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**

**«Решение систем линейных уравнений методом Гаусса с выбором ведущего элемента»**

**Выполнил**:

студент группы 381903-2

Тимаков Н. Г.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В. Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи. 3](#_Toc41659007)

[Метод решения**.** 4](#_Toc41659008)

[Руководство пользователя. 9](#_Toc41659009)

[Описание программной реализации. 11](#_Toc41659010)

[Подтверждение корректности. 12](#_Toc41659011)

[Заключение. 13](#_Toc41659012)

[Приложение 14](#_Toc41659013)

# Постановка задачи.

Реализовать программу, решающую систему линейных уравнений методом Гаусса с выбором ведущего элемента. Для этого необходимо написать шаблонный класс вектор. Унаследовать от него класс матрица, которая является шаблоном класса вектор от вектора. Унаследовать от класса квадратных матриц класс линейных уравнений, содержащий метод Гаусса.

**Метод решения.**

1.Реализован шаблонный класс вектор Vector с конструктором по умолчанию, возвращающим вектор размера 1, конструктором, создающим динамический массив из заданного количества элементов, методом print, выводящим вектор на экран и перегруженными операторами индексирования [] и присваивания =.

using namespace std;

template <typename T>

class Vector

{

public:

Vector()

{

SIZE = 1;

ptrVector = new T[SIZE];

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

{

ptrVector[i] = 0;

}

}

Vector(int \_SIZE, T\* value)

{

SIZE = \_SIZE;

ptrVector = new T[SIZE];

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

{

ptrVector[i] = value[i];

}

}

Vector(int \_SIZE, T value = T())

{

SIZE = \_SIZE;

ptrVector = new T[SIZE];

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

{

ptrVector[i] = value;

}

}

Vector(const Vector& other)

{

delete[]this->ptrVector;

this->SIZE = other.SIZE;

this->ptrVector = new T[other.SIZE];

for (int i = 0; i < other.SIZE; i++)

{

this->ptrVector[i] = other.ptrVector[i];

}

return \*this;

}

~Vector()

{

delete[] ptrVector;

}

T& operator[](int i)

{

return ptrVector[i];

}

Vector& operator = (const Vector& other)

{

delete[]this->ptrVector;

this->ptrVector = new T[other.SIZE];

for (int i = 0; i < other.SIZE; i++)

{

this->ptrVector[i] = other.ptrVector[i];

}

return \*this;

}

void printVector() const

{

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

{

cout << ptrVector[i] << "\n";

}

}

protected:

T\* ptrVector;

int SIZE;

};

2. Реализован класс Matrix – наследник Vector, являющийся шаблоном вектора Vector от вектора Vector.

using namespace std;

template <typename T>

class Matrix : public Vector <Vector <T> >

{

public:

Matrix(int N) : Vector<Vector <T>>(N, Vector <T>(N)) {}

};

3. Реализована шаблонная функция Мod, возвращающая модуль переданного в неё значения.

template <typename T>

T Mod(T x)

{

if (x < 0)

{

return -x;

}

else

{

return x;

}

}

4. Реализован шаблонный класс Linear\_System, наследуемый от класса Matrix. Данный класс содержит реализацию метода Гаусса Gauss, в ходе которого матрица приводится к треугольному виду, система проверяется на отсутствие любых или частных решений и выводится результат: решение системы или соответствующее сообщение о невозможности её решить, метод printMatrix, выводящий расширенную матрицу, метод Fill, позволяющий пользователю заполнить матрицу элементами, и метод swap, меняющий местами строчки.

template <typename T>

class Linear\_Sistem : public Matrix <T>

{

private:

T\* ptr;

public:

Linear\_Sistem(int N, T \_ptr = T()) : Matrix<T>(N)

{

ptr = new T[N];

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

{

ptr[i] = \_ptr;

}

}

~Linear\_Sistem()

{

delete[]ptr;

}

void printMatrix()

{

for (int i = 0; i < this->SIZE; i++)

{

for (int j = 0; j < this->SIZE; j++)

{

cout << this->ptrVector[i][j] << '\t';

}

cout << "\t| " << ptr[i];

cout << endl;

}

cout << endl;

}

void Fill()

{

for (int i = 0; i < this->SIZE; i++)

{

for (int j = 0; j < this->SIZE; j++)

{

cout << "Enter [" << j + 1 << "] equation" << "[" << i + 1 << "] coefficient:" << endl;

cin >> this->ptrVector[i][j];

}

}

for (int i = 0; i < this->SIZE; i++)

{

cout << "Enter " << i + 1 << " right values" << endl;

cin >> ptr[i];

}

}

void minus(int val1, int val2, T\* ptr, int index)

{

T ptrIndex = this->ptrVector[val2][index];

for (int i = 0; i + index < this->SIZE; i++)

{

this->ptrVector[val2][i + index] -= this->ptrVector[val1][i + index] \* ptrIndex / this->ptrVector[val1][index];

}

ptr[val2] -= ptr[val1] \* ptrIndex / this->ptrVector[val1][index];

}

void swap(int x, int y, T\* ptr)

{

if ((x < this->SIZE) && (y < this->SIZE))

{

T tmp;

tmp = ptr[x];

ptr[x] = ptr[y];

ptr[y] = tmp;

T\* ptrtmp = new T[this->SIZE];

for (int i = 0; i < this->SIZE; i++)

{

ptrtmp[i] = this->ptrVector[x][i];

this->ptrVector[x][i] = this->ptrVector[y][i];

this->ptrVector[y][i] = ptrtmp[i];

}

delete[]ptrtmp;

}

else

{

cout << "Error!";

exit(0);

}

}

Vector <T> Gauss()

{

for (int j = 0; j < this->SIZE; j++)

{

int i = 0;

int indexMax = i + j;

T max = Mod<T>(this->ptrVector[i + j][j]);

for (i + j + 1; i + j + 1 < this->SIZE; i++);

{

if (Mod<T>(this->ptrVector[i + j][j]) > max)

{

indexMax = i + j;

}

}

swap(j, indexMax, ptr);

for (int n = j + 1; n < this->SIZE; n++)

{

if (this->ptrVector[j][j] != 0)

{

for (int k = j; k < this->SIZE; k++)

{

this->ptrVector[n][k] += this->ptrVector[j][k] \* (-(this->ptrVector[n][j] / this->ptrVector[j][j]));

}

ptr[n] += ptr[j] \* (-(this->ptrVector[n][j] / this->ptrVector[j][j]));

}

else

{

cout << "Error! Division by zero!" << endl;

exit(3);

}

}

}

for (int i = 0; i < this->SIZE; i++)

{

T sum = 0;

for (int j = 0; j < this->SIZE; j++)

{

sum += Mod<T>(this->ptrVector[i][j]);

}

if ((sum == 0) && (ptr[i] == 0))

{

cout << "No solution!" << endl;

exit(3);

}

if ((sum == 0) && (ptr[i] != 0))

{

cout << "No solution!" << endl;

exit(2);

}

}

T\* result = new T[this->SIZE];

for (int i = this->SIZE - 1; i >= 0; i--)

{

T tmp = 0;

for (int j = i + 1; j < this->SIZE; j++)

{

tmp += this->ptrVector[i][j] \* result[j];

}

result[i] = (ptr[i] - tmp) / this->ptrVector[i][i];

}

return Vector<T>(this->SIZE, result);

}

};

# Руководство пользователя.

1. После компиляции программы откроется меню, в котором пользователю будет предложено ввести размер вектора квадратной матрицы.

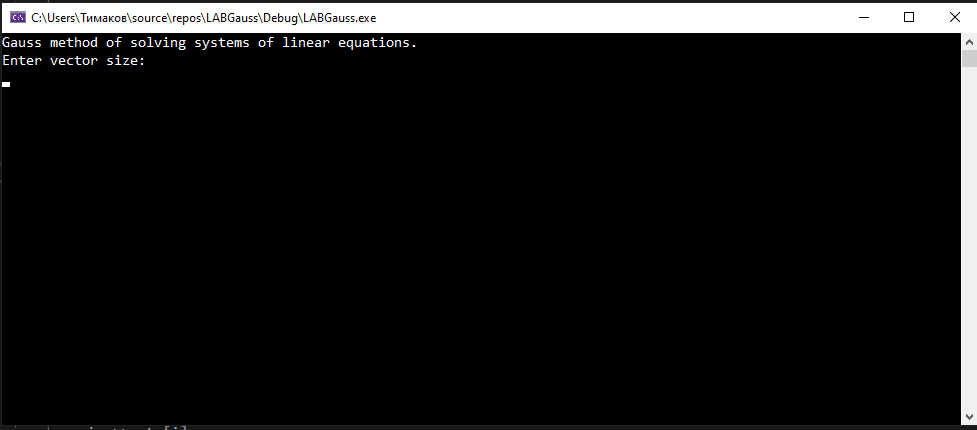


Рис. 1. Пользовательское меню. Ввод размера матрицы.

2. Затем пользователю будет предложено ввести коэффициенты уравнения

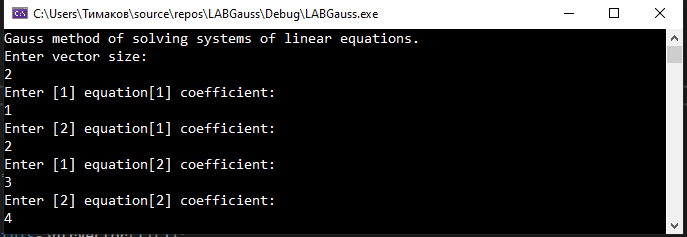


Рис. 2. Ввод коэффициентов в матрицу размером 2.

3. Требуется ввести значения правых частей уравнений.

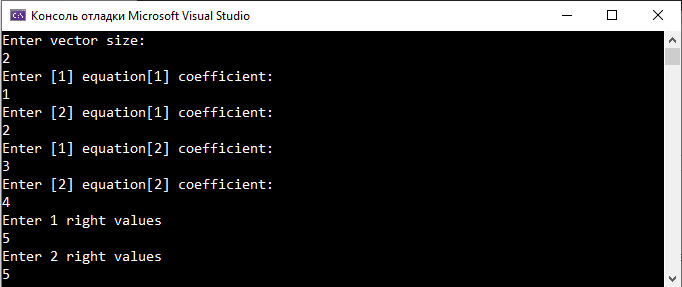


Рис. 3. Ввод значений правой части уравнения.

4. Проводится решение системы линейных уравнений, приведение матрицы к треугольному виду и вывод ответа.

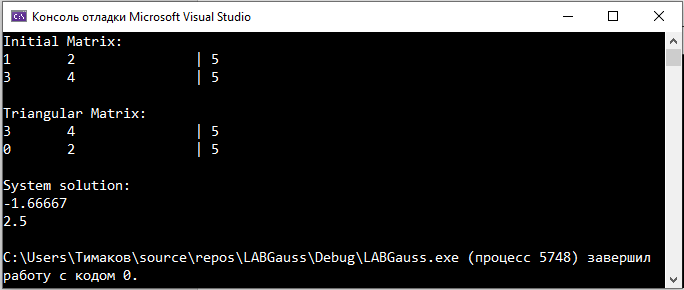


Рис. 4. Решение СЛАУ.

# Описание программной реализации.

1. LABGauss.cpp

Главный файл, который соединяет в единое целое все заголовочные файлы.

Подключены Vector.h, Matrix.h и Linear\_System.h.

2. Vector.h

Заголовочный файл с реализацией шаблонного класса My\_vector.

3. Mod.h

Заголовочный файл с реализацией шаблонной функции Mod.

4. Matrix.h

Заголовочный файл с реализацией шаблонного класса квадратных матриц Matrix.

Подключен Vector.h.

5. Linear\_System.h

Заголовочный файл с реализацией шаблонного класса Linear\_System.

Подключены Vector.h, Matrix.h и Mod.h.

# Подтверждение корректности.

Проверим корректность алгоритма решения методом Гаусса. Для этого используем метод Check класса Linear\_System.

Код метода проверки:

void Check()

{

T\* Check = new T[this->SIZE];

cout << "\nValues to Check: " << endl;

for (int i = 0; i < this->SIZE; i++)

{

for (int j = 0; j < this->SIZE; j++)

{

Check[i] += this->ptrVector[i][j] \* ptr[j];

}

cout << ptr[i] << "\t";

}

}

# Заключение.

В ходе лабораторной работы был разработан алгоритм метода Гаусса, решающий систему линейных уравнений, с выбором ведущего элемента. Метод Гаусса работает корректно и решает СЛАУ.

# Приложение

Vector <T> Gauss()

{

for (int j = 0; j < this->SIZE; j++)

{

int i = 0;

int indexMax = i + j;

T max = Mod<T>(this->ptrVector[i + j][j]);

for (i + j + 1; i + j + 1 < this->SIZE; i++);

{

if (Mod<T>(this->ptrVector[i + j][j]) > max)

{

indexMax = i + j;

}

}

swap(j, indexMax, ptr);

for (int n = j + 1; n < this->SIZE; n++)

{

if (this->ptrVector[j][j] != 0)

{

for (int k = j; k < this->SIZE; k++)

{

this->ptrVector[n][k] += this->ptrVector[j][k] \* (-(this->ptrVector[n][j] / this->ptrVector[j][j]));

}

ptr[n] += ptr[j] \* (-(this->ptrVector[n][j] / this->ptrVector[j][j]));

}

else

{

cout << "Error! Division by zero!" << endl;

exit(3);

}

}

}

for (int i = 0; i < this->SIZE; i++)

{

T sum = 0;

for (int j = 0; j < this->SIZE; j++)

{

sum += Mod<T>(this->ptrVector[i][j]);

}

if ((sum == 0) && (ptr[i] == 0))

{

cout << "No solution!" << endl;

exit(3);

}

if ((sum == 0) && (ptr[i] != 0))

{

cout << "No solution!" << endl;

exit(2);

}

}

T\* result = new T[this->SIZE];

for (int i = this->SIZE - 1; i >= 0; i--)

{

T tmp = 0;

for (int j = i + 1; j < this->SIZE; j++)

{

tmp += this->ptrVector[i][j] \* result[j];

}

result[i] = (ptr[i] - tmp) / this->ptrVector[i][i];

}

return Vector<T>(this->SIZE, result);

}