 Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

До лабораторної роботи №4

**На теми:**  “Розв’язування систем лінійних алгебраїчних рівнянь методом Гауса та методом LU-розкладу”

**З дисципліни:** “Чисельні методи”

**Лектор:**

доц. каф. ПЗ

Мельник Н.Б.

**Виконав:**

ст. гр. ПЗ-18

Юшкевич А.І.

**Прийняв:**

проф. каф. ПЗ

Гавриш В.І.

« … » … 2023 р.

∑ = \_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

**Теми**: розв’язування систем лінійних алгебраїчних рівнянь методом Гауса та методом LU-розкладу.

**Мета**: ознайомлення на практиці з методом Гауса та методом LU- розкладу розв’язування систем лінійних алгебраїчних рівнянь. Код програмної реалізації подано у додатку.

**Завдання**

Скласти програму розв’язування системи лінійних алгебраїчних рівнянь методами Гауса з вибором головного елемента та *LU* -розкладу.

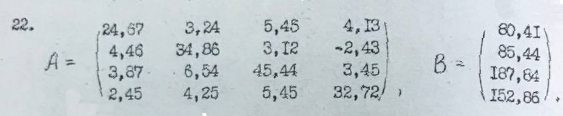


Рис. 1. Система лінійних алгебраїчних рівнянь

**Метод Гаусса**

Найвідомішим точним методом розв’язування систем лінійних алгебраїчних рівнянь є метод Гауса, суть якого полягає в тому, що систему рівнянь, яку необхідно розв’язати, зводять до еквівалентної системи з верхньою (або нижньою) трикутною матрицею. Невідомі знаходять послідовними підстановками, починаючи з останнього рівняння перетвореної системи. Точність результату та витрачений на його отримання час у більшості випадків залежить від алгоритму формування трикутної матриці системи. У загальному випадку алгоритм методу Гауса складається з двох етапів – прямого та зворотного ходу. Під час **прямого ходу** СЛАР перетворюють до еквівалентної системи з верхньою трикутною матрицею. **Зворотній хід** дає змогу визначити елементи вектору невідомих, починаючи з останнього рівняння системи, підставляючи послідовно відповідні елементи цього вектору, отримані на попередньому кроці.

**Метод Гауса з вибором головного елемента**

Серед елементів матриці A виберемо найбільший за модулем елемент, який називають головним елементом. Далі перетворюємо матрицю A так: від кожного i -го неголовного рядка віднімаємо почленно головний рядок, помножений на mi . У результаті отримуємо матрицю, у якій всі елементи k -го стовпця, за винятком pk a , дорівнюють нулеві. Відкидаючи цей стовпець і головний рядок, отримуємо нову матрицю A1 з меншою на одиницю кількістю рядків та стовпців. Такі самі дії повторюємо над матрицею A1 і отримуємо матрицю A2 і т.д. Ці перетворення продовжуємо доти, поки не отримаємо матрицю, що містить один рядок з двох елементів, який вважаємо головним. Об’єднаємо всі головні рядки, починаючи від останнього. Після деяких перестановок вони утворять трикутну матрицю, еквівалентну до початкової матриці A.

**Метод LU-розкладу**

Розв’язуючи систему лінійних алгебраїчних рівнянь даним методом, матрицю A коефіцієнтів системи розкладають на добуток двох матриць – нижньої трикутної матриці L , елементи головної діагоналі якої не дорівнюють нулеві та верхньої трикутної U , на головній діагоналі якої містяться одиниці. Розв’язування матричного рівняння виконуємо за два етапи: спочатку розв’язуємо матричне рівняння, а потім. Такий підхід суттєво спрощує отримання розв’язку порівняно з методом Гауса для випадку, коли маємо кілька систем рівнянь з однаковою матрицею коефіцієнтів А, оскільки матриці L та U визначають один раз. Розв’язування систем LY  B та UX  Y називають прямим та оберненим ходом відповідно. Спочатку розглянемо прямий хід методу. Завдяки трикутній формі матриці L вектор Y легко визначають. Для цього матричне рівняння перепишемо у розгорнутому вигляді. При виконанні оберненого ходу компоненти вектору X визначають зі системи рівнянь.

**Основні етапи обчислювального алгоритму для розв’язування системи лінійних алгебраїчних рівнянь методом Гауса, реалізованого у програмному продукті мовою C++**

1. внесення даних в конструкторі SystemSolver() (рис. 2);
2. виклик метода Gauss() (рис. 3), що реалізує знаходження коренів системи лінійних рівнянь методом Гауса;
3. знаходження визначника заданої матриці за допомогою методу FindDeterminant() (рис. 4)
4. виклик метода GaussItself() (рис. 5), який за допомогою елементарних перетворень утворює верхню трикутну матрицю;
5. вивід результату виконання в консоль (рис. 6).
6. перевірка точності отриманого розв’язку системи лінійних рівнянь (рис. 9).

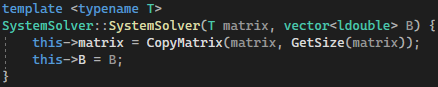


Рис. 2. Конструктор SystemSolver()

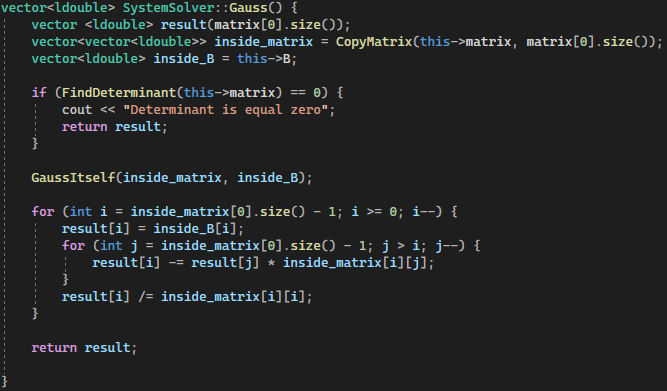


Рис. 3. Метод Gauss()

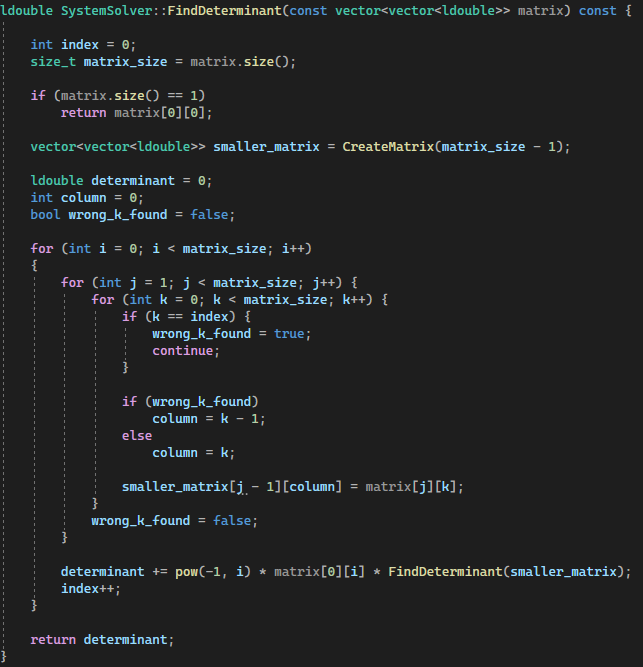


Рис. 4. Метод FindDeterminant()

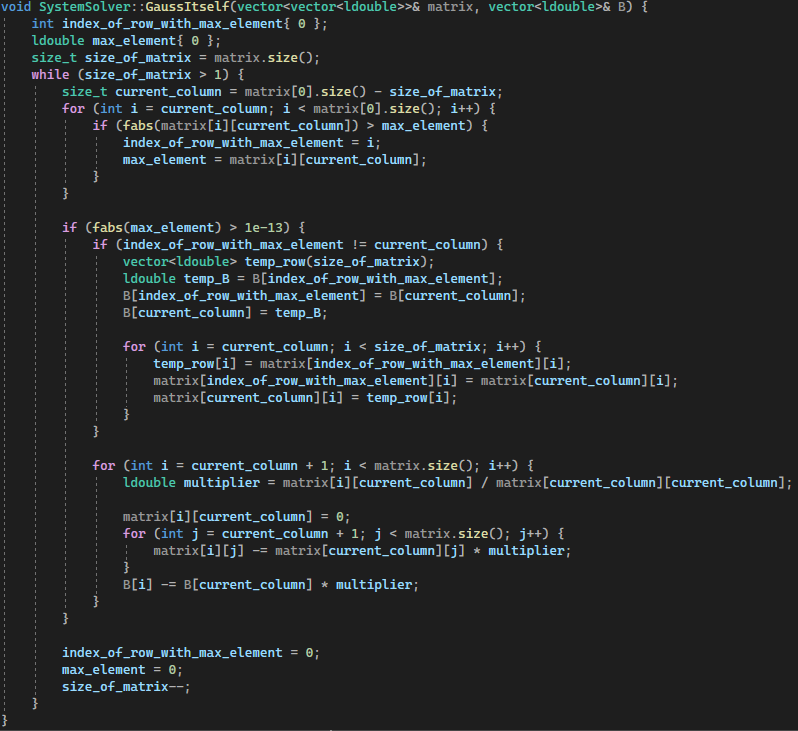


Рис. 5. Метод GaussItself()

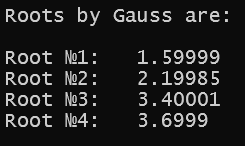


Рис. 6. Результат виконання програмної реалізації методу Гауса

**Основні етапи обчислювального алгоритму для розв’язування системи лінійних алгебраїчних рівнянь методом LU-розкладу, реалізованого у програмному продукті мовою C++**

1. внесення даних в конструкторі SystemSolver() (рис. 2);
2. виклик метода LU() (рис. 7), що реалізує знаходження коренів системи лінійних рівнянь методом LU-розкладу;
3. вивід результату виконання в консоль (рис. 8).
4. перевірка точності отриманого розв’язку системи лінійних рівнянь (рис. 9).

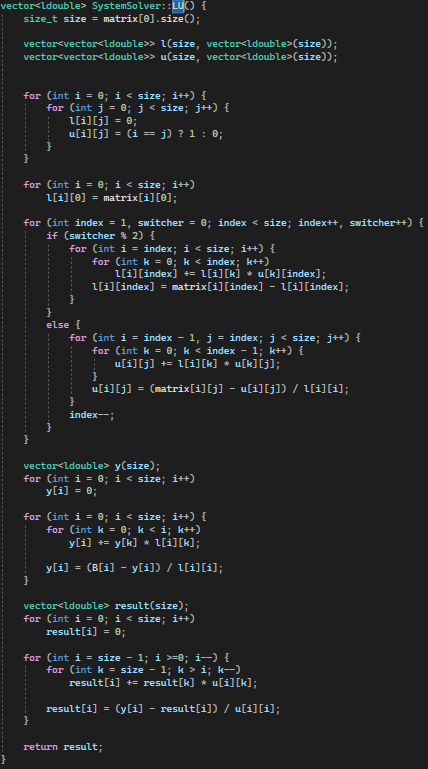


Рис. 7. Метод LU ()

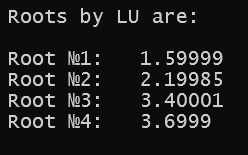


Рис. 8. Результат виконання програмної реалізації методу LU-розкладу

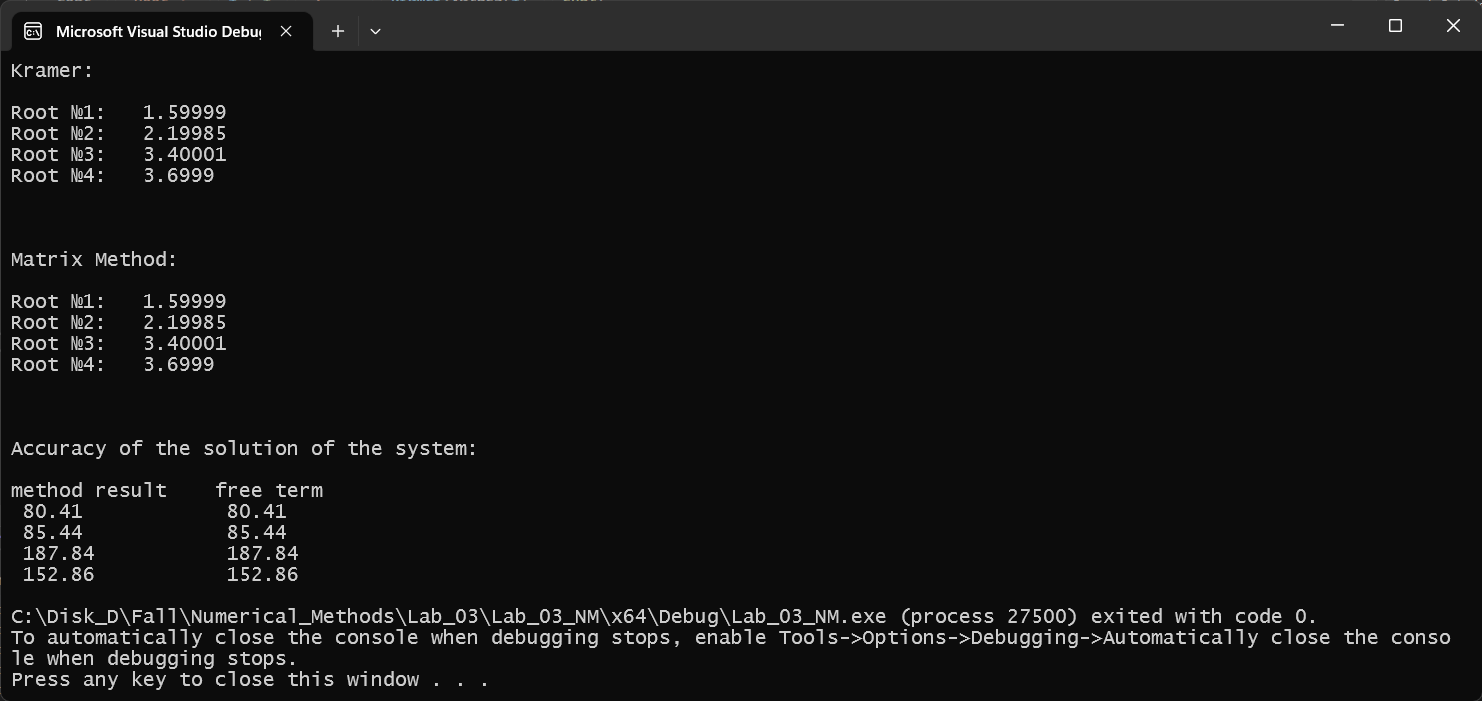


Рис. 9. Перевірка точності отриманого розв’язку системи лінійних рівнянь

**Висновки**

У результаті виконання лабораторної роботи розробив програму розв’язування системи лінійних алгебраїчних рівнянь методами Гауса з вибором головного елемента та *LU* -розкладу для заданої системи лінійних алгебраїчних рівнянь.

**Додаток**

**Header.h:**

#pragma once

#include <iostream>

#include <cmath>

#include <vector>

using namespace std;

typedef long double ldouble;

class SystemSolver {

private:

vector<ldouble> B;

vector<vector<ldouble>> matrix;

template <typename T>

vector<vector<ldouble>> CopyMatrix(const T matrix, const size\_t size);

template <typename T>

size\_t GetSize(const T matrix) const;

vector<vector<ldouble>> CreateMatrix(const size\_t size) const;

ldouble FindDeterminant(const vector<vector<ldouble>> matrix) const;

void GaussItself(vector<vector<ldouble>>& matrix, vector<ldouble>& B);

public:

template <typename T>

SystemSolver(T matrix, vector<ldouble> B);

vector<ldouble> Gauss();

vector<ldouble> LU();

};

template <typename T>

size\_t SystemSolver::GetSize(const T matrix) const {

size\_t result{ 0 };

if(matrix != nullptr)

result = sizeof(matrix[0]) / sizeof(matrix[0][0]);

return result;

}

template <typename T>

vector<vector<ldouble>> SystemSolver::CopyMatrix(const T matrix, const size\_t size) {

vector<vector<ldouble>> new\_vector(size, vector<ldouble>(size));

do {

new\_vector = CreateMatrix(size);

if (new\_vector.empty())

break;

for (int i = 0; i < size; i++) {

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

new\_vector[i][j] = matrix[i][j];

}

}

} while (false);

return new\_vector;

}

template <typename T>

SystemSolver::SystemSolver(T matrix, vector<ldouble> B) {

this->matrix = CopyMatrix(matrix, GetSize(matrix));

this->B = B;

}

**Functions.cpp:**

#include "Header.h"

vector<vector<ldouble>> SystemSolver::CreateMatrix(const size\_t size) const{

vector<vector<ldouble>> new\_matrix(size, vector<ldouble>(size));

return new\_matrix;

}

ldouble SystemSolver::FindDeterminant(const vector<vector<ldouble>> matrix) const {

int index = 0;

size\_t matrix\_size = matrix.size();

if (matrix.size() == 1)

return matrix[0][0];

vector<vector<ldouble>> smaller\_matrix = CreateMatrix(matrix\_size - 1);

ldouble determinant = 0;

int column = 0;

bool wrong\_k\_found = false;

for (int i = 0; i < matrix\_size; i++)

{

for (int j = 1; j < matrix\_size; j++) {

for (int k = 0; k < matrix\_size; k++) {

if (k == index) {

wrong\_k\_found = true;

continue;

}

if (wrong\_k\_found)

column = k - 1;

else

column = k;

smaller\_matrix[j - 1][column] = matrix[j][k];

}

wrong\_k\_found = false;

}

determinant += pow(-1, i) \* matrix[0][i] \* FindDeterminant(smaller\_matrix);

index++;

}

return determinant;

}

vector<ldouble> SystemSolver::Gauss() {

vector <ldouble> result(matrix[0].size());

vector<vector<ldouble>> inside\_matrix = CopyMatrix(this->matrix, matrix[0].size());

vector<ldouble> inside\_B = this->B;

if (FindDeterminant(this->matrix) == 0) {

cout << "Determinant is equal zero";

return result;

}

GaussItself(inside\_matrix, inside\_B);

for (int i = inside\_matrix[0].size() - 1; i >= 0; i--) {

result[i] = inside\_B[i];

for (int j = inside\_matrix[0].size() - 1; j > i; j--) {

result[i] -= result[j] \* inside\_matrix[i][j];

}

result[i] /= inside\_matrix[i][i];

}

return result;

}

void SystemSolver::GaussItself(vector<vector<ldouble>>& matrix, vector<ldouble>& B) {

int index\_of\_row\_with\_max\_element{ 0 };

ldouble max\_element{ 0 };

size\_t size\_of\_matrix = matrix.size();

while (size\_of\_matrix > 1) {

size\_t current\_column = matrix[0].size() - size\_of\_matrix;

for (int i = current\_column; i < matrix[0].size(); i++) {

if (fabs(matrix[i][current\_column]) > max\_element) {

index\_of\_row\_with\_max\_element = i;

max\_element = matrix[i][current\_column];

}

}

if (fabs(max\_element) > 1e-13) {

if (index\_of\_row\_with\_max\_element != current\_column) {

vector<ldouble> temp\_row(size\_of\_matrix);

ldouble temp\_B = B[index\_of\_row\_with\_max\_element];

B[index\_of\_row\_with\_max\_element] = B[current\_column];

B[current\_column] = temp\_B;

for (int i = current\_column; i < size\_of\_matrix; i++) {

temp\_row[i] = matrix[index\_of\_row\_with\_max\_element][i];

matrix[index\_of\_row\_with\_max\_element][i] = matrix[current\_column][i];

matrix[current\_column][i] = temp\_row[i];

}

}

for (int i = current\_column + 1; i < matrix.size(); i++) {

ldouble multiplier = matrix[i][current\_column] / matrix[current\_column][current\_column];

matrix[i][current\_column] = 0;

for (int j = current\_column + 1; j < matrix.size(); j++) {

matrix[i][j] -= matrix[current\_column][j] \* multiplier;

}

B[i] -= B[current\_column] \* multiplier;

}

}

index\_of\_row\_with\_max\_element = 0;

max\_element = 0;

size\_of\_matrix--;

}

}

vector<ldouble> SystemSolver::LU() {

size\_t size = matrix[0].size();

vector<vector<ldouble>> l(size, vector<ldouble>(size));

vector<vector<ldouble>> u(size, vector<ldouble>(size));

for (int i = 0; i < size; i++) {

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

l[i][j] = 0;

u[i][j] = (i == j) ? 1 : 0;

}

}

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

l[i][0] = matrix[i][0];

for (int index = 1, switcher = 0; index < size; index++, switcher++) {

if (switcher % 2) {

for (int i = index; i < size; i++) {

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

l[i][index] += l[i][k] \* u[k][index];

l[i][index] = matrix[i][index] - l[i][index];

}

}

else {

for (int i = index - 1, j = index; j < size; j++) {

for (int k = 0; k < index - 1; k++) {

u[i][j] += l[i][k] \* u[k][j];

}

u[i][j] = (matrix[i][j] - u[i][j]) / l[i][i];

}

index--;

}

}

vector<ldouble> y(size);

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

y[i] = 0;

for (int i = 0; i < size; i++) {

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

y[i] += y[k] \* l[i][k];

y[i] = (B[i] - y[i]) / l[i][i];

}

vector<ldouble> result(size);

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

result[i] = 0;

for (int i = size - 1; i >=0; i--) {

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

result[i] += result[k] \* u[i][k];

result[i] = (y[i] - result[i]) / u[i][i];

}

return result;

}

**Lab\_04\_NM.cpp:**

#include <iostream>

#include "Header.h"

using namespace std;

int main()

{

const size\_t size{ 4 };

ldouble matrix[size][size] = { { 24.67, 3.24, 5.45, 4.13},

{ 4.46, 34.86, 3.12, -2.43},

{ 3.87, 6.54, 45.44, 3.45},

{ 2.45, 4.25, 5.45, 32.72} };

vector<ldouble> B{ 80.41, 85.44, 187.84, 152.86 };

SystemSolver ss(matrix, B);

vector<ldouble> result = ss.Gauss();

cout << "Roots by Gauss are: " << endl << endl;

for (int i = 0; i < result.size(); i++) {

cout << "Root №" << i + 1 << ": " << result[i] << endl;

}

cout << endl << endl << endl;

result = ss.LU();

cout << "Roots by LU are: " << endl << endl;

for (int i = 0; i < result.size(); i++) {

cout << "Root №" << i + 1 << ": " << result[i] << endl;

}

cout << endl << endl << endl;

ldouble sum{ 0 };

for (int i = 0; i < size; i++) {

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

sum += matrix[i][j] \* result[j];

}

cout << sum << "\t" << B[i] << endl;

sum = 0;

}

return 0;

}