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 382003-1

Смолин К.А.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**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

Необходимо написать программу, реализующую **метод Гаусса с выбором ведущего элемента.**

1. Реализовать шаблонный класс вектор.
2. Написать шаблонный класс матрица, вектор от веторов.
3. Реализовать класс (или метод класса матрица), который на вход принимает правую часть в виде аргумента и вычисляет решение системы уравнение.
4. Вывести результат в виде вектора.
5. Написать интерфейс для пользователя.

# Метод решения

Для решения системы линейных уравнений **методом Гаусса**, необходимо привести матрицу к треугольному виду путём **элементарных преобразований**. Нам доступны следующие действия:

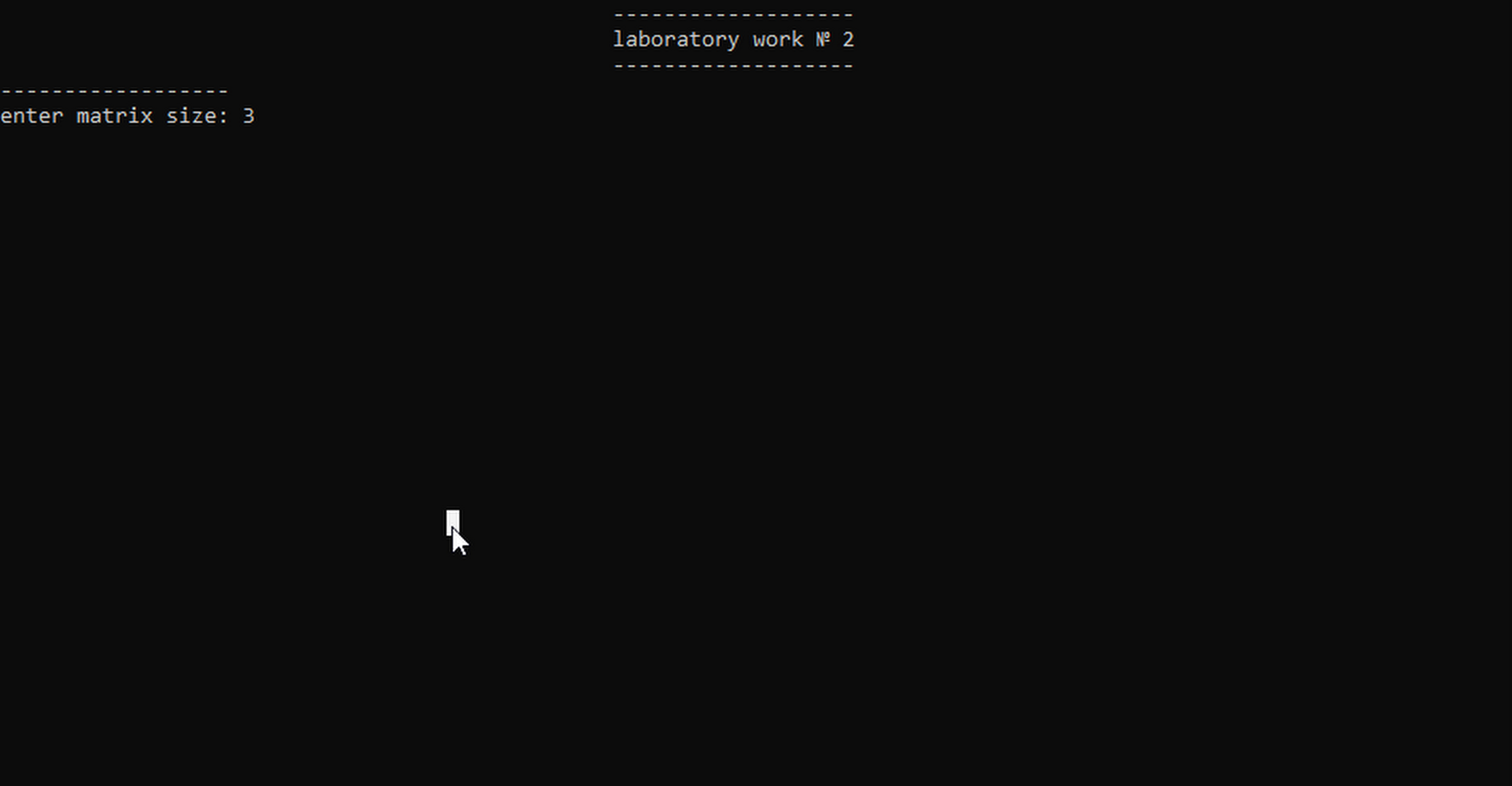
1. Перестановка местами любых двух строк матрицы
2. Умножение на ненулевую константу любой строки матрицы
3. Прибавление к любой строке матрицы другой строки, умноженной на ненулевую константу.
4. Метод Гаусса состоит из 2 основных частей **– прямой ход, обратный ход.**

**Прямой ход** – последовательный выбор элементов матрицы в i-м столбце, начиная с , в качестве ведущих и зануление остальных элементов в том же столбце путём элементарных преобразований. Далее происходит переход к следующему справа столбцу, в котором элемент строкой ниже не равен нулю.

**Обратный ход** – при наличии решений происходит вычисление значений неизвестных. Данная операция происходит последовательно от последнего уравнения к первому.

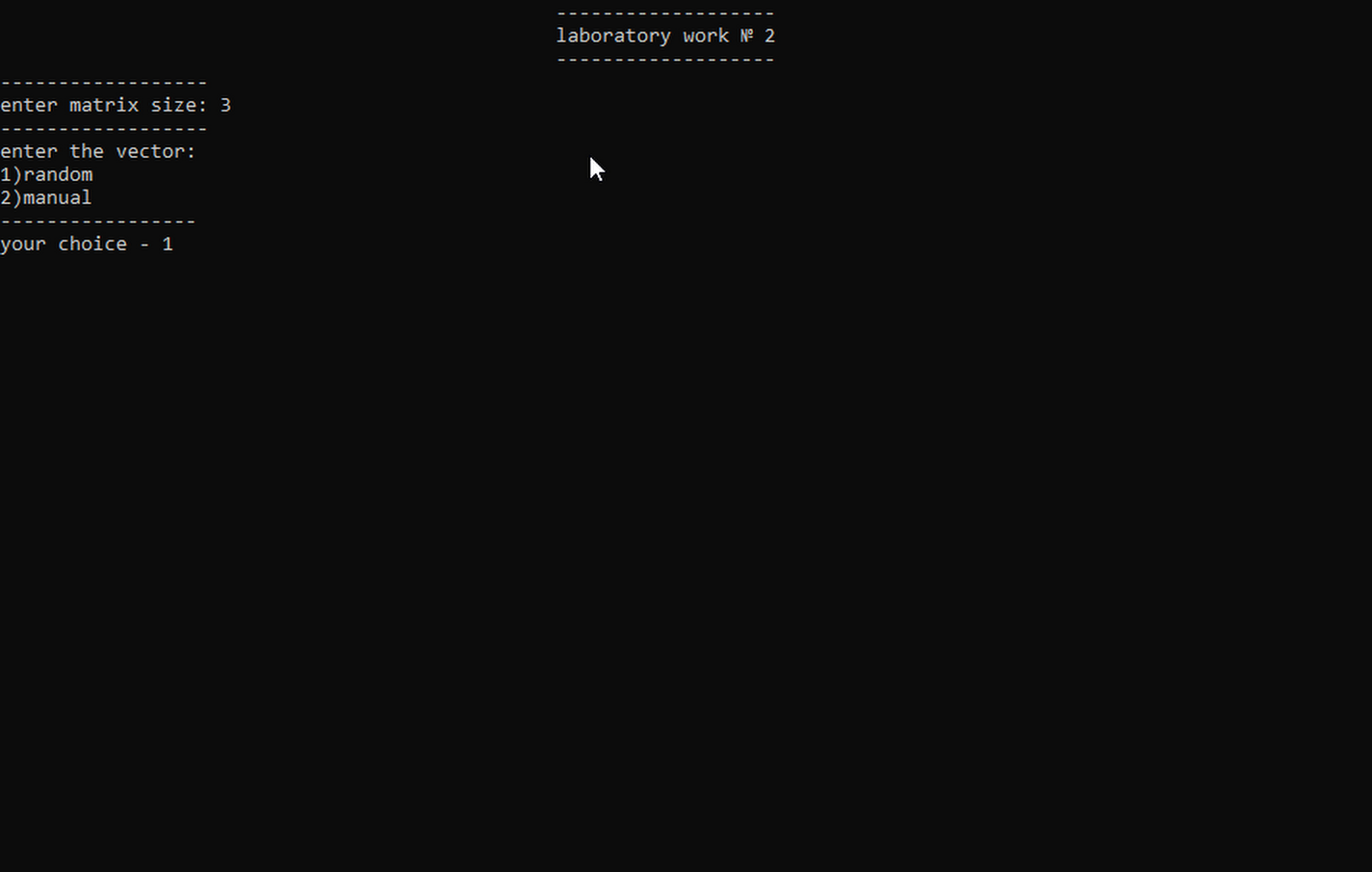
# Руководство пользователя

При запуске программы пользователю предлагается ввести размер квадратной матрицы, который также станет размером вектора правой части.



Далее необходимо ввести вектор правой части. Есть два варианта:

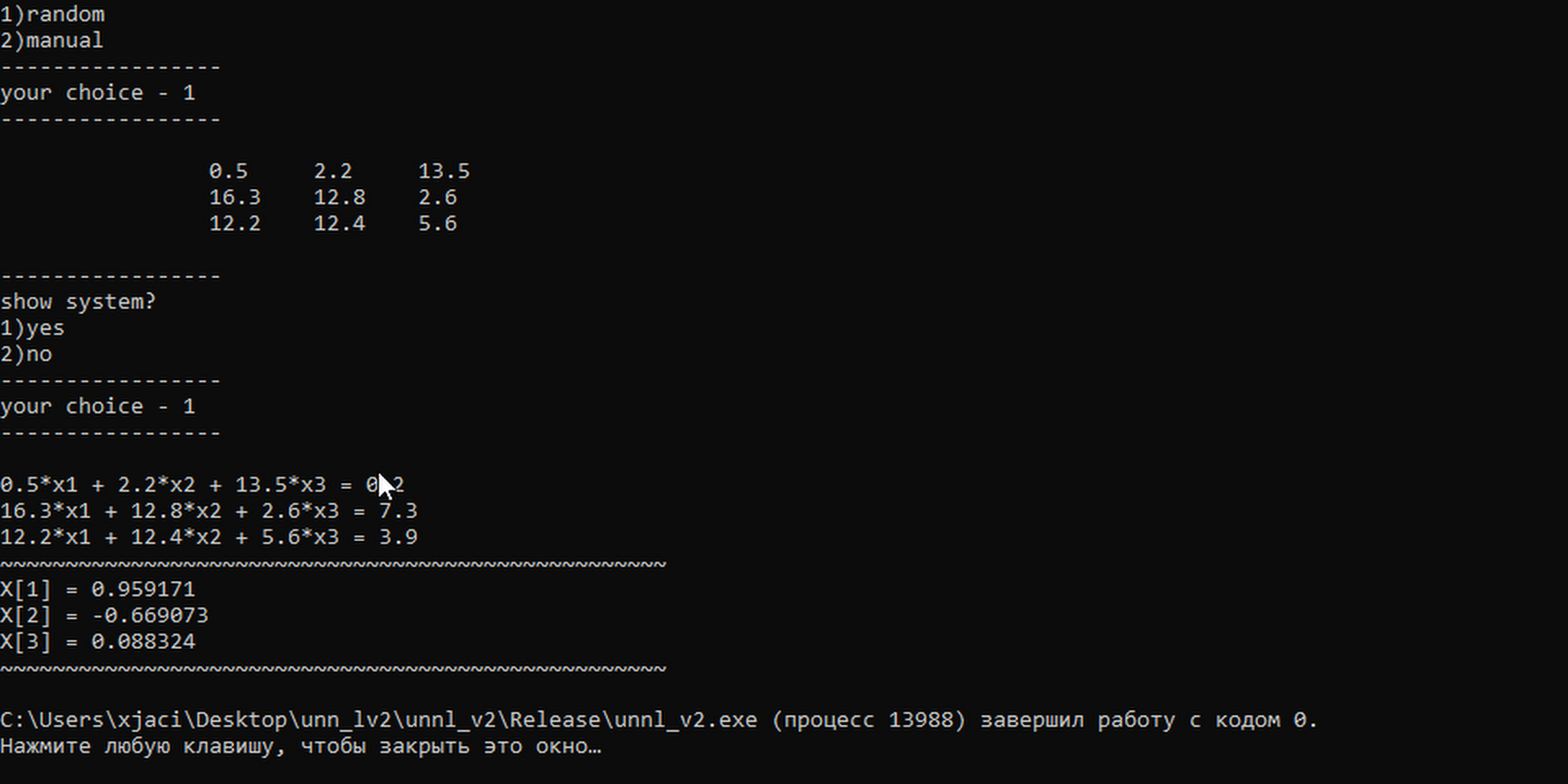
* Рандомный вектор
* Ввести вручную



Далее также предлагается два варианта на ввод матрицы – **рандомный, ввод вручную**.



После ввода матрицы пользователь может увидеть полную систему линейных уравнений. Для этого необходимо ввести 1, если необходимо показать систему и 2, если этот шаг можно пропустить. Также сразу будет показан ответ в виде вектора, независимо от того, выбрал пользователь 1 или 2 на шаге отображения полной системы.



# Описание программной реализации

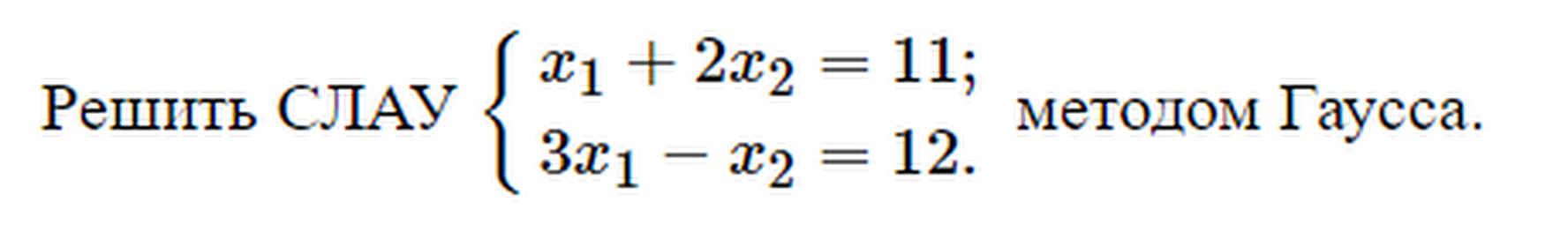
В проекте находятся 4 файла, 2 исходных, 2 заголовочных.

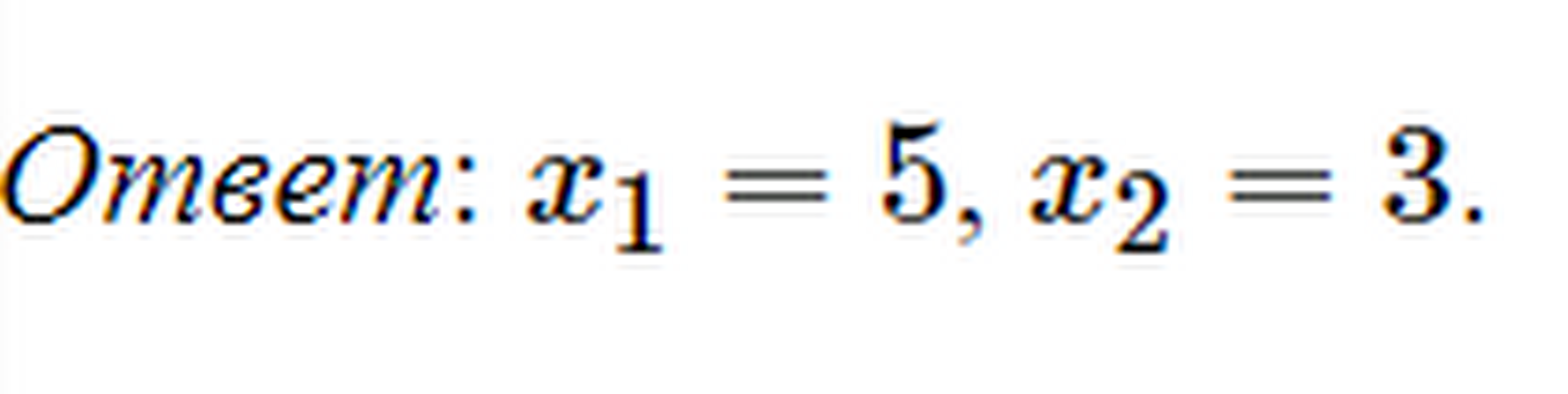
1. Файл classes.h содержит реализацию класса вектор, матрицы, всех методов, содержит метод, решающий систему уравнений
2. Main.cpp – инициализирует интерфейс пользователя
3. Menu.h – подключается к main и содержит всего одну функция для инициализации меню
4. Menu.cpp – содержит реализацию интерфейса пользователя

# Подтверждение корректности

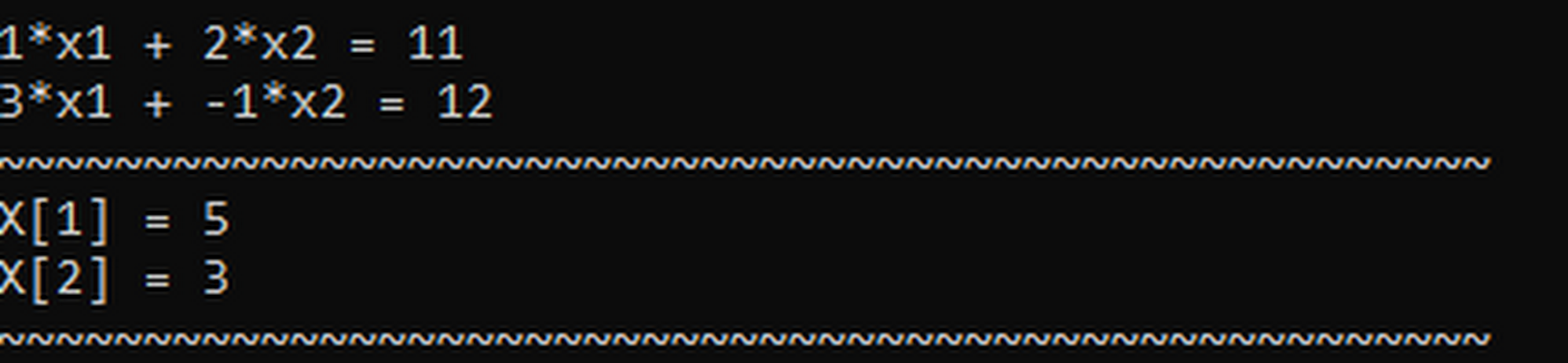
Для подтверждения корректности были использованы онлайн ресурсы решения систем линейных уравнений. При проверке решений были получены те же результаты, что и при работе программы.

# Результаты экспериментов

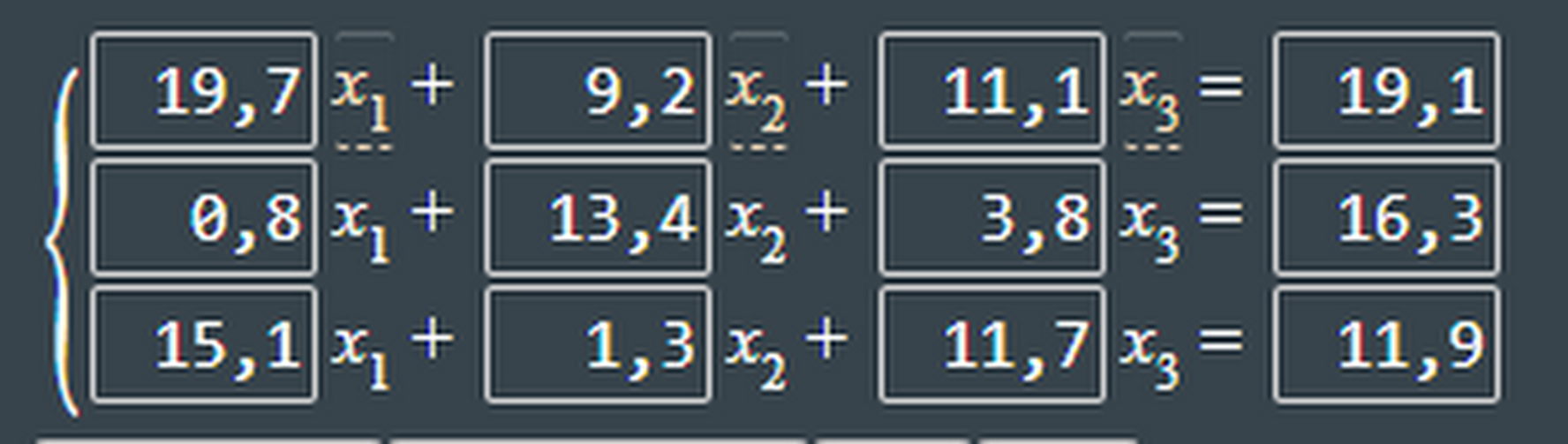
Пример 1: 

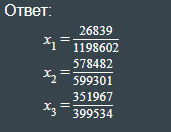


Результат работы программы:

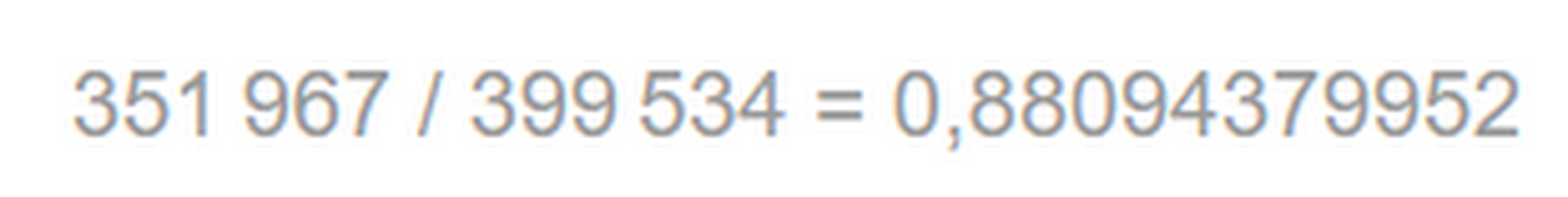


Пример 2(рандомный ввод):

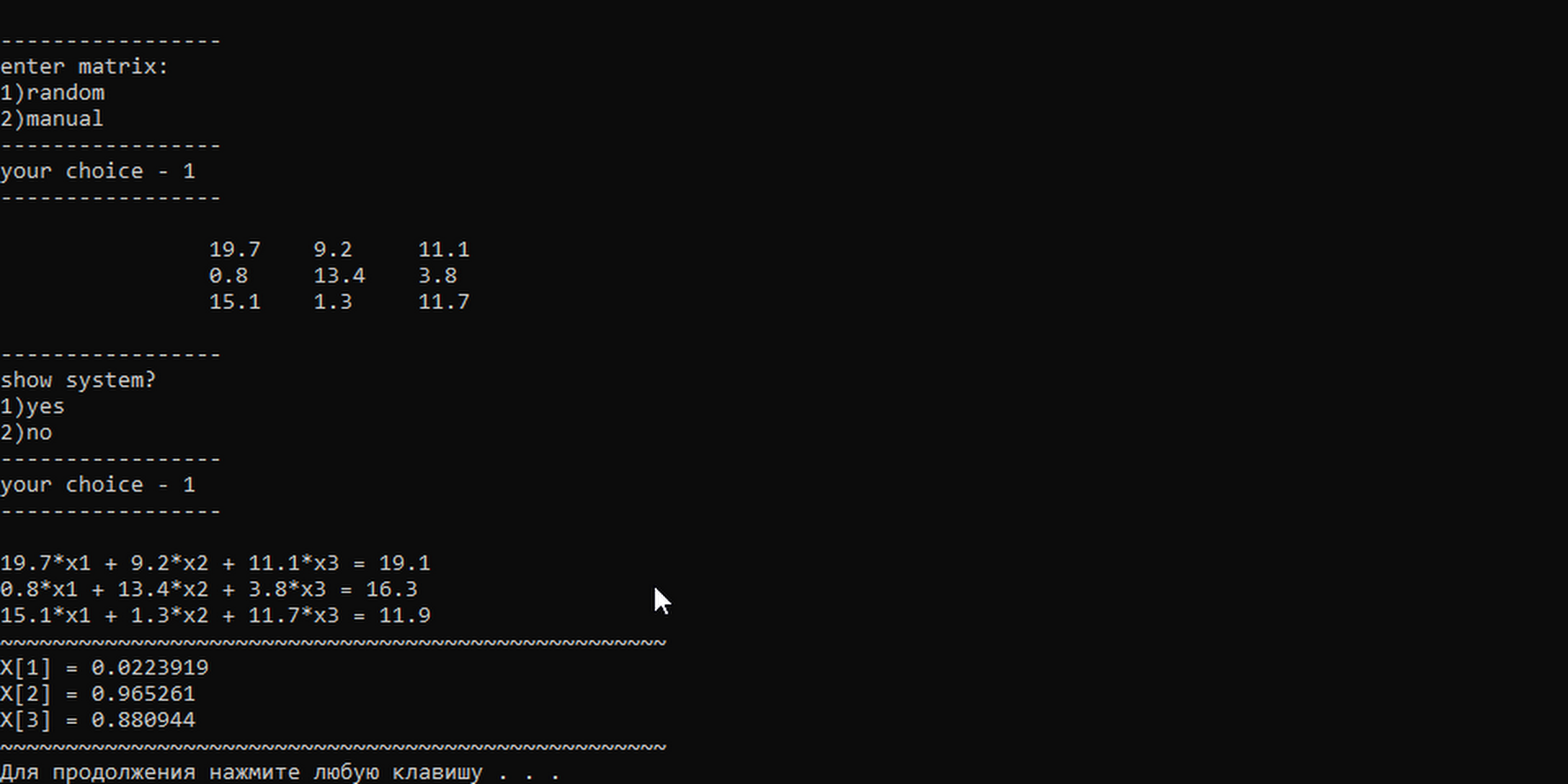


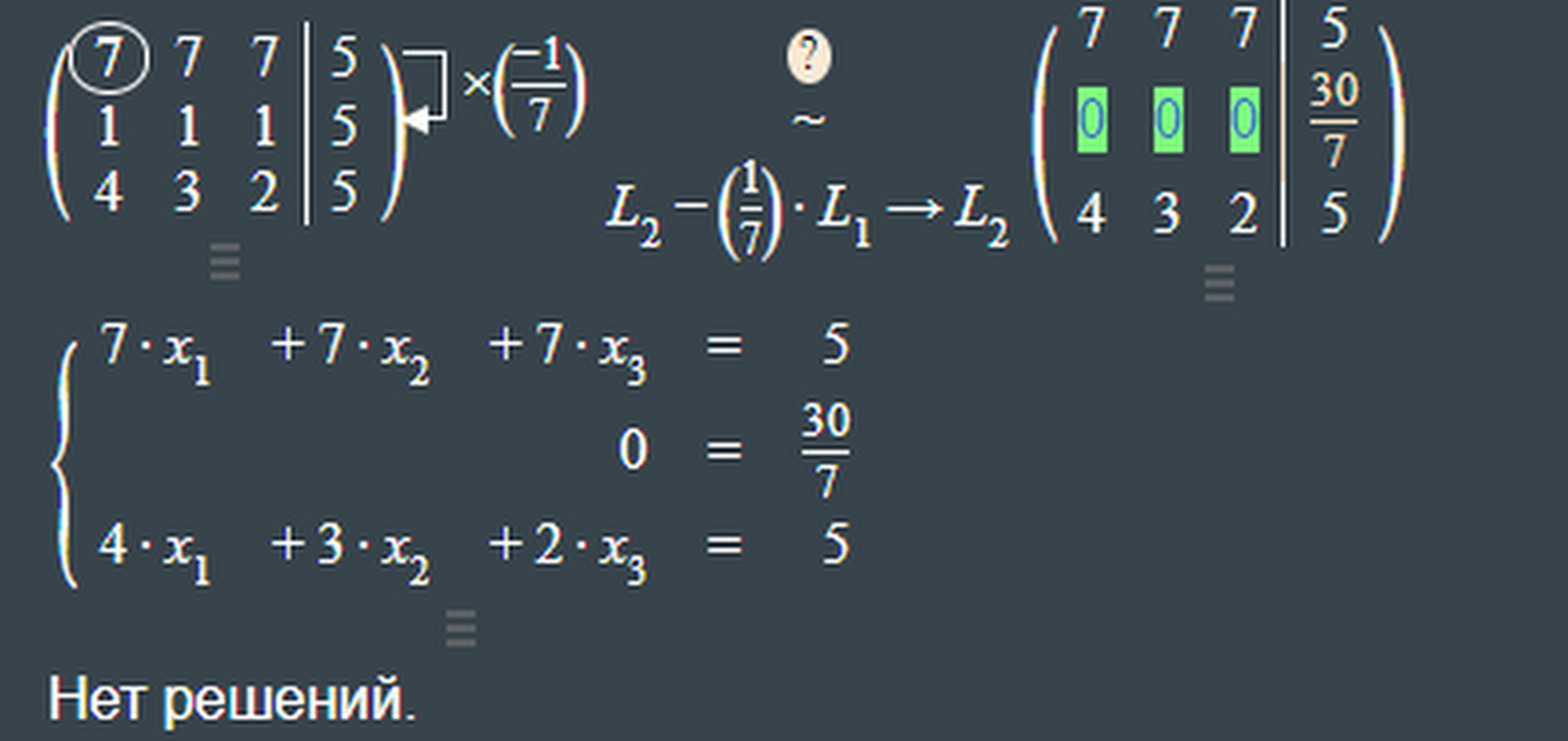


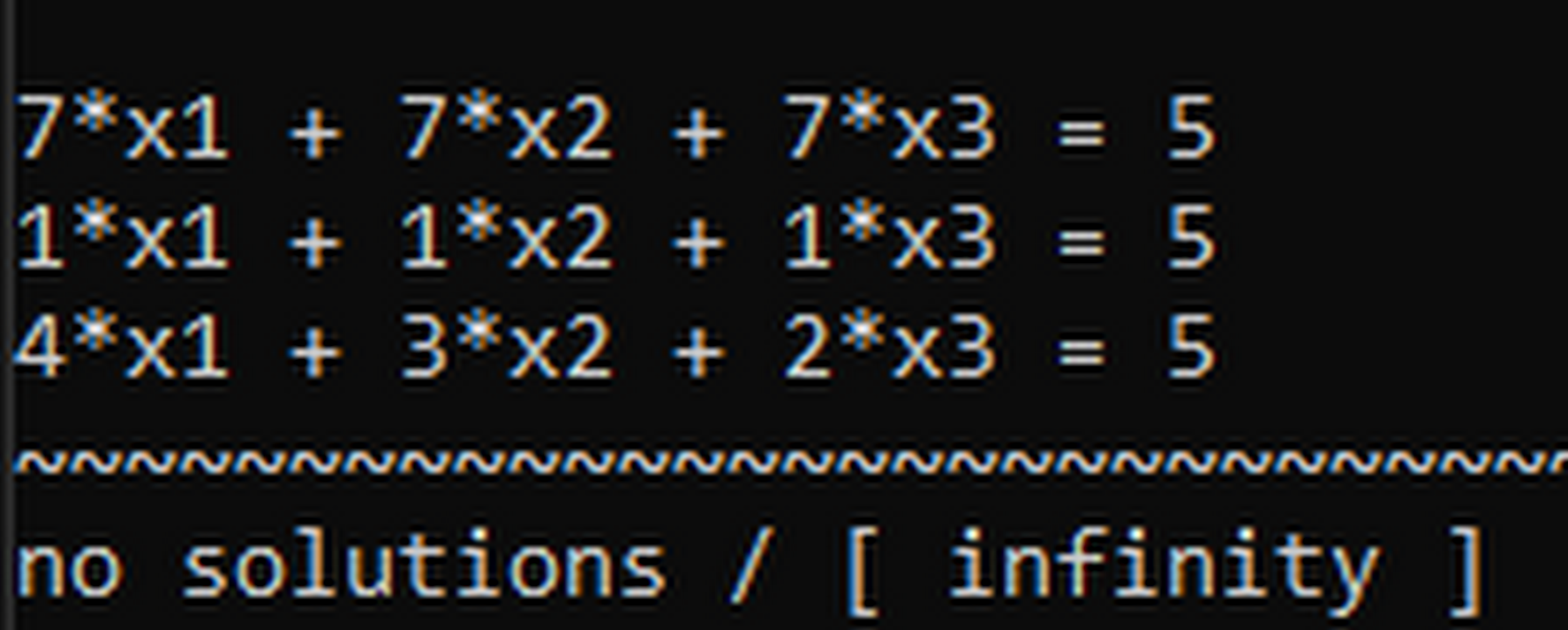


Результат работы программы:



Пример 3:





По данным экспериментов видно, что все варианты совпадают с результатами, полученными с помощью онлайн-ресурсов.

# Заключение

Итак, поставленная задача выполнена: реализованы шаблонные классы **Vector**, **Matrix**, реализован метод Гаусса с выбором ведущего элемента, вывод решения в случае его единственности либо сообщения об отсутствии решений.

# Приложение

#include <iostream>

#include <cmath>

const double eps = 1e-8;

template <typename T>

class vector {

protected:

T\* data;

size\_t size;

public:

vector() { size = 0; data = nullptr; }

vector(size\_t \_size) {

size = \_size;

data = new T[size];

}

vector(vector& sec) {

size = sec.size;

data = new T[size];

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

data[i] = sec.data[i];

}

void set\_size(int \_size)

{

if (size != \_size)

{

delete[] data;

data = new T[\_size];

size = \_size;

}

}

T get\_data(size\_t ind){

return data[ind];

}

size\_t get\_size() {

return size;

}

void set\_random() {

srand(static\_cast<unsigned int>(time(0)));

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

data[i] = (T)(rand() % 200) / 10.0;

}

}

void set\_manual() {

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

{

std::cin >> data[i];

}

}

void vec\_swap(size\_t first, size\_t sec)

{

if (first != sec)

{

T tmp = data[first];

data[first] = data[sec];

data[sec] = tmp;

}

}

void print() {

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

std::cout << data[i] << std::endl;

}

}

vector& operator=(const vector& sec)

{

if (size != sec.size)

{

delete[] data;

data = nullptr;

size = 0;

data = new T[sec.size];

size = sec.size;

}

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

{

this->data[i] = sec.data[i];

}

return \*this;

}

vector<T>& operator \*=(T second) {

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

data[i] \*= second;

return \*this;

}

vector<T>& operator /=(T second) {

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

data[i] /= second;

return \*this;

}

vector& operator-=(vector second) {

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

data[i] -= second.data[i];

return \*this;

}

vector<T>& operator+=(vector<T> a)

{

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

{

data[i] += a[i];

}

return \*this;

}

vector operator+(vector& second) {

if (size != second.size) {

std::cout << "sizes are not equal" << std::endl;

exit(-2);

}

vector result = \*this;

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

result.data[i] += second.data[i];

return result;

}

vector operator-(vector& second) {

if (size != second.size) {

std::cout << "sizes are not equal" << std::endl;

exit(-3);

};

vector result = \*this;

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

result.data[i] -= second.data[i];

return result;

}

friend vector<T> operator\*(const vector<T>& a, T b)

{

vector<T> result(a.size);

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

{

result[i] = a[i] \* b;

}

return result;

}

friend vector operator\*(T b, const vector<T>& a)

{

vector<T> result(a.size);

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

{

result[i] = a[i] \* b;

}

return result;

}

T& operator[](size\_t i)

{

/\*if (i < 0 || i >= size)

{

std::cout << "out of arr";

exit(-4);

}\*/

return data[i];

}

const T& operator[](size\_t i) const

{

/\*if (i < 0 || i >= size)

{

std::cout << "out of arr";

exit(-5);

}\*/

return data[i];

}

friend std::ostream& operator<<(std::ostream& out, const vector<T>& vec)

{

out << "\t";

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

out << vec[i] << "\t";

out << '\n';

return out;

}

friend std::istream& operator>>(std::istream& in, vector<T>& vec)

{

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

in >> vec[i];

return in;

}

~vector() {

delete[] data;

}

};

template <typename T>

class matrix : public vector<vector<T>> {

public:

matrix(int \_size) : vector<vector<T>>(\_size)

{

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

{

this->data[i].set\_size(this->size);

}

}

//~matrix(){}

void set\_random\_matrix()

{

srand(static\_cast<unsigned int>(time(0)));

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

{

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

{

this->data[i][j] = (T)(rand() % 200) / 10.0;

}

}

}

void set\_manual\_matrix()

{

srand(static\_cast<unsigned int>(time(0)));

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

{

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

{

std::cin >> this->data[i][j];

}

}

}

void show\_system(vector<T> right\_side) {

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

for (size\_t j = 0; j < this->size; j++) {

std::cout << this->data[i][j] << "\*x" << j+1;

if (j <= this->size - 2)

std::cout << " + ";

}

std::cout << " = " << right\_side[i] << std::endl;

}

}

size\_t max\_col(size\_t col)

{

T max = this->data[col][col];

size\_t num = col;

for (size\_t i = col + 1; i < this->size; i++)

{

if (abs(max) < abs(this->data[i][col]))

{

max = this->data[i][col];

num = i;

}

}

return num;

}

void swap(int i, int j)

{

if (i != j)

{

vector<T> tmp = this->data[i];

this->data[i] = this->data[j];

this->data[j] = tmp;

}

}

void triangle\_lower(matrix<T>& a, size\_t n, vector<T>& vec)

{

for (size\_t i = 0; i < n - 1; ++i) { // n - 1

size\_t imax = max\_col(i);

if (i != imax) {

swap(i, imax);

vec.vec\_swap(i, imax);

}

if (abs(a[i][i]) < eps)

{

std::cout << "System is linearly dependent\n\n";

}

for (size\_t j = i + 1; j < n; ++j) {

T mul = -a[j][i] / a[i][i];

vec[j] += vec[i] \* mul;

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

a[j][k] += a[i][k] \* mul;

}

}

}

vector<T> sol(n);

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

T temp = 0;

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

temp += a[i][j] \* sol[j];

}

sol[i] = (vec[i] - temp) / a[i][i];

if (abs(sol[i]) < eps)

{

sol[i] = 0;

}

}

size\_t p = 0;

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

if (a[i][i] == 0)

{

for (size\_t j = 0; j < n; j++)

if (a[i][j] == 0)

p = p + 1;

if (p == n && sol[i] != 0)

{

std::cout << "no solutions / [ infinity ] ";

exit(-9);

}

}

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

std::cout << "X["<< i+1 << "]" << " = " <<sol.get\_data(i) << "\t\t" << std::endl;

}

};