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**

**«Шаблонный метод Гаусса с выбором ведущего элемента»**

**Выполнил**:

студент группы 382003-1

Устинов А.В.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

# Содержание

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 6](#_Toc26962564)

[Описание программной реализации 9](#_Toc26962565)

[Подтверждение корректности 13](#_Toc26962566)

[Результаты экспериментов 14](#_Toc26962567)

[Заключение 16](#_Toc26962568)

[Приложение 17](#_Toc26962569)

# Постановка задачи

Задание требует написать программу, реализующую **метод Гаусса с выбором ведущего элемента.** Способ проектирования состоит в создании шаблонного класса **Vector**, наследованного от него как вектор векторов шаблонного класса **Matrix** и шаблонного класса **GaussianMethod**, наследованного от **Matrix** либо содержащего объект класса **Matrix** в себе. Класс **GaussianMethod** должен иметь метод **Solve**:он принимает вектор свободных коэффициентов системы уравнений и возвращает вектор-решение, если таковое единственно, в противном случае выдаётся сообщение об отсутствии решений или бесконечности их количества.

# Метод решения

Основным алгоритмом программы является **метод Гаусса** – метод решения системы линейных алгебраических уравнений, основанный на исключении переменных путём **элементарных преобразований**:

1) сложение одних уравнений с другими, умноженными на число;

2) умножение уравнений на число;

3) перестановка уравнений.

Эти действия удобно производить в матричной форме – СЛАУ при этом записывается в форме расширенной матрицы.

Пусть заданы матрица коэффициентов и .

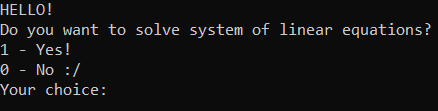
Стандартный метод Гаусса состоит из 2 этапов:

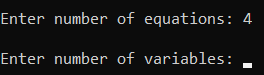
1) **Прямой ход** – последовательный выбор элементов матрицы в i-м столбце, , начиная с , в качестве ведущих и зануление остальных элементов в том же столбце путём прибавления к соответствующим строкам расширенной матрицы i-й строки, умноженной на подходящий коэффициент. Далее происходит переход к следующему справа столбцу, в котором элемент строкой ниже не равен нулю. Если на одной из итераций строка матрицы нулевая, а  **–** нет, то система не имеет решений. Если же по окончании итераций количество ненулевых строк меньше количества переменных, то система имеет бесконечно много решений. Во всех остальных случаях, точнее, когда по окончании итераций количество ненулевых строк равно количеству переменных, решение есть и единственно.

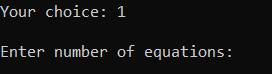
**2) Обратный ход** – при наличии решений происходит вычисление значений неизвестных. Пусть – столбцы , в которых выбирались ведущие элементы. Тогда неизвестные в соответствующих столбцах назовём зависимыми переменными, остальные – свободными переменными. Сначала выражается через и свободные переменные, затем – через свободные переменные, и также , и так далее: зависимые переменные выражаются через свободные, следующие за ними по индексу зависимые и элементы **.** При единственности решения свободных переменных нет, и каждое неизвестное будет получаться не зависящим от любых параметров.

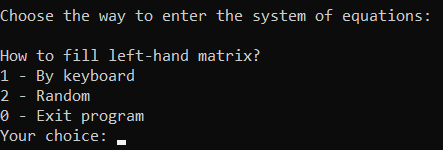
В данной работе приведён **метод Гаусса с выбором ведущего элемента**. Он неотличим от стандартного алгоритма за тем исключением, что на каждой итерации в качестве ведущего элемента выбирается не просто следующий ненулевой элемент, а наибольший по модулю в столбце. Оправдано это тем, что при реализации в программе при использовании в СЛАУ вещественных коэффициентов возможно деление на малый по модулю (в сравнении с другими элементами матрицы) элемент, что при следующих операциях приведёт к сильной потере точности. По этой причине стандартный метод вычислительно неустойчив. Выбор же наибольшего по модулю элемента значительно снижает потерю точности при делении и последующих действиях, поэтому модифицированный метод Гаусса более устойчивым.

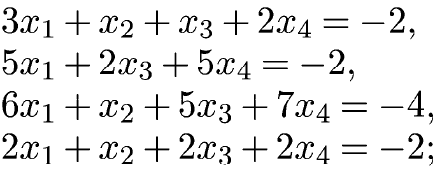
# Руководство пользователя

После запуска программы пользователю предлагается выбор: решать систему уравнений или выйти (далее при каждой возможности выбора также можно закрыть программу), выбрать нужное можно вводом соответствующего числа:

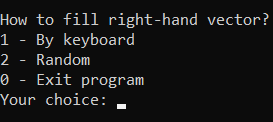
После выбора первого варианта предлагается ввести количество уравнений и затем количество неизвестных:



Далее требуется выбрать способ ввода системы уравнений: с клавиатуры или заполнением случайными числами (**замечание**: матрица коэффициентов при неизвестных и вектор свободных коэффициентов вводятся отдельно!):

При выборе варианта **1** потребуется ввести матрицу коэффициентов, числа вводятся построчно, так же, как располагаются коэффициенты в самой системе. **Пример:** для СЛАУ ниже ввод матрицы должен осуществляться так:

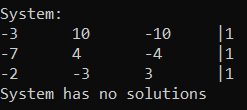
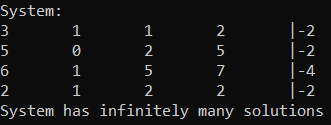
При выборе варианта 2 матрица заполняется случайными числами.

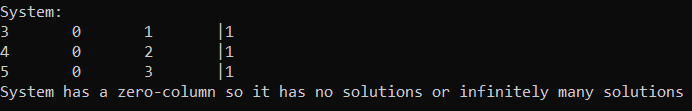
За вводом матрицы следует ввод вектора свободных коэффициентов, которые стоят в правой части каждого уравнения:



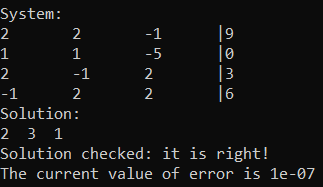
После выбора варианта **1** требуется ввести вектор свободных коэффициентов, как показано выше (вводится в строку!), вариант **2** же означает, что вектор заполняется случайными числами.

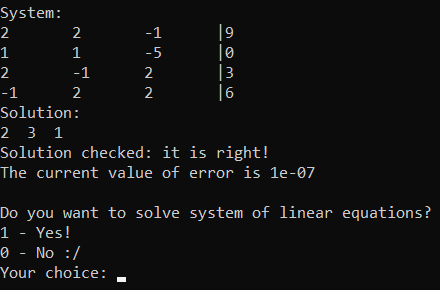
После заполнения системы происходит её вывод, чтобы можно было посмотреть значения в уравнениях, и затем выводится сообщение о наличии решений. Возможно следующее:

******1) Решений нет 2) Решений бесконечно много**

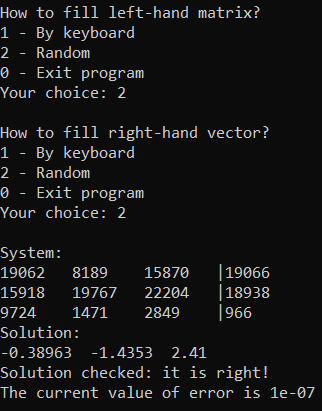
**3) В матрице коэффициентов есть нулевой столбец** (что лишено смысла с точки зрения СЛУ, т.к. будет содержаться фиктивная переменная, но этот случай должен обрабатываться)

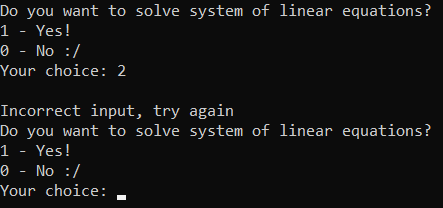
**4) Решение единственно** (тогда выводится решение; также производится его проверка и выводится сообщение, верно ли решение или нет при данной величине погрешности)



После решения данной системы пользователю снова предалагается выбрать, решать ли очередную СЛУ или выйти:

Пример решения случайной СЛУ:

**Замечание:** при вводе неподходящего символа при выборе выдаётся сообщение о некорректности ввода, и предлагается выбрать заново:



Программа состоит из одного файла (**main.cpp**), в котором реализованы все функции: функции сортировок, вспомогательные функции и главная (**main**).

# Описание программной реализации

Проект состоит из нескольких файлов:

* **Main.cpp** – файл с функцией **main**, вызывает функцию **menu** из **menu.cpp**.
* **Menu.cpp** – файл с реализацией пользовательского интерфейса, содержит функцию **menu**.
* **Menu.h** – заголовочный файл для **Menu.cpp**, содержит объявление **menu**.
* **TemplateVector.h** – заголовочный файл с реализацией шаблонного класса **Vector** и функцией **error,** которая выводит сообщения о состоянии СЛУ и ошибках при работе с **Vector**.
* **MatrixGauss.h** – заголовочный файл с реализацией шаблонных классов **Matrix** и **GaussianMethod**.

Класс **Vector<T>:** реализует вектор, поля класса - динамический массив **data** элементов типа **T** и размер массива **size**. В классе определены **конструктор с параметром** (принимает размер и образец элемента для заполнения массива), **конструктор копирования**, **операторы >> и <<** (ввод и вывод вектора соответственно), **оператор =** (присваивание размера и каждого элемента принимаемого вектора) и **арифметические операторы** **+, -, +=, -=** (поэлементное сложение/вычитание векторов, принимают вектор), **\*, /, \*=, /=** (умножение/деление на коэффициент, принимают объект типа **T**), **оператор []** (возвращает элемент вектора под заданным индексом), **логические операторы ==** и **!=** (поэлементное сравнение векторов) и метод **GetSize** (возвращает размер вектора).

Класс **Matrix<T>:** наследник класса **Vector<Vector<T>>** (вектор векторов), реализует матрицу, из нового содержит в себе только перегруженный **оператор \*** для матричного умножения (принимает вектор, возвращает вектор).

Класс **GaussianMethod<T>:** основной класс программы, наследник **Matrix<T>,** реализует решение СЛАУ.Новые поля: **zero** - нулевой элемент типа **T** (необходим для сравнения с ним элементов системы) и **HasUniqueSolution** типа **bool** (отражает наличие единственного решения системы).

Методы:

* **Конструктор с параметром** (принимает количество уравнений, образец строки для заполнения и ноль типа **T**).
* **bool ZeroColumnCheck(const GaussianMethod<T>& system, size\_t row, size\_t column)** – проверка матрицы коэффициентов на наличие нулевого столбца. Его наличие означает, что система не может иметь единственного решения.
* **void CheckSystem(GaussianMethod<T>& system, size\_t column\_size, size\_t row\_size, size\_t current\_column, Vector<T>& b)** – проверка состояния системы на каждой итерации метода Гаусса. Метод производит проверку каждой строки расширенной матрицы. Если в какой**-**то строке все коэффициенты в левой части нулевые, а в правой – нет, система не имеет решений. Также подсчитывается количество ненулевых строк: если номер текущей итерации превосходит это число, то невозможно выбрать очередной ведущий элемент в столбце, что означает наличие свободных переменных и отсутствие единственного решения. При выполнении одного из двух последних условий значение флага **HasUniqueSolution** становится **false**, а на экран выводится сообщение об отсутствии решений или их бесконечном количестве.
* **int SearchMaxInColumn(const GaussianMethod<T>& system, size\_t column\_size, size\_t column)** – поиск ведущего элемента в столбце от строки с индексом **column** и ниже. Ищется наибольший по модулю ненулевой элемент, и возвращается его индекс. Если же все элементы нулевые, переменная, соответствующая данному столбцу, свободная, и единственного решения нет; при этом возвращается значение -1.
* **void MatrixZeroTrimmer(GaussianMethod<T>& system, size\_t m, size\_t n)** – зануление близких к нулю элементов матрицы при вещественном типе **T**. В методе Гаусса происходит зануление элементов в столбце вычитанием из одной строки другой, и из-за особенностей вещественных типов результат вычитания может оказаться ненулевым, что может существенно повлиять на итоговое решение СЛУ. Для устранения этой проблемы и используется этот метод: элемент становится равным нулю, если его значение по модулю меньше **eps** (погрешность можно менять, на данный момент её значение – ).
* **void VectorZeroTrimmer(Vector<T>& vector, size\_t n)** – то же, что для матрицы, только для вектора свободных коэффициентов.
* **void CheckSolution(const GaussianMethod<T>& system, const Vector<T>& sol, const Vector<T>& rhs)** – проверка полученного решения системы (см. [**Подтверждение корректности**](#_Подтверждение_корректности)).
* **Vector<T> Solve(Vector<T>& b)** – метод, осуществляющий собственно решение СЛУ. Принимает вектор свободных коэффициентов и возвращает вектор-решение.

**Прямой ход:** вызов **ZeroColumnCheck**, затем начало итераций метода. В каждой итерации вызывается **CheckSystem**, затем **SearchMaxInColumn.** Если есть ведущий элемент, его строка ставится на главную диагональ, затем она делится на этот элемент, и зануляются все элементы в том же столбце ниже ведущей строки вычитанием этой строки, умноженной на соответствующий коэффициент. Далее применяются **MatrixZeroTrimmer** и **VectorZeroTrimmer**. Если на какой-то итерации выясняется, что у системы нет единственного решения, немедленно возвращается нулевой вектор (вектор, содержащий только нули). Обработка этого осуществляется в **menu.cpp** на основании значения **HasUniqueSolution** (см. описание **menu.cpp**).

**Обратный ход:** если единственное решение всё-таки есть, матрица левой части будет верхней треугольной. Начинается вычисление элементов вектора-решения **solution**. Пусть – матрица в левой части, – вектор-столбец в правой части, – вектор-решение, тогда (индексация в стиле C++):

, .

Далее вызываются **VectorZeroTrimmer** для **solution**, **CheckSolution** для проверки, и возвращается **solution**.

**Menu.cpp: void menu()** – функция с пользовательским интерфейсом. Выводит сообщения и осуществляет приём и обработку вводимых данных ([см. выше](#_Руководство_пользователя)). Состоит из цикла без условия, который выполняется, пока пользователь не введёт 0 во время выбора, и 3 циклов, обрабатывающих ввод. Первый обрабатывает то, будет ли пользователь решать СЛУ или нет (если будет, то надо ввести количество уравнений и неизвестных), второй – заполнение матрицы левой части, третий – заполнение вектора правой части (заполнение случайными числами производится функцией **rand**). Если пользователь вводит некорректное значение на любой стадии выбора, выдаётся сообщение о некорректном вводе, и будет снова предлагаться сделать выбор до тех пор, пока не будет получено корректное значение.

В случае решения СЛУ происходит ввод количества уравнений и неизвестных, затем создаются необходимые объекты (матрица и векторы). Как только система введена, происходят её вывод и решение. Если значение **HasUniqueSolution** есть **true**, происходит вывод решения, иначе ничего не происходит, потому что сообщение о решениях системы уже будет выведено.

# Подтверждение корректности

Если решение СЛУ единственное, проверка корректности производится в методе **CheckSolution** (см. выше). Там осуществляется умножение матрицы левой части на вектор-столбец решения; если решение верное, должен получиться вектор-столбец правой части. Если векторы совпадают, выводится сообщение, что решение верное, в противном случае – сообщение, что решение неверное. Сравнение векторов идёт при заданной погрешности (на данный момент – ); она также выводится на экран.

Если единственного решения нет, проверка того, что это так, не осуществляется, и здесь всё зависит от погрешности при выполнении операций и применении **MatrixZeroTrimmer** и **VectorZeroTrimmer**.

# Результаты экспериментов

Тестирование производилось на 10 СЛУ над полем действительных чисел.

Прежде СЛУ были решены вручную, затем производились решение через программу и сравнение результатов.

|  |  |  |
| --- | --- | --- |
| СЛУ | Решение (вручную) | Решение (программа) |
|  | Бесконечно много решений |  |
|  | Бесконечно много решений |  |
|  | (-2, 2, -3, 3) |  |
|  | Бесконечно много решений |  |
|  | Нет решений |  |
|  | Нет решений |  |
|  | (2, 0, 0, 0) |  |
|  | Бесконечно много решений |  |
|  | (8, -5, 10,-5) |  |
|  | (2,0,-2,-2,1) |  |

На всех тестах программа отработала корректно.

# Заключение

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

# Приложение

**1) TemplateVector.h**

void error(int code) { // Выдача сообщений об ошибках и завершение программы

switch (code) {

case 1:

cerr << "Error: can't complete the operation - vectors have different length" << endl;

exit(1);

break;

case 2:

cerr << "System has no solutions" << endl << endl;

break;

case 3:

cerr << "System has infinitely many solutions" << endl << endl;

break;

case 4:

cerr << "System has a zero-column so it has no solutions or infinitely many solutions" << endl << endl;

break;

default:

cout << "Can't do anything with this input" << endl << endl;

break;

}

}

template <typename T>

class Vector {

protected:

T\* data;

size\_t size;

public:

Vector(size\_t \_size = 0, const T& sample = T()) {

size = \_size;

data = new T[size];

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

data[i] = sample;

}

Vector(const Vector& vec) {

size = vec.size;

data = new T[size];

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

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

}

size\_t GetSize() { return size; }

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

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

ost << std::setprecision(5) << vec.data[i] << "\t";

}

return ost;

}

friend std::istream& operator>> (std::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) {

if (this->size != vec.size) error(1);

Vector result = \*this;

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

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

return result;

}

Vector operator- (const Vector& vec) {

if (this->size != vec.size) error(1);

Vector result = \*this;

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

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

return result;

}

Vector operator\* (const T& coefficient) {

Vector result = \*this;

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

result.data[i] \*= coefficient;

return result;

}

Vector operator/ (const T& coefficient) {

Vector result = \*this;

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

result.data[i] /= coefficient;

return result;

}

Vector& operator+= (const Vector& vec) {

if (this->size != vec.size) error(1);

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

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

return \*this;

}

Vector& operator-= (const Vector& vec) {

if (this->size != vec.size) error(1);

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

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

return \*this;

}

Vector& operator\*= (const T& coefficient) {

\*this = \*this \* coefficient;

return \*this;

}

Vector& operator/= (const T& coefficient) {

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

data[i] /= coefficient;

return \*this;

}

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

if (i >= size && i < 0) {

cout << "Reading out of bounds!!!" << endl;

exit(1);

}

else return data[i];

}

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

if (i >= size && i < 0) {

cout << "Reading out of bounds!!!" << endl;

exit(1);

}

else 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);

}

~Vector() {

delete[] data;

}

};

**2) MatrixGauss.h**

template <typename T>

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

public:

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

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

size\_t column\_size = this->GetSize();

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

Vector<T> result(column\_size);

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

for (int j = 0; j < row\_size; j++)

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

return result;

}

};

template <typename T>

class GaussianMethod : public Matrix<T> {

protected:

T zero; // Ноль типа Т (необходим для сравнения с ним элементов)

bool HasUniqueSolution; // Индикатор единственности решения

public:

GaussianMethod(size\_t column, const Vector<T>& sample, const T& zero) : Matrix<T>(column, sample) {

this->zero = zero;

HasUniqueSolution = true;

}

bool GetHasUniqueSolution() { return HasUniqueSolution; }

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

size\_t column\_size = this->GetSize(); // Количество строк в матрице

size\_t row\_size = (\*this)[0].GetSize(); // Количество столбцов в матрице (без учёта столбца свободных коэффициентов)

Vector<T> solution(row\_size, zero); // Вектор решения

size\_t max\_point; // Позиция ведущего элемента в столбце

Vector<T> zero\_vector(row\_size, zero); // Нулевой вектор (возвращается из метода, когда нет единственного решения)

// Прямой ход метода Гаусса

// Есть нулевой столбец => нет единственного решения

if (ZeroColumnCheck(\*this, column\_size, row\_size)) {

error(4);

HasUniqueSolution = false;

return zero\_vector;

}

else {

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

CheckSystem(\*this, column\_size, row\_size, i, b); // Проверка СЛАУ на решения

// Если нет решений/их бесконечно много

if (!HasUniqueSolution) return zero\_vector;

// Поиск наибольшего по модулю элемента в столбце

max\_point = SearchMaxInColumn(\*this, column\_size, i);

if (max\_point != -1) {

// Ставим ведущий элемент на главную диагональ

std::swap<Vector<T>>((\*this)[i], (\*this)[max\_point]);

std::swap<T>(b[i], b[max\_point]);

// Нормирование строки

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

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

b[i] /= diag\_elem;

// Зануляем всё ниже ведущего элемента

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

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

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

b[j] -= b[i] \* coeff;

}

// Зануляем близкие к нулю значения вещественных типов в матрице

MatrixZeroTrimmer(\*this, column\_size, row\_size);

VectorZeroTrimmer(b, column\_size);

}

// Если рассмотренные элементы в столбце - нули, есть свободная переменная => решений бесконечно много

else {

error(3);

HasUniqueSolution = false;

return zero\_vector;

}

}

}

// Обратный ход метода Гаусса

// Вычисление вектора-решения, если решение единственное

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

solution[i] = b[i];

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

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

}

VectorZeroTrimmer(solution, row\_size); // Зануляем близкие к нулю значения вещественных типов в решении

CheckSolution(\*this, solution, b); // Проверка решения

return solution;

}

};

**3) menu.cpp**

#define current\_zero 0 // Ноль текущего типа данных для СЛУ

typedef double current\_type; // Текущий тип данных для СЛУ

#pragma warning (push)

#pragma warning (disable: 4244)

using std::cout;

using std::cin;

using std::endl;

void menu() {

cout << "HELLO!" << endl;

short code0, code1, code2; // Переменные для хранения вариантов выбора при вводе

int column\_size, row\_size; // Кол-во уравнений и переменных в системе

for(;;) {

short flag0 = 0, flag1 = 0, flag2 = 0; // Флаги для обработки некорректного ввода

// Будет ли юзер решать систему или выйдет из программы? 1 - да, 0 - нет (закрытие программы)

cout << "Do you want to solve system of linear equations? " << endl << "1 - Yes!" << endl << "0 - No :/" << endl;

cout << "Your choice: ";

cin >> code0;

cout << endl;

// Обработка выбора о решении СЛУ

do {

switch (code0) {

case 0:

exit(0);

default:

// Введено что-то кроме 0 либо 1

cout << "Incorrect input, try again" << endl;

cout << "Do you want to solve system of linear equations? " << endl << "1 - Yes!" << endl << "0 - No :/" << endl;

cout << "Your choice: ";

cin >> code0;

cout << endl;

break;

case 1: {

// Юзер будет решать систему

flag0 = 1;

// Ввод количества уравнений

do {

cout << "Enter number of equations: ";

cin >> column\_size;

cout << endl;

if (column\_size <= 0) cout << "Incorrect input, try again" << endl;

} while (column\_size <= 0);

// Ввод количества неизвестных

do {

cout << "Enter number of variables: ";

cin >> row\_size;

cout << endl;

if (row\_size <= 0) cout << "Incorrect input, try again" << endl;

} while (row\_size <= 0);

cout << endl;

Vector<current\_type> sample(row\_size); // Строка-образец для инициализации матрицы

GaussianMethod<current\_type> system(column\_size, sample, current\_zero); // Матрица коэффициентов

Vector<current\_type> right\_hand\_vector(column\_size); // Вектор-столбец правой части

Vector<current\_type> solution(row\_size); // Вектор решения

Vector<current\_type> zero\_vector(row\_size); // Нулевой вектор (индикатор отсутствия единственного решения)

cout << "Choose the way to enter the system of equations: " << endl << endl;

// Как заполнить матрицу? 1 - с клавиатуры, 2 - случайно; 0 - закрыть программу

cout << "How to fill left-hand matrix?" << endl

<< "1 - By keyboard" << endl << "2 - Random" << endl << "0 - Exit program" << endl;

cout << "Your choice: ";

cin >> code1;

cout << endl;

// Обработка выбора о заполнении матрицы

do {

switch (code1) {

case 0:

exit(0);

case 1:

flag1 = 1;

cout << "Enter the matrix of coefficients in left-hand side: " << endl;

cin >> system; // Ввод матрицы коэффициентов в левой части

cout << endl;

break;

case 2:

flag1 = 1;

srand(time(NULL));

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

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

system[i][j] = rand();

break;

default:

// Введено что-то кроме 0, 1 либо 2

cout << "Incorrect input, try again" << endl;

cout << "How to fill left-hand matrix?" << endl

<< "1 - By keyboard" << endl << "2 - Random" << endl << "0 - Exit program" << endl;

cout << "Your choice: ";

cin >> code1;

cout << endl;

break;

}

} while (flag1 == 0);

// Как заполнить вектор свободных коэффициентов? 1 - с клавиатуры, 2 - случайно; 0 - закрыть программу

cout << "How to fill right-hand vector?" << endl

<< "1 - By keyboard" << endl << "2 - Random" << endl << "0 - Exit program" << endl;

cout << "Your choice: ";

cin >> code2;

cout << endl;

// Обработка выбора о заполнении правого вектора

do {

switch (code2) {

case 0:

exit(0);

case 1:

flag2 = 1;

cout << "Enter the coefficients of right-hand vector: " << endl;

cin >> right\_hand\_vector; // Ввод столбца свободных коэффициентов в правой части

cout << endl;

break;

case 2:

flag2 = 1;

srand(time(NULL));

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

right\_hand\_vector[i] = rand();

break;

default:

// Введено что-то кроме 0, 1 либо 2

cout << "Incorrect input, try again" << endl;

cout << "How to fill right-hand vector?" << endl

<< "1 - By keyboard" << endl << "2 - Random" << endl << "0 - Exit program" << endl;

cout << "Your choice: ";

cin >> code2;

cout << endl;

break;

}

} while (flag2 == 0);

// Вывод системы (расширенная матрица) и её решение

cout << "System: " << endl;

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

cout << system[i] << "|" << right\_hand\_vector[i] << endl;

solution = system.Solve(right\_hand\_vector);

if (system.GetHasUniqueSolution()) { // Если решение единственно, выводим его

// Вывод решения в аккуратном виде (не через оператор << класса Vector, там возникли неудобства)

cout << "Solution: " << endl;

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

cout << std::setprecision(5) << solution[i] << " ";

}

cout << endl << endl;

}

break;

}

}

} while (flag0 == 0);

}

}