Київський національний університет

імені Т.Шевченка

Звіт

до лабораторної роботи №2

на тему:

«Розв’язання систем лінійних алгебраїчних рівнянь**»**

Студентки 2 курсу

Групи К-24

Факультету комп’ютерних наук

та кібернетики

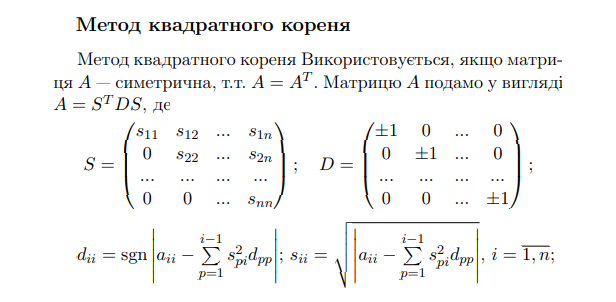
Віткіної Анни

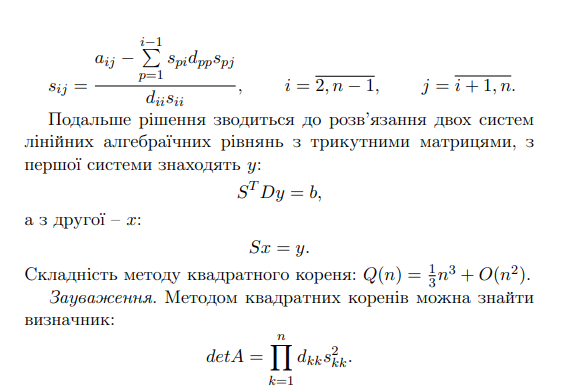
Київ-2022

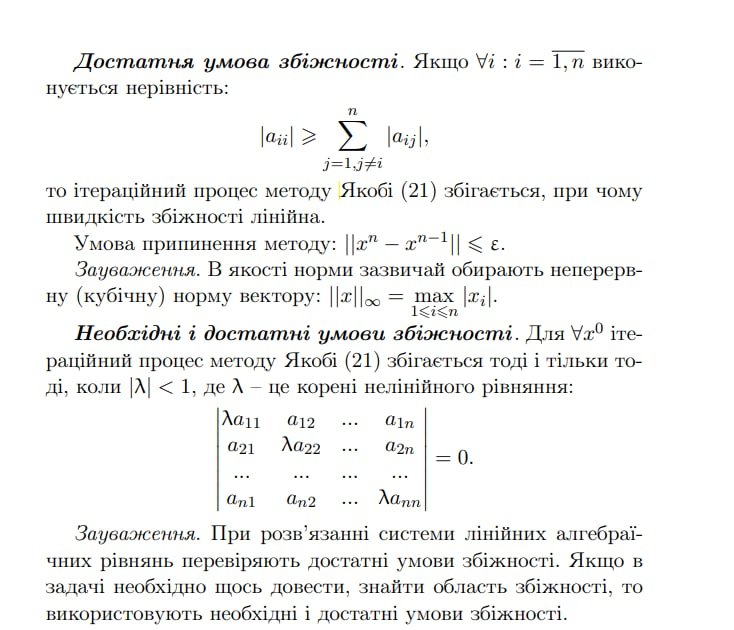
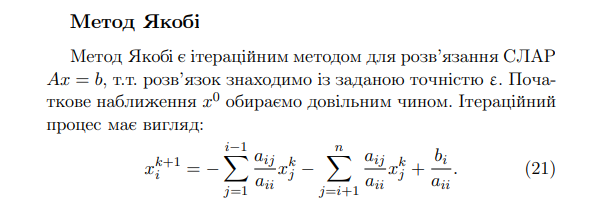
Постановка задачі. Варіант №5

Написати програму, яка розв’язує систему лінійних алгебраїчних рівнянь методом квадратних коренів та методом Якобі.

Теоретичні матеріали, використані у лабораторній робот**і**

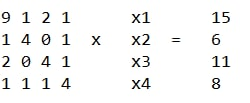
****

****



**Задача і деякі обчислення**

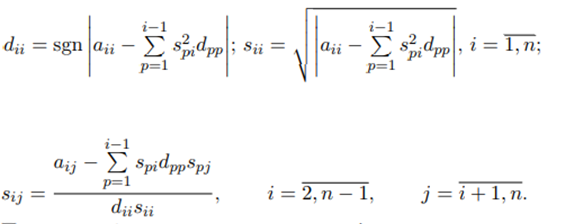
# A b

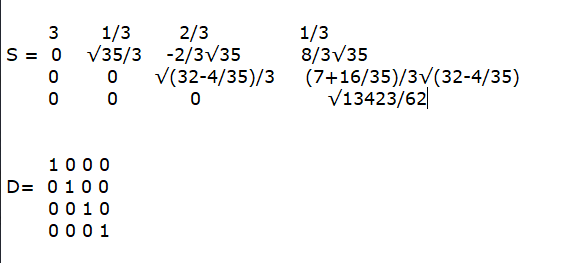


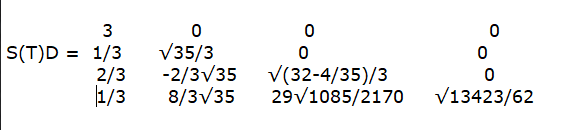
*Розв’язати систему рівнянь.*

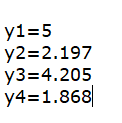
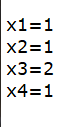
1) Метод квадратного кореня:

Знаходимо матриці S i D.



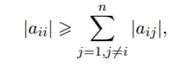




2) Метод Якобі

Умова збіжності:



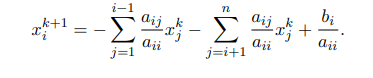
9=a11>=1+2+1

4=a22>=1+0+1

4=a33>=2+0+1

4=a44>=1+1+1

Формула ітерації:



**Опис програм**

**I.Метод квадратного кореня:**

#include<iostream>

#include "math.h"

#include <iomanip>

using namespace std;

template <class Value>

int sign(Value Val) {

if (Val == 0.) return 0;

if (Val > 0.) return 1;

else return -1;

}

bool IsMartOK(float a[4][4]) **//Перевірка матриці на правильність**

{

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

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

{

if (a[i][j] != a[j][i])

{

return 0;

}

}

return 1;

}

void PrintMatrix(float a[4][4]) **//вивід матриці на екран**

{

cout << "========================" << endl;

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

{

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

{

cout<< a[i][j] << " ";

}

cout << endl;

}

}

float findDET(float matS[4][4], float matD[4][4]) **//detA**

{

float result = 1;

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

{

result \*= matS[i][i] \* matS[i][i] \* matD[i][i];

}

return result;

}

void Generate(float matS[4][4], float matD[4][4], float matA[4][4]) **//знайти S i D**

{

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

{

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

{

matS[i][j] = 0;

matD[i][j] = 0;

}

}

matD[0][0] = sign(matA[0][0]);

matS[0][0] = sqrt(matA[0][0]);

float temp;

temp = (matD[0][0] \* matS[0][0]);

matS[0][1] = matA[0][1] / temp;

matS[0][2] = matA[0][2] / temp;

matS[0][3] = matA[0][3] / temp;

temp = matA[1][1] - matS[0][1] \* matS[0][1] \* matD[0][0];

matD[1][1] = sign(temp);

matS[1][1] = sqrt(abs(temp));

temp = (matD[1][1] \* matS[1][1]);

matS[1][2] = (matA[1][2] - matS[0][1] \* matS[0][2] \* matD[0][0]) / temp;

matS[1][3] = (matA[1][3] - matS[0][1] \* matS[0][3] \* matD[0][0]) / temp;

temp = matA[2][2] - matS[0][2] \* matS[0][2] \* matD[0][0] - matS[1][2] \* matS[1][2] \* matD[1][1];

matD[2][2] = sign(temp);

matS[2][2] = sqrt(abs(temp));

matS[2][3] = (matA[2][3] - matS[0][2] \* matS[0][3] \* matD[0][0] - matS[1][2] \* matS[1][3] \* matD[1][1]) / (matD[2][2] \* matS[2][2]);

temp = matA[3][3] - matS[0][3] \* matS[0][3] \* matD[0][0] - matS[1][3] \* matS[1][3] \* matD[1][1] - matS[2][3] \* matS[2][3] \* matD[2][2];

matD[3][3] = sign(temp);

matS[3][3] = sqrt(abs(temp));

}

int main()

{

setlocale(LC\_ALL, "rus");

float matS[4][4];

float matD[4][4];

float x[4];

float y[4];

float matA[4][4] = {{9,1,2,1}, {1,4,0,1},{2,0,4,1},{1,1,1,4}}; //**Наш приклад**

float b[4] = {15,6,11,8};

bool q = IsMartOK(matA);

if (q) cout << "Матриця пiдходить" << endl;

else cout << "Матриця не пiдходить" << endl;

Generate(matS, matD, matA);

PrintMatrix(matS);

PrintMatrix(matD);

//

float tempmat[4][4];

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

{

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

{

if (matD[j][j] == 1)

tempmat[i][j] = matS[j][i];

else tempmat[i][j] = (-1) \* matS[j][i];

}

}

//cout << endl << endl;

//PrintMatrix(tempmat);

**//Знаходимо вектор у**

y[0] = b[0] / tempmat[0][0];

y[1] = (b[1] - y[0] \* tempmat[1][0]) / tempmat[1][1];

y[2] = (b[2] - y[0] \* tempmat[2][0] - y[1] \* tempmat[2][1]) / tempmat[2][2];

y[3] = (b[3] - y[0] \* tempmat[3][0] - y[1] \* tempmat[3][1] - y[2] \* tempmat[3][2]) / tempmat[3][3];

**//Знаходимо вектор х**

x[3] = y[3] / matS[3][3];

x[2] = (y[2] - matS[2][3] \* x[3]) / matS[2][2];

x[1] = (y[1] - matS[1][2] \* x[2] - matS[1][3] \* x[3]) / matS[1][1];

x[0] = (y[0] - matS[0][1] \* x[1] - matS[0][2] \* x[2] - matS[0][3] \* x[3]) / matS[0][0];

cout << "x1 = " << x[0] << " x2 = " << x[1] << " x3 = " << x[2] << " x4 = " << x[3] << endl;

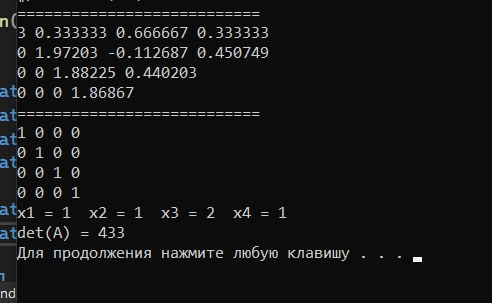
cout << "det(A): " << findDET(matS, matD) << endl;

system("pause");

return 0;

}

Результат:



**ІІ. Метод Якобі та обумовлене число**

float normvector(float a[4]) //**норма вектора**

{

float max = abs(a[0]);

if (abs(a[1]) > max) max = abs(a[1]);

if (abs(a[2]) > max) max = abs(a[2]);

if (abs(a[3]) > max) max = abs(a[3]);

return max;

}

float matnorm(float a[4][4]) **//норма матриці**

{

float temp[4] = { 0,0,0,0 };

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

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

{

temp[i] += abs(a[i][j]);

}

return normvector(temp);

}

void Inv(float A[4][4], int N) //**пошук оберненої матриці**

{

float temp;

float E[4][4];

N = 4;

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

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

{

E[i][j] = 0.0;

if (i == j)

E[i][j] = 1.0;

}

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

{

temp = A[k][k];

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

{

A[k][j] /= temp;

E[k][j] /= temp;

}

for (int i = k + 1; i < N; i++)

{

temp = A[i][k];

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

{

A[i][j] -= A[k][j] \* temp;

E[i][j] -= E[k][j] \* temp;

}

}

}

for (int k = N - 1; k > 0; k--)

{

for (int i = k - 1; i >= 0; i--)

{

temp = A[i][k];

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

{

A[i][j] -= A[k][j] \* temp;

E[i][j] -= E[k][j] \* temp;

}

}

}

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

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

A[i][j] = E[i][j];

}

bool IsMarrOK(float m[4][4]) {/**/перевірка коректності матриці для методу Якобі**

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

{

if (m[i][i] < m[i][0] + m[i][1] + m[i][2] + m[i][3] - m[i][i])

return 0;

}

return 1;

}

int main()

{

setlocale(LC\_ALL, "ukr");

float A[4][4] = { {9,1,2,1} ,{1,4,0,1} ,{2,0,4,1} ,{1,1,1,4}, };

float b[4] = { 15,6,11,8 };

float x[4];

float temp[4];

float eps = 0.05**;//значення для точності**

float shouldcontinue[4];

for (int i = 0; i < 4; i++) **//перше наближення**

{

x[i] = 0;

}

int count = 1;

do {

temp[0] = (-A[0][1] \* x[1] - A[0][2] \* x[2] - A[0][3] \* x[3] + b[0]) / A[0][0];//**наступна ітерація**

temp[1] = (-A[1][0] \* x[0] - A[1][2] \* x[2] - A[1][3] \* x[3] + b[1]) / A[1][1];

temp[2] = (-A[2][1] \* x[1] - A[2][0] \* x[0] - A[2][3] \* x[3] + b[2]) / A[2][2];

temp[3] = (-A[3][1] \* x[1] - A[3][2] \* x[2] - A[3][0] \* x[0] + b[3]) / A[3][3];

for(int i=0; i<4; i++) shouldcontinue[i] = abs(x[i] - temp[i]);

x[0] = temp[0];

x[1] = temp[1];

x[2] = temp[2];

x[3] = temp[3];

cout << "Норма векторiв: " << setprecision(10) << normvector(shouldcontinue) << endl;

count++;

cout << "Iтерацiя " << count << ": x1 = " << setprecision(5) << x[0] << " x2 = " << x[1] << " x3 = " << x[2] << " x4 = " << x[3] << endl;

} while (normvector(shouldcontinue) > eps); **//вимоги для завершення розв’язання**

cout << "Норма векторiв: " << normvector(shouldcontinue) << endl;

cout << "x1 = " << setprecision(5) << x[0] << " x2 = " << x[1] << " x3 = " << x[2] << " x4 = " << x[3] << endl;

cout << "==========================================" << endl;

**//пошук і виведення обумовленого числа матриці**

float norm\_a = matnorm(A);

Inv(A, 4);

cout << "Обернена матриця:" << endl;

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

{

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

cout << A[i][j] << " ";

cout << std::endl;

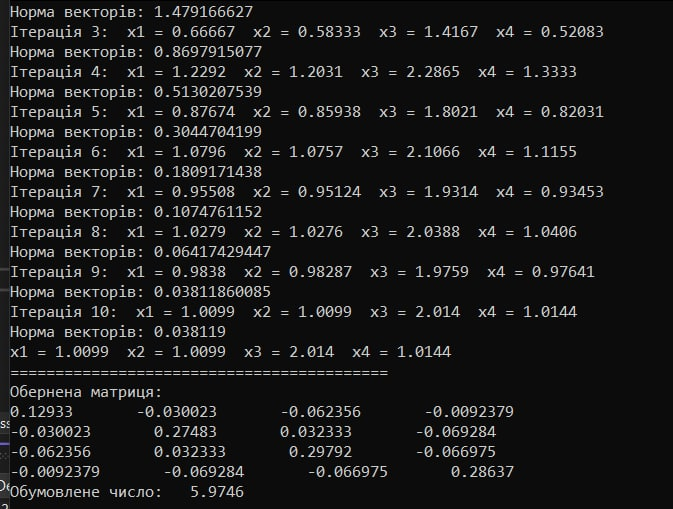
}

double norm\_a1 = matnorm(A);

cout << "Обумовлене число: " << norm\_a \* norm\_a1 << endl;

return 0;

}

Результати: 

**Висновки**

Під час написання даної лабораторної роботи я змогла детальніше розглянути методи на практиці, зрозуміти принцип їхньої роботи та відмінності. Для даного рівняння підійшли обидва методи. Методи квадратного кореня і Якобі різні і слід використовувати в різних випадках.