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 ситем линейных уравнений методом Гаусса с выбором ведущего элемента»**

**Выполнил**:

студент группы 381903-2

Егоров Д. В.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Написать программу, решающую систему линейных уравнений методом Гаусса с выбором ведущего элемента, для этого написать шаблонный класс вектор, унаследовать от него класс квадратных матриц которые являются шаблоном вектор от вектора, унаследовать от класса квадратных матриц класс линейных уравнений, содержащий метод Гаусса.

# Метод решения

1. Написан шаблонный класс вектор My\_vector с конструктором по умолчанию, возвращающим вектор размера 1, конструктором создающим динамический массив из заданного количества элементов, методом print, выводящим вектор на экран, перегруженными операторами = и [].

template <typename T>

class My\_vector

{

public:

My\_vector()

{

m\_x = new T[1];

size = 1;

}

My\_vector(int \_size, T \_value = T())

{

size = \_size;

m\_x = new T[\_size];

for (int i = 0; i < \_size; i++)

{

m\_x[i] = \_value;

}

}

My\_vector(int \_size, T\* values)

{

size = \_size;

m\_x = new T[\_size];

for (int i = 0; i < \_size; i++)

{

m\_x[i] = values[i];

}

}

My\_vector(const My\_vector& v)

{

delete[]this->m\_x;

this->size = v.size;

this->m\_x = new T[v.size];

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

{

this->m\_x[i] = v.m\_x[i];

}

return \*this;

}

T& operator[](int i)

{

return m\_x[i];

}

My\_vector& operator=(const My\_vector& v)

{

delete[]this->m\_x;

this->size = v.size;

this->m\_x = new T[v.size];

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

{

this->m\_x[i] = v.m\_x[i];

}

return \*this;

}

void print()

{

cout << "(";

for (int i = 0; i < size - 1; i++)

{

cout << m\_x[i] << "\t";

}

cout << m\_x[size - 1] << ")";

}

~My\_vector()

{

delete[]m\_x;

}

private:

protected:

T\* m\_x;

int size;

};

1. Написалн класс квадратной матрицы Matrix, наследник класса My\_vector, являющийся шаблоном My\_vector от My\_vector.

template < typename T >

class Matrix : public My\_vector< My\_vector < T > >

{

public:

Matrix(int lines) : My\_vector < My\_vector < T > >(lines, My\_vector < T >(lines)) {}

};

1. Написана шаблонная функция mod, возвращающую модуль переданного значения.

template <typename T>

T mod(T a) {

if (a > 0)

{

return a;

}

else

{

return -a;

}

}

1. Написан шаблонный класc Linear\_system, содержащий метод print, выводящий расширенную матрицу, метод swap, меняющий местами строчки и реализацию метода Гаусса gauss\_method, в ходе которого матрица приводится к треугольному виду, система проверяется на отсутствиелюбых или частных решений и выводится результат:решение системы или соответствующее сообщение о невозможности её решить.

template < typename T >

class Linear\_system : public Matrix<T>

{

public:

Linear\_system(int n, T \_rv = T()) : Matrix <T>(n)

{

rv = new T[n];

for (int i = 0; i < n; i++)

{

rv[i] = \_rv;

}

}

void filler()

{

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

{

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

{

cout << "Введите [" << j + 1 << "] коэффициент" << "[" << i + 1 << "] уравнения:" << endl;

cin >> this->m\_x[i][j];

}

}

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

{

cout << "Введите " << i + 1 << " правое значение" << endl;

cin >> rv[i];

}

}

void swap(int str1, int str2)

{

if (str1 < this->size && str2 < this->size)

{

T r\_tmp;

r\_tmp = rv[str1];

rv[str1] = rv[str2];

rv[str2] = r\_tmp;

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

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

{

tmp[i] = this->m\_x[str1][i];

this->m\_x[str1][i] = this->m\_x[str2][i];

this->m\_x[str2][i] = tmp[i];

}

delete[]tmp;

}

else

{

cout << "Неправильная строка!";

exit(1);

}

}

void print()

{

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

{

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

{

cout << this->m\_x[i][j] << "\t";

}

cout << "| " << rv[i] << endl;

}

cout << endl;

}

My\_vector <T> gauss\_method()

{

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

{

int i = 0;

int max\_index = i + j;

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

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

{

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

{

max\_index = i + j;

}

}

swap(j, max\_index);

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

{

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

{

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

{

this->m\_x[n][k] += this->m\_x[j][k] \* (-(this->m\_x[n][j] / this->m\_x[j][j]));

}

rv[n] += rv[j] \* (-(this->m\_x[n][j] / this->m\_x[j][j]));

}

else

{

cout << "Ошибка! Произошло деление на 0" << endl;

exit(3);

}

}

}

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

{

T sum = 0;

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

{

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

}

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

{

cout << "Система не совместна, нет решений." << endl;

exit(2);

}

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

{

cout << "Система не имеет частных решений." << endl;

exit(3);

}

}

T\* res = 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] \* res[j];

}

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

}

return My\_vector<T>(this->size, res);

}

~Linear\_system()

{

delete[]rv;

}

private:

T\* rv;

};

# Руководство пользователя

Пользователь должен ввести размер матрицы, равный количеству переменных в уравнении системы

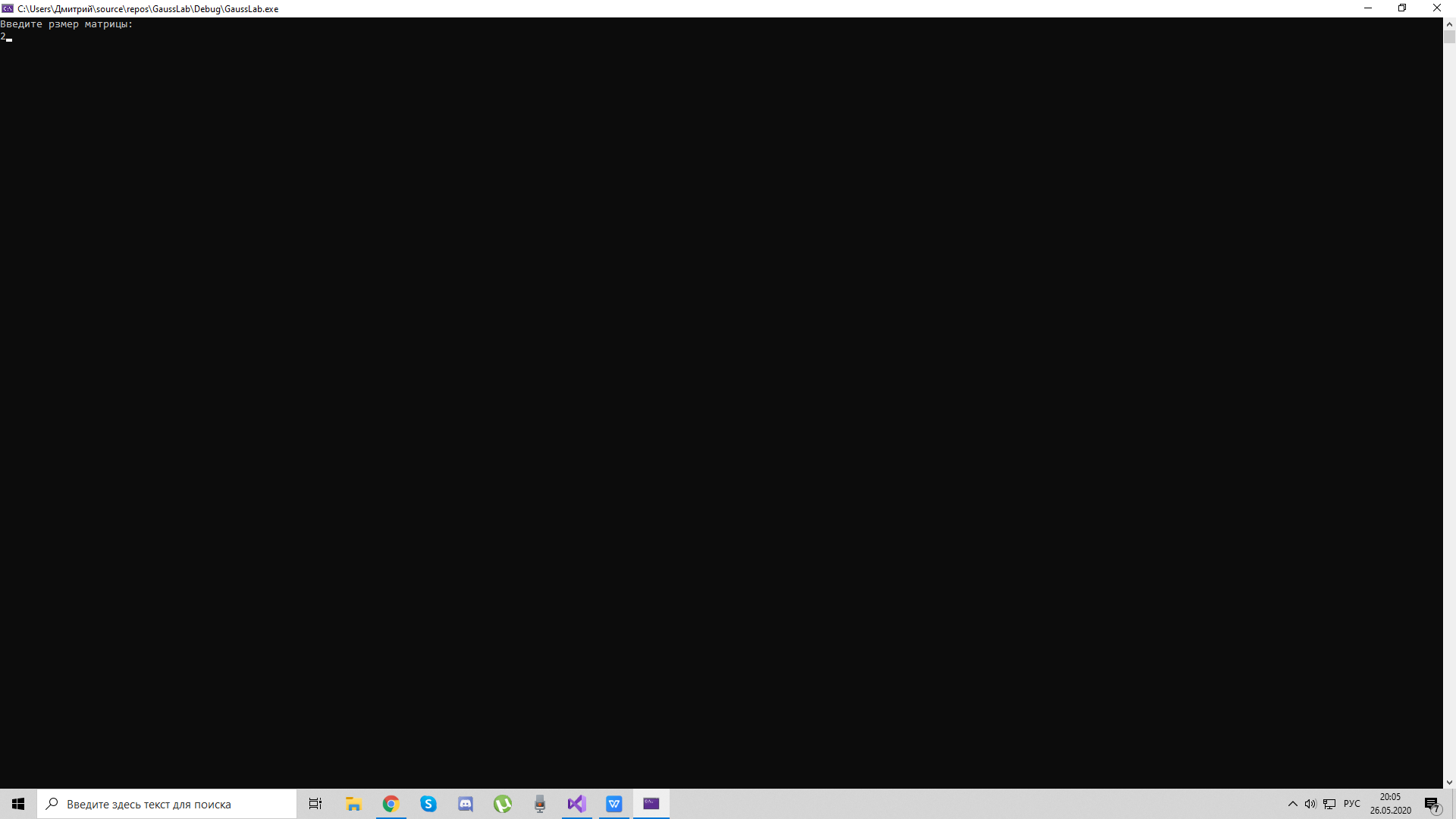


Рис.1 Ввод размера матрицы.

Далее нужно сделать последовательный ввод коэффициентов при переменных уравнений

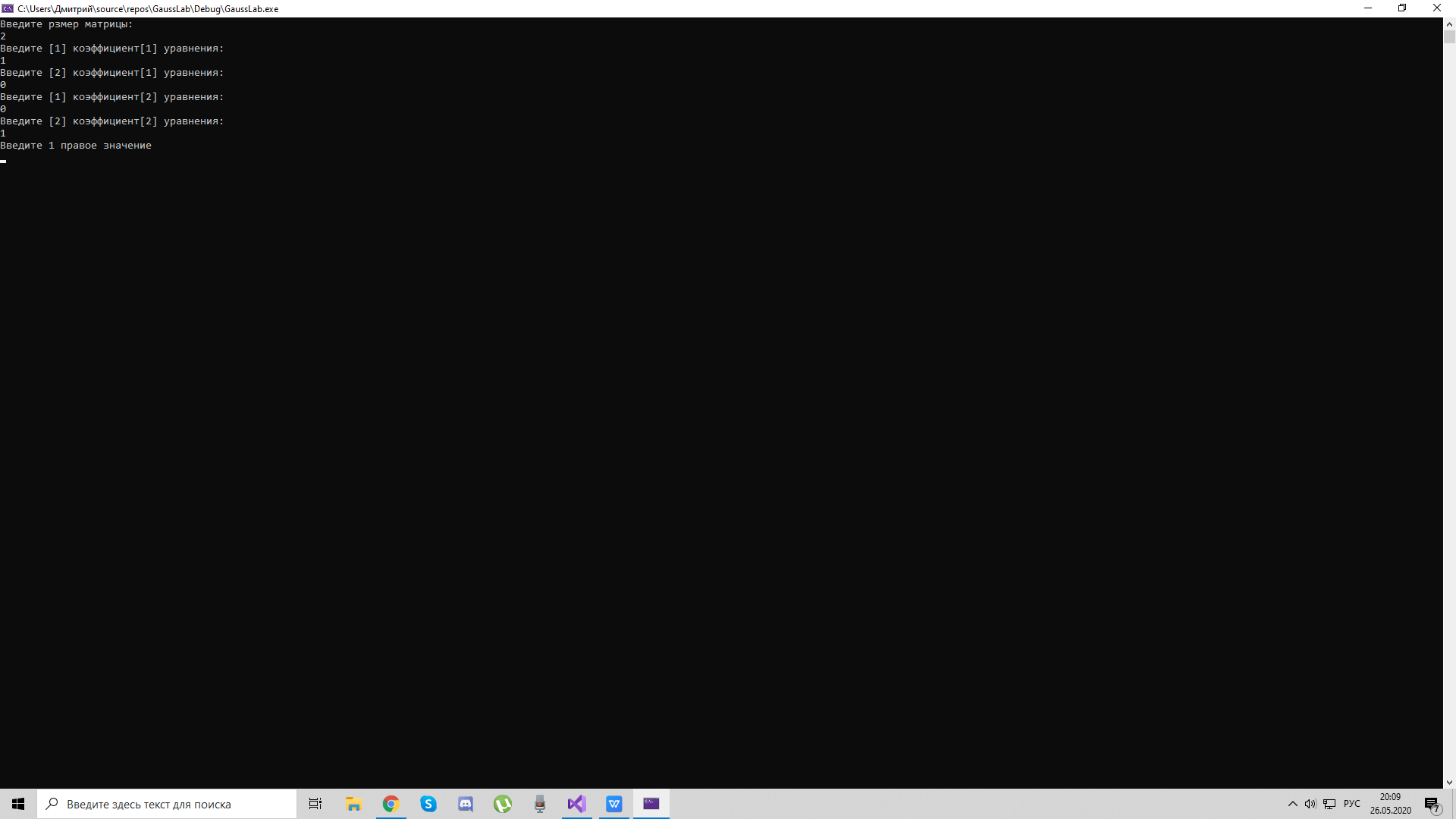


Рис.2 Ввод коэффициентов.

Затем нужно ввести правую часть - то, чему равно каждое из уравнений в рамках данной системы

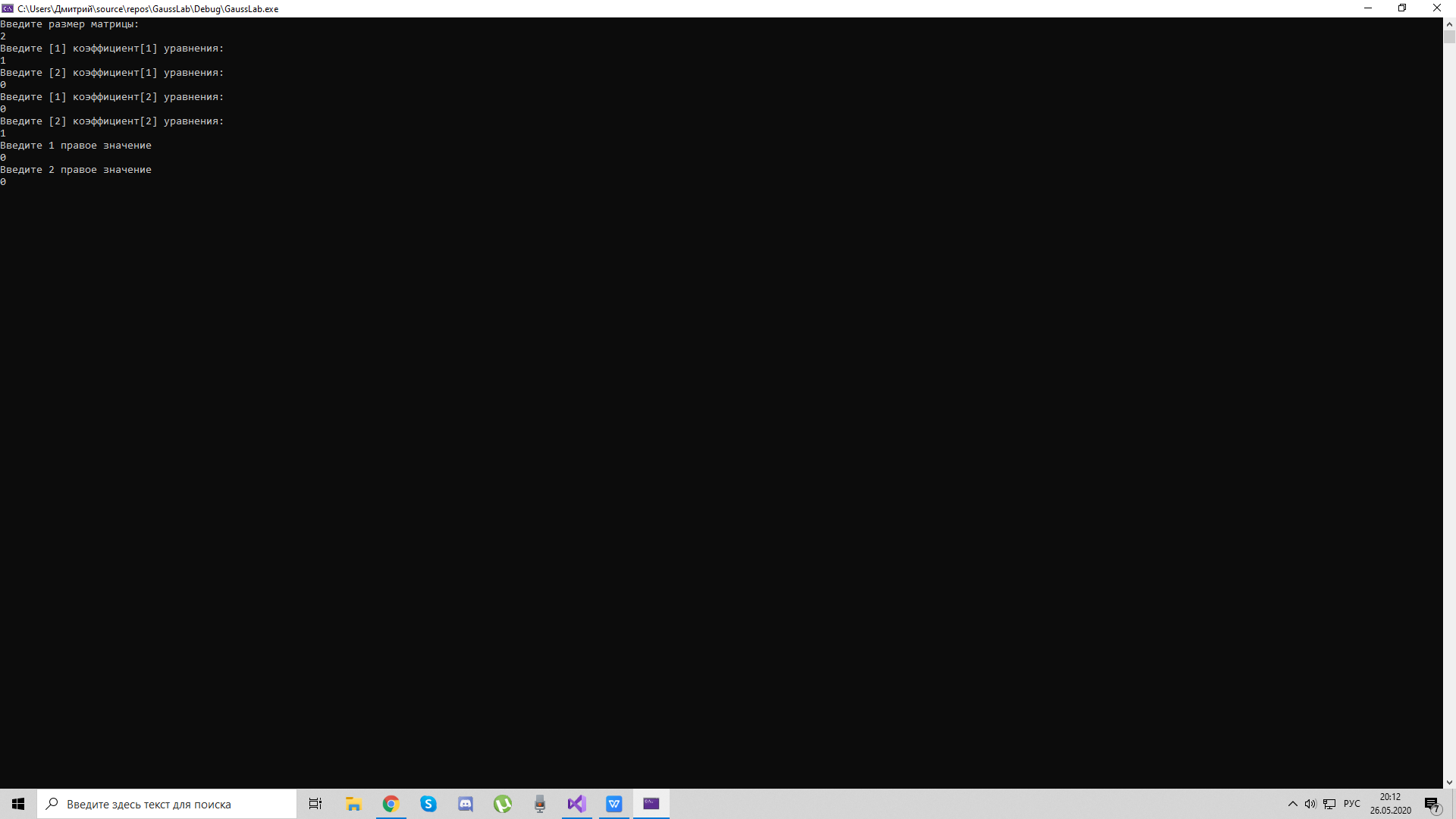


Рис.3 Ввод правых значений.

После проделанных действий на экран будет выведен ответ.

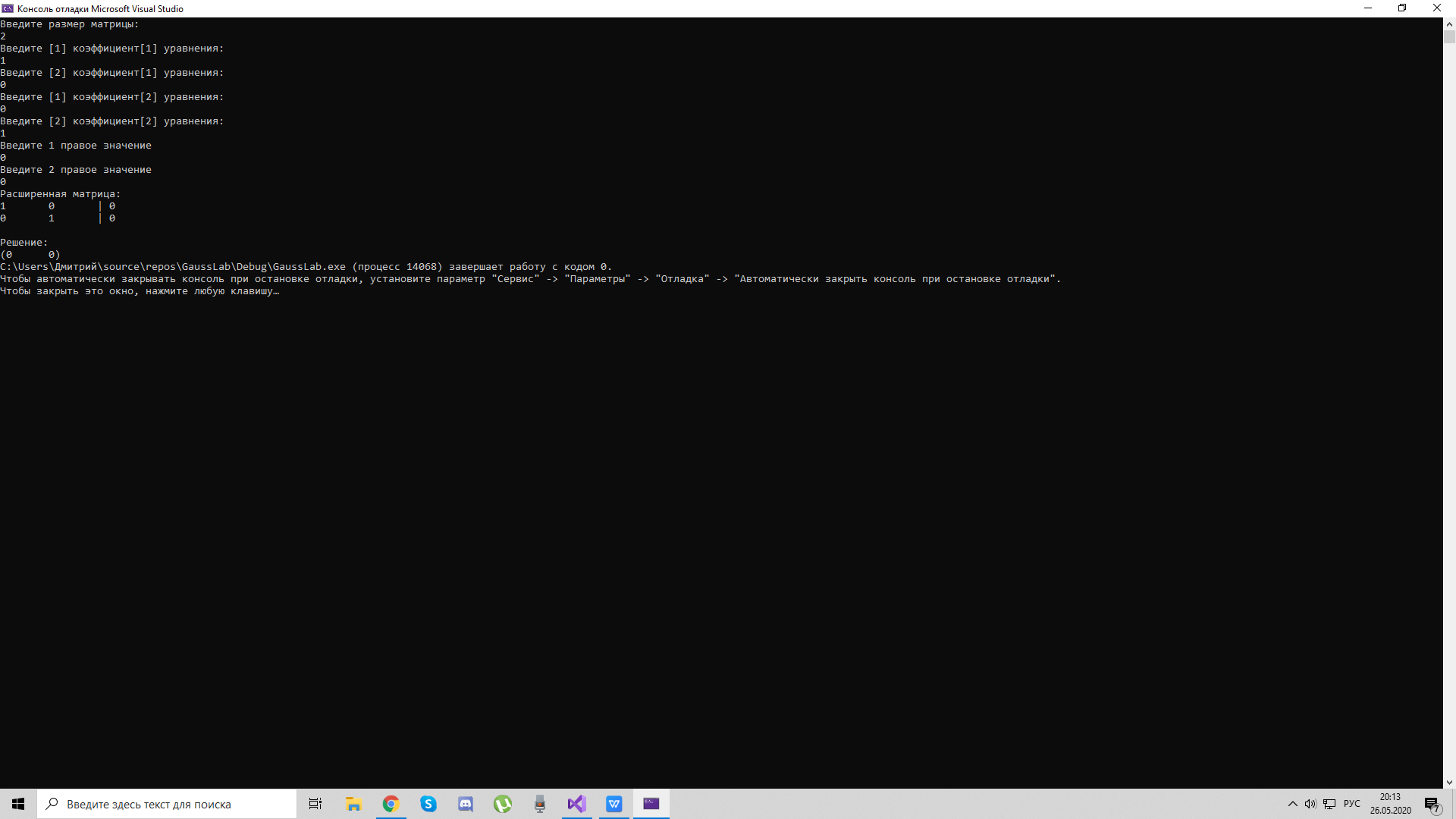


Рис.4 Ответ.

# Описание программной реализации

1. Main.cpp

Файл соединяющий в единое целое заголовочные файлы

1. Vector.h

Заголовочный файл с реализацией шаблонного класса My\_vector.

1. Matrix.h

Заголовочный файл с реализацией шаблонного класса квадратных матриц Matrix

1. Linear\_system.h

Заголовочный файл с реализацией шаблонного класса Linear\_system.

1. Mod.h

Заголовочный файл с реализацией шаблонной функции mod.

# Подтверждение корректности

Проверим корректность умножив начальную матрицу на полученный вектор решений.

Код функции для проверки:

void check()

{

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

cout << endl;

cout << "Правое значение при проверке:" << endl;

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

{

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

{

check[i] += this->m\_x[i][j]\*rv[j];

}

cout << rv[i] << "\t";

}

}

# Результаты экспериментов

1. При размере матрицы 2, последовательно введёных коэффициентах: (1, 2), (1, 6), правой части:(0,0) результат:(0,0).
2. При размере матрицы 3, последовательно введённых коэффициентах:(1, 3, 5),(1, 2, 10),(0, 0, 0), правой части(1, 1, 1), результат: система несовместна, нет решений.
3. При размере матрицы 3, последовательно введённых коэффициентах:(1, 3, 5),(1, 2, 10),(0, 0, 0), правой части(1, 1, 0), результат: система не имеет частных решений.

# Заключение

В ходе работы реализован метод Гаусса, решающий систему линейных уравнений с выбором ведущего элемента.

# Приложение

template <typename T>

class My\_vector

{

public:

My\_vector()

{

m\_x = new T[1];

size = 1;

}

My\_vector(int \_size, T \_value = T())

{

size = \_size;

m\_x = new T[\_size];

for (int i = 0; i < \_size; i++)

{

m\_x[i] = \_value;

}

}

My\_vector(int \_size, T\* values)

{

size = \_size;

m\_x = new T[\_size];

for (int i = 0; i < \_size; i++)

{

m\_x[i] = values[i];

}

}

My\_vector(const My\_vector& v)

{

delete[]this->m\_x;

this->size = v.size;

this->m\_x = new T[v.size];

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

{

this->m\_x[i] = v.m\_x[i];

}

return \*this;

}

T& operator[](int i)

{

return m\_x[i];

}

My\_vector& operator=(const My\_vector& v)

{

delete[]this->m\_x;

this->size = v.size;

this->m\_x = new T[v.size];

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

{

this->m\_x[i] = v.m\_x[i];

}

return \*this;

}

void print()

{

cout << "(";

for (int i = 0; i < size - 1; i++)

{

cout << m\_x[i] << "\t";

}

cout << m\_x[size - 1] << ")";

}

~My\_vector()

{

delete[]m\_x;

}

private:

protected:

T\* m\_x;

int size;

};