

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
«Решение СЛУ методом Гаусса»

Выполнил:

студент группы 3824Б1ПМ1

Лазарев Н.П.

Проверила:

Бусько П.В.

Нижний Новгород
2025

Содержание

Введение	3
Постановка задачи	4
Руководство пользователя	5
Описание программной реализации	6
Результаты экспериментов	7
Заключение	8
Литература	9
Приложение	10

Введение

Решение систем линейных алгебраических уравнений (СЛАУ) является ключевой задачей в области вычислительной математики и находит широкое применение в науке и инженерии. Среди численных методов особое место занимает метод Гаусса с выбором ведущего элемента по столбцу, который обеспечивает высокую точность и устойчивость вычислений. В данной работе представлена программная реализация этого алгоритма на языке C++. Проект разработан с использованием объектно-ориентированных принципов и шаблонного программирования, что обеспечивает его универсальность и модульность.

Постановка задачи

Основной целью данной работы является создание программного продукта на C++, способного решать СЛАУ с квадратной матрицей при помощи метода Гаусса. Для выполнения этой цели были определены следующие ключевые задачи:

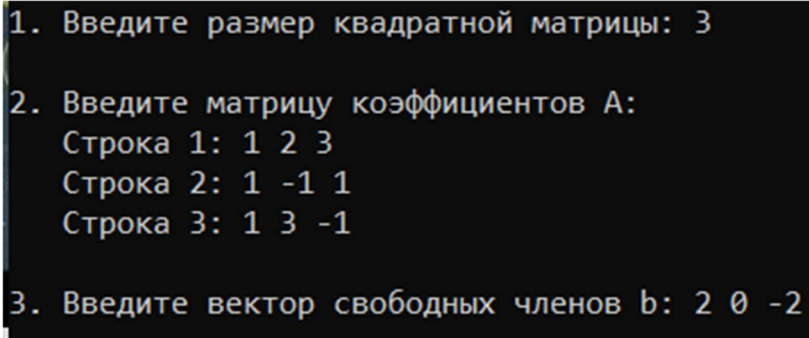
1. Разработка шаблонного класса **vector<T>**: Создать шаблонный класс для представления одномерного массива, который расширяет функциональность стандартного **std::vector<T>** и предоставляет удобный вывод в консоль.
2. Проектирование класса **SquareMatrix<T>**: Реализовать базовый шаблонный класс для квадратной матрицы, инкапсулирующий двумерную структуру данных и предоставляющий основные операции, такие как доступ к строкам и их обмен.
3. Создание класса **SLAE<T>**: Разработать класс для представления системы линейных алгебраических уравнений, который наследует свойства и методы класса **SquareMatrix<T>**.
4. Реализация метода **gaussMethod**: В классе **SLAE<T>** имплементировать основной вычислительный метод, реализующий алгоритм Гаусса с выбором главного элемента. Метод должен принимать в качестве аргумента вектор свободных членов и возвращать вектор-решение.
5. Обеспечение отказоустойчивости: Предусмотреть обработку исключительных ситуаций, в частности, при работе с вырожденными матрицами, для которых решение не существует или не является единственным.

Руководство пользователя

Запуск программы:

Программа ожидает от пользователя ввода следующих данных через консоль:

1. Размер квадратной матрицы
2. Матрицу коэффициентов системы уравнений
3. Вектор свободных членов (правых частей уравнений)



```
1. Введите размер квадратной матрицы: 3
2. Введите матрицу коэффициентов A:
  Строка 1: 1 2 3
  Строка 2: 1 -1 1
  Строка 3: 1 3 -1
3. Введите вектор свободных членов b: 2 0 -2
```

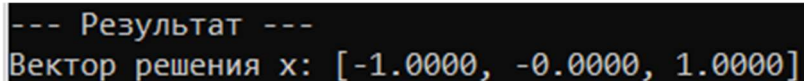
Рис. 1. Пример входных данных

Выполнение:

Программа автоматически выполняет прямой ход метода Гаусса и обратную подстановку.

Результат:

Вектор решений выводится в консоль:



```
--- Результат ---
Вектор решения x: [-1.0000, -0.0000, 1.0000]
```

Рис. 2. Пример вывода решений

Описание программной реализации

Структура проекта:

Проект состоит из одного файла **main.cpp**, в котором реализована вся логика. Структура программы включает в себя три ключевых шаблонных класса.

Ключевые классы:

1. **Vector<T>**

- Шаблонный класс, наследуемый от **std::vector<T>**. Он полностью заимствует функциональность стандартного вектора и дополняет его перегруженным оператором вывода **<<** для удобного отображения содержимого в консоли.

2. **SquareMatrix<T>**

- Шаблонный класс, служащий основой для представления квадратной матрицы, что инкапсулирует объект **Vector<Vector<T>>**. Класс предоставляет оператор **[]** для доступа к строкам и метод **swapRows()** для их перестановки, что является ключевой операцией в методе Гаусса.

3. **SLAE<T>**

- Класс, моделирующий систему линейных алгебраических уравнений. Он является наследником класса **SquareMatrix<T>**.
- Основной метод этого класса — **gaussMethod(Vector<T>& b)**. Он реализует полный цикл решения СЛАУ:
 - Создание копий: Чтобы не изменять исходные данные, метод работает с копиями матрицы и вектора свободных членов.
 - Прямой ход: Итеративно приводит матрицу к верхнетреугольному виду. На каждом шаге выполняется поиск максимального по модулю элемента в текущем столбце (выбор ведущего элемента) и перестановка строк для повышения численной устойчивости.
 - Проверка на вырожденность: Перед каждым шагом деления выполняется проверка, не является ли диагональный элемент близким к нулю. Если это так, генерируется исключение **std::runtime_error**.
 - Обратный ход: После приведения матрицы к треугольному виду вычисляет компоненты вектора решения, двигаясь от последнего уравнения к первому.

Результаты экспериментов

Тестовая система уравнений:

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 2 \\ x_1 - x_2 + x_3 = 0 \\ x_1 + 3x_2 - x_3 = -2 \end{cases}$$

Результаты: $x_1 = -1, x_2 = 0, x_3 = 1$

Решение совпадает с аналитическим.

Заключение

В рамках выполненной работы был успешно разработан и протестирована программа для решения систем линейных алгебраических уравнений методом Гаусса с выбором ведущего элемента. Применение объектно-ориентированного подхода позволило создать логически структурированный и легко читаемый код. Использование шаблонов C++ делает программу универсальной, позволяя работать с различными числовыми типами данных (например, *float*, *double*). Тестирование на конкретном примере показало точность и корректность вычислений, подтвердив надежность реализованного алгоритма.

Литература

1. Кнут Д. Э. Искусство программирования – Издательский дом Вильямс, 2000. – Т. 3.
2. Керниган Б., Ритчи Д. Язык программирования СИ //М.: Финансы и статистика. – 1992.
3. Седжвик Р., Уэйн К. Алгоритмы на C++: краткий курс. 2013.
4. Лафоре Р. Объектно-ориентированное программирование в C++. — М.: Вильямс, 2009. — 864 с.
5. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. — М.: Вильямс, 2018.

Приложение

main.cpp

```
template <typename T>
class Vector : public std::vector<T> {
public:
    using std::vector<T>::vector;

    friend std::ostream& operator<<(std::ostream& os, const Vector<T>& vec) {
        os << "[";
        for (size_t i = 0; i < vec.size(); ++i) {
            os << std::fixed << std::setprecision(4) << vec[i];
            if (i < vec.size() - 1) {
                os << ", ";
            }
        }
        os << "]";
        return os;
    }
};

template <typename T>
class SquareMatrix {
protected:
    Vector<Vector<T>> matrix;
    size_t size;

public:
    SquareMatrix() : size(0) {}

    explicit SquareMatrix(size_t n) : size(n) {
        matrix.resize(n, Vector<T>(n, 0));
    }

    size_t getSize() const {
        return size;
    }

    Vector<T>& operator[](size_t index) {
        if (index >= size) throw std::out_of_range("Индекс строки матрицы вне
диапазона");
        return matrix[index];
    }

    const Vector<T>& operator[](size_t index) const {
        if (index >= size) throw std::out_of_range("Индекс строки матрицы вне
диапазона");
        return matrix[index];
    }

    void swapRows(size_t row1, size_t row2) {
        if (row1 < size && row2 < size) {
            std::swap(matrix[row1], matrix[row2]);
        }
    }

    friend std::ostream& operator<<(std::ostream& os, const SquareMatrix<T>& mat) {
        for (size_t i = 0; i < mat.getSize(); ++i) {
            for (size_t j = 0; j < mat.getSize(); ++j) {
                os << std::setw(10) << std::fixed << std::setprecision(4) <<
mat[i][j];
            }
            os << std::endl;
        }
    }
};
```

```

        return os;
    }
};

template <typename T>
class SLAE : public SquareMatrix<T> {
public:
    using SquareMatrix<T>::SquareMatrix;

    Vector<T> gaussMethod(Vector<T>& b) {
        if (this->size != b.size()) {
            throw std::invalid_argument("Размер вектора правой части не совпадает с
размером матрицы.");
        }

        SquareMatrix<T> temp_matrix = *this;
        Vector<T> temp_b = b;
        size_t n = this->size;

        for (size_t i = 0; i < n; ++i) {
            size_t pivot_row = i;
            for (size_t k = i + 1; k < n; ++k) {
                if (std::abs(temp_matrix[k][i]) >
std::abs(temp_matrix[pivot_row][i])) {
                    pivot_row = k;
                }
            }

            temp_matrix.swapRows(i, pivot_row);
            std::swap(temp_b[i], temp_b[pivot_row]);

            if (std::abs(temp_matrix[i][i]) < 1e-12) {
                throw std::runtime_error("Матрица вырождена или близка к
вырожденной. Решений нет или их бесконечно много.");
            }

            for (size_t k = i + 1; k < n; ++k) {
                T factor = temp_matrix[k][i] / temp_matrix[i][i];
                for (size_t j = i; j < n; ++j) {
                    temp_matrix[k][j] -= factor * temp_matrix[i][j];
                }
                temp_b[k] -= factor * temp_b[i];
            }
        }

        Vector<T> solution(n);
        for (int i = n - 1; i >= 0; --i) {
            T sum = 0;
            for (size_t j = i + 1; j < n; ++j) {
                sum += temp_matrix[i][j] * solution[j];
            }
            solution[i] = (temp_b[i] - sum) / temp_matrix[i][i];
        }

        return solution;
    }
};

void read_value(double& value) {
    while (!(std::cin >> value)) {
        std::cout << " Ошибка: Введите корректное число. Попробуйте еще раз: ";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
}

```