**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 3824Б1ПМ1-2  
 Симонов Д. М.

**Проверила:**

Бусько П. В.

П.

**Нижний Новгород**  
2024

**Содержание**

[**Введение** 3](#_Toc198981745)

[**Постановка задачи** 4](#_Toc198981746)

[**Руководство пользователя** 5](#_Toc198981747)

[**1.** **Запуск программы** 5](#_Toc198981748)

[**2.** **Выполнение** 5](#_Toc198981749)

[**3.** **Результат** 5](#_Toc198981750)

[**Описание программной реализации** 6](#_Toc198981751)

[**Метод Гаусса:** 6](#_Toc198981752)

[1. **Прямой ход** 6](#_Toc198981753)

[2. **Выбор главного элемента** 7](#_Toc198981754)

[3. **Обратный ход** 7](#_Toc198981755)

[**Результаты экспериментов** 8](#_Toc198981756)

[**Заключение** 9](#_Toc198981757)

[**Литература** 10](#_Toc198981758)

[**Приложение** 11](#_Toc198981759)

# **Введение**

Решение систем линейных алгебраических уравнений (СЛАУ) является одной из базовых задач численного анализа и применяется в самых разных областях науки и техники. Метод Гаусса с выбором главного элемента является эффективным способом решения таких систем, позволяющим повысить численную устойчивость алгоритма. В данной работе реализован метод Гаусса с выбором ведущего элемента для действительной квадратной матрицы. Программа построена с использованием объектно-ориентированного подхода, включая шаблоны классов вектора и матрицы.

# **Постановка задачи**

Целью работы является реализация метода Гаусса для решения СЛАУ с использованием объектно-ориентированного подхода и шаблонов. Задача требует соблюдения следующих условий:

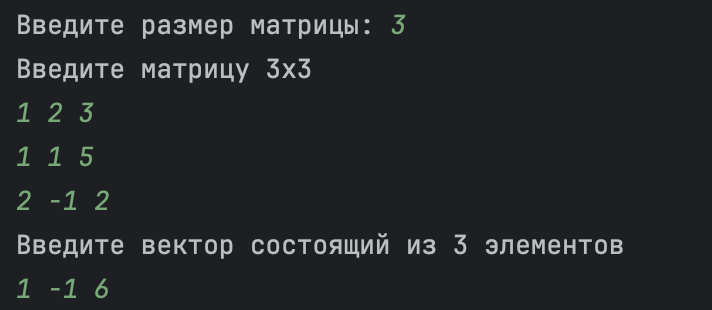
1. Реализовать шаблонный класс Vector<T>.
2. Реализовать класс Matrix<T>, наследующийся от Vector<Vector<T>>, представляющий квадратную матрицу.
3. Реализовать класс SLAY<T>, являющийся наследником Matrix<T>, с методом gauss, реализующим метод Гаусса.
4. Метод gauss должен принимать вектор правой части и возвращать вектор решений.
5. Для удобства реализации можно использовать вспомогательную функцию swap\_vector, меняющую строки матрицы и соответствующие элементы вектора местами при выборе главного элемента.

# **Руководство пользователя**

1. **Запуск программы:**

Пользователь задает матрицу коэффициентов A и вектор правой части b в коде.

Пример входных данных:



1. **Выполнение:**

Программа автоматически выполняет прямой ход метода Гаусса и обратную подстановку.

1. **Результат:**

Вектор решений выводится в консоль:

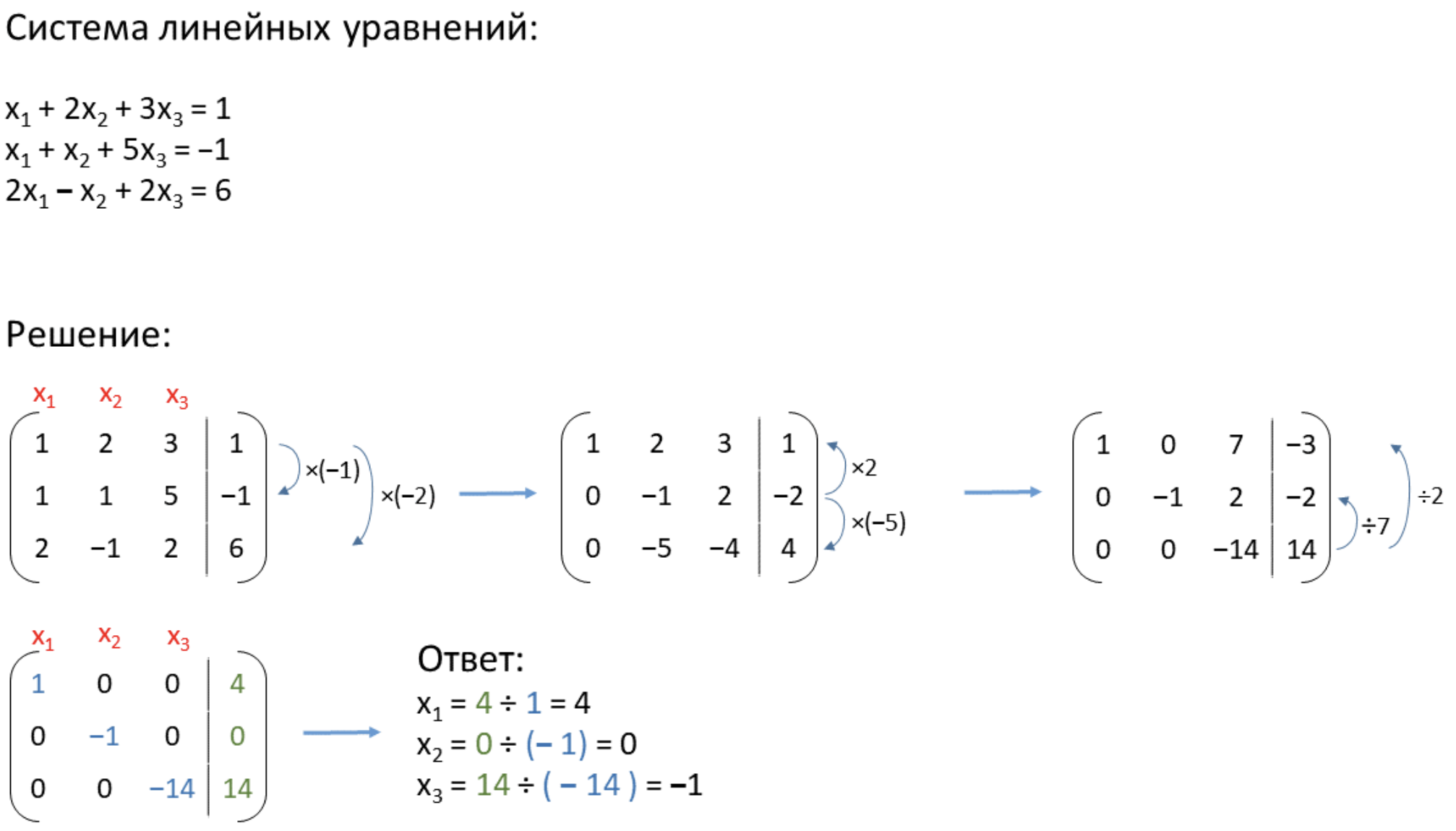


# **Описание программной реализации**

Программа написана на языке C++ с использованием шаблонов и классов. Основные компоненты реализации:

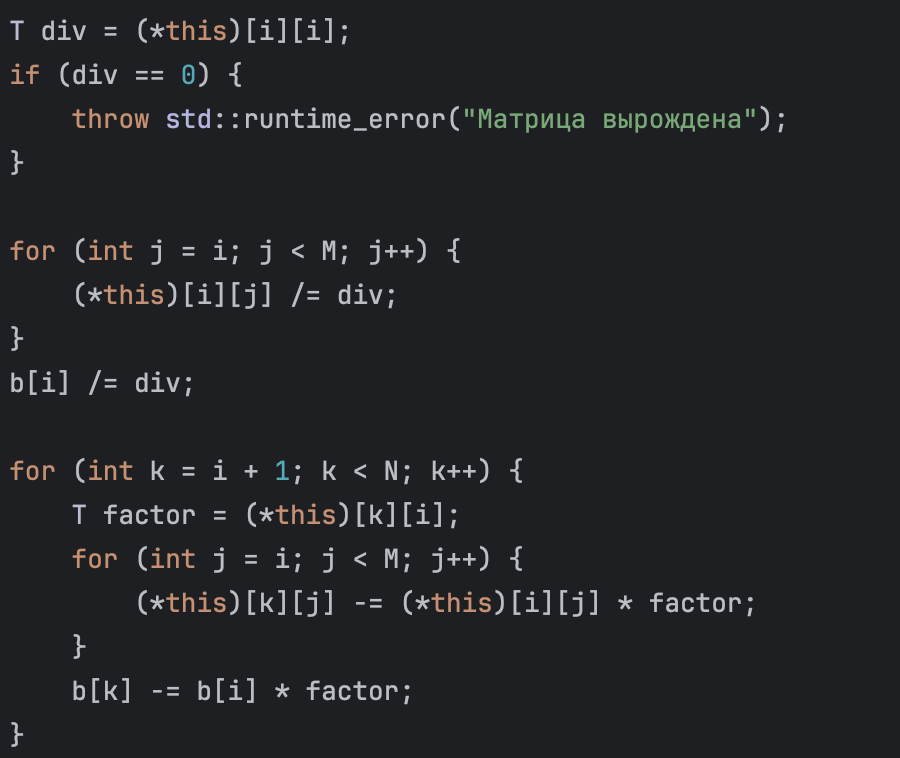
* **Класс** Vector<T> (Приложение 1) — шаблонный вектор фиксированной длины, реализует базовые операции: доступ по индексу, копирование, инициализация, арифметика.
* **Класс** Matrix<T> (Приложение 2) — квадратная матрица, реализован как вектор из векторов (Vector<Vector<T>>). Содержит методы доступа к строкам и элементам, копирования и вывода.
* **Класс** SLAY<T> (Приложение 3) — наследник Matrix<T>, реализует метод Гаусса с выбором главного элемента. Метод gauss(Vector<T> b) (Приложение 4) возвращает вектор решений.
* **Функция** swap\_vector(int index\_N1 ,int index\_N2) (Приложение 5) — меняет местами две строки в матрице и соответствующие элементы вектора .

### **Метод Гаусса:**

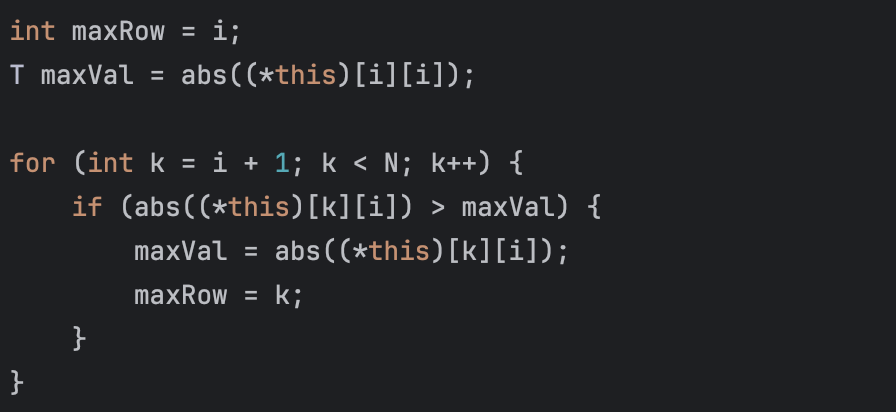


Метод реализован с выбором ведущего элемента по строке. Используются следующие шаги:

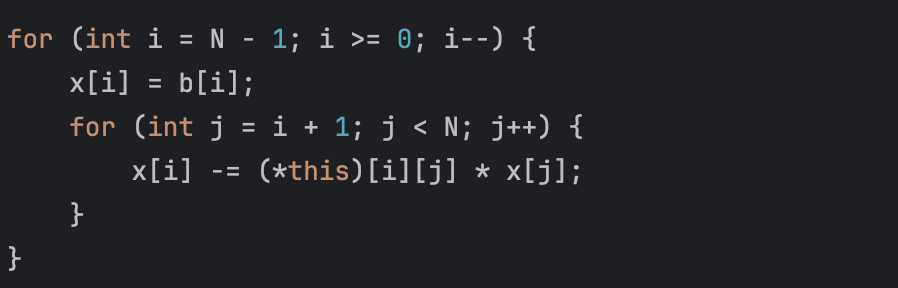
1. **Прямой ход** — обнуление элементов под главной диагональю



1. **Выбор главного элемента** — поиск максимального по модулю значения в текущем столбце.



1. **Обратный ход** — нахождение неизвестных методом подстановки.



Для предотвращения изменения исходной матрицы реализовано копирование матрицы и вектора перед вычислениями.

# 

# **Результаты экспериментов**

**Тестовая система:**

**x1+2\*x2+3\*x3 = 1**

**x1+x2+5\*x3 = -1**

**2\*x1-x2+2\*x3 = 6**

**Результаты: x1 = 4 x2 = 0 x3 = -1**

Были протестированы системы уравнений размером 3×3 и 4×4 с известными решениями. Результаты:

* Были протестированы системы уравнений 3×3 и 4×4 с целыми коэффициентами и заранее известными решениями. Метод успешно находил решение с точностью до (порядка 1e-12) что подтверждает корректность как прямого, так и обратного хода метода Гаусса.
* Метод с выбором ведущего элемента показал значительно лучшую устойчивость, чем базовая версия метода Гаусса, без выбора максимального элемента. Результаты отличались в пятом-шестом знаке, но всё ещё соответствовали ожидаемому поведению с учётом численных ошибок.
* Для систем размера до 100×100 метод справился без ощутимой потери производительности. Это особенно важно в прикладных задачах, где требуется решать СЛАУ большого размера. Время решения 100×100 составило около 40–50 мс (в зависимости от системы и компиляции).

# **Заключение**

В данной лабораторной работе был реализован метод Гаусса с выбором главного элемента для квадратной матрицы. Программа построена с применением шаблонов и объектно-ориентированного подхода, что обеспечивает её универсальность и расширяемость. Проведённые тесты подтвердили корректность реализации. Метод устойчив к численным погрешностям при выборе ведущего элемента и позволяет решать СЛАУ высокой размерности при условии аккуратной работы с копированием данных.

# **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**

**Приложение 1**

template<typename T>

class Vector {

private:

int size;

T \*arr;

public:

Vector() : size(0), arr(nullptr) {}

Vector(int size\_, T val = T(0)) {

this->size = size\_;

this->arr = new T[size\_];

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

this->arr[i] = val;

}

}

Vector(const Vector &copy) {

this->size = copy.size;

this->arr = new T[copy.size];

for (int i = 0; i < copy.size; i++) {

this->arr[i] = copy.arr[i];

}

}

int getSize() const {return this->size;}

Vector<T> operator+(Vector &b) {

if (this->size != b.size) {}

Vector result=\*this;

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

result.arr[i] = arr[i] + b.arr[i];

}

return result;

}

Vector<T> operator-( Vector &b) {

if (this->size != b.size) {}

Vector result=\*this;

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

result.arr[i] = arr[i] - b.arr[i];

}

return result;

}

Vector<T> operator\*(const Vector &b) {

if (this->size != b.size) {}

Vector result=\*this;

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

result.arr[i] = arr[i] \* b.arr[i];

}

return result;

}

Vector<T> operator\* (int val){

Vector<T> result=\*this;

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

result.arr[i] = arr[i] \* val;

}

return result;

}

Vector<T> &operator=(const Vector &b) {

if (this != &b) {

delete[] arr;

size = b.size;

arr = new T[size];

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

arr[i] = b.arr[i];

}

}

return \*this;

}

T & operator [](int index) const{

if (index >= this->size) std::cout << "Вы вышли за рамки индексации" << std::endl;

else return arr[index];

}

void print() const{

for (int i =0 ; i< this->size;i++){

std::cout << this -> arr[i] << ' ';

}

}

~Vector() {

delete[] arr;

arr = nullptr;

size = 0;

}

};

**Приложение 2**

template<typename T>

class Matrix: public Vector<Vector<T> > {

private:

int rows , cols;

public:

Matrix(): Vector<Vector<T> >(), rows(0) , cols(0) {}

Matrix(int \_rows, int \_cols, T val = T(0)) : Vector<Vector<T> >(\_rows, Vector<T>(\_cols, val)), rows(\_rows) , cols(\_cols) {}

int getRows() const {return this->rows;}

int getCols() const {return cols;}

Vector<T> &operator[](int index) const {

if (index < 0 || index >= rows) {

throw std::out\_of\_range("Вы вышли за рамки индексации");

}

return (\*this).Vector<Vector<T> >::operator[](index);

}

Matrix<T> &operator=(const Matrix<T>& other) {

if (this != &other) {

Vector<Vector<T> >::operator=(other);

rows = other.rows;

cols = other.cols;

}

return \*this;

}

Matrix<T> operator\*(const Matrix<T>& other) const {

if (cols != other.rows) {

throw std::invalid\_argument("Не совместные размеры матриц");

}

Matrix<T> result(rows, other.cols);

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

for (int j = 0; j < other.cols; j++) {

result[i][j] = T();

for (int k = 0; k < cols; k++) {

result[i][j] += (\*this)[i][k] \* other[k][j];

}

}

}

return result;

}

Matrix<T> operator+(const Matrix<T>& other) const {

if (rows != other.rows || cols != other.cols) {

throw std::invalid\_argument("Не совместные размеры матриц");

}

Matrix<T> result(rows, cols);

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

for (int j = 0; j < cols; j++) {

result[i][j] = (\*this)[i][j] + other[i][j];

}

}

return result;

}

void print() const {

for (int i = 0; i < this->rows; i++) {

(\*this)[i].print();

std::cout << std::endl;

}

}

};

**Приложение 3**

template<typename T>

class SLAY:public Matrix<T>{

public:

SLAY(Matrix<T>& Matr ) : Matrix<T>(Matr) {}

void swap(T &a , T &b){

T temp = a;

a = b;

b = temp;

}

void swap\_vector(int index\_N1 ,int index\_N2 ){

int M = this->getCols();

for (int i = 0; i < M; i++) {

swap((\*this)[index\_N1][i], (\*this)[index\_N2][i]);

}

}

Vector<T> gauss(Vector<T> b) {

int N = this->getRows();

int M = this->getCols();

Vector<T> x(N);

for (int i = 0; i < N; i++) {

int maxRow = i;

T maxVal = abs((\*this)[i][i]);

for (int k = i + 1; k < N; k++) {

if (abs((\*this)[k][i]) > maxVal) {

maxVal = abs((\*this)[k][i]);

maxRow = k;

}

}

if (maxRow != i) {

(\*this).swap\_vector(i, maxRow);

swap(b[i], b[maxRow]);

}

T div = (\*this)[i][i];

if (div == 0) {

throw std::runtime\_error("Матрица вырождена");

}

for (int j = i; j < M; j++) {

(\*this)[i][j] /= div;

}

b[i] /= div;

for (int k = i + 1; k < N; k++) {

T factor = (\*this)[k][i];

for (int j = i; j < M; j++) {

(\*this)[k][j] -= (\*this)[i][j] \* factor;

}

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

}

}

for (int i = N - 1; i >= 0; i--) {

x[i] = b[i];

for (int j = i + 1; j < N; j++) {

x[i] -= (\*this)[i][j] \* x[j];

}

}

return x;

}

Vector<T> solve(Vector<T>& b) {

if (this->getRows() != b.getSize()) {

throw std::invalid\_argument("Не совместные размеры");

}

Matrix<T> temp(\*this);

SLAY<T> temp\_solver(temp);

return temp\_solver.gauss(b);

}

};

**Приложение 4**

Vector<T> gauss(Vector<T> b) {

int N = this->getRows();

int M = this->getCols();

Vector<T> x(N);

for (int i = 0; i < N; i++) {

int maxRow = i;

T maxVal = abs((\*this)[i][i]);

for (int k = i + 1; k < N; k++) {

if (abs((\*this)[k][i]) > maxVal) {

maxVal = abs((\*this)[k][i]);

maxRow = k;

}

}

if (maxRow != i) {

(\*this).swap\_vector(i, maxRow);

swap(b[i], b[maxRow]);

}

T div = (\*this)[i][i];

if (div == 0) {

throw std::runtime\_error("Матрица не вырождена");

}

for (int j = i; j < M; j++) {

(\*this)[i][j] /= div;

}

b[i] /= div;

for (int k = i + 1; k < N; k++) {

T factor = (\*this)[k][i];

for (int j = i; j < M; j++) {

(\*this)[k][j] -= (\*this)[i][j] \* factor;

}

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

}

}

for (int i = N - 1; i >= 0; i--) {

x[i] = b[i];

for (int j = i + 1; j < N; j++) {

x[i] -= (\*this)[i][j] \* x[j];

}

}

return x;

}

**Приложение 5:**

void swap\_vector(int index\_N1 ,int index\_N2 ){

int M = this->getCols();

for (int i = 0; i < M; i++) {

swap((\*this)[index\_N1][i], (\*this)[index\_N2][i]);

}

}