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»**

**Выполнила**:

студентка группы 3821Б1ПМ2

Миронова А. А.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В. Д.

Нижний Новгород

2022

**Содержание**

Постановка задачи 3

Метод решения 4

Руководство пользователя 6

Описание программной реализации 8

Подтверждение корректности 9

Результаты экспериментов 10

Заключение 13

Приложение 14

# Постановка задачи

Целью лабораторной работы является реализация метода Гаусса с выбором ведущего элемента, используя реализованный самостоятельно шаблонный класс вектор, а также класс квадратная матрица, являющийся шаблоном класса вектор от вектора. Метод Гаусса должен принимать на вход систему линейных алгебраических уравнений, а выводить на экран вектор значений. Нужно описать реализацию и алгоритмы работы программы. Необходимо подтвердить корректность работы сознанного алгоритма.

# 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 состоит из 2 основных частей **–** прямой и обратный ход**.**

**Прямой ход:**

Исключения Гаусса основаны на последовательном исключении переменных по одной до тех пор, пока не останется только одно уравнение с одной переменной в левой части. 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. После того, как указанные преобразования были совершены, указанный процесс продолжается до тех поор, пока не останется уравнение с одной неизвестной.

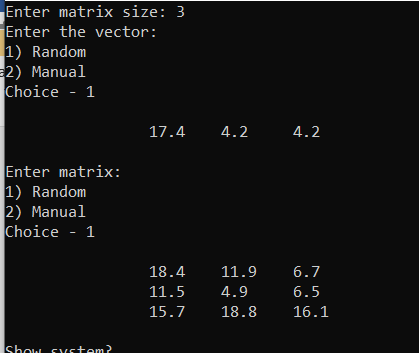
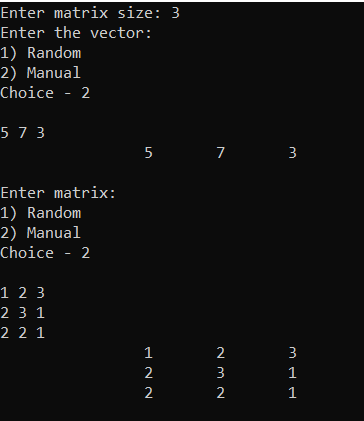
**Обратный ход:**

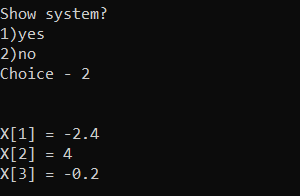
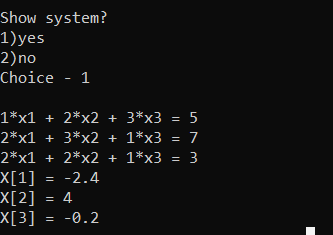
При наличии решений происходит вычисление значений неизвестных. Данная операция происходит последовательно от последнего уравнения к первому.

В работе приведён метод Гаусса с выбором ведущего элемента. На каждой итерации в качестве ведущего элемента выбирается наибольший по модулю в столбце, что значительно снижает потерю точности при делении и последующих действиях.

# Руководство пользователя

1. После запуска программы появляется окно ввода, где для начала предлагается ввести число, которое задает размерность квадратной матрицы;
2. Далее предлагается два варианта: рандомное заполнение(1) и ввод вручную(2) на ввод сначала вектора правой части, а потом матрицы;

 1. 2.

1. После ввода матрицы можно посмотреть полную систему линейных уравнений (1, чтобы показать систему и 2, для того чтобы это пропустить). Также сразу будет показан ответ в виде вектора или сообщение о том, что решений нет.

# Описание программной реализации

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

1. 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. Laba\_3.cpp – реализация 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

Тестирование производилось на 5 СЛУ. Вот часть из проверенных решений:

|  |  |  |
| --- | --- | --- |
| **СЛУ** | **Решение вручную** | **Решение программы** |
|  | (2; -3; -1) |  |
|  | (3; 1; 4; 6) |  |
|  | Нет решений (система несовместна) |  |
|  | Нет решений (система несовместна) |  |
|  | Бесконечно много решений |  |

СЛУ были решены вручную и с помощью онлайн калькулятора метода Гаусса, затем производились решение через программу и сравнение результатов.

На всех тестах программа отработала корректно. Для проверки использовался онлайн-калькулятор:

https://ru.onlinemschool.com/math/assistance/equation/gaus/

# Заключение

В ходе лабораторной работы был реализован метод Гаусса с выбором ведущего элемента, используя реализованный самостоятельно шаблонный класс вектор, а также класс квадратная матрица. Метод Гаусса принимает на вход систему линейных алгебраических уравнений, а выводит на экран вектор значений. Также была описана реализация и алгоритмы работы программы, экспериментально подтверждена корректность работы созданного алгоритма.

# 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)

{

return data[i];

}

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

{

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";

exit(-9);

}

}

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

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

}

};