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

Шокуров Д. А.

**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

Задача данной лабораторной работы – реализовать программу для решения системы линейных уравнений методом Гаусса. Для этого использовать шаблонный класс матриц, наследуемый от вектора векторов.

# Метод решения

Метод Гаусса

Идея:

1. Привести матрицу к ступенчатому виду.

Выбрать среди элементов столбца наибольший ненулевой элемент и переместить строку, содержащую данный элемент, в верхнее положение, делая ее первой. Далее вычесть из каждой следующей строки первую, умноженную на отношение первого элемента этих строк на первый элемент первой строки. Далее повторить процедуру, для следующих столбов, без учета всех предыдущих столбцов и предыдущих строк. Если среди элементов столбца нет ненулевых, то перейти к следующему столбцу.

1. Выражение базисных переменных

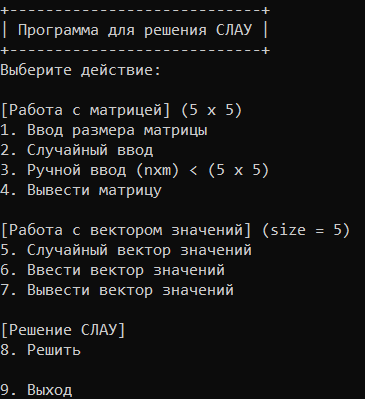
Начиная с последней строки подставлять получившиеся значения базисных переменных в строки выше. Если в результате выражения квадратной СЛУ оказалась нулевая строка, то либо решений бесконечно много (если элемент вектора значений напротив соответствующей базисной переменной равен нулю), либо решений нет (если элемент вектора значений не равен нулю). В остальных случаях решение существует и единственно.

Сложность алгоритма – O (), где N – размер квадратной матрицы

# Руководство пользователя

1.1 Главное меню

Главное меню программы содержит 9 команд: 4 по работе с матрицей, 3 для работы с вектором значений, команду «Решить» и команду «Выход». Для выбора команды необходимо ввести в консоль её номер.

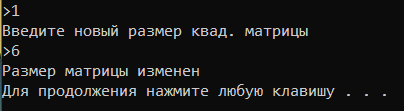


1.1 Главное меню

Далее приведено описание команд:

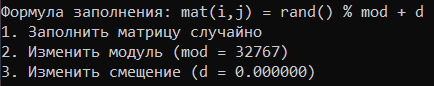
**Работа с матрицей**

1) «Ввод размера матрицы»: Для выбора введите 1. Для изменения размера матрицы введите число, равное её новому размеру.



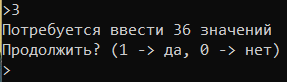
1.2 Окно выбора размера матрицы

2) «Случайный ввод»: Для выбора введите 2. Выводит меню для настройки диапазона значений и смещения случайных чисел для заполнения матрицы



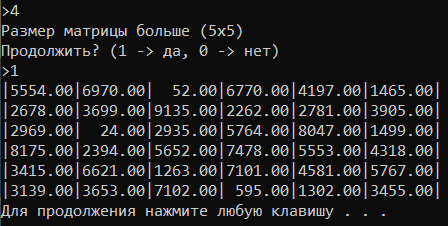
1.3 Меню настройки заполнения матрицы случайными числами

3) «Ручной ввод»: Для выбора введите 3. Введитечисел для заполнения матрицы. При размере матрицы больше попросит подтвердить ввод. Для подтверждения введите 1, для отмены 0.



1.4 Подтверждение ручного ввода

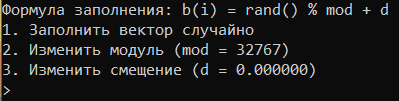
4) «Вывести матрицу»: Для выбора введите 4. Выводит матрицу



1.5 Вывод матрицы

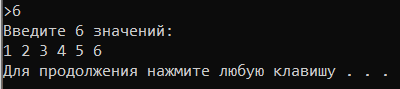
**Работа с вектором значений**

5) «Случайный вектор значений»: Для выбора введите 5. По аналогии с командой «случайный ввод» предлагает заполнить вектор значений.



2.1 Меню настройки заполнения вектора значений случайными числами

6) «Ввести вектор значений»: Для выбора введите 6. Введите N чисел для заполнения вектора значений

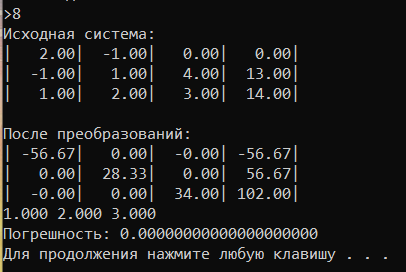


2.2 Заполнение вектора значений

7) «Вывести вектор значений»: Для выбора введите 7. Выводит вектор значений

**Решение СЛАУ**

8) «Решить»: Для выбора введите 8. Находит решение СЛАУ методом Гаусса. Выводит решение если оно есть с учетом погрешности. Если система несовместна, то выводит сообщение «Решений нет». В случае совместности системы при линейной зависимости строк выводит сообщение «Решений бесконечно много».

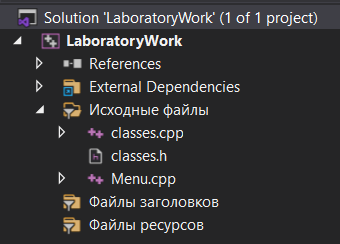


2.3 Пример работы команды «Решить»

9) «Выход»: Для выбора введите 9. Завершает работу программы.

# Описание программной реализации

Для удобства программа была разделена на два файла «classes.cpp», «Menu.cpp». К «classes.cpp» прилагается заголовок.



2.1 Решение

Файл «Menu.cpp» содержит точку входа в программу - функцию main(), в которой вызываются два метода Initialize() и UpdateMenu()

**Menu.cpp**

В данном файле располагается реализация меню программы.

Глобальные переменные:

matrix<double> mat – матрица с вещественными коэффициентами

vector<double> b – вектор значений

int limit – ограничения на ручной ввод и вывод

int mod – модуль, по которому будут генерироваться случайные числа

double d – переменная для хранения смещения диапазона значений случайных чисел

const int Exit – код выхода из программы

**void UpdateMenu();**

Для обработки ввода в консоль используется метод UpdateMenu(), который отображает главное меню программы и обрабатывает ввод консоли. При наличии команды, соответствующей введенному числу, данный метод вызывает ее, в противном случае выводит в консоль «Команда не найдена».

**Initialize();**

Предварительная инициализация переменных n, limit, mod, d и присваивание размеров обьектам matrix<double> mat и vector<double> b.

Вызов дальнейших методов зависит от введенного в консоль числа.

1. **void input\_size();**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 1. В данном методе осуществляется проверка на корректность введенных данных и вызов у обьектов mat и b методов resize(), которые меняют размер матрицы и вектора значений.

1. **void random\_input();**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 2. Вызывает меню настроек случайного ввода и заполняет матрицу случайными числами типа double.

1. **void input\_matrix();**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 3.

В данном методе происходит запрос на ввод чисел для заполнения матрицы.

1. **void show\_matrix()**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 4.

Выводит элементы матрицы в консоль с точностью до двух знаков после запятой.

1. **void input\_vector()**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 5.

В данном методе происходит запрос на ввод N чисел для заполнения вектора значений

1. **void random\_vector()**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 6. Вызывает меню настроек случайного ввода и заполняет вектор значений случайными числами типа double.

1. **void show\_vector()**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 7. Выводит вектор значений в консоль с точностью до трех знаков после запятой.

1. **void solve\_system()**

Данная функция вызывается из UpdateMenu() в случае ввода пользователем 8. Создает обьект класса System<double> и вызывает у него метод solve(), который возвращает результат выполнения вычислений и в случае единственности решения заносит результат в вектор значений с его последующим выводом в консоль с учетом погрешности.

**Classes.cpp**

В данном файле хранится реализация шаблонных классов vector<T>, Line<T>, matrix<T> и System<T>.

**vector<T> -** шаблонный класс для динамического выделения массива данных.

**переменные:**

T\* data – указатель на массив

size\_t \_size – размер массива

**Конструкторы:**

**vector() –** конструктор по умолчанию. Вызывает vector (size\_t size) с size=0.

**vector(size\_t size)** – конструктор с параметром. Принимает в качестве параметра size размер вектора.

**vector(const vector<T>& second)** – конструктор копирования. Вызывает конструктор vector(size\_t size) с передачей размера вектора и копирует данные в свой массив.

**Методы:**

**C модификатором static:**

**void copy\_data(T\* to\_array, T\* from\_array, size\_t size)** – функция для переноса size элементов из массива from\_array в to\_array. Обычно вызывается с присваиванием переменной size размера меньшего из двух массивов. Не является публичным.

**void copy\_data (vector<T>& to, const vector<T>& from)** – перегруженная версия copy\_data. Принимает два вектора из которого и в который будет происходить копирование массивов. Вызывает стандартный copy\_data(), с указанием в качестве size размера меньшего из массивов. Не является публичным.

**void swap(vector<T>& x, vector<T>& y) –** меняет указатели и размеры векторов x и y местами. Является публичным.

**Без модификатора static:**

**void realloc(size\_t new\_size)** – функция для перевыделения памяти под массив data с удалением данных. Используется в операторе присваивания, так как оператор не требует сохранения исходного массива data. Не является публичным.

**void resize(size\_t new\_size) –** функция для изменения размера массива data с сохранением данных. Если новый размер больше старого, тогда перевыделяет память для массива и переносит все значения в новый массив. Если новый размер меньше старого, тогда также перевыделяет память, но в новый массив попадают элементы с индексами от 0 до new\_size – 1. Если новый размер равен старому, то ничего не происходит. Является публичным.

**Line<T> -** шаблонный класс для работы с элементами массива экземпляра класса vector<T>. Используется для удобной работы со строками матрицы и для ограничения возможностей по изменению размера строк матрицы.

**Конструкторы:**

**Line(vector<T>& v) –** конструктор с параметром. Принимает вектор, с элементами массива которого будут происходить арифметические операции.

**Переменные:**

T\*& data – ссылка на массив вектора

size\_t& \_size – ссылка на размер массива вектора

**Операторы:**

**Line<T>& operator = (const Line<T>& second) –** выполняет присваивание массиву data массива second.data

**Line<T>& operator += (const Line<T>& second) –** выполняет поэлементное сложение с присваиванием значений в исходный массив data.

**Line<T>& operator -= (const Line<T>& second)** – выполняет поэлементное вычитание с присваиванием значений в исходный массив data.

**Line<T>& operator \*= (const T& second)** - выполняет поэлементное умножение на элемент second.

**matrix : protected vector<vector<T>> -** шаблонный класс матриц

**Переменные:**

size\_t &\_n, m – размеры матрицы

vector<T>\*& \_data – ссылка на наследуемый массив vector<T>\* data

**Конструкторы**:

**matrix()** –конструктор по умолчанию. Создает матрицу размера 1х0

**matrix(size\_t n, size\_t m)** – конструктор с параметрами. Принимает в качестве n количество строк, в качестве m – количество столбцов матрицы.

**matrix(const matrix<T>& mat)** – конструктор копирования. Создает матрицу тех же размеров, что и mat и копирует значения.

**Функции:**

**void swap(size\_t u, size\_t v) –** меняет строки с номерами u и v местами, если они существуют и их номера не совпадают. Является публичным.

**void resize(size\_t new\_n, size\_t new\_m) –** меняет размер массива с сохранением данных (по аналогии с одноименным методом класса vector<T>). Является публичным.

**System<T> -** шаблонный класс для решения СЛАУ

**Переменные:**

matrix<T> \_mat – матрица коэффициентов

**Конструкторы:**

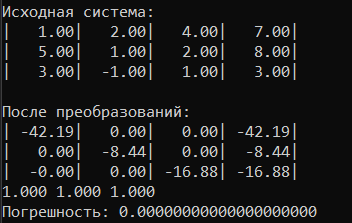
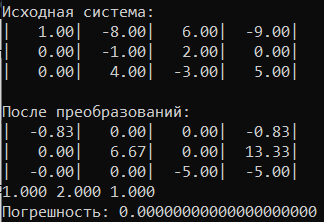
**System(const matrix<T>& mat) –** конструктор с параметрами. Принимает в качестве параметра матрицу коэффициентов.

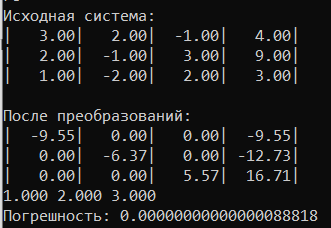
**Функции:**

**int gaussian\_method(vector<T>& b) –** функция для решения СЛАУ методом Гаусса. В качестве параметра принимает вектор значений. Возвращает 2 если решений бесконечно много, 1 если существует единственное решение и 0 если решения нет. Единственное решение хранится во входном векторе b.

# Подтверждение корректности

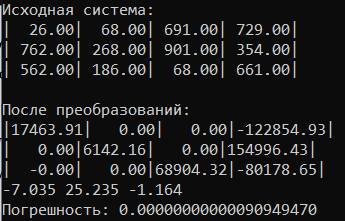
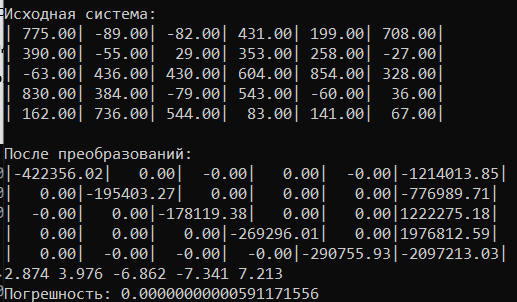
Для подтверждения корректности алгоритму были поданы СЛАУ, решения которых были найдены с помощью онлайн калькулятора



Кроме того, в программе присутствует участок кода, в котором, в случае единственности решения, исходная матрица коэффициентов умножается на ответ, найденный алгоритмом и сравнивается с исходным вектором b. Сумма разностей элементов b и столбца значений, полученного умножением, складывается по модулю и составляет так называемую погрешность, величину которой можно увидеть после завершения работы алгоритма в консоли программы.

Примеры:

Как можно заметить величина погрешности растет с увеличением значений матрицы и с увеличением размера матрицы

# Результаты экспериментов

По результатам экспериментов видно, что метод Гаусса корректно находит решение СЛАУ.

# Заключение

Выполняя лабораторную работу я реализовал класс матриц, наследуемый от вектора векторов и алгоритм для решения СЛАУ методом Гаусса.

# Приложение

//classes.cpp

#include <ostream>

#include <istream>

#include <iostream>

using namespace std;

template <typename T> class Line;

template <typename T> class vector {

protected:

T\* data;

size\_t \_size;

static void copy\_data(vector<T>& to, const vector<T>& from) {

copy\_data(to.data, from.data, min(to.\_size, from.\_size));

}

static void copy\_data(T\* to\_array, T\* from\_array, size\_t size) {

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

to\_array[i] = from\_array[i];

}

}

//reallocate memory with data deletion

void realloc(size\_t new\_size) {

if (new\_size == \_size)return;

delete[] data;

\_size = new\_size;

data = new T[\_size];

}

void set\_zero(T\* data, size\_t size) {

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

data[i] = 0;

}

}

public:

friend Line<T>;

vector() :vector(0) {}

vector(const vector<T>& second) :vector(second.\_size) {

copy\_data(\*this, second);

}

vector(size\_t size) {

\_size = size;

data = new T[\_size];

set\_zero(data, size);

}

size\_t size() {

return \_size;

}

//reallocate memory with data copying

void resize(size\_t new\_size) {

if (new\_size == \_size)return;

T\* temp\_data = new T[new\_size];

set\_zero(temp\_data, new\_size);

copy\_data(temp\_data, data, min(\_size, new\_size));

delete[] data;

data = temp\_data;

\_size = new\_size;

}

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

#ifdef DEBUG

if (index >= \_size) exit(1);

#endif // DEBUG

return data[index];

}

//copied how much fits

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

if (this != &second) {

//this->realloc(second.\_size);

copy\_data(\*this, second);

}

return \*this;

}

~vector() {

delete[] data;

}

static void swap(vector<T>& x, vector<T>& y) {

if (&x == &y)return;

T\* temp\_data = x.data;

size\_t temp\_size = x.\_size;

x.data = y.data;

x.\_size = y.\_size;

y.data = temp\_data;

y.\_size = temp\_size;

}

};

template<typename T> std::ostream& operator << (std::ostream& ost, vector<T>& second) {

size\_t si = second.size();

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

ost.precision(3);

ost.setf(std::ios::left);

ost.width(5);

ost << second[i] << " ";

}

return ost;

};

template <typename T> class Line

{

protected:

T\*& data;

size\_t& \_size;

public:

Line(vector<T>& v) : data(v.data), \_size(v.\_size) {}

size\_t size() {

return \_size;

}

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

#ifdef DEBUG

if (index >= \_size) exit(1);

#endif // DEBUG

return data[index];

}

void set\_zero() {

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

data[i] = 0;

}

}

Line<T>& operator = (const Line<T>& second) {

if (\_size != second.\_size) exit(1);

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

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

}

return \*this;

}

Line<T>& operator += (const Line<T>& second) {

if (\_size != second.\_size) exit(1);

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

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

}

return \*this;

}

Line<T>& operator -= (const Line<T>& second) {

if (\_size != second.\_size) exit(1);

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

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

}

return \*this;

}

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

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

data[i] \*= second;

}

return \*this;

}

};

//perceived as an array of strings

template <typename T> class matrix :

protected vector<vector<T>>

{

protected:

size\_t& \_n = this->\_size, \_m;

vector<T>\*& \_data = this->data;

public:

matrix() :matrix(1, 0) {}

matrix(size\_t n, size\_t m) {

if (n == 0) n = 1;

\_n = n; \_m = m;

\_data = new vector<T>[\_n];

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

\_data[i].resize(\_m);

}

}

matrix(const matrix<T>& mat) {

\_n = 1; \_m = 0;

\_data = new vector<T>[\_n];

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

\_data[i].resize(\_m);

}

(\*this) = mat;

}

void swap(size\_t u, size\_t v) {

if ((u >= \_n) || (v >= \_n))return;

vector<T>::swap(\_data[u], \_data[v]);

}

size\_t size() {

return \_n;

}

void resize(size\_t new\_n, size\_t new\_m) {

if (new\_n == 0) new\_n = 1;

if (new\_n == \_n && new\_m == \_m)return;

vector<T>\* temp\_data = new vector<T>[new\_n];

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

temp\_data[i].resize(new\_m);

if (i < \_n) {

temp\_data[i] = \_data[i];

}

}

delete[] \_data;

\_data = temp\_data;

\_n = new\_n; \_m = new\_m;

}

Line<T> operator[](size\_t index) {

#ifdef DEBUG

if (index >= \_n) exit(1);

#endif // DEBUG

return Line<T>(\_data[index]);

}

const Line<T> operator[](size\_t index) const {

#ifdef DEBUG

if (index >= \_n) exit(1);

#endif // DEBUG

return Line<T>(\_data[index]);

}

matrix<T>& operator = (const matrix<T>& second) {

this->resize(second.\_n, second.\_m);

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

\_data[i] = second.\_data[i];

}

return \*this;

}

matrix<T> operator \*(const matrix<T>& second) {

if (\_m != second.\_n) exit(567);

size\_t n = \_n, m = \_m, p = second.\_m;

matrix<T> ans(n, p);

//auxiliary variables

vector<T> temp(p);

Line<T> temp\_line(temp);

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

ans[line].set\_zero();

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

temp\_line.set\_zero();

temp\_line = second[count];

temp\_line \*= this->\_data[line][count];

ans[line] += temp\_line;

}

}

return ans;

}

};

template <typename T> std::ostream& operator <<(std::ostream& ost, matrix<T>& mat) {

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

ost << "|";

for (size\_t j = 0;j < mat[i].size();j++) {

ost.precision(2);

ost.setf(std::ios::right);

ost.width(7);

ost << mat[i][j] << "|";

}

ost << "\n";

}

return ost;

};

template <typename T> std::istream& operator >>(std::istream& ist, matrix<T>& mat) {

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

for (size\_t j = 0;j < mat[i].size();j++) {

ist >> mat[i][j];

}

}

return ist;

};

template<typename T> class System {

private:

matrix<T> \_mat;

public:

System(const matrix<T>& mat) {

\_mat = mat;

};

/\*if solve is inf then 2

if solve is one then return 1 and answer in b

if system have not answer then 0

if -1 is error\*/

int gaussian\_method(vector<T>& b) {

if (b.size() != \_mat.size()) return -1;

matrix<T> mat = \_mat;

mat.resize(mat.size(), mat[0].size() + 1);

size\_t position = mat[0].size() - 1;

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

mat[i][position] = b[i];

}

cout << "Исходная система:\n";

cout << mat << "\n";

size\_t mi = min(mat.size(), mat[0].size());

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

T max = 0;

size\_t number = leader;

//find max abs

for (size\_t num\_row = leader;num\_row < mat.size();num\_row++) {

if (abs(max) <= abs(mat[num\_row][leader])) {

max = mat[num\_row][leader];

number = num\_row;

}

}

mat.swap(number, leader);

if (mat[leader][leader] != 0) {

for (size\_t i = leader + 1;i < mat.size();i++) {

if (mat[i][leader] == 0)continue;

mat[i] \*= (mat[leader][leader] / mat[i][leader]);

mat[i] -= mat[leader];

}

}

}

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

if (mat[i][i] != 0) {

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

if (mat[j][i] != 0) {

mat[j] \*= (mat[i][i] / mat[j][i]);

mat[j] -= mat[i];

}

}

b[i] = mat[i][mat[0].size() - 1] / mat[i][i];

}

}

cout << "После преобразований:\n";

cout << mat;

bool flag = true;

bool inf = false;

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

flag = true;

for (int j = 0;j < mat[0].size() - 1;j++) {

if (mat[i][j] != 0)flag = false;

}

if (flag) {

if (mat[i][mat[0].size() - 1] != 0) {

return 0;//система несовместна

}

else {

inf = true;//решений бесконечно много

}

}

}

if (inf)return 2;

return 1;

};

};