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**:

студентка группы 382003-1

Шамонина А.С.

**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

Требовалось написать программу, содержащую реализацию метода Гаусса с выбором ведущего элемента для решения системы линейных уравнений, программа должна включать в себя реализацию шаблонных классов. На вход подаётся квадратная матрица коэффициентов СЛАУ и вектор свободных коэффициентов, на выходе должен быть вектор решений СЛАУ.

# Метод решения

В основе метода Гаусса лежит идея последовательного исключения неизвестных, приводящая исходную систему с квадратной матрицей к легко разрешимой системе с верхней треугольной матрицей.

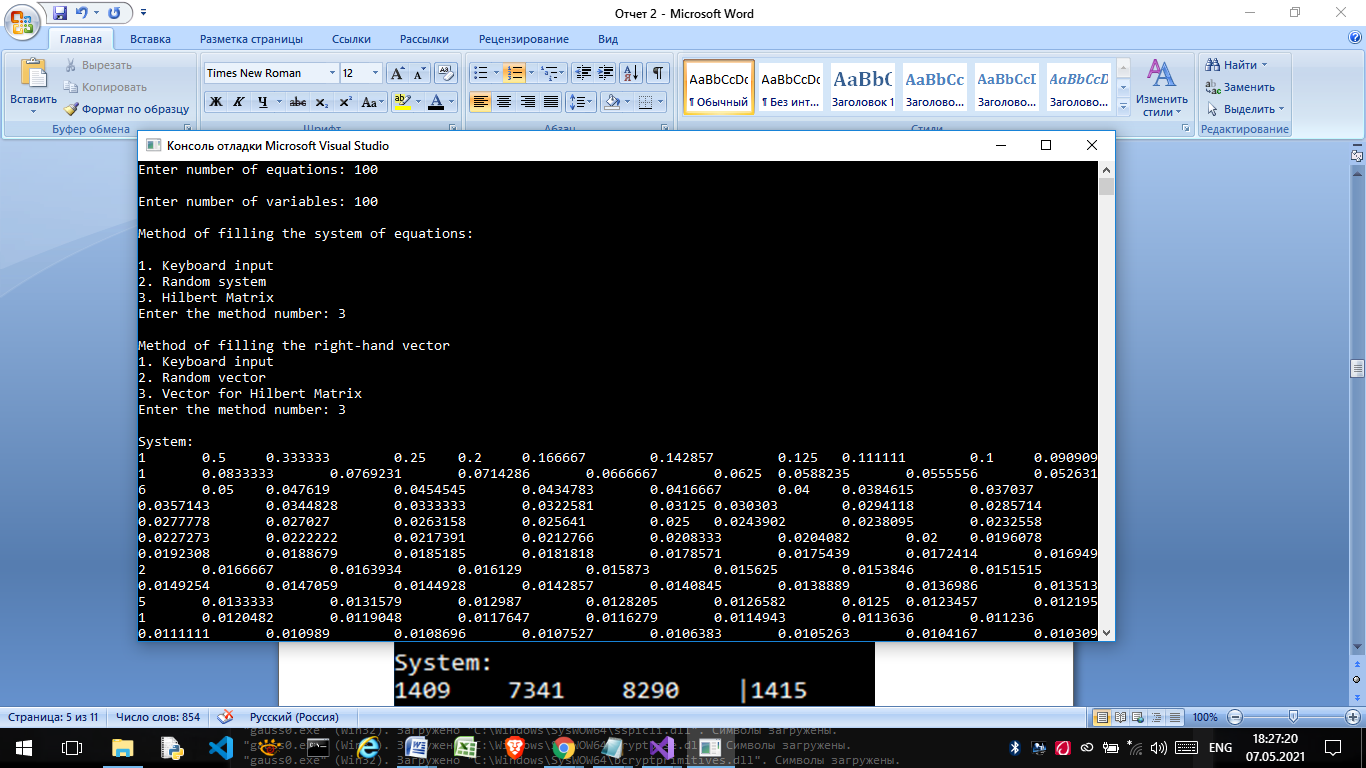
В методе Гаусса с выбором главного элемента на каждом шаге исключения i-го неизвестного в качестве ведущего используется уравнение (с i-го по n-ое), содержащее максимальный по модулю коэффициент – главный элемент. При этом в качестве него может использоваться один из коэффициентов i-го столбца, i-ой строки или всей непреобразованной части матрицы.

**Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа.**

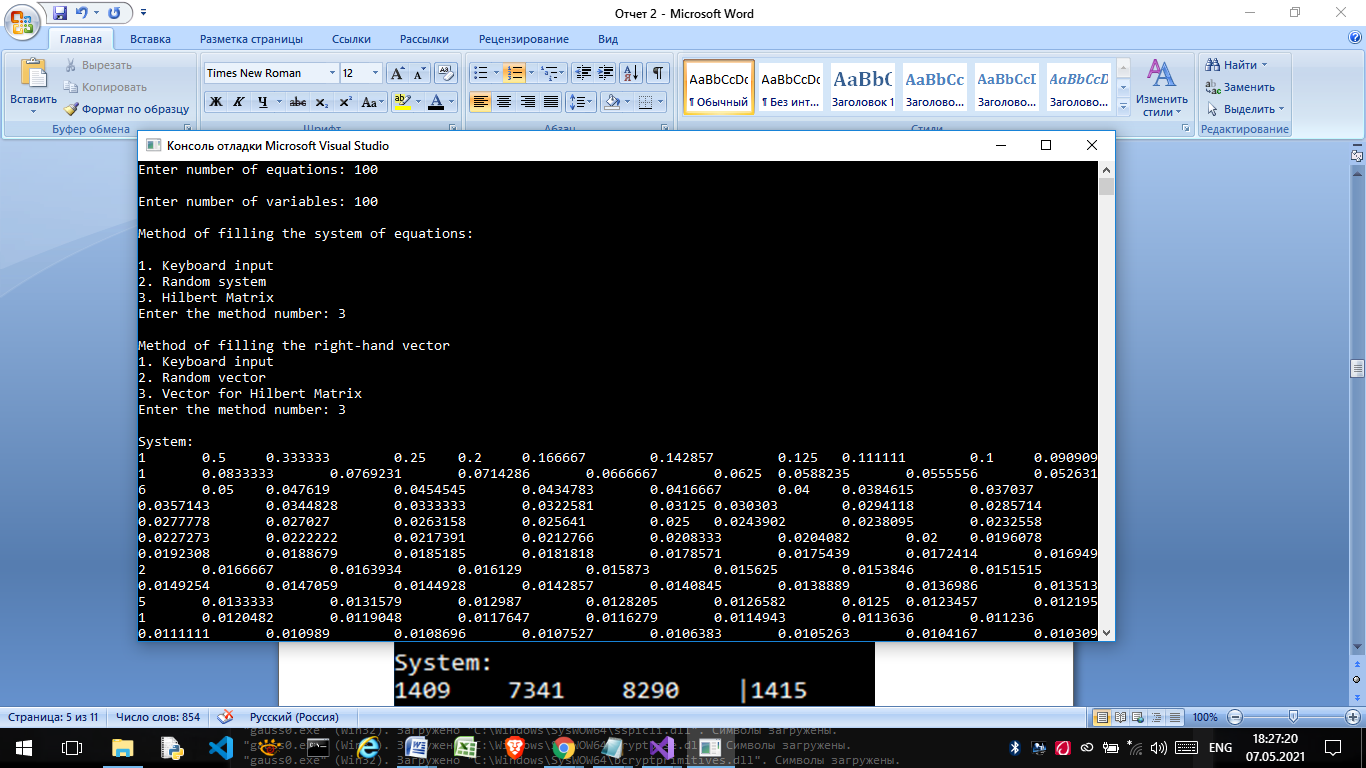
1. На первом этапе осуществляется прямой ход, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна/не имеет решений/имеет бесконечно много решений. Для этого среди элементов первого столбца матрицы выбирают максимальный по модулю, перемещают содержащую его строку в крайнее верхнее положение, делая эту строку первой. Далее ненулевые элементы первого столбца всех нижележащих строк обнуляются путём вычитания из каждой строки первой строки, домноженной на отношение первого элемента этих строк к первому элементу первой строки.
2. На втором этапе осуществляется обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить  систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» наверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.

# Руководство пользователя

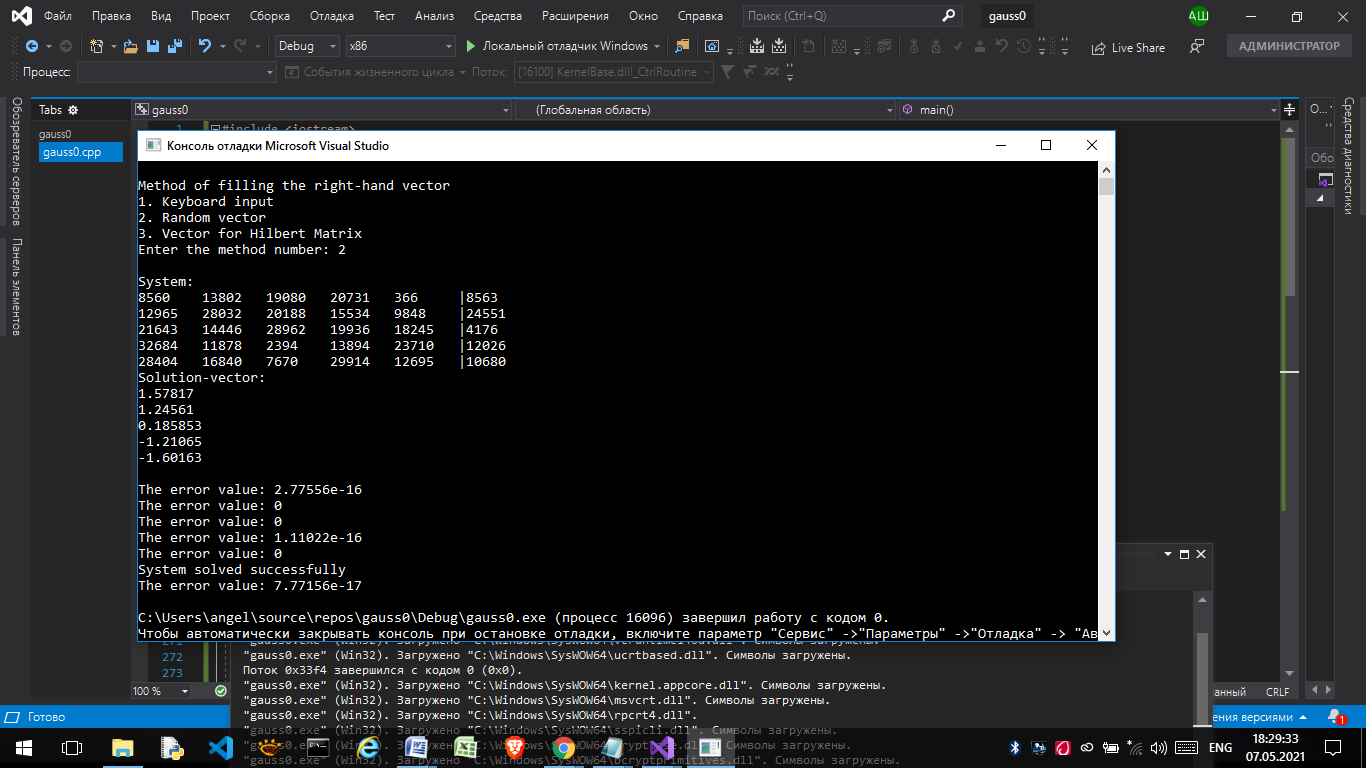
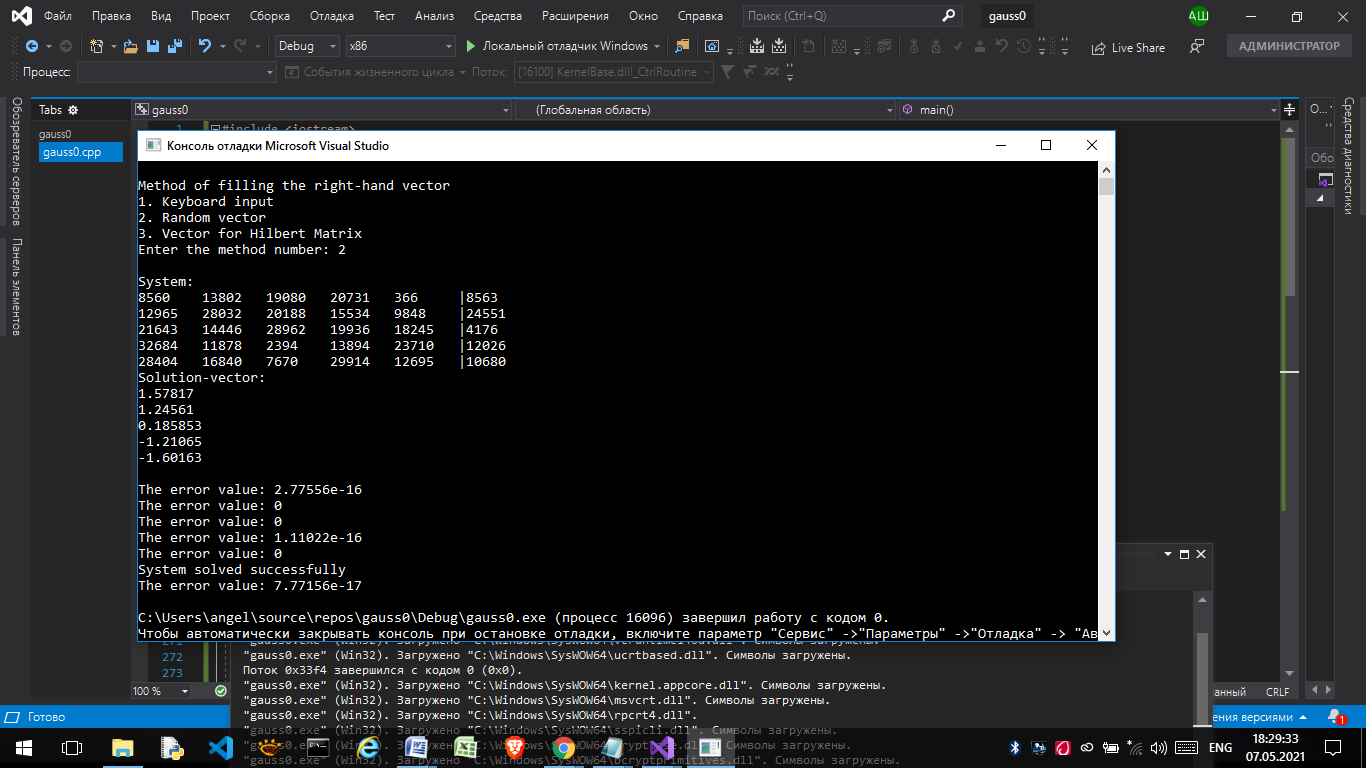
При запуске программы появляется окно ввода, в котором предлагается ввести количество уравнений в системе, которую пользователь собирается решить, и неизвестных в ней. Далее появится меню, в котором можно выбрать способ задания системы.



Затем, после ввода количество уравнений и неизвестных, и выбора способа задания системы, появляется меню, в котором в окно ввода предлагается ввести количество свободных коэффициентов и выбрать способ их задания.



В результате работы программы будет выведена исходная система уравнений и вектор её решений(или же сообщение о том, что решений нет или их бесконечно много), так же будет выведен результат проверки решений на корректность и размер погрешности

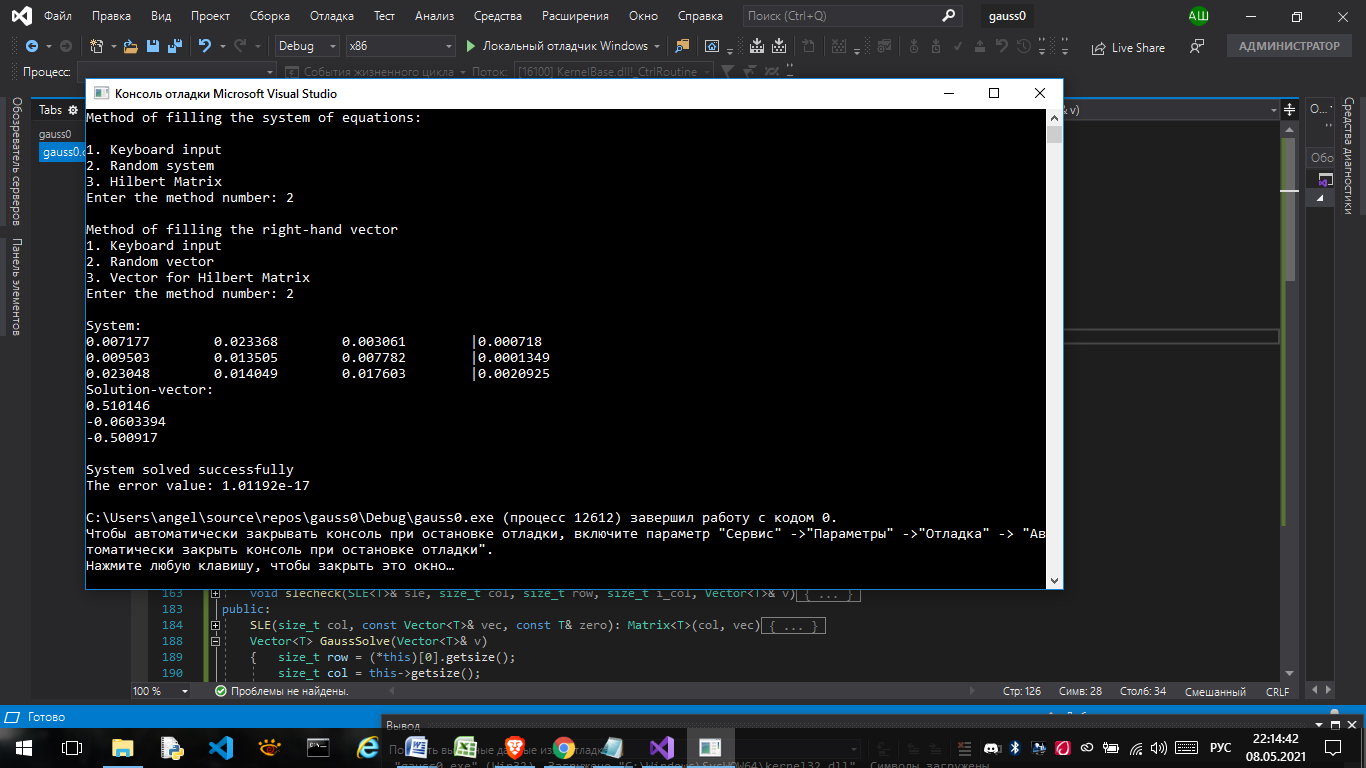


# Описание программной реализации

|  |  |  |
| --- | --- | --- |
| Список функций/ классов | Что делает | Аргументы/Свойства и методы |
| class Vector | Представляет собой модель арифметического вектора | Поля: размер size вектора, указатель на выделенный участок памяти  Методы: Конструкторы(копирования, деструктор) getsize() взятие размера вектора, перегрузки бинарных операций(+,-,\*(умножение вектора на скаляр), перегрузка операции присваивания, логических операций(==,!=), операции индексирования, перегрузка операций ввода и вывода |
| class Matrix | Наследник класса Vector Vector<Vector<T>>, является моделью квадратной матрицы | Содержит перегрузку операции \* как умножение матрицы на вектор |
| class SLE | Наследник класса Matrix, модель системы линейных уравнений | Содержит в себе функции для работы с СЛАУ, рассмотренные ниже, и метод Гаусса для нахождения вектора решений СЛАУ |
| int maxsrch | Ищет максимальный элемент в заданном столбце, так же считает ненулевые элементы, чтобы найти возможные свободные переменные | Принимает ссылку на СЛАУ, размер столбца и индекс текущего элемента в столбце, ниже которого ищется наибольший |
| bool nullcol | Поиск нулевых столбцов | Принимает ссылку на СЛАУ, количество уравнений и неизвестных |
| void slecheck | Поиск свободных переменных и проверка системы на нулевые строки | Принимает ссылку на СЛАУ, количество уравнений и неизвестных |
| Vector<T> Gauss | Принимает правую часть СЛАУ и возвращает вектор её решений. Прямой ход Гаусса запускается в зависимости от проверки на наличие нулевых столбцов(иначе выдает сообщение о бесконечном числе решений), ход заключается в поиске максимального элемента в столбце, перестановке строк так, чтобы он оказался на главной диагонали. Затем строка нормируется(делится на элемент, лежащий на главной диагонали), а элементы, лежащие ниже ведущего зануляются. Обратный ход запускается если решение единственное. | Принимает ссылку на вектор свободных коэффициентов, возвращает вектор решений при условии, что решение единственное, если решений бесконечно много или не существует, то выводится нулевой вектор. |
| void check | Производит проверку решения умножением матрицы СЛАУ на вектор решений | Принимает ссылки на СЛАУ, вектор свободных коэффициентов, вектор решения |

# Подтверждение корректности

Для подтверждения корректности решения системы линейных уравнений в программе используется вызов функции check которая производит проверку результата метода Гаусса путем умножения матрицы СЛАУ на вектор её решений. Если решение окажется корректным(сравнивается копия вектора свободных коэффициентов с результатом умножения матрицы на вектор решения)), будет произведен вывод фразы «System solved successfully», в худшем случае «System solved unsuccessfully» и в случаях, когда решений бесконечно много или не существует. Так же во всех случаях выводится значение погрешности



# Результаты экспериментов

Для уменьшения погрешности вычислений нами используется Метод Гаусса с выделением главного элемента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Значение погрешности*/Размер матрицы | 5х5 | 10х10 | 30х30 | 50х50 | 100х100 |
| Целочисленные значения | 2.20934e-15 | 1.80411e-17 | 1.32938e-16 | 6.67339e-16 | 2.85605e-16 |
| Дробные значения | 5.68989e-17 | 1.89085e-17 | 2.23895e-16 | 6.27901e-16 | 6.72951e-16 |
| Матрица Гильберта | 1.05247e-08 | 0.5>10e-8 | 1.68408e+07>10e-8 | 4.28416e+09>10e-8 | 3.17952e+11>10e-8 |

**Тесты:**

|  |  |  |
| --- | --- | --- |
| Системы | Программа | Ответ |
|  |  | **(**0.791,-1.191,1.083,0.777) |
|  |  | (0.8,1,1.2,1.4) |
|  |  | Бесконечно много решений |
| https://www.mathelp.spb.ru/book1/Gauss%20Elimination%20Method_files/image18.jpg |  | Нет решений |
| https://function-x.ru/chapter3/sys153.gif |  | **(**0.666,-2.388,1.444,-0.388) |
| https://function-x.ru/chapter3/sys173.gif |  | Бесконечно много решений |
|  |  | Нет решений |

# Заключение

В ходе работы была написана программа, решающая системы линейных уравнений методом Гаусса, содержащая шаблонные классы, реализующие модели вектора, матрицы и СЛАУ. Так же были описаны руководство пользователя и способ проверки результата решения системы на корректность. Были приведены результаты экспериментов.

# Приложение

class Vector {

protected:

T\* data;

size\_t size;

public:

Vector() {

size = 0;

data = new T[0];

}

Vector(size\_t size) {

this->size = size;

data = new T[size];

}

Vector(size\_t size, T v) {

this->size = size;

data = new T[size];

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

data[i] = v;

}

Vector(const Vector& vec)

{

size = vec.size;

data = new T[size];

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

data[i] = vec.data[i];

}

~Vector() {

delete[] data;

}

size\_t getsize() { return size; }

friend ostream& operator<<(ostream& ost, const Vector& vec) {

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

ost << vec.data[i] << "\t";

}

return ost;

}

friend istream& operator>>(istream& ist, const Vector& vec) {

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

ist >> vec.data[i];

return ist;

}

Vector& operator=(const Vector& vec) {

if (this == &vec) return \*this;

if (size != vec.size) {

size = vec.size;

delete[] data;

data = new T[size];

}

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

data[i] = vec.data[i];

return \*this;

}

Vector operator+(const Vector& vec) {

Vector result = \*this;

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

{

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

}

return result;

}

Vector operator- (const Vector& vec) {

Vector result = \*this;

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

{

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

}

return result;

}

Vector operator\* (const T& n) {

Vector result = \*this;

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

{

result.data[i] \*= n;

}

return result;

}

Vector operator/(const T& n) {

Vector result = \*this;

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

{

result.data[i] /= n;

}

return result;

}

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

return data[i];

}

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

return data[i];

}

bool operator==(const Vector& vec) {

if (this == &vec) return true;

if (size != vec.size) return false;

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

{

if (data[i] != vec.data[i]) return false;

}

return true;

}

bool operator!= (const Vector& vec) {

return !(\*this == vec);

}

};

template <class T>

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

public:

Matrix(size\_t size, const Vector<T>& vec): Vector<Vector<T>>(size, vec) {}

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

size\_t col = this->getsize();

size\_t row = (\*this)[0].getsize();

Vector<T> result(col, 0);

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

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

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

}

}

return result;

}

};

template <class T>

class SLE : public Matrix<T> {

protected:

double null = 0;

bool flagsol; // есть ли единственное решение

int maxsrch(const SLE<T>& sle, size\_t col, size\_t i\_col)// поиск максимума в столбце

{ T max\_el = sle[i\_col][i\_col]; //максимум

size\_t max\_ind = i\_col; // индекс максимума в столбце

int count = 0; // подсчет ненулевых элементов в столбце

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

if (sle[i][i\_col] != null) count++;

if (sle[i][i\_col] \* sle[i][i\_col] > max\_el \* max\_el) {

max\_el = sle[i][i\_col];

max\_ind = i;

}

}

if (count == 0) max\_ind = -1;

return max\_ind;

};

bool nullcol(const SLE<T>& sle, size\_t row, size\_t col) {//проверка нулевых столбцов

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

{ int count = 0;//количество ненулевых элементов в столбце

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

if (sle[j][i] != null) count++;

if (count == 0) return true;

}

return false;

};

void slecheck(SLE<T>& sle, size\_t col, size\_t row, size\_t i\_col, Vector<T>& v) // проверка на нулевые строки и свободные переменные

{ int count\_nozerorow = 0; // подсчет ненулевых строк

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

{ int count = 0; // подсчет ненулевых элементов строки

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

{ if (sle[i][j] != null) count++;

}

if ((count == 0) && (v[i] != null))

{ cout << "No solutions" << endl << endl;;

flagsol = false;

return;

}

else if (!(count == 0 && v[i] == null)) count\_nozerorow++;

}

if (count\_nozerorow < i\_col + 1)

{ cout << "Infinitely many solutions" << endl << endl;

flagsol = false;

return;

}

};

public:

SLE(size\_t col, const Vector<T>& vec, const T& zero): Matrix<T>(col, vec)

{ this->null = zero;

flagsol = true;

}

Vector<T> GaussSolve(Vector<T>& v)

{ size\_t row = (\*this)[0].getsize();

size\_t col = this->getsize();

size\_t ved; // индекс ведущего

Vector<T> sol\_vec(row);

Vector<T> zero\_vec(row);

// Прямой ход Гаусса

if (!(nullcol(\*this, col, row)))

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

{ slecheck(\*this, col, row, i, v);

if (!flagsol) return zero\_vec;

ved = maxsrch(\*this, col, i);

if (ved != -1)

{ swap<Vector<T>>((\*this)[i], (\*this)[ved]);

swap<T>(v[i], v[ved]);

T diag\_el = (\*this)[i][i];

(\*this)[i] = (\*this)[i] / diag\_el;

v[i] = v[i] / diag\_el;

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

{ T n = (\*this)[j][i];

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

v[j] = v[j] - (v[i] \* n);

}

}

else

{ cout << "Infinitely many solutions" << endl << endl;

flagsol = false;

return zero\_vec;

}

}

}

else {

cout << "No solutions or infinitely many solutions " << endl << endl;

flagsol = false;

return zero\_vec;

}

// Обратный ход Гаусса

for (int i = row - 1; i >= 0; i--)

{ sol\_vec[i] = v[i];

for (int j = row - 1; j >= i + 1; j--)

{ sol\_vec[i] = sol\_vec[i] - ((\*this)[i][j] \* sol\_vec[j]);

}

}

return sol\_vec;

}

};

template <class T>

void check(const SLE<T>& sle, const Vector<T>& sol, const Vector<T>& vec\_copy) {

Vector<T> check = (Matrix<T>)sle \* sol;

double eps = 10e-7;

bool checkflag = true;

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

{

p += abs(check[i] - vec\_copy[i]);

if (abs(check[i] - vec\_copy[i]) > eps) checkflag = false;

}

p/= check.getsize();

if (checkflag) cout << "System solved successfully" << endl << "The error value: " << p << endl;

else cout << "System solved unsuccessfully " << endl << "The error value: " << p << endl;

}