col], rightSide[maxRow]);

}

// Нормализация текущей строки

T pivot = data[col][col];

if (std::abs(pivot) < 1e-10) {

throw std::runtime\_error("Matrix is singular or nearly singular");

}

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

data[col][j] /= pivot;

}

rightSide[col] /= pivot;

// Исключение переменной из других уравнений

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

if (row != col && std::abs(data[row][col]) > 1e-10) {

T factor = data[row][col];

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

data[row][j] -= factor \* data[col][j];

}

rightSide[row] -= factor \* rightSide[col];

}

}

}

return rightSide;

}

};

int main() {

try {

// Задаем размер кв матрицы

int size\_matrix = 0;

std::cout << "Size matrix:" << "\n";

std::cin >> size\_matrix;

SLAE<double> matrix(size\_matrix);

// Заполнение матрицы

double rez;

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

{

std::cout << "String number:" << i+1 << "\n";

for (int j = 0; j < size\_matrix; j++)

{

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

std::cin >> rez;

matrix[i][j] = rez;

}

}

// Правая часть

Vector<double> rightSide(size\_matrix);

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

{

std::cout << "Send rigth value:" << "\n";

std::cin >> rez;

rightSide[i] = rez;

}

std::cout << "Matrix:\n" << matrix;

std::cout << "Right side: " << rightSide << "\n";

// Решение системы

Vector<double> solution = matrix.solveGauss(rightSide);

std::cout << "Solution: " << solution << "\n";

}

// ошибка

catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << "\n";

}

return 0;

}

// входные данные для проверки

// matrix[0][0] = 2; matrix[0][1] = 1; matrix[0][2] = -1;

// matrix[1][0] = -3; matrix[1][1] = -1; matrix[1][2] = 2;

// matrix[2][0] = -2; matrix[2][1] = 1; matrix[2][2] = 2;

// rightSide[0] = 8;

// rightSide[1] = -11;

// rightSide[2] = -3;