Министерство образования Республики Беларусь

Учреждение образования “Брестский Государственный Технический Университет”

Кафедра ИИТ

Лабораторная работа №2

По дисциплине Вычислительная математика

"**Решение систем линейных алгебраических уравнений (СЛАУ)"**

Выполнил: студент 1 курса

Группы АС-59

Быбко Т.А.

Проверил: Пролиско Е.Е.

Брест 2021

Вариант 46

Цель: применить на практике три метода для решения СЛАУ, написать программу их выполнения.

Условие задания: решить систему линейных алгебраических уравнений, заданных расширенной матрицей

-10.1 -8.8 -0.3 -118.1

-9.4 12.1 -1.9 -40.1

2.1 -5.7 -10.0 -76.0

Решение получить методом Гаусса, а также методом простых итераций и методом Зейделя с точностью до 10^(-6).

Ход работы:

Напишем программу на С++, выполняющая методы Гаусса, простых итераций и Зейделя для решения СЛАУ.

Код программы:

#include <iostream>

#include <iomanip>

#include <stdlib.h> //Для функции max()

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

cout << setprecision(10); //Определяем точность выводимых значений

//В матрицах cначала строки, а потом столбцы!

double A[3][3] = { {-10.1, -8.8, -0.3}, {-9.4, 12.1, -1.9}, {2.1, -5.7, -10.0} };

double B[3][1] = { {-118.1}, {-40.1}, {-76.0} };

double X[3][1] = { {0}, {0}, {0} };

double opr = 0.0; //Определитель

cout << " Исходная матрица: " << endl;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

cout << setw(6) << A[i][j] << " ";

cout << " | ";

cout << setw(6) << B[i][0] << endl;

double diagDOWN= A[0][2]\*A[1][1]\*A[2][0]; //рассчитываем нижнюю (побочную) диагональ

double diagUP = A[0][0]\*A[1][1]\*A[2][2]; //рассчитываем верхнюю (главную) диагональ

opr = diagUP - diagDOWN; //находим определитель матрицы

}

cout << "-------------------------------------------------------------------------------------------------------------" << endl;

cout << "Применим метод Гаусса" << endl;

cout << "Определитель матрицы: " << opr << ", не равен нулю, а значит система имеет единственное решение." << endl;

cout << "Применим прямой ход метода Гаусса: " << endl;

double AB[3][4];

cout << "Расширенная матрица: " << endl;

for (int i = 0; i < 3; i++) //создаём расширенную матрицу

{

for (int j = 0; j < 3; j++)

{

AB[i][j] = A[i][j];

cout << setw(6) << AB[i][j] << " ";

}

AB[i][3] = B[i][0];

cout << setw(8) << AB[i][3] << endl;

}

cout << "Расширенная матрица после прямого хода: " << endl;

double first = 0.0, second = 0.0;

for(int i = 0; i < 3; i++) //приводим матрицу к ступенчатому виду

{

for (int j = 0; j < 3; j++) //определяем "опорный" элемент в нужной строке

{

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

{

first = AB[i][j];

break;

}

}

for (int j = 0; j < 4; j++) /\*делим всё на "опорный" элемент, чтобы получить единицу в нужном столбце и облегчить дальнейшие вычисления \*/

{

AB[i][j] /= first;

}

for (int k = i + 1; k < 3; k++) //перебираем строки, ниже текущей

{

for (int j = 0; j < 3; j++) //определяем второй "опорный" элемент

{

if (AB[k][j] != 0)

{

second = AB[k][j];

break;

}

}

for (int j = 0; j < 4; j++) //сам процесс "Обнуления"

{

AB[k][j] = AB[k][j] - AB[i][j] \* second;

}

}

}

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 4; j++)

cout << setw(15) << AB[i][j] << " ";

cout << endl;

}

cout << "Применим обратный ход метода Гаусса и найдём корни: " << endl;

//решаем полученную при прямом ходе систему

X[2][0] = AB[2][3];

X[1][0] = AB[1][3] - AB[1][2] \* X[2][0];

X[0][0] = AB[0][3] - AB[0][2] \* X[2][0] - AB[0][1] \* X[1][0];

cout << "x1 = " << X[0][0] << " x2 = " << X[1][0] << " x3 = " << X[2][0] << endl;

cout << "-------------------------------------------------------------------------------------------------------------" << endl;

cout << "Применим итерационные методы." << endl;

double E = 1e-6; //Точность

/\*изменённая матрица под итерационные методы (выражая х1 из первого уравнения, х2 из второго и х3 из третьего): \*/

double C[3][3] = { {0, -0.87128713, -0.02970297}, {0.77685950, 0, 0.15702479}, {0.21, -0.57, 0} };

//матрица свободных членов:

double D[3][1] = { {11.69306931}, {-3.31404959}, {7.6} };

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

cout << setw(15) << C[i][j] << " ";

cout << " | ";

cout << setw(15) << D[i][0] << endl;

}

double KubNorm = 0.0;

//Определим кубическую норму

KubNorm = max(fabs(C[0][0] + C[0][1] + C[0][2]), fabs(C[1][0] + C[1][1] + C[1][2]) );

KubNorm = max(KubNorm, fabs(C[2][0] + C[2][1] + C[2][2]) );

//Т.к. функция max() принимает только два значения, сравнение пришлось проводить дважды

cout << "Кубическая норма матрицы равна: " << KubNorm << " меньше 1, значит итерационные методы сходятся к точному решению" << endl;

cout << "-------------------------------------------------------------------------------------------------------------" << endl;

cout << "Применим метод простых итераций" << endl;

double nev = 0.0; //Переменная для невязки

double X0[3] = { 0 }; //Начальное приближение

double X1[3] = { 0 }; //Следующая итерация

double razn = 0.0; //Понадобится для условия остановки цикла

int iter = 0; //подсчёт итераций

do //Сам процесс

{

X1[0] = 1 / A[0][0] \* (B[0][0] - A[0][1] \* X0[1] - A[0][2] \* X0[2]);

X1[1] = 1 / A[1][1] \* (B[1][0] - A[1][0] \* X0[0] - A[1][2] \* X0[2]);

X1[2] = 1 / A[2][2] \* (B[2][0] - A[2][0] \* X0[0] - A[2][1] \* X0[1]);

razn = max(fabs(X1[0] - X0[0]), fabs(X1[1] - X0[1]));

razn = max(razn, fabs(X1[2] - X0[2]));

X0[0] = X1[0];

X0[1] = X1[1];

X0[2] = X1[2];

iter++;

} while (KubNorm / (1 - KubNorm) \* razn > E); // Условие продолжения цикла

cout << "x1 = " << X1[0] << " x2 = " << X1[1] << " x3 = " << X1[2] << endl;

cout << "Количество итераций: " << iter << endl;

cout << "Невязка функции: ";

nev = max(fabs(B[0][0] - A[0][0] \* X1[0] - A[0][1] \* X1[1] - A[0][2] \* X1[2]), fabs(B[1][0] - A[1][0] \* X1[0] - A[1][1] \* X1[1] - A[1][2] \* X1[2]));

nev = max(nev, fabs(B[2][0] - A[2][0] \* X1[0] - A[2][1] \* X1[1] - A[2][2] \* X1[2]));

cout << nev << endl;

cout << "-------------------------------------------------------------------------------------------------------------" << endl;

cout << "Применим метод Зейделя" << endl;

for (int i = 0; i < 3; i++)

{

X0[i] = 0;

X1[i] = 0;

}

//Изменённая матрица под метод Зейделя (разложим на две треугольные (C = C1 + C2))

double C2[3][3] = { {0, -0.87128713, -0.02970297}, {0, 0, 0.15702479}, {0, 0, 0} };

double KubNorm2 = 0.0; //Кубическая норма матрицы С2

KubNorm2 = max(fabs(C2[0][0] + C2[0][1] + C2[0][2]), fabs(C2[1][0] + C2[1][1] + C2[1][2]));

//Т.к. 3 строка вся в нулях, поэтому достаточно одной функции max()

iter = 0; //Обнуление счётчика

do //Сам процесс метода

{

X1[0] = 1 / A[0][0] \* (B[0][0] - A[0][1] \* X0[1] - A[0][2] \* X0[2]);

X1[1] = 1 / A[1][1] \* (B[1][0] - A[1][0] \* X1[0] - A[1][2] \* X0[2]);

X1[2] = 1 / A[2][2] \* (B[2][0] - A[2][0] \* X1[0] - A[2][1] \* X1[1]);

razn = max(fabs(X1[0] - X0[0]), fabs(X1[1] - X0[1]));

razn = max(razn, fabs(X1[2] - X0[2]));

X0[0] = X1[0];

X0[1] = X1[1];

X0[2] = X1[2];

iter++;

} while (KubNorm2 / (1 - KubNorm) \* razn > E); // Условие продолжения цикла

cout << "x1 = " << X1[0] << " x2 = " << X1[1] << " x3 = " << X1[2] << endl;

cout << "Количество итераций: " << iter << endl;

cout << "Невязка функции: ";

nev = max(fabs(B[0][0] - A[0][0] \* X1[0] - A[0][1] \* X1[1] - A[0][2] \* X1[2]), fabs(B[1][0] - A[1][0] \* X1[0] - A[1][1] \* X1[1] - A[1][2] \* X1[2]) );

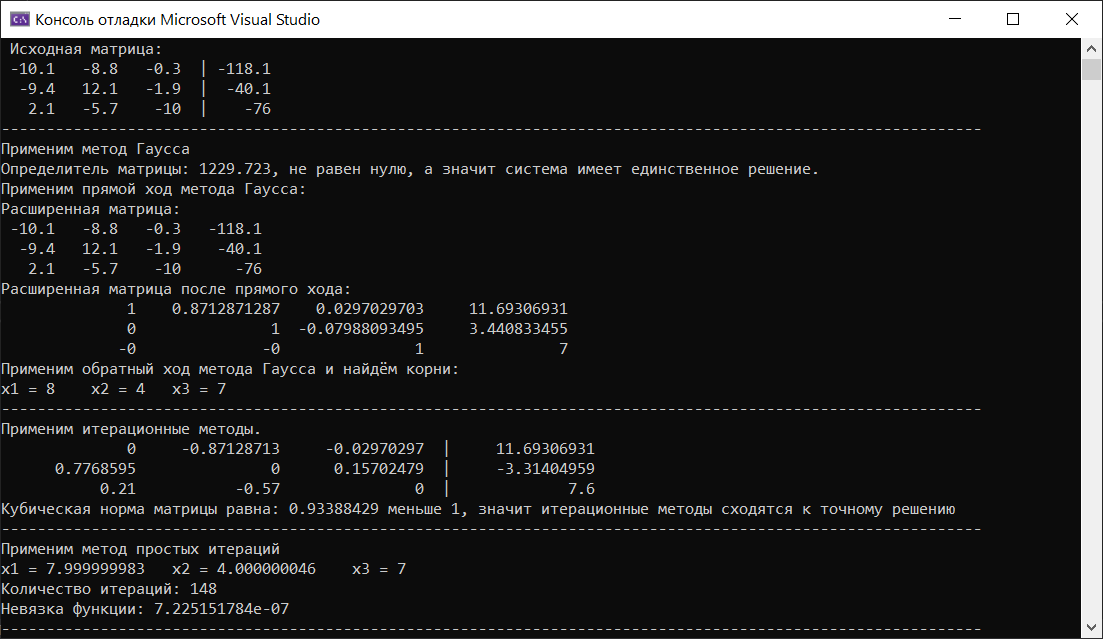
nev = max(nev, fabs(B[2][0] - A[2][0] \* X1[0] - A[2][1] \* X1[1] - A[2][2] \* X1[2]) );

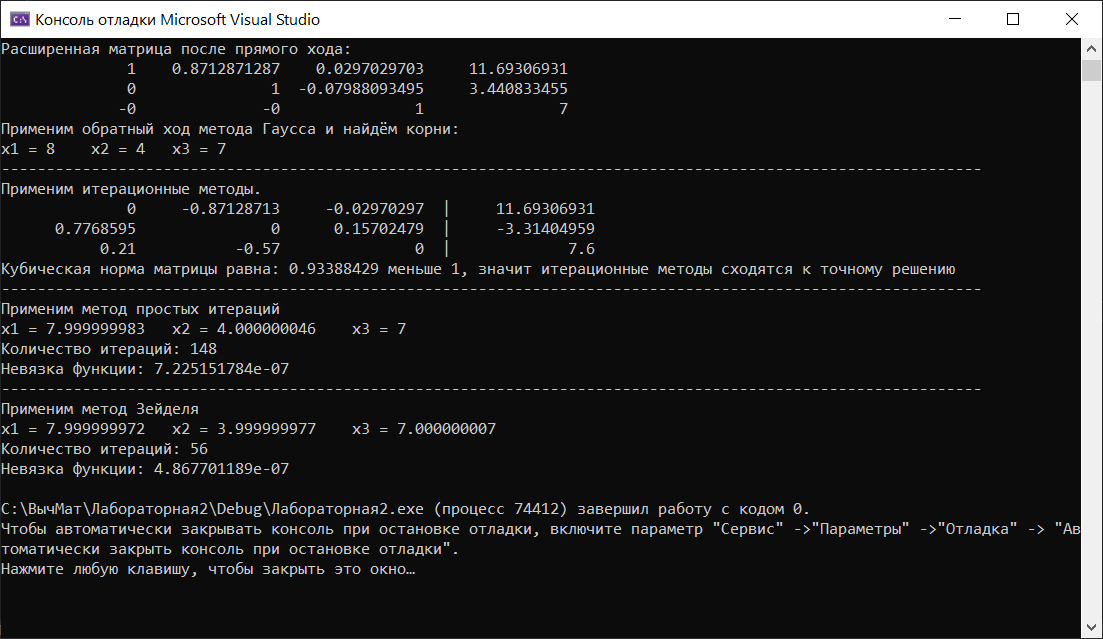
cout << nev << endl;

return 0;

}

Результаты выполнения программы:





Вывод: СЛАУ была решена всеми тремя необходимыми способами. На удивление, наиболее быстро реализованным методом стал именно метод Гаусса, возможно из-за того, что мы изучали его ранее в первом семестре, что сэкономило время на изучение теории. Однако его реализация тоже не была простой, особенно это касается попытки автоматизировать обратных ход метода, для которого пришлось использовать множество циклов (в том числе тройной). Итерационные методы были не так сложны в реализации, как в самом освоении. Но на погружении в теоретические сведения и реализации метода на С++ дело не закончилось. Дело в том, что пришлось выявлять множество ошибок и исправлять их, на что уходило очень много времени. Также, как видно из выше прикреплённых скриншотов, метод простых итераций имеет 148 итераций, что довольно много. Долгое время я думал, что ошибка у меня в программе или я не так понял теорию, однако через некоторое время я решил проверить свою программу и вставил в неё данные из методички. Всё сработало как надо. После этого я также вставил в программу начальные данные из нескольких других случайных вариантов, результаты также были удовлетворительными. После всего этого, я сделал вывод, что такое количество итераций – норма для моего варианта.