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 3821Б1ПМ2

Соколов В.Д.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

Оглавление

[Постановка задачи 3](#_Toc105321701)

[Метод Гаусса 4](#_Toc105321702)

[Вектор 4](#_Toc105321703)

[Матрица 6](#_Toc105321704)

[Система линейных алгебраических операций 10](#_Toc105321705)

[Алгоритм метода Гаусса 12](#_Toc105321706)

[Руководство пользователя 15](#_Toc105321707)

[Описание программной реализации 22](#_Toc105321708)

[Подтверждение корректности 24](#_Toc105321709)

[Заключение 27](#_Toc105321710)

# Постановка задачи

Целью лабораторной работы являлась реализовать на языке программирования Си++ Метод Гаусса для нахождения вектора.

Метод Гаусса нужно реализовать для данных типа double. В программе должны присутствовать:

1. Собственный шаблонный класс вектор (+матрица,+СЛАУ)
2. Собственный шаблонный класс матрицы, наследуемый от класса вектора.
3. Метод Гаусса для нахождения вектора
4. Проверка корректности вычисления нужного вектора

Нужно описать программную реализацию и алгоритмы работы, а также подтвердить корректность реализации метода Гаусса. Необходимо провести эксперименты для подтверждения сложности, описать способ проведения экспериментов и сделать вывод по полученным результатам.

Метод Гаусса нужно реализовать для данных типа double.

# Метод Гаусса

## Вектор

Вектор представляет собой массив из одного столбца и ***n*** строк.

Вектор обозначают строчной латинской буквой (a).

Если векторы подобны (параллельны друг другу), т.е. если из одного вектора можно получить другой домножив его на определённое число, то они называются *коллинеарными*.

Если все элементы вектора равны нулю, то вектор называется *нулевым*. Нулевой вектор коллинеарен любому другому вектору. 0 – нулевой вектор.

Два вектора называются равными если их элементы попарно равны между собой.

Вектор в линейной алгебре отображается так, как показано на рис.1.

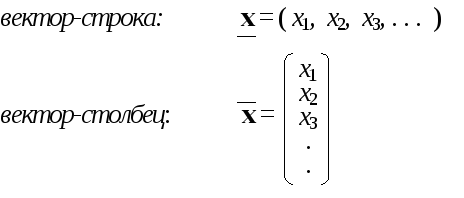


Рисунок Отображение вектора

**Линейные операции** **:**

* Операция сложения

Суммой векторов ***a*** и ***b*** является вектор c элементы которого равны сумме элементов из ***a*** и ***b***.

* Умножение вектора на скаляр

Произведение вектора ***a*** на скаляр ***k*** называется вектор c подобный (коллинеарен) вектору ***a***. Элементы вектора c равны элементам вектора ***a*** помноженным на скаляр ***k***.

Операции сложение векторов и умножение вектора на скаляр называются *линейными операциями*.

**Свойства линейных операций :** ( a,b,c – вектора; k,q – скаляры )

1. a + b = b + a ( коммутативность сложения )
2. (a + b) + c = a + (b + c) ( ассоциативность сложения )
3. a + 0 = a 0 – нулевой вектор
4. a + (-a) = 0
5. 1 \* a = a
6. k \* (q \* a) = (k \* q) \* a
7. (k + q) \* a = k \* a + q \* a
8. k \* (a + b) = k \* a + k \* b

## Матрица

Матрица представляет собой массив из n строк и m столбцов (рис 2).

Матрицу обозначают заглавной латинской буквой (A), а её элементы – соответствующими строчными буквами с индексами (aij). Первый индекс нумерует строки, второй – столбцы.

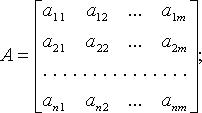


Рисунок Матрица размером n X m

Простейшие операции с матрицами :

* Умножение матрицы на число

При умножении матрицы на число получится матрица того же размера, элементы которой будут равны произведению элементов и числа (рис.3).

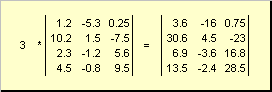


Рисунок Умножение матрицы на число

* Сложение (вычитание) матриц

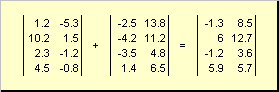
При сложении (вычитании) матриц получится матрица того же размера, элементы которой будут равны сумме (разности) элементов первой матрицы и элементов второй матрицы (рис.4).

Рисунок Сложение матриц

Умножение матриц :

Матрицы можно перемножать только в том случае, если они имеют соответствующие размерности: количество столбцов первой матрицы должно быть равно количеству строк второй матрицы. При этом:

A[n x m] \* B[m x n] = C[n x n]

B[m x n] \* A[n x m] = G[m x m] C[n x n] =/= G[m x m]

Умножение матрицы происходит по формуле :



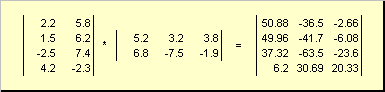
Наглядно работу формулы можно увидеть на примере (рис.5).

Рисунок Произведение матриц

Умножение матриц можно объяснить ещё проще. Чтобы получить какой-либо элемент итоговой матрицы нам нужно перемножить элементы строки первой матрицы на элементы столбца второй матрицы и сложить их (рис. 6).

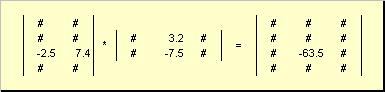


Рисунок Получение элемента произведения матриц

Произведение матриц некоммутативно, но ассоциативно:

(A \* B) \* C = A \* (B \* C)

Кроме того, умножение матриц дистрибутивно:

A \* (B + C) = A \* B + A \* C

А также: A \* 0 = 0

Квадратная матрица

Если число столбцов матрицы равно числу её строк, то такая матрица называется *квадратной*.

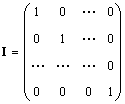
*Единичной* матрицей (E) называется матрица, у которой все элементы равны нулю, за исключением диагональных, которые равны 1 (рис.7)

Рисунок Единичная матрица

Очевидно, что: A \* E = E \* A = A

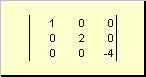
Матрица называется *диагональной*, если все её элементы, кроме диагональных равны нулю (рис.8).

Рисунок Пример диагональной матрицы

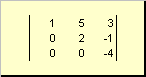
Матрица A называется верхней *треугольной*, если все её элементы, лежащие ниже диагонали, равны нулю (рис.9).

Рисунок Пример верхней треугольной матрицы

Аналогично определяется и нижняя треугольная матрица.

AT – *транспонированная матрица*. Чтобы получить транспонированную матрицу нужно поменять строки со столбцами (перевернуть матрицу на 90o по часовой стрелке).

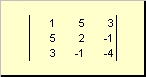
Матрица A называется *симметричной*, если AT = A. Иными словами aij = aji (рис.10).

Рисунок Пример симметричной матрицы

Матрица A называется ортогональной, если

AT \* A = A \* AT = E

Матрица называется нормальной, если

AT \* A = A \* AT

## Система линейных алгебраических операций

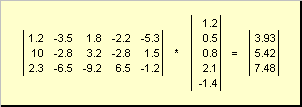
Вектор ***x*** можно умножать на матрицу **A** подходящей размерности. При этом вектор-столбец умножается справа (**A** \* ***x***), а вектор строка – слева (***x***T \* **A**). Если размерность вектора m, а размерность матрицы ***n*** x ***m*** то в результате получится вектор размерности ***n*** (рис.11).

Рисунок Пример умножения матрицы на вектор

Если матрица **A** – квадратная (размерность ***n*** x ***n*** ), то вектор *b* = **A** \* *x* имеет ту же размерность, что и *x*.

Системы линейных уравнений

Rank(A) – ранг матрицы: количество линейно независимых строк.

Строку можно представить как вектор-строку, что, по сути, как транспонированный вектор-столбец. Если вектор не коллинеарен ни одному другому вектору, то он линейно независим.

det(A) – определитель матрицы: [скалярная величина](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%B0%D0%BB%D1%8F%D1%80%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0), которая характеризует ориентированное «растяжение» или «сжатие» многомерного [евклидового пространства](https://ru.wikipedia.org/wiki/%D0%95%D0%B2%D0%BA%D0%BB%D0%B8%D0%B4%D0%BE%D0%B2%D0%BE_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE) после преобразования матрицей; имеет смысл только для [квадратных матриц](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0). Формула определителя:

|  |  |  |
| --- | --- | --- |
| det(A) = | Σ | (-1)N(α1,α2,...,αn)·aα11·aα22·...·aαnn |

(α1, α2,…, αn)

Пусть **A** – матрица размером n x n , а *b* – вектор размерности n. Рассмотрим уравнение:

**A** \* *x* = *b*

относительно вектора *x ,* размерности n. По сути – это система из n линейных уравнений с n неизвестными *x*1 ,…, *x*n :

| a11 \* [x2] + a12 \* [x2] + … + a1n \* [xn]

| a21 \* [x2] + a22 \* [x2] + … + a2n \* [xn]

| … … …

| an1 \* [x2] + an2 \* [x2] + … + ann \* [xn]

Решение существует только в том случае, когда

rank(A) = rank(B) = r

где **B** – это расширенная матрица размерности n x (n+1) , состоящая из матрицы **A**, дополненной столбцом *b*, **B** = (**A** *b*). В противном случае уравнения несовместны.

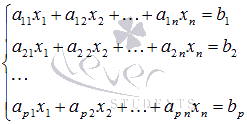
Если r = n = n , то решение единственно

*x* = **A**-1 \* *b*

Если r < n, то существует множество различных решений, которые можно выразить через линейную комбинацию n – r векторов.

Система однородных уравнений **A** \* *x* = *0* с квадратной матрицей **A** (n x n) имеет нетривиальное решение (x 0) тогда и только тогда, когда det(A) =0. Если r = rank(A) < n , то существуют n-r линейно независимых решений.

Если с системой линейных алгебраических уравнений



произвести следующие действия

* Поменять местами два уравнения
* Умножить обе части какого-либо уравнения на произвольное и отличное от нуля действительное число k
* К обеим частям какого-либо уравнения прибавить соответствующие части другого уравнения, умноженные на произвольное число k

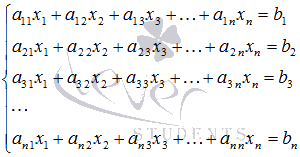
то получится эквивалентная система, которая имеет такие же решения.

Для расширенной матрицы системы линейных алгебраических уравнений эти действия будут означать проведение элементарных преобразований со строками:

* Перестановка двух строк местами
* Умножение всех элементов какой-либо строки матрицы T на отличное от нуля число k
* Прибавление к элементам какой-либо строки матрицы соответствующих элементов другой строки, умноженных на произвольное число k

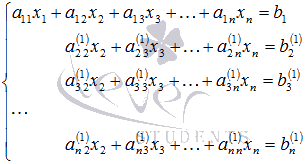
## Алгоритм метода Гаусса

Пусть нам потребуется решит систему из n линейных алгебраических уравнения с n неизвестными переменными вида



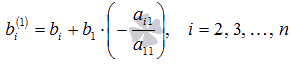
И пусть определитель её основной матрицы отличен от нуля.

Будем считать, что , так мы всегда можем этого добиться перестановкой местами уравнений системы. Исключим неизвестную переменную x1 из всех уравнений системы, начиная со второго. Для этого ко второму уравнению системы прибавим первое, умноженное на , к третьему уравнению прибавим первое, умноженное на , и так далее, к n-ому уравнению прибавим первое, умноженное на . Система уравнений после таких преобразований примет вид



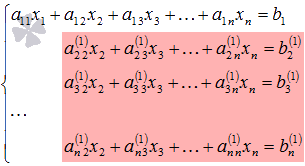
где

формула

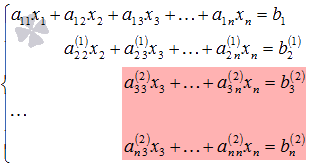


К такому же результату мы бы пришли, если бы выразили x1 через другие неизвестные переменные в первом уравнении системы и полученное выражение подставили во все остальные уравнения. Таким образом, переменная x1 исключена из всех уравнений, начиная со второго.

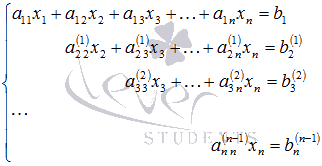
Далее действуем аналогично, но лишь с частью полученной системы



Будем считать, что (в противном случае мы переставим местами вторую строку с k-ой, где ). Приступаем к исключению неизвестной переменной x2 из всех уравнений начиная с третьего. Останется



Так продолжаем прямой ход метода Гаусса пока система не примет вид



С этого момента начинаем обратный ход метода Гаусса: вычисляем xn из последнего уравнения как , с помощью полученного значения xn находим xn-1 из предпоследнего уравнения, и так далее, находим x1 из первого уравнения.

Таким образом мы нашли вектор *x* ( x1 , x2 , … , xn ).

# Руководство пользователя

Программа LabRab\_2semestr-2 обладает удобным меню (на русском языке), в котором объясняются необходимые от пользователя действия для получения нужного ему ответа (вектора x).

Запуская программу, пользователь сначала увидит краткую заметку про СЛАУ\* и метод Гаусса. Затем идёт пример, наглядно демонстрирующий решение задачи с использованием метода Гаусса.

Матрицу нельзя решить.

Решить можно только Систему Линейных Алгебраических Уравнений

По СЛАУ можно построить матрицу.

Матрицу мы не можем решить, но зато можем упростить по методу Гаусса.

Пример СЛАУ и построенной по СЛАУ матрицы

СЛАУ:

3x + 5y + 9z = 28

7x + 7y + 7z = 42

x + 13z = 16 \_

v - вектор

Построенная по СЛАУ матрица: A\*v=b

( 3 5 9 | 28 )

( 7 7 7 | 42 )

( 1 0 13 | 16 )

Упрощенная матрица по методу Гаусса (использовалась данная программа):

( 3 0 0 | 9 ) --> 3\*x = 9

( 0 -4.66667 0 | -9.33333 ) --> -4.66667\*y = -9.33333

( 0 0 15 | 15 ) --> 15\*z = 15

Полученный вектор v = ( 3 , 2 , 1 ) , что означает x=3 y=2 z=1

Планируется делать вывод программы похожим данной схеме.

Далее идёт выбор типа данных:

В предоставленном списке рабочим вариантом будет только тип double. Меню кажется бессмысленным, но оно остаётся в программе как наработка для возможного следующего обновления проекта. Также никто не отрицал возможность использования этой программы для дополнения других проектов.

Пользователю просто нужно ввести число, под которым стоит необходимый ему тип данных.

Пока осуществим вариант только с double.

Другие выдают ошибку!

<>< МЕНЮ ><> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |

| > Выберите тип данных |

| |

| |

| 1) double [рациональные] |

| |

| 2) другого нет [ X ] |

| |

| |

| > Введите цифру нужного вам типа данных |

| |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Ваш тип данных >

При вводе других, не обозначенных в списке, чисел программа не завершается, а предлагает пользователю ввести другое число:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| |

| Ошибка ввода ! |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

> Ожидалась одна из цифр: 1 или х [2] x

Ошибка

Под числом [2] не стоит ни один тип данных, следовательно ввод [2] некорректен и пользователю нужно ввести другое число.

Далее пользователю нужно задать размер матрицы. Необходимо ввести положительное целочисленное значение, чтобы программа работала.

Для значения размера матрицы используется тип данных int.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| |

| > Теперь Вам нужно ввести размер матрицы |

| <( Матрица A будет иметь размер [ n x n ] ) --> матрица А - квадратная: > |

| длина строки = n |

| длина столбца = n |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Ваш размер матрицы A[n x n]: n =

Вводим число

Если всё-таки ввести число, неподходящее условию (к примеру отрицательное), то выведется ошибка (исключение)

<!> Ошибка памяти: неверное расположение в динамической памяти

и программа завершится. Второго шанса, как при выборе типа данных, не будет. Пользователю придётся перезапускать программу.

И последнее

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| |

| > Вам нужно поочерёдно ввести значения элементов матрицы |

| > элементы пишутся в строку ! |

| строка 1: x1 -> y1 -> z1 |

| строка 2: x2 -> y2 -> z2 |

| и т.д. |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

x1 + y1 + z1 = b1 ( (1) (2) (3) | (10) )

x2 + y2 + z2 = b2 ( (4) (5) (6) | (11) )

x3 + y3 + z3 = b3 ( (7) (8) (9) | (12) )

Круглые скобки показывают в какой последовательности пользователь вводит элементы в матрице

Когда пользователь будет вводить элементы матрицы, программа будет сопровождать его объяснениями, деталями и некоторыми указаниями.

1-ой, 2-ой, 3-ей и т.д.

> Введите значения элементов на k-ой строке

и

Чему равно уравнение в первой строке , 2-ой, 3-ей и т.д.

> Введите значение вектора b

Когда пользователь введёт последний элемент он увидит

Матрица построена. Для получения ответа нажмите на любую клавишу...

что будет сигналом о том, что больше от пользователя не требуется никаких сложных действий. Также необходимо учесть, что хоть и написано “Матрица построена”, но это не факт. Если в решении будет ошибка или бесконечное множество решений, то упрощённая матрица не будет показана. Тогда пользователь увидит только полное СЛАУ и полную матрицу.

Далее идёт лист с ответом (всё что ниже будет написано выведется на экран)

Там, где находятся круглые скобки () лежат заданные пользователем числа (в показанной последовательности)

Там, где находятся квадратные скобки [] лежат значения вектора x (пользователь их не вводил)

В СЛАУ не подставлены значения вектора x

Полная матрица, построенная при помощи СЛАУ

> СЛАУ

1> (1)\*[x0] + (2)\*[x1] = (b1)

2> (3)\*[x0] + (4)\*[x1] = (b2)

> Её матрица:

( (1) (2) | (b1) )

( (3) (4) | (b2) )

Упрощённая диагональная матрица.

Там, где находятся волнистые скобки {} лежат изменённые значения.

В каждой строке ( 1> ; 2> ) остаётся лишь одна неизвестная.

> Упрощённая СЛАУ:

1> {1}\*[x0] + 0\*[x1] = {b1}

2> 0\*[x0] + {2}\*[x1] = {b2}

> Её матрица:

( {1} 0 | {b1} )

( 0 {2} | {b2} )

Программа не будет выдавать не диагональную матрицу т.к. у этой матрицы будет бесконечное множество решений, а это исключение. Ведь так мы не сможем найти нужный нам вектор x.

Там, где находятся знаки неравенства <> лежат полученные программой значения вектора x.

<><--| Полученный вектор: X = (<x1>, <x2>)

Ответ, который искал пользователь

СЛАУ была решена [правильно] с погрешностью (<P>), что меньше максимально допустимой погрешности (1e-10)

Матрица будет решена [правильно] когда максимальная погрешность будет меньше 10-10 (1e-10).

Иначе - матрица решена [неправильно].

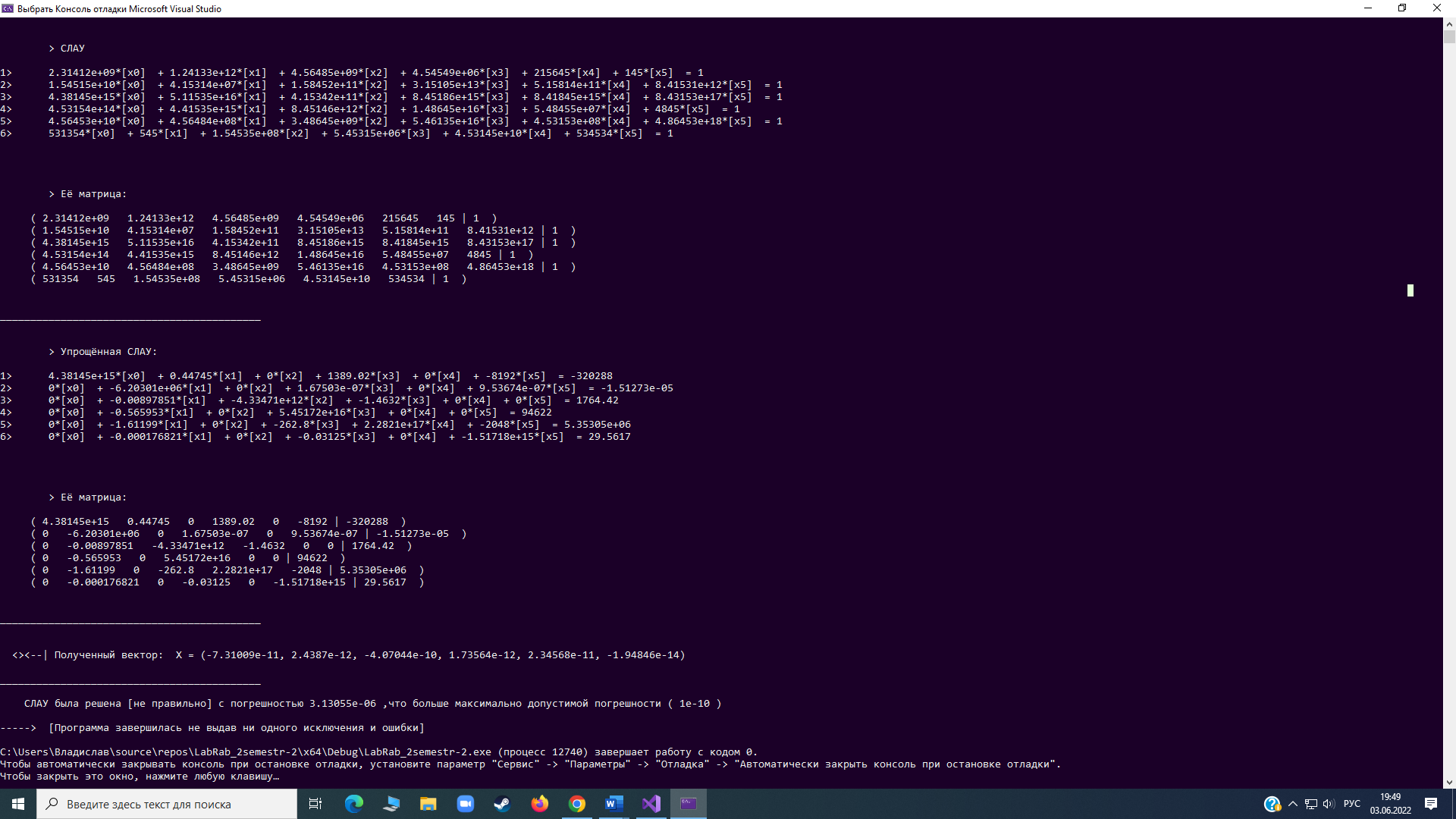
Погрешность (<P>) – это максимальна погрешность, выявленная при подстановке значений нашего вектора в уравнение и сравнения полученного ответа (b’) с заданным пользователем (b)

В конце появится сообщение:

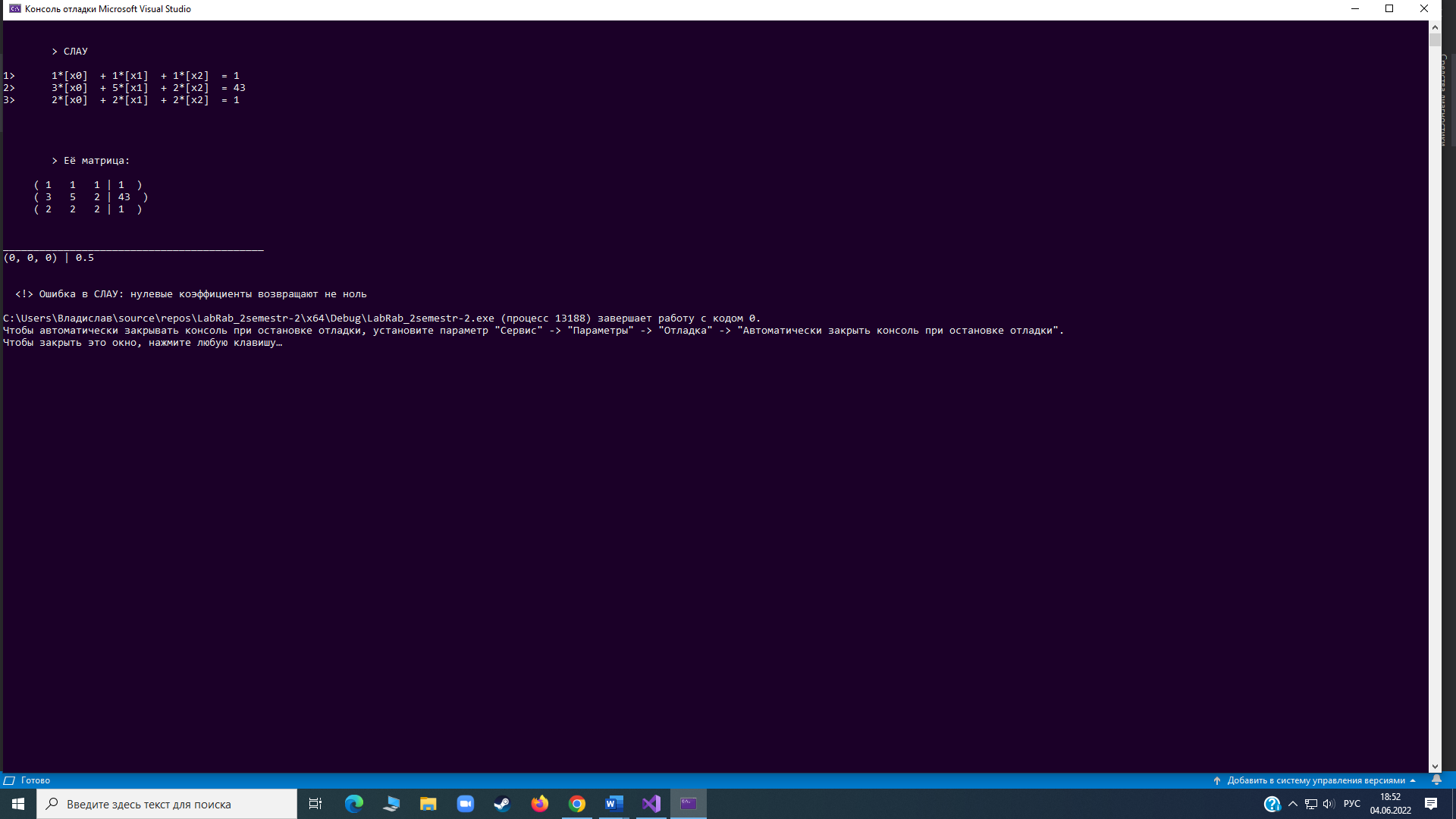
что означает, что во время работы программы не были зафиксированы ошибки в решении или исключения.

-----> [Программа завершилась не выдав ни одного исключения и ошибки]

В дополнении (в качестве бонуса) рассмотрим неправильно решённый вариант (1) и вариант с какой-либо ошибкой (2) :

1)

2)



# Описание программной реализации

Exctptions.h

Class MyException

{

MyException(int n) – подставляем номер ошибки

void show() – показывает описание конкретной ошибки

}

Vector.h

Class Vector

{

Vector(int n) – создаёт массив (вектор)

void change\_size(int n) – заменяет размер вектора на число n

void set() – задание собственного вектора (b)

~Vector() – удаляет вектор

Vector& operator=(Vector& a) – перегружает оператор присваивания

Vector& operator-=(Vector& a) – перегрузка операции разности

Vector& operator\*=(T s) - перегрузка операции умножения

ostream& operator<<(ostream& out) – перегрузка операции вывода

}

}

Matrix(…).h

Class Matrix

{

void set() - пользователь вводит элементы матрицы

Matrix(int n) :Vector<Vector<T>>(n) - строим матрицу

Matrix(int n,Matrix<T>& M) :Vector<Vector<T>>(n) - Строим матрицу для упрощения

}

}

Main.cpp

Int main()

{

…

try

{

…

Slau<double> answ(n); - заводим СЛАУ(+вектор,+матрица)

answ << cout; - вывод на экран то, что дано

answ.Method\_Gaussa(); - метод Гаусса и вывод ответа

}

Catch (MyException& e) - ловим исключения

{

…

}

}

SLAU.h

Class Slau

{

Slau(int n) : Matrix<T>(n) – создаём вектор x и задаём свой вектор b

string Whole\_Check(Vector<T>& B, Matrix<T>& M) – подсчитывание погрешности

void Method\_Gaussa() – метод Гаусса по упрощению матрицы

ostream& operator<<(ostream& out) – выводит СЛАУ

}

# Подтверждение корректности

Для подтверждения корректности программы мы воспользуемся примитивным, но самым точным, методом : поиск погрешности.

Мы можем регулировать когда работа программы нас устраивает, а когда нет. Мы просто устанавливаем свой порог, который погрешность переступить не может. В данной программе порог 10-10.

Погрешность считается модулем разности нашего b и получившегося b’ с использованием x найденном программой.

b’ высчитывается по формуле: A\*x = b’ , где A – матрица

, x – найденный программой

Для проверки корректности реализации программы были выбраны несколько вариантов СЛАУ, примеры при различных ситуациях :

1) СЛАУ с большими элементами (нормальная)

Матрица

( 0 1.24453e+07 341314 1.23413e+08 | 314123 )

( 4.21431e+07 4.31412e+06 1.43412e+06 1234 | 41 )

( 123 1242 14 1314 | 414 )

( 123 143 14 2421 | 123 )

Упрощённая матрица:

( 4.21431e+07 0 0 0 | -6.13948e+07 )

( 0 1.24453e+07 0 0 | 1.68593e+06 )

( 0 0 6.23786 0 | 264.501 )

( 0 0 0 -6555.83 | 841.669 )

Погрешность: 3.725 \* 10-9 🡪 Матрица решена некорректно ( погрешность > 10-10)

2) СЛАУ с четырёхзначными элементами (и меньше)

Матрица

( 3142 6436 3478 3562 | 5482 )

( 4612 9569 5637 2479 | 9662 )

( 5327 5434 1234 1000 | 7502 )

( 7324 1234 5345 5783 | 6305 )

Упрощённая матрица:

( 7324 0 0 0 | 6737.07 )

( 0 8791.94 0 0 | 4327.53 )

( 0 0 -340.847 0 | -117.683 )

( 0 0 0 -23506.2 | 11726.3 )

Погрешность: 3.638 \* 10-12

3) СЛАУ с малыми элементами

Матрица

( 0.0001234 0.0006473 0.0005473 0.000549 | 0.000341 )

( 4.324e-05 2.15e-05 7.48e-05 6.47e-05 | 4.12e-05 )

( 7.583e-05 6.5e-05 6.43e-05 1.11e-05 | 7.34e-05 )

( 9.899e-05 1.387e-05 8.4e-05 1e-05 | 7.3534e-05 )

Упрощённая матрица:

( 0.0001234 0 0 0 | 7.27618e-05 )

( 0 -0.000505386 0 0 | -0.000159105 )

( 0 0 2.72601e-05 0 | 3.54409e-06 )

( 0 0 0 2.33281e-05 | -2.84654e-07 )

Погрешность: 5.421 \* 10-20

4) Диагональная матрица (контролируемая)

Матрица

( 1542 0 0 0 | 3084 )

( 0 8325 0 0 | 8325 )

( 0 0 83024 0 | 10378 )

( 0 0 0 86960 | 1087 )

Упрощённая матрица:

( 1542 0 0 0 | 3084 )

( 0 8325 0 0 | 8325 )

( 0 0 83024 0 | 10378 )

( 0 0 0 86960 | 1087 )

Погрешность: 0 X = (2, 1, 0.125, 0.0125)

Мы сами построили матрицу, зная X. Мы можем сравнить X полученный программой с нашим.

# Заключение

В ходе лабораторной работы на языке программирования Си++ был реализован метод Гаусса по сокращению матрицы. Был реализован поиск по сокращённой матрицы вектора x, из произведения A\*x = b , где A-матрица и x-вектор. Была реализована поверка вычисляющая погрешность и выяснявшая корректность программы по этой погрешности. Были созданы шаблонные классы Slau,Matrix,Vector , а также собственный класс исключений (MyException). Были проведены проверки используя примерные решения задачи (описанные в пункте “подтверждение корректности”).

Список литературы:

1. Алексей Померанцев “Матрицы и векторы” – издательство 2007 года. (Российское хемометрическое общество).

Сайт: <https://rcs.chemometrics.ru/old/Tutorials/matrix.htm#ch101>

1. “Системы, решение систем уравнений и неравенств” – математика на cleverstudents.ru.

Сайт: <http://www.cleverstudents.ru/systems/solving_systems_Gauss_method.html>