Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра Автоматизированных Систем Управления (АСУ)

**Отчет по контрольной (лабораторной) работе № 1**

**по дисциплине «Вычислительная математика»**

Выполнил: ст. гр. з-422П8-5

Жданов А.А. «\_\_» \_\_\_2023г.

Проверил: к.т.н., доц. каф. АСУ \_\_\_\_\_\_\_\_

Романенко В. В. «\_\_» \_\_\_2023г.

Томск 2023

**Оглавление**

1 РЕШЕНИЕ УРАВНЕНИЙ С ОДНОЙ ПЕРЕМЕННОЙ 3

1.1 Задание 3

1.2 Теоретический материал 3

1.3 Алгоритм решения 3

1.4 Результаты работы программы 4

2 РЕШЕНИЕ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ 6

2.1 Задание 6

2.2 Теоретический материал 6

2.3 Алгоритм решения 7

2.4 Результаты работы программы 8

Выводы 12

Список использованных источников 14

Приложение А Листинги программ 15

**1. РЕШЕНИЕ УРАВНЕНИЙ С ОДНОЙ ПЕРЕМЕННОЙ**

**1.1 Задание**

Написать программу отделения корней их уточнения методом дихотомии.

Входные данные:

функция f(x);

интервал [a,b];

точность по аргументу и по функции.

Выходные данные: корни ξ , точность;

значения функции f (ξ);

количество итераций n ;

количество вычислений функции f (x) ;

время счета.

Вариант 6: f(x) = - 3cos(x)

**1.2 Теоретический материал**

Для решения уравнения был избран метод дихотомии или половинного деления. У данного метода теоретическая идея довольно проста — делить заданный отрезок пополам (отсюда, собственно, и название), подставлять значение срединной точки в уравнение до тех пор, пока разница между правой и левой частями уравнения не будет равной, или менее заданной точности, либо длина отрезка также не станет менее заданной (тогда, решение находится на одном из концов разделенного отрезка). Для задания лабораторной работы в программе устанавливается точность равная 0,0001 как по функции, так и по аргументу.

**1.3 Алгоритм решения**

В программе реализованы 3 метода: метод непосредственно половинного деления; метод возвращающий значение заданной функции в какой-либо точке; метод определяющий знак функции в какой-либо точке.

Функция осуществляющая метод половинного деления получает в качестве параметров значения начала и конца заданного отрезка (в программе этот интервал определен как [0,2]), значение длины отрезка, и значение функции в точке. Далее запускается цикл деления отрезка пополам до тех пор пока абсолютное значение функции в точке не будет равным или менее установленного параметра, если это значение превышает установленный параметра запускается условный оператор, который переназначает начало или конец отрезка в зависимости от знака функции в точке — если знак функции в начале отрезка совпадает со знаком функции в текущей точке, то началу отрезка присваиватся значение середины отрезка, в том случае если знак не совпадает значение присваивается концу отрезка, и процесс деления пополам начинается вновь.

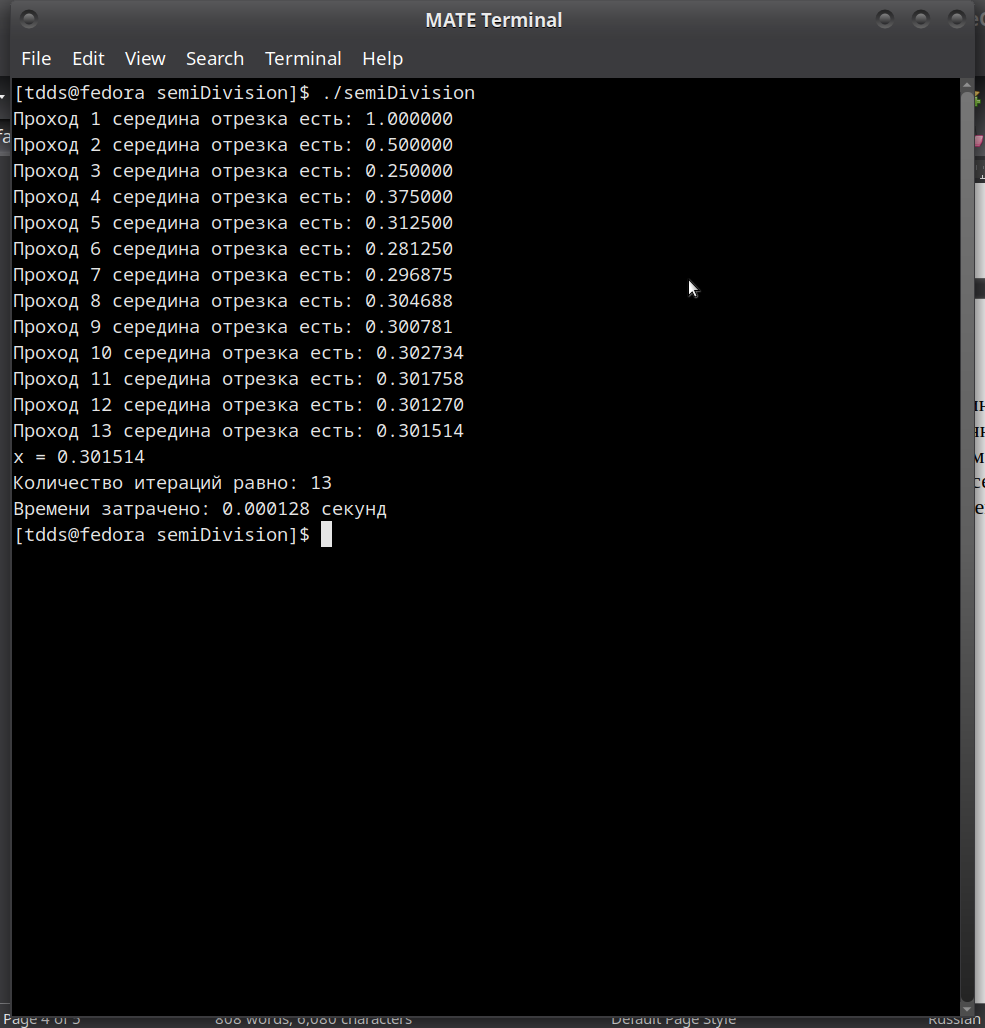
Метод возвращающий значение функции в точке получает параметр, являющийся нулем на первом шаге, а далее этим параметром становится середина отрезка. Метод подставляет полученный параметр в функцию (согласно варианту), вычисляет её значение, и возвращает его.

Метод определяющий знак функции в точке принимает как параметр значение текущей точки и возвращает 1, либо -1, соответственно знаку текущей точки.

В главной функции находятся: счетчик времени выполнения программы; выражения выводящие на экран все подставлямые середины отрезков, количество итераций, и количество затраченного на расчет времени.

**1.4 Результаты работы программы**

Результат работы программы показан на рис. 1

Рисунок 1.

**2. РЕШЕНИЕ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ**

**2.1 Задание**

Написать программу решения системы линейных алгебраических уравнений методом Гаусса.

Входные данные: порядок системы n; матрица системы A; правая часть системы b.

Выходные данные: промежуточные векторы и матрицы; решение системы; невязка.

Вариант 6:

3.2x+5.4y+4.2z+2.2p=11.4

2.1x+3.2y+3.1z+1.1p=9.2

1.2x+0.4y-0.8z-0.8p=0.4

4.7x+10.4y+9.7z+9.7p=30.4

Написать программу вычисления определителя матрицы методом Гаусса. Входные данные: порядок системы n; матрица системы A.

Выходные данные: значение определителя.

Написать программу вычисления обратной матрицы методом Гаусса.

Входные данные: порядок системы n ; матрица системы A.

Выходные данные: промежуточные векторы и матрицы; обратная матрица; невязка.

**2.2 Теоретический материал**

Метод Гаусса отличается относительно малым количеством итераций при решении СЛАУ. Кроме того, при помощи него хорошо описаны в литературе методы обращения матриц, и вычисления определителей.

Метод заключается в следующем — для начала, при помощи элементарных преобразований, система приводится к т.н. треугольному (сверху или снизу) виду, т.е. уравнения системы содержат столько неизвестных с ненулевыми коэффициентами, каков их номер по счету в системе (для треугольных снизу систем, для систем треугольных сверху — наоборот). Далее, в действие приводится метод обратной подстановки, т.е. неизвестные вычисляются в обратном порядке, и затем последовательно подставляются в уравнения все еще содержащие неизвестные, тем самым, постепенно находя все неизвестные.

Вычисление определителя методом Гаусса вообще не представляет какой-либо сложности — его значение равно произведению коэффициентов, стоящих на главной диагонали, приведенной к треугольному виду матрицы.

Для вычисления обратной матрицы также используется метод Гаусса, с тем отличием, что вместо свободных членов заданной СЛАУ, подставляются значения единичной матрицы «постолбцово». Т.е. получается четыре системы уравнений имеющих одну и ту же матрицу и различные свободные члены. Решая эти четыре системы уравнений получится 16 значений неизвестных, которые в свою очередь являются значениями обратной матрицы. Невязка вычисляется подстановкой членов столбца в первоначальную систему, и сравнивается со свободными членами единичной («постолбцово») матрицы. Разность между ними и будет невязкой обратной матрицы.

**2.3 Алгоритм работы программы**

Программы состоит из «пустых» методов, которые ничего не возвращают, но преобразуют переменные. Переменными здесь являются матрицы, а также векторы.

Вначале система проверяет нет ли у заданной матрицы нулей на осевых элементах (элементах, стоящих на главной диагонали). Метод проверки основан на цикличном проходе по всем строкам, а затем по всем элементам строки. Если находится нуль, то к строке применяются элементарные преобразования — каждый член следующей строки умножается на -1 и прибавляется к текущей. Здесь можно отметить, что подобные преобразования несут некий риск, т.к. возможна ситуация, когда в следующей строке соответствующий член тоже равен нулю, и преобразование не получится. Но подразумевается, что в учебных заданиях не может быть сингулярных, или систем без решения.

Далее вызывается метод преобразующий, проверенную, и при необходимости, преобразованную матрицу системы, к треугольному виду. Описывать алгоритм метода видится нецелесообразным т.к. в листинге программы даны соответствующие комментарии, см. Приложение А. Можно лишь отметить, что при помощи циклов, пробегающих по строкам и коэффициентам программы, применяя элементарные арифметические операции реализован метод Гаусса для решения СЛАУ.

Затем вызывается функция реализующая метод обратной подстановки, также как и предыдущий основан на циклах проходящих по строкам и коэфициентам, но в обратном порядке. Начиная снизу, решает уравнение с одной неизвестной, далее подставляя уже вычисленные значения в уравнения с неизвестными, и также решает уравнение с одной неизвестной. Найденные неизвестные записываются в вектор, и выводятся на экран.

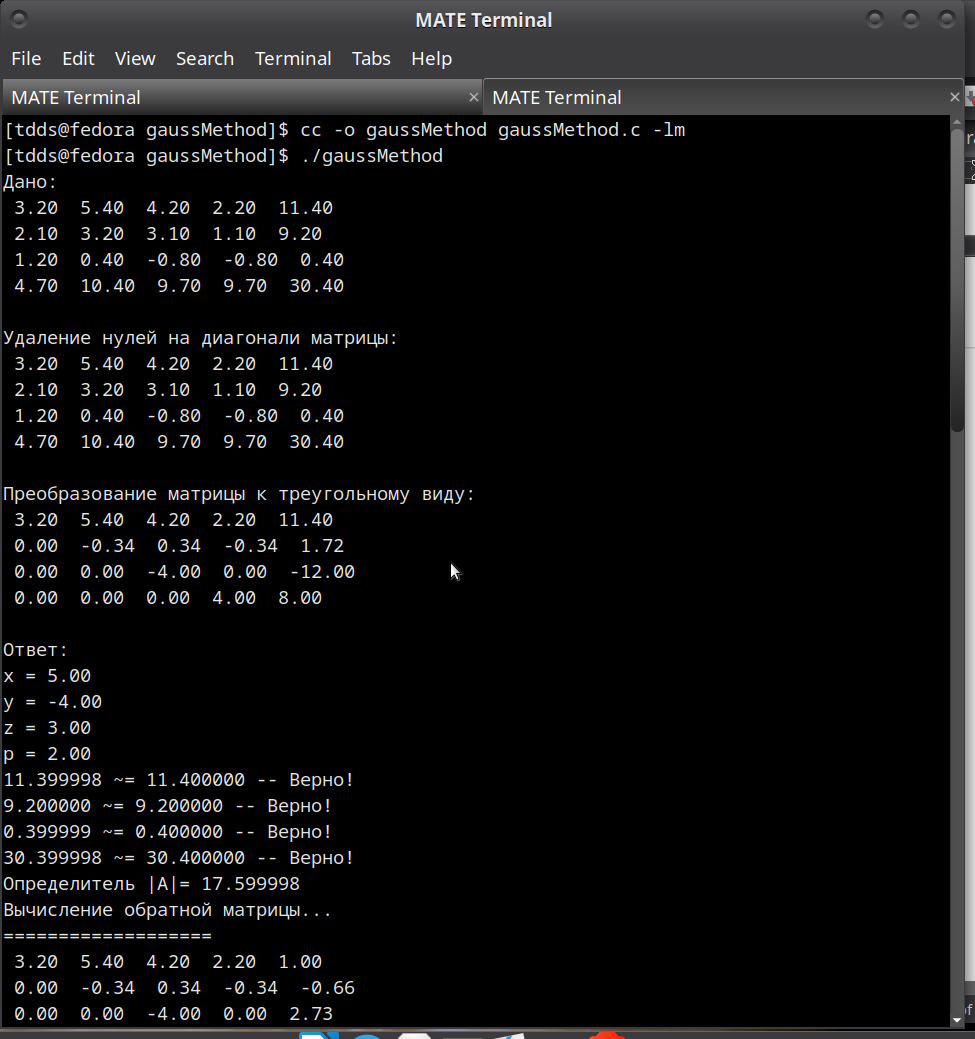
Следующий метод вычисляет невязку в решениях СЛАУ. Опять же, при помощи пары циклов проходящих по строкам/коэффициентам, подставляя соответствующие решения в произведение к соответствующим коэффициентам, метод суммирует эти произведения и сравнивает со свободным членом первоначальной системы. Если разница в сравнении не превышает одной тысячной, на экран выводится сообщение о верности решения.

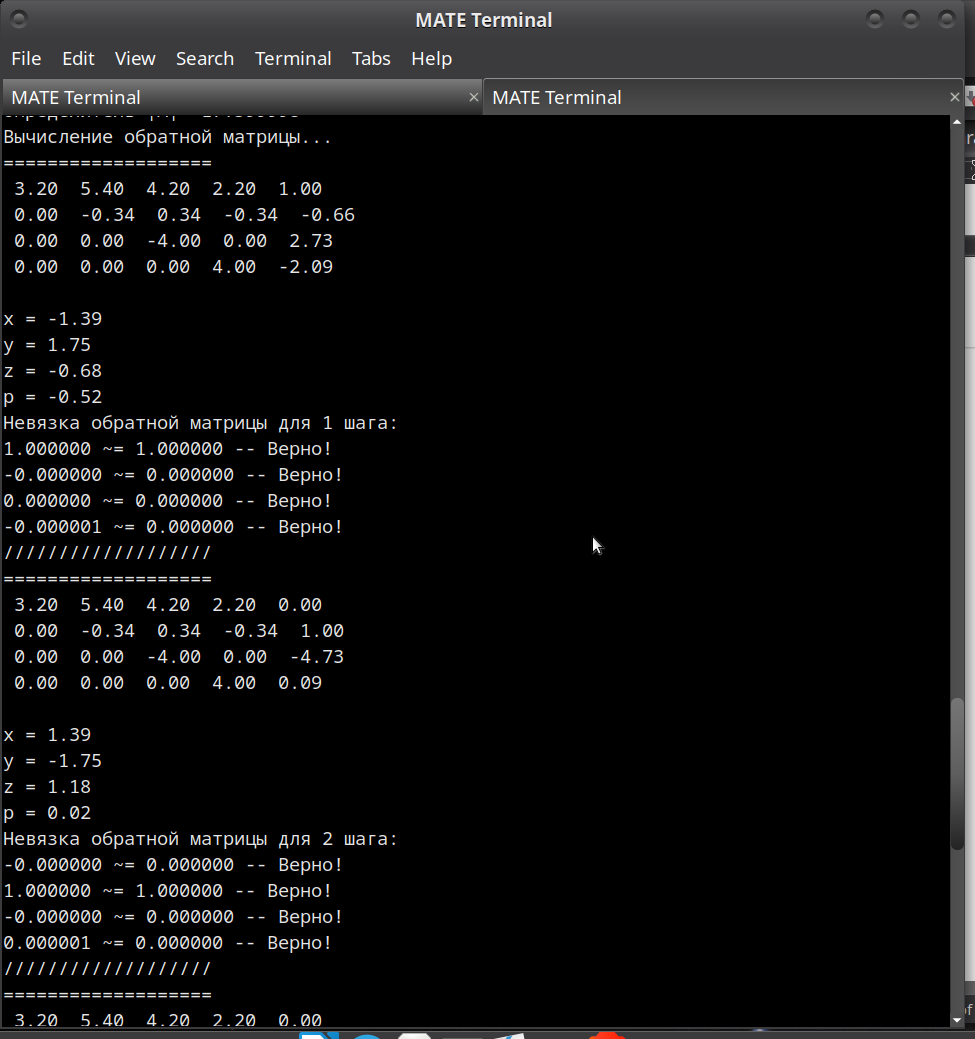
Задачи 6.2.1, 6.2.2, 6.2.3 решаются в одной программе, т.к. видится нецелесообразным разрабатывать три почти идентичные программы, тем более, что матрицы для последних двух задач берутся из СЛАУ задачи 6.2.1. Поэтому, следующий метод вычисляет определитель матрицы, просто перемножая коэффициенты на главной диагонали, уже преобразованной, методом выше, к треугольному виду матрицы. Здесь также используются вложенные циклы. После выполнения метода, вызыватся выражение выводящее значение определителя на экран.

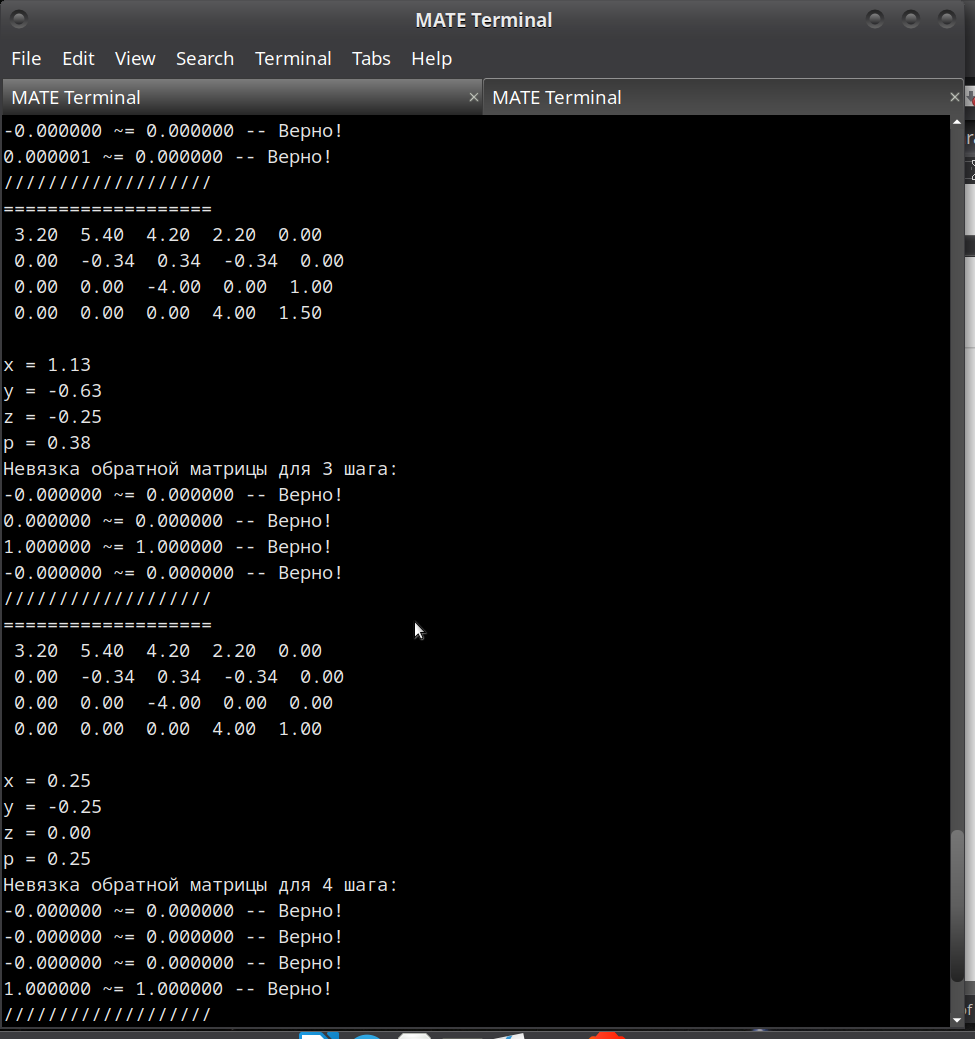
Последний вызванный метод вычисляет обратную матрицу, получая параметром матрицу из СЛАУ задачи 6.2.1. В методе инициализируются две вспомогательные матрицы, одна из которых никак не преобразовывается а просто сохраняется в первоначальном виде для вычисления невязки обратной матрицы. Невязка здесь вычисляется потому, что для вычисления обратной матрицы решаются четыре системы уравнений, что очевидно, влечет за собой риск погрешности вычислений. В методе четыре вложенных цикла: первый цикл назначает номер прохода — на каждом вызываются уже описанные выше методы: приведение матрицы к треугольному виду; метод обратной подстановки; вывод обратной матрицы на экран; вычисление невязки для СЛАУ с соответствующими символами Кронекера. В первый цикл вложены еще два цикла — в первом из них назначается соответствующий символ Кронекера, проходя по строкам матрицы, во втором заполняется обратная матрица. В первом вложенном цикле содержится еще один вложенный цикл который проходя по коэффициентам строки заполняет вспомогательные матрицы, которые будут использованы в качестве параметров для методов содержащихся в наружном цикле.

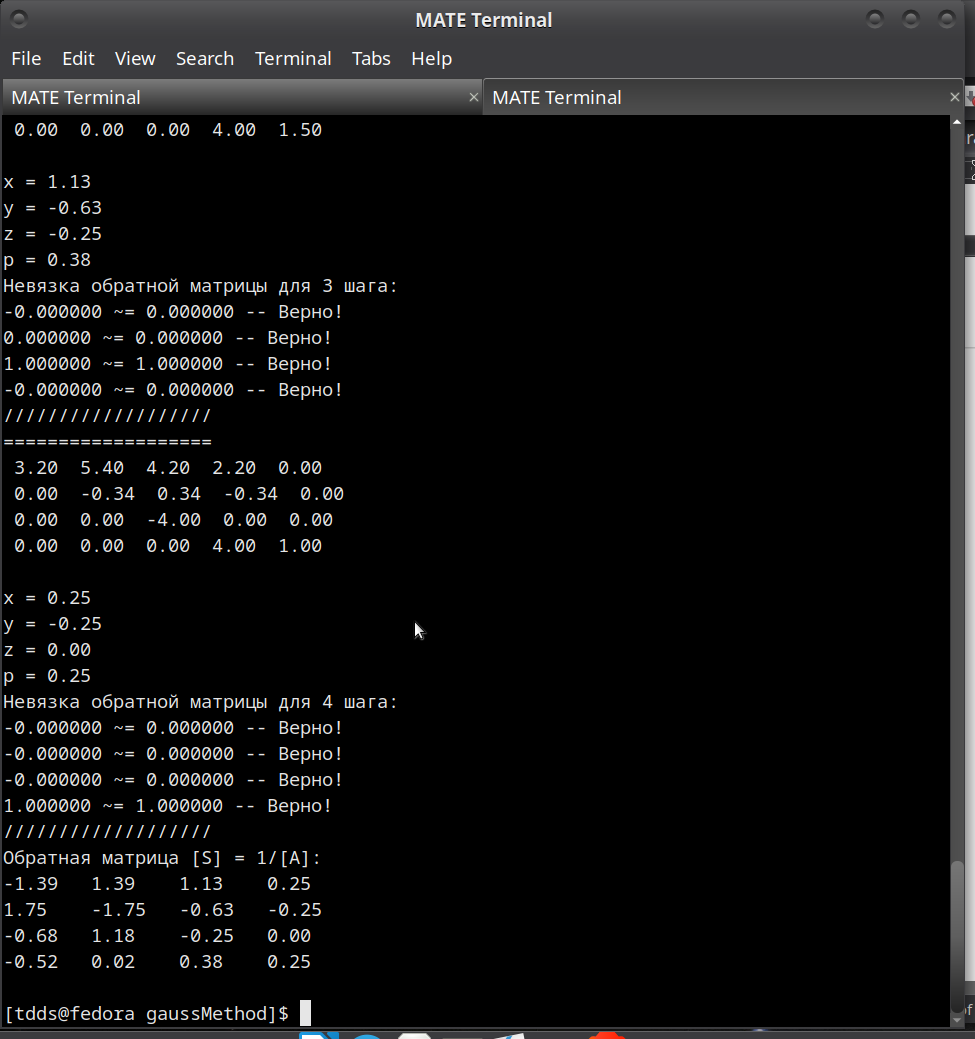
**2.4 Результат работы программы**

На рис.2, рис.3, рис.4, и рис.5 приведены результаты программы.

Рисунок 2.

Рисунок 3.

Рисунок 4.

Рисунок 5.

**Выводы**

В ходе лабораторной работы применены алгоритмы численных методов для написания соответствующих програм на языке С. Очевидно преимущество во времени решения подобных задач именно численными методами, к примеру для решения СЛАУ 4 ранга методом Гаусса потребовалось 16 преобразований изначальной матрицы, плюс четыре подстановки обратным ходом. Для вычисления обратной матрицы потребовалось 80 действий, плюс решение элементарных уравнений. Понятно, в рамках учебных примеров приведены, очевидно, не самые сложные уравнения и СЛАУ. Однако, если бы нужно было обратить матрицу с рангом, к примеру, 100 то потребовались бы усилия целого коллектива. Более того, аналитическими методами решаются очень малый процент задач, здесь конечно, численные методы вне всякой конкуренции, не взирая на погрешности в вычислениях, которые также вычисляются, оцениваются и принимаются во внимание при решении реальных задач.

**Список использованных источников**

1. Мицель А. А. Вычислительные методы : учебное пособие / А. А. Мицель. — Томск : ЭльКонтент, 2013. — 198 с.

2. Мицель А. А., Романенко В. В. Вычислительная математика : методические указания по выполнению контрольной и лабораторных работ / А. А. Мицель, В. В. Романенко. – Томск : ФДО, ТУСУР, 2019. – 119 с.

3. Н.И. Данилина, [и др]. Численные методы. - Москва «Высш. Школа», 1976. 386с.

4. C How to Program: with an introduction to C++, 8 st edition, ISBN 978-0-13-397689-2, by Paul Deitel and Harvey Deitel published by Pearson Education © 2016. - 1005p.

5. B.M. Harwani C Programming Cookbook. Over 40 recipes exploring data structures, pointers, interprocess communication, and database in C Published by Packt Publishing Ltd. Livery Place 2019. - 335p.

6. <https://www.geeksforgeeks.org/c-programming-language/?ref=shm>

Приложение А

Листинг 1. Решение нелинейных уравнений

#include <stdio.h>

#include <math.h>

#include <time.h>

float dihot(float a, float b, float e, float e1);//Инициализация метода дихотомии

float func( float x);// Инициализация метода заданной функции

int sign(float x);// Инициализация метода определяющего знак

int n = 0;

int main()

{

clock\_t start, end;

start = clock();

printf("x = %f\n", dihot(0, 2, 0.0001, 0.0001));// Вывод на экран решения уравнения. Параметром вывода является результат метода реализующего, собственно, метод дихотомии. В свою очередь, параметрами функции являются начало и конец отрезка, на котором находится решение, эти цифры можно менять; точность с которой будет решаться уравнение, значение, с которым будет сравниваться значение заданной функции на предмет приближения к нулю (по известной теореме, это и будет решением уравнения) соответственно.

end = clock();

double time\_taken = ((double)(end - start))/CLOCKS\_PER\_SEC;

printf("Количество итераций равно: %d\n", n);

printf("Времени затрачено: %f секунд\n",time\_taken);

}

float func( float x)//Метод возвращает значение заданной функции, вычисленной от текущего параметра

{

return sqrt(4\*x+7)-3\*cos(x); //Вариант 6

}

int sign(float x)//Метод возвращает 1 либо -1 в зависимости от поступившего параметра. Параметром является текущее вычисленное х.

{

int res;

res=0;

if (x<0) res=-1;

if (x>0) res= 1;

return res;

}

float dihot(float a, float b, float e, float e1)//Метод посредством цикла while делит заданный отрезок пополам до тех пор пока разница между концами отрезка не будет меньше или равна параметру е, либо абсолютное значение функции в текущей точке приблизится к нулю менее чем параметр е1

{

float x;

while (b-a>e)

{

n++;

x=(a+b)/2;// Вычисление значения искомой переменной, собственно, половинное деление/метод дихотомии

printf("Проход %d середина отрезка есть: %f \n", n, x);

if (fabs(func(x))<e1)

break; // Если асолютное значение функции в соответсвующей точке приближается к нулю с заданной точностью, цикл прекращается, и метод возвращает текущее значение х, которое и является решением уравнения с заданной точностью

if (sign(func(a))==sign(func(x))) {

a=x; // Значению начальной точки присваивается значение текущей искомой переменной, если значение функции в начально точке равно значению функции в точке х

}else

b=x; // Если предыдущее условие не выполнено значение переменной х присваивается концу заданного отрезка, на котором ищется решение

}

return x; //Метод возвращает значение переменной удовлетворяющее первому условию в цикле выше

}

Листинг 2. Решение СЛАУ методом Гаусса, вычисление определителя матрицы, обращение матрицы

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <stdbool.h>

#define row 4 //Установка константы количества уравнений/переменных в системе

void beTriangleMatrix (float system[row][row+1]); // Инициализация методов

void checkerZero (float system[row][row+1]);

void reverseSubs (float system[row][row+1]);

void printer (float system[row][row+1], int params);

void descrepancy(float system[row][row+1]);

float unvar[row]; //Инициализация массива значений искомых переменных.

float determinant = 1; // Инициализация переменной опеределителя и присвоение значения по умолчанию

float sMatrix[row][row+1];// Инициализация двумерного массива обратной матрицы [S]

void checkerZero (float system[row][row+1]) // Метод избавляется от нулей на осевых элементах

{

for (int i=0; i<row; i++) // Цикл пробегает по всем уравнениям системы

{

for(int j=0; j<row+1; j++) // Цикл пробегает по всем коэффициентам каждого уравнения

{

while (i==j && system[i][j]==0 && i!=row-1) // Если номер уравнения в системе равен номеру коэффициента, а коэффициент в свою очередь равен нулю, то к каждому коэффициенту прибавляется соотоветствующий коэффициент следующего в системе уравнения, умноженный на -1 (элементарные преобразования системы линейных уравнений)

{

for(j=0; j<row+1; j++)

{

system[i][j] += (system[i+1][j])\*(-1);

}

}

while (i==j && system[i][j]==0 && i==row-1) // Т.к. для последнего уравнения в системе нет следующего, что очевидно, к соответствующим коэффициентам последнего прибавляются коэффициенты предыдущего, также умноженные на -1, что не противоречит правилам элементарных преобразований

{

for(j=0; j<row+1; j++)

{

system[i][j] += (system[i-1][j])\*(-1);

}

}

}

}

}

void beTriangleMatrix (float system[row][row+1]) //Метод получает на вход матрицу и преобразует ее к т.н. треугольному (сверху) виду

{

float divVal; // Инициализация переменной для замены частного коэффициентов.

for (int k=0; k<row; k++)// Цикл назначает номер шага исключения переменной

{

for (int i=k+1; i<row; i++)// Цикл пробегает по всем уравнениям системы, при том, что номер уравнения системы равен номеру шага исключения увеличенного на единицу, т.е. следующего

{

float divVal = system[i][k]/system[k][k];//Вспомогательная переменная назначается как результат деления коэффициента следующего уравнения на соответсвующий коэффициент текущего уравнения

for (int j=k+1; j<=row; j++)// Цикл пробегает по всем коэффициентам текущего уравнения(строки)

system[i][j] -= system[k][j]\*divVal;// Каждый последующий коэффициент уравнения равен разнице между соответствующим коэффициентом на предыдущим шаге и коэффициентом на текущем шаге помноженным на вспомогательную переменную значение которой описано при ее назначении см. соответствующий комментарий

system[i][k]=0;// Коэффициенты не учавствующие в последующем расчете приравниваются к нулю

}

}

}

void reverseTriangle(float system[row][row+1] ) //Метод получает на вход матрицу и вычисляет обратную ей матрицу

{

float auMat[row][row+1];// Инициализация вспомогательной матрицы, в которой правой части присваиваются соответствующие значения символа Кронекера

float auMat2[row][row+1];// Инициализация вспомогательной матрицы, в которой правой части присваиваются соответствующие значения символа Кронекера, эта матрица не преобразовывается для вычисления невязки обратной матрицы

printf("Вычисление обратной матрицы...\n");

for (int t = 0; t<row; t++)

{

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

{

auMat[i][row] = 0; //Присвоение правой части соответсвующего уравнению соответствующего символа Кронекера

auMat2[i][row] = 0;// --//--

auMat[t][row] = 1;// --//--

auMat2[t][row]= 1; // --//--

for(int j=0; j<row; j++)// Присвоение левой части соответствующих значений изначальной матрицы

{

auMat[i][j] = system[i][j];

auMat2[i][j] = system[i][j];

}

}

puts("===================");

beTriangleMatrix(auMat);// Вызов метода приводящего полученную матрицу к треугольному виду

printer(auMat, 1);// Вывод на экран преобразованной треугольной матрицы

reverseSubs(auMat); //Вызов функции для проведения обратной подстановки текущей преобразованной матрицы

printf("Невязка обратной матрицы для %d шага: \n", t+1);

descrepancy(auMat2); //Вызов метода для вычисления невязки обратной матрицы

for(int u = 0; u<row; u++) // Цикл заполняет столбцы искомой обратной матрицы

{

sMatrix[u][t] = unvar[u]; // Решения соответcтвующей системы являются соответствующим столбцом обратной матрицы

//printf("%.2f\t\n", sMatrix[u][t]);// Вывод решений на экран

}

puts("///////////////////");

}

}

void reverseSubs (float system[row][row+1])// Метод реализует метод обратной подстановки. Параметром является какая-либо матрица.

{

char solution[]="xyzpq"; //Инициализация массива искомых переменных.

for(int i=row-1; i>=0; i--) // Цикл решает систему уравнений методом обратной подстановки опираясь на значение последней переменной назначенной ранее

{

unvar[i] = system[i][row];// Назначение неизвестной переменной с номером, который соответствует номеру уравнения (счет уравнений ведется снизу вверх, что следует из названия метода)

for(int j=i+1; j<row; j++)// Цикл вычисляет вспомогательное значение на основе предыдущих значений и соответствующих значений треугльной матрицы

{

unvar[i] -= system[i][j]\*unvar[j]; // В теле цикла назначенное значение переназначается. В данном случае это значение является вспомогательным.

}

unvar[i] /= system[i][i]; // Окончательное вычисление и присвоение соответствующего значения в массиве(наполнение массива вычисленными значениями)

}

for(int i=0; i<row; i++)//Цикл выводит на экран вычисленные значения переменных

printf("%c = %.2f\n", solution[i], unvar[i]);

}

void printer (float system[row][row+1], int params) //Метод выводит на экран матрицу, поступающую на вход как параметр метода, второй параметр вводится для того, чтобы выводить матрицу иной размерности

{

if(params !=1)

printf("Обратная матрица [S] = 1/[А]:\n");

for (int i=0; i<row; i++)// Цикл выводит на экран значения матрицы, если значение на диагонали равно нулю оно выделяется квадратными скобками.

{

if(params == 1){

for (int j=0; j<row+1; j++)

{

if (system[i][j] == 0 && i==j)

{

printf(" [%.2f]",system[i][j]);

} else {

printf(" %.2f ",system[i][j]);

}

}

} else if(params == 0) {

for(int j = 0; j<row; j++) {

printf("%.2f\t", system[i][j]);

}

}

puts("");

}

puts("");

}

void auxilaryRate(float system[row][row+1], float auxilaryRate[row][row+1])// Метод записывает вспомогательную матрицу для вычисления обратной матрицы, т.к. исходная уже приведена к треугольному виду

{

for (int i=0; i<row; i++) // Цикл пробегает по всем уравнениям системы

{

for(int j=0; j<row+1; j++) // Цикл пробегает по всем коэффициентам каждого уравнения

{

auxilaryRate[i][j] = system[i][j];

}

}

}

void descrepancy(float system[row][row+1])// Метод вычисляет невязку с точностью до тысячной доли

{

float sum;

bool truFal;

float bVal;

for (int i=0; i<row; i++) // Цикл пробегает по всем уравнениям системы

{

sum = 0;

for(int j=0; j<row; j++) // Цикл пробегает по коэффициентам каждого уравнения, кроме свободного члена b

{

sum += system[i][j]\*unvar[j];//Собственно, вычисление невязки

if (sum - system[i][row] <= 0.001)

truFal = true;

bVal = system[i][row];//Переменной присваивается значение свободного члена

}

if (truFal == true)

printf ("%f ~= %f -- Верно!\n", sum, bVal);//Если правые и левые части уравнения отличаются на одну тысячную или менее выводится сообщение о верности решения

}

}

void determ(float system[row][row+1])// Метод вычисляет определитель матрицы методом Гаусса используя уже преобразованную, треугольную матрицу, хотя параметром может быть любая матрица соответствующей размерности, но тогда, значение переменной determinant не имеет смысла

{

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

{

float multy;

for( int j = 0; j<row; j++)

{

if (i == j )

multy = system[i][j];

}

//printf("%f\n", multy);

determinant \*= multy;

}

}

int main ()

{

float rateAu[row][row+1];//Инициализация первоначальной матрицы

float rate[row][row+1] = { // Матрица заданной системы уравнений, предствленная в виде двумерного массива. Вариант 6.

{3.2, 5.4, 4.2, 2.2, 11.4},

{2.1, 3.2, 3.1, 1.1, 9.2},

{1.2, 0.4, -0.8, -0.8, 0.4},

{4.7, 10.4, 9.7, 9.7, 30.4}

};

auxilaryRate(rate, rateAu);//Вызов метода для сохранения первоначальной матрицы

printf("Дано: \n");

printer(rate, 1); //Вывод на экран первоначальной матрицы

printf("Удаление нулей на диагонали матрицы:\n");

checkerZero(rate); // Преобразование матрицы к виду: без нулей на осевых элементах

printer(rate, 1);//Вывод на экран прeобразованной матрицы

printf("Преобразование матрицы к треугольному виду:\n");

beTriangleMatrix(rate);// Преобразование матрицы к треугольному виду

printer(rate, 1);//Вывод на экран треугольной матрицы

printf("Ответ:\n");

reverseSubs(rate);//Метод обратной подстановки и вывод на экран значений соответствующих переменных, решенной системы линейных уравнений

descrepancy(rateAu);//Вызов метода вычисляющего невязку. Проверка.

determ(rate);//Вычисление определителя методом Гаусса.

printf("Определитель |A|= %f\n", determinant);// Вывод на экран значения определителя для заданной матрицы

//printer(rateAu, 1);// Вывод вспомогательной (первоначальной,т.к. матрица rate уже преобразована в треугольную) матрицы на экран

reverseTriangle(rateAu);// Вызов метода для вычисления обратной матрицы, и ее невязки

printer(sMatrix, 0);// Вывод обратной матрицы на экран

}