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

Медведев. М. 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

Передо мной стояла следующая задача: написать программу, реализующую метод Гаусса с выбором ведущего элемента в строке. Для этого необходимо создать следующие шаблонные классы: Vector, наследованный от Vector класс Matrix, наследованный от класса Matrix класс ecuationsSystem. Функция, решающая систему, должна принимать расширенную матрицу, в которой последний столбец – вектор свободных значений. Оператор () будет возвращать вектор решений.

# Метод решения

Метод решения – это алгоритм Гаусса. Он был назван в честь немецкого математика Карла Фридриха Гаусса. Он используется для решения систем линейных алгебраических уравнений. В процессе решения матрица приводится к треугольному виду путём элементарных преобразований. Треугольный вид позволяет исключить в каждом уравнении несколько переменных. Если решение единственное, то с помощью выражения одних переменных через другие, пользователь получит ответ. Основные элементарные преобразования над матрицей:

1. Перестановка строк матрицы местами;
2. Умножение или деление строки на число;
3. Сложение одной строки матрицы, умноженной на число (в т.ч. на 1), с другой.

Алгоритм решения СЛАУ методом Гаусса подразделяется на 2 этапа:

1. Прямой ход;
2. Обратный ход;

Суть первого этапа заключается в том, что путём элементарных преобразований над строками матрицу приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна. Для этого среди элементов первого столбца матрицы выбирают ненулевой и переставляют содержащую его строку в самое верхнее положение. Затем происходит обнуление всех элементов, находящихся под ведущим. Это происходит следующим образом: из всех строк, кроме первой, вычитается первая строка, умноженная на коэффициент, полученный путём отношения первого элемента этих строк к ведущему элементу первой строки. Затем пользователь мысленно спускается на ступеньку вниз (ведущим будет элемент, находящийся на месте а(2, 2)) и проделывает те же самые действия. И дальше по аналогии.

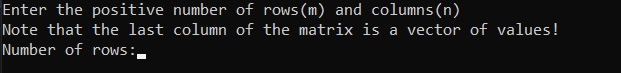
Суть второго этапа заключается в том, чтобы выразить все получившиеся ведущие переменные через неведущие и построить ФСР или получить единственное решение. Эта процедура начинается с последнего элемента, так как переменная при нём получается автоматически. Затем его подставляют в предыдущие уравнения и выражают другие переменные. Эти действия повторяются до самой первой (верхней) строки включительно.

Мне было поручено реализовать этот алгоритм с одним условием: ведущим элементом в строке должен быть максимальным по модулю. Это позволит избежать большой потери точности.

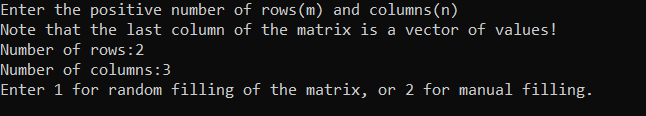
# Руководство пользователя

Для удобной работы с программой был реализован понятный пользователю интерфейс. Всё, что требуется – это следование инструкциям, которые выводятся в консоли.

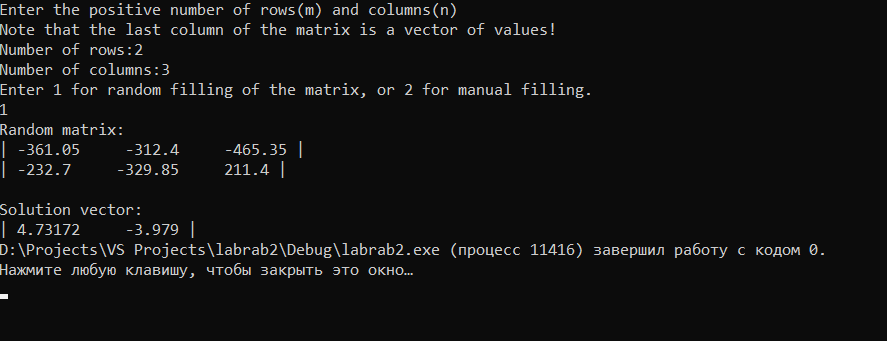
После запуска программы пользователю будет предложено задать количество строк и столбцов, учитывая то, что значения должны быть положительны:



Затем будет предложено выбрать способ заполнения матрицы (1 – случайно, 2 – вручную):



После этого в консоль выведется исходная матрица и вектор решения:



Для закрытия программы нужно нажать любую клавишу.

# Описание программной реализации

Моя программа состоит из четырёх файлов: labrab2.cpp, Vector.h, Matrix.h, gauss.h.

labrab2.cpp – файл с функцией main, в которой реализован пользовательский интерфейс, вызов функций, ввод матрицы (случайно или вручную), вывод вектора решения.

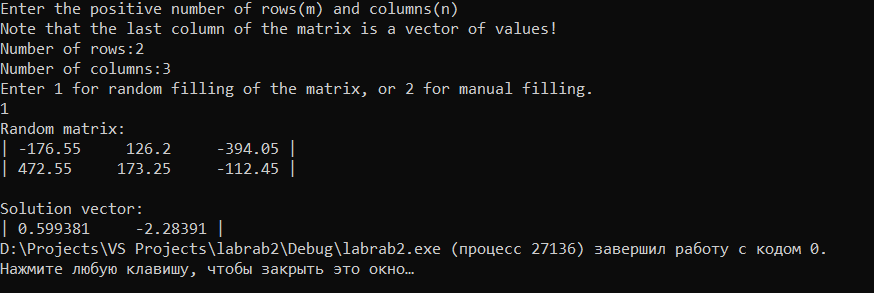
Vector.h – файл с реализацией шаблонного класса Vector<T>. В самом классе представлены поля data – динамический массив элементов вектора типа <T> - и размер массива size (размер вектора), определены конструктор с параметрами, конструктор копирования, операторы [] (возвращает элемент вектора под заданным индексом), << (вывод вектора) и >> (ввод вектора), = (присваивание размера и каждого элемента принимаемого вектора), +, +=, -, -= (поэлементное сложение и вычитание векторов), \*, \*=, /, /= (умножение и деление на коэффициент), функции swap (меняет вектора местами), resize, random\_fill\_in (заполняет вектор случайными элементами), метод getsize (возвращает размер вектора).

Matrix.h – файл, содержащий шаблонный класс Matrix<T>. Matrix<T> является наследником класса Vector<Vector<T>>. Он реализует матрицу, содержит конструкторы с параметрами и копирования, функции MaxIndexInCol (поиск максимального элемента в столбце), rand\_matr (заполнение матрицы случайными числами), методы geth (возвращает размер столбца) и getw (возвращает размер строки), операторы << и >>, \* (используется для матричного умножения).

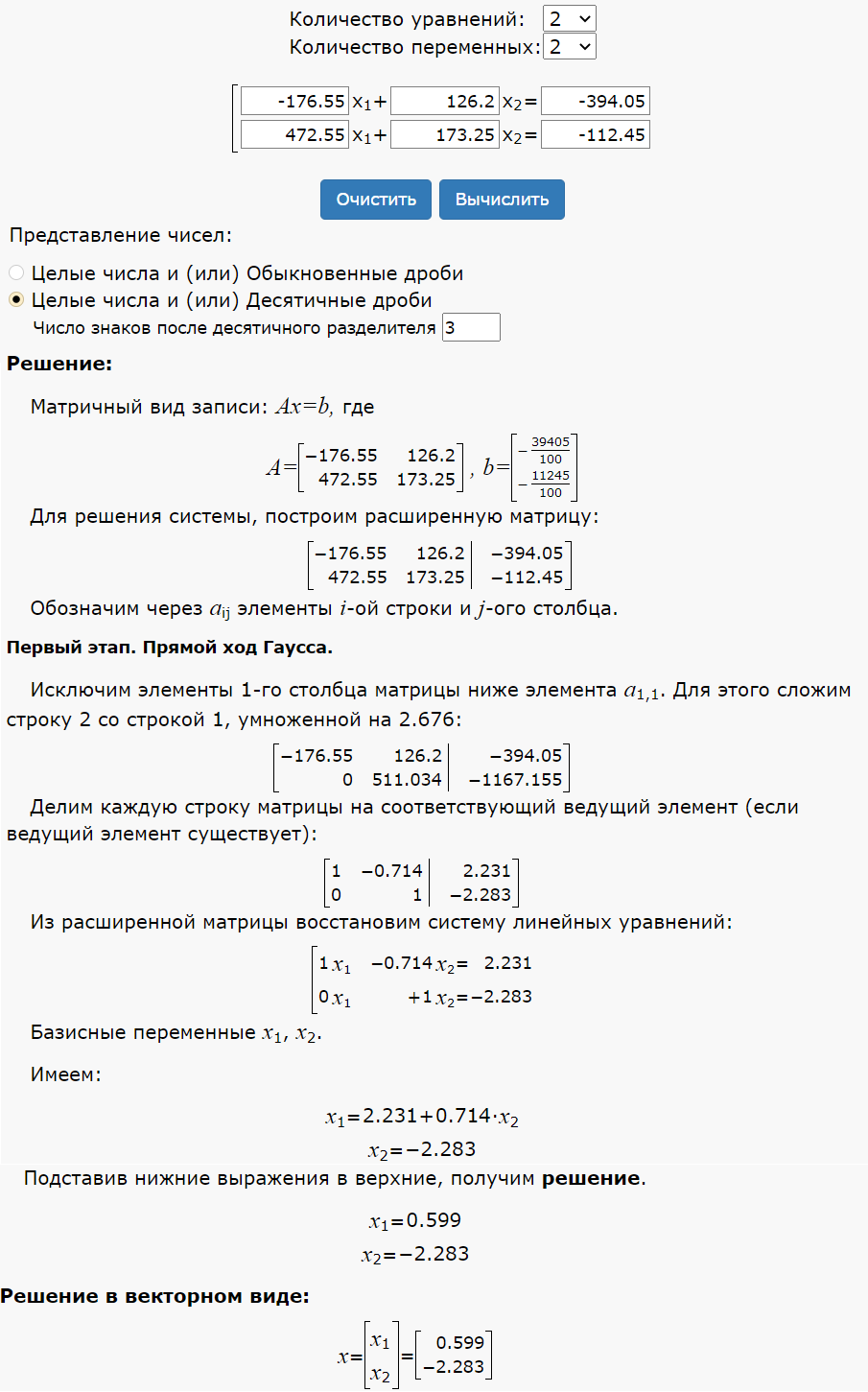
gauss.h – файл, содержащий класс equationsSystem<T> – наследник Matrix<T>. Он реализует решение СЛУ, содержит поля m, err, compare. В нём представлен оператор() (принимает расширенную матрицу, решает СЛУ и возвращает вектор значений), класс-функтор (в конструкторе принимает матрицу), перечисление eqResults.

# Подтверждение корректности

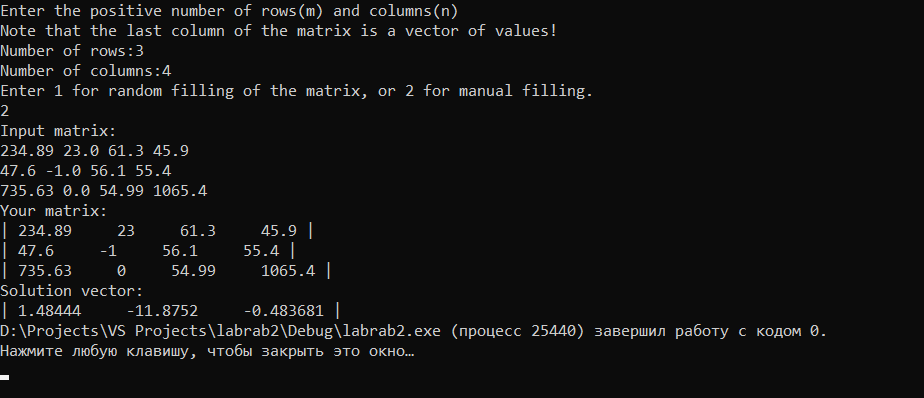
Для проверки корректности я поставил несколько экспериментов над программой и сверился с онлайн-калькуляторами.



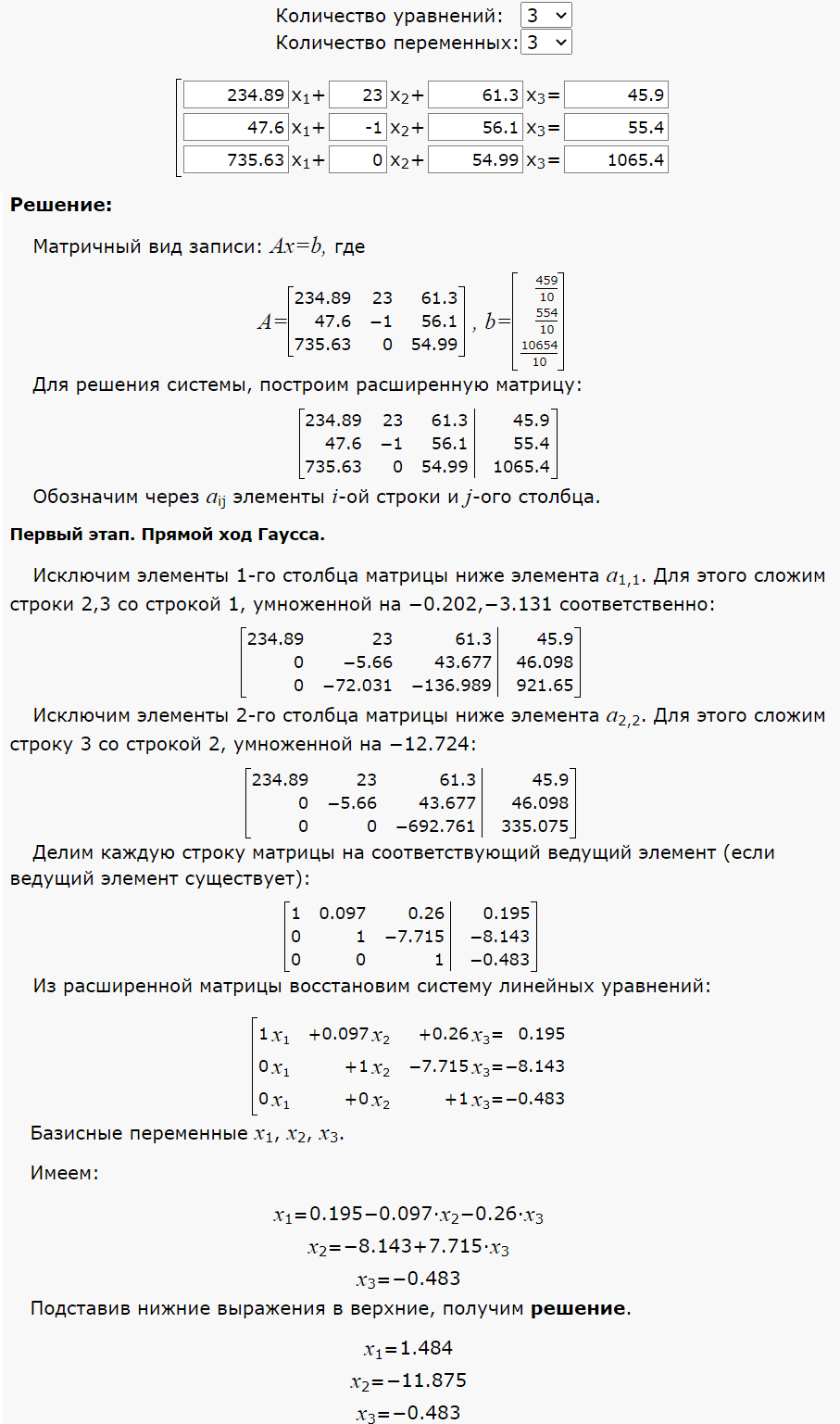
Результат в калькуляторе:



2)



Результат в калькуляторе:



Благодаря этим двум экспериментам можно убедиться, что программа работает корректно для случайной матрицы и для матрицы, введённой вручную.

# Результаты экспериментов

Результаты экспериментов занесены в таблицу.

|  |  |
| --- | --- |
| Пример СЛУ | Решение |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Если сделать проверку решения, то можно убедиться, что результат работы программы верный.

# Заключение

Таким образом, проделав данную работу, я написал программу, реализующую метод Гаусса с выбором ведущего элемента в строке. В процессе этого были созданы шаблонные классы Vector, Matrix, ecuationsSystem. Были поставлены соответствующие эксперименты над программой, показавшие её корректность.

# Приложение

Не обесценивая весь код, я считаю, что главной частью является та, в которой представлен алгоритм Гаусса, потому что его реализация является главной задачей моей работы.

enum class eqResults { OK, INCONSISTENT, INF\_SOLUTIONS, NOT\_SQUARE, UNKNOWN };

template <typename T>

class equationsSystem

{

public:

equationsSystem(const Matrix<T>& matrix) : m(matrix), err(eqResults::UNKNOWN) {}

Vector<T> operator()()

{

this->err = eqResults::UNKNOWN;

if (m.geth() + 1 != m.getw())

{

this->err = eqResults::NOT\_SQUARE;

}

else

{

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

{

size\_t row = this->m.MaxIndexInCol(i);

if (row != i)

{

swap(m[i], m[row]);

}

if (m[i][i] == 0)

{

if (m[i][m.getw() - 1] != 0)

{

for (size\_t j = i + 1; j + 1 < m.getsize(); j++)

{

if (m[i][j] != 0)

{

this->err = eqResults::INF\_SOLUTIONS;

break;

}

}

if (this->err == eqResults::UNKNOWN)

{

this->err = eqResults::INCONSISTENT;

}

}

else

{

this->err = eqResults::INF\_SOLUTIONS;

}

break;

}

else

{

for (size\_t j = i + 1; j < m.geth(); j++)

{

T tmp = m[j][i] / m[i][i];

for (size\_t k = i + 1; k < m.getw(); k++)

{

if (compare(m[j][k], tmp \* m[i][k]) )

{

m[j][k] = 0;

}

else

{

m[j][k] -= tmp \* m[i][k];

}

}

m[j][i] = 0;

}

}

}

}

Vector<T> ans(m.getw() - 1, T(0));

T sum;

if (this->err == eqResults::UNKNOWN)

{

for (int i = m.geth() - 1; i >= 0; i--)

{

sum = 0;

for (size\_t j = i + 1; j + 1 < m.getw(); j++)

{

sum += m[i][j] \* ans[j];

}

ans[i] = (m[i][m.getw() - 1] - sum) / m[i][i];

}

this->err = eqResults::OK;

}

return ans;

}

eqResults getErr()

{

return this->err;

}

private:

Matrix<T> m;

eqResults err;

bool compare(const T& l, const T& r)

{

if (is\_floating\_point<T>::value)

{

return std::abs(l - r) < std::numeric\_limits<T>::epsilon() \* 1000;

}

else

{

return l == r;

}

}

};