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 группы 381903-2

Киселев Д. Д.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](#_Toc41409418)

[Метод решения 4](#_Toc41409419)

[Руководство пользователя 5](#_Toc41409420)

[Описание программной реализации 7](#_Toc41409421)

[Подтверждение корректности 9](#_Toc41409422)

[Заключение 10](#_Toc41409423)

[Приложение 11](#_Toc41409424)

# Постановка задачи

Необходимо реализовать метод Гаусса для действительной квадратной матрицы с выбором ведущего элемента

Необходимо реализовать шаблонный класс вектор

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

Необходимо реализовать класс СЛАУ, который является наследником квадратной матрицы и у него есть метод Гаусса.

Метод Гаусса — классический метод решения системы линейных алгебраических уравнений (СЛАУ). Это метод последовательного исключения переменных, когда с помощью элементарных преобразований система уравнений приводится к равносильной системе треугольного вида, из которой последовательно, начиная с последних (по номеру), находятся все переменные системы.

# Метод решения

* Реализован шаблонный класс Vector, хранящий в себе массив заданного размера.

Для данного класса написан конструктор по умолчанию, задающий вектору размер – 2, значения – 0.

Написан конструктор копирования, перегружен оператор =.

Реализован метод v\_print, выводящий значения вектора в консоль.

* Реализован класс Matrix, являющийся шаблоном класса Vector от Vector.

Для данного класса реализован метод m\_print, выводящий в консоль квадратную матрицу.

* Реализован класс SLAE, являющийся наследником класса Matrix.

При создании элемента класса SLAE вызывается конструктор для класса Matrix, создающий матрицу заданного размера.

Реализован метод fill, позволяющий ввести значения матрицы системы и вектора правой части.

Реализован метод s\_print, выводящий в консоль СЛАУ.

Реализован метод swap, меняющий строки системы местами.

Реализован метод gauss – метод Гаусса. Данный метод приводит СЛАУ к треугольному виду. Если в процессе вычислений в системе появляются малые по модулю значения, то такие значения заменяются на 0 (реализована возможность указать, начиная с какого значения число считается равным нулю). Реализована проверка решений (программа не выполнится, если решений нет, если решение неоднозначно, либо если в процессе приведения матрицы к треугольному виду происходит деление на 0). Найденные решения данный метод возвращает в виде элемента класса Vector.

* Реализована шаблонная функция abs, возвращающая модуль заданного типа данных.

# Руководство пользователя

Запустить программу, в консоли ввести размерность матрицы системы:

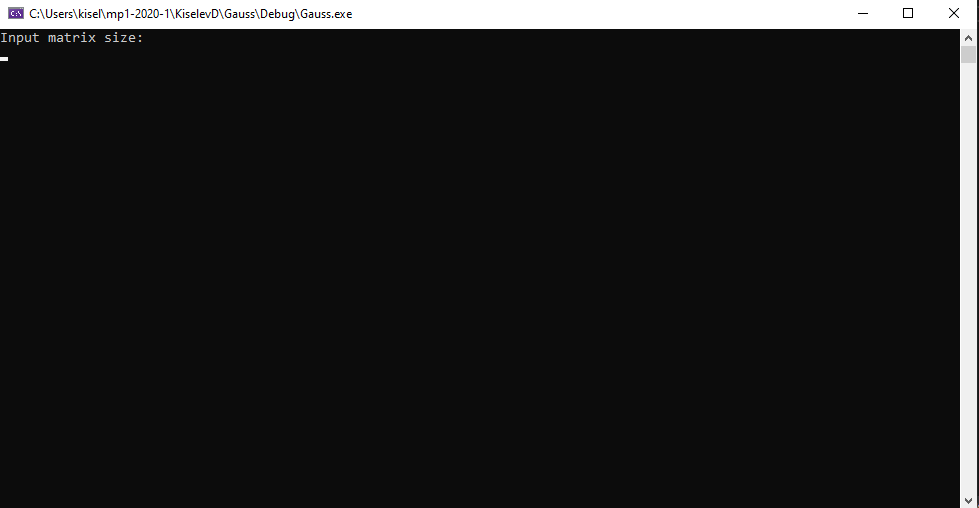


Рисунок 1. Ввод размерности.

Далее требуется построчно ввести элементы матрицы системы:

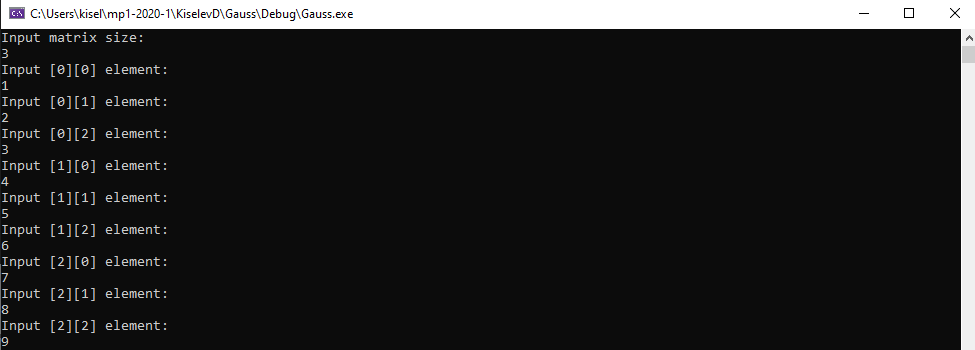


Рисунок 2. Ввод значений матрицы системы.

Далее требуется ввести значения вектора правой части:

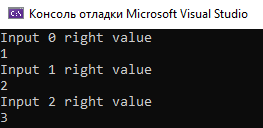


Рисунок 3. Ввод значений вектора правой части.

После ввода все данных, в консоль выводится: введенная матрица; матрица, приведенная к треугольному виду; в случае, если есть решения – вектор решений. Если решений нет – сообщение об ошибке:

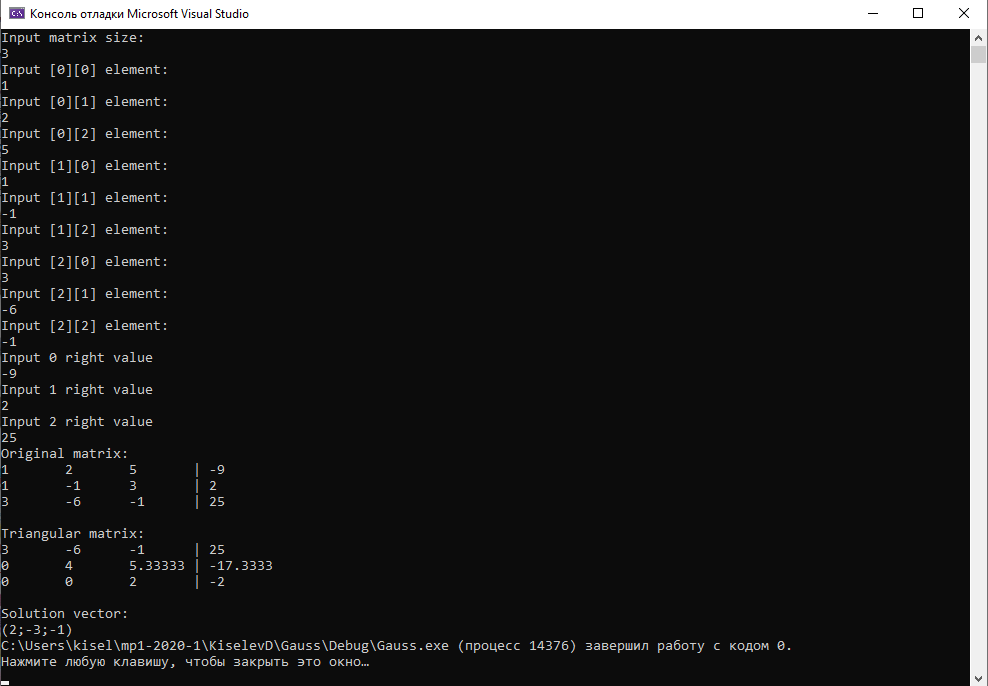


Рисунок 4. Результат выполнения программы.

# Описание программной реализации

Исходные файлы:

* Main.cpp

Файл с подключением заголовочных файлов, инициализацией переменных, вызовом метода Гаусса.

Файлы заголовков:

* vector.h

Реализация класса Vector и его методов:

template <typename T>

class Vector {

public:

Vector() {}

Vector(int \_size, T \_value = T()) {…}

Vector(int \_size, T \* values) {…}

Vector(const Vector& other) {…}

~Vector(){…}

void v\_print(){…}

T& operator[](int i) {…}

Vector& operator=(const Vector& other) {…}

protected:

T\* m\_x;

int size;

};

* matrix.h

Реализация класса Matrix и его методов:

template < typename T >

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

public:

Matrix(int n) : Vector < Vector < T > >(n, Vector < T >(n)) {}

void m\_print() {…}

private:

};

* slae.h

Реализация класса SLAE и его методов:

template < typename T >

class SLAE : public Matrix<T>

{

public:

SLAE(int n, T \_rv = T()) : Matrix <T>(n) {…}

~SLAE() {…}

void fill() {…}

void s\_print() {…}

void swap(int str1, int str2) {…}

Vector <T> gauss() {…}

private:

T\* rv;

};

* abs.h

Реализация шаблонной функции abs:

template <typename T>

T abs(T a) {

if (a < 0)

{

return -a;

}

else

{

return a;

}

}

# Подтверждение корректности

Для подтверждения корректности необходимо подставить вектор решений x системы в правую часть СЛАУ, после чего будет получен ранее введенный вектор b.

# Заключение

В ходе лабораторной работы был разработан метод Гаусса для решения системы линейных уравнений.

Метод Гаусса корректно работает и решает систему линейных уравнений.

# Приложение

Vector <T> gauss() {

T error = 0.0000001;

//making it look triangular

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

{

int i = 0;

int max\_abs\_index = i + j;

T max\_abs = abs<T>(this->m\_x[i + j][j]);

for (i + j + 1; i + j + 1 < this->size; i++);

{

if (abs<T>(this->m\_x[i+j][j]) > max\_abs)

{

max\_abs\_index = i+j;

}

}

swap(j, max\_abs\_index);

for (int l = j + 1; l < this->size; l++)

{

if (this->m\_x[j][j] != 0)

{

T rate = -(this->m\_x[l][j] / this->m\_x[j][j]);

for (int k = j; k < this->size; k++)

{

this->m\_x[l][k] += this->m\_x[j][k] \* rate;

}

rv[l] += rv[j] \* rate;

}

else

{

cout << "Division by 0 when searching rate!" << endl;

exit(3);

}

}

}

//removing the error

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

{

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

{

if (abs(this->m\_x[i][j]) < error)

{

this->m\_x[i][j] = 0;

}

}

if (abs(rv[i]) < error)

{

rv[i] = 0;

}

}

//checking for missing solutions

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

{

T sum = 0;

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

{

sum += abs<T>(this->m\_x[i][j]);

}

if ((sum == 0) && (rv[i] != 0))

{

cout << "No solution: 0 \* x = C, C!=0" << endl;

exit(2);

}

if ((sum == 0) && (rv[i] == 0))

{

cout << "Cant solve: the equation depends on the variable" << endl;

exit(3);

}

}

//find solutions

T\* sol = new T[this->size];

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

T tmp = 0;

for (int j = i + 1; j < this->size; j++) {

tmp += this->m\_x[i][j] \* sol[j];

}

sol[i] = (rv[i] - tmp) / this->m\_x[i][i];

}

return Vector<T>(this->size, sol);

}