Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

Про виконання лабораторної роботи № 6

**На тему:**

«Розв’язування перевизначених систем лінійних алгебраїчних рівнянь»

**Лектор:**

доц. каф. ПЗ

Мельник Н. Б.

**Виконала:**

ст. гр. ПЗ-18

Юшкевич А.І.

**Прийняв:**

проф. каф. ПЗ

Гавриш В.І.

« … » … 2023 р.

∑ = \_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

**Тема**: розв’язування перевизначених систем лінійних алгебраїчних рівнянь.

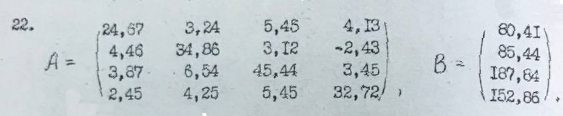
**Мета**: ознайомлення на практиці з методами розв’язування перевизначених систем лінійних алгебраїчних рівнянь.

**Завдання**

Варіант 8

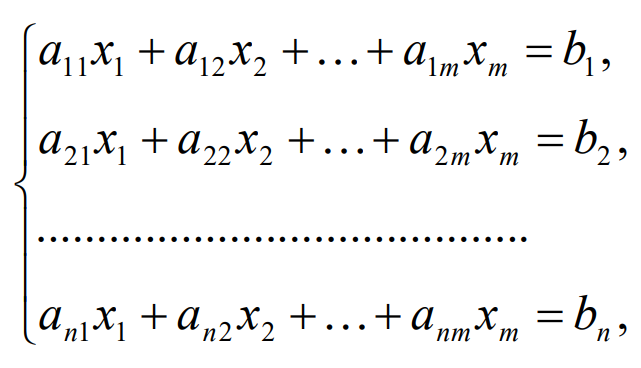
Розв’язати перевизначену систему лінійних алгебраїчних рівнянь методом найменших квадратів. Отриману відповідну нормальну систему розв’язати методом квадратного кореня.

Код програмної реалізації подано у додатку.



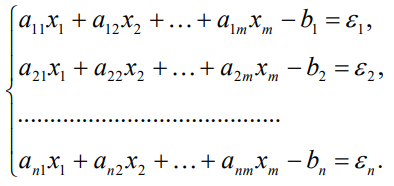
**Метод найменших квадратів для розв'язування перевизначених СЛАР**

Розглянемо систему лінійних алгебраїчних рівнянь, у якій кількість рівнянь є більшою за кількість невідомих

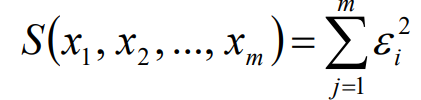
**** (рис. 1)

де n > m.

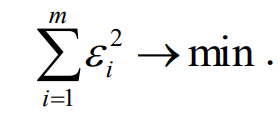
Знайдемо розв’язок системи х1, х2, ..., хm наближено, але щоб він задовольняв усі рівняння системи (рис. 1), а саме

 (рис. 2)

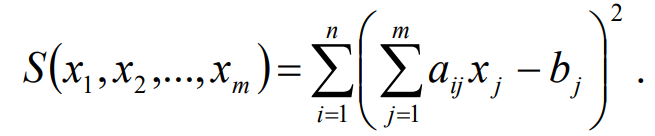
Розв’язок системи (рис. 2) будемо знаходити з використанням умови мінімізації суми квадратів відхилень, тобто з умов мінімізації функції

 (рис. 3)

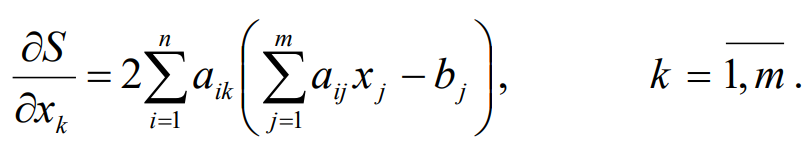
і вимагатимемо, щоб виконувалась умова

**** (рис. 4)

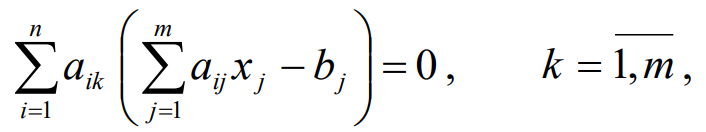
Проведемо деякі перетворення над системою (рис. 2), використовуючи умову (рис. 4). Розглянемо функцію

**** (рис. 5)

Необхідною умовою мінімуму функції від багатьох змінних є рівність нулеві її частинних похідних. Використаємо цей факт і продиференціюємо функцію (рис. 5) за змінними i x (i =1,m). У результаті отримаємо

**** (рис. 6)

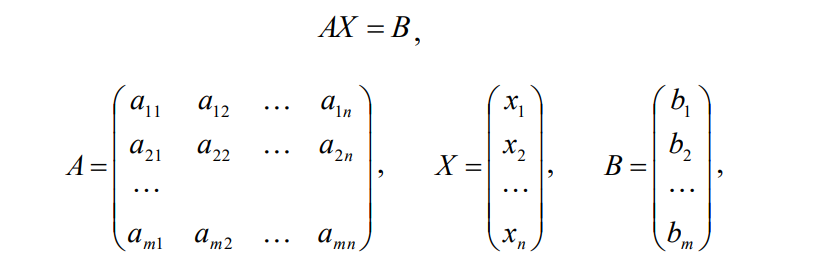
Прирівнявши вирази (рис. 6) до нуля, отримаємо нормальну систему рівнянь

**** (рис. 7)

в якій кількість рівнянь системи дорівнює кількості невідомих. Нормальні системи лінійних алгебраїчних рівнянь характеризуються тим, що матриці їх коефіцієнтів завжди є симетричними, а діагональні елементи - додатніми.

Систему (рис. 7) розв'язують довільними прямими або ітераційними методами. Якщо матриця коефіцієнтів системи рівнянь (рис. 7) є додатньо визначеною (визначник матриці є більшим за нуль), то рекомендують для її розв’язування використовувати метод квадратного кореня.

Запишемо систему лінійних алгебраїчних рівнянь (рис. 1) у матричному вигляді

 (рис. 8)

де A - матриця коефіцієнтів системи розмірності m × n, X – матриця-стовпець невідомих розмірності m × 1, B - матриця-стовпець вільних членів системи розмірності m × 1.

Матричне рівняння (рис. 8) помножимо на транспоновану матрицю AТ до матриці A . У результаті отримаємо матричне рівняння

NX = C (рис. 9)

де N – матриця коефіцієнтів нормальної системи

N = AТ А, (рис. 10)

C -стовпець вільних членів

С = AТ В. (рис. 11)

Розв’язавши нормальну систему лінійних алгебраїчних рівнянь, отримаємо її точний розв’язок (якщо використано прямі методи) або наближений розв’язок (якщо використано ітераційні методи). Отриманий розв’язок буде наближеним для СЛАР (рис. 1).

**Метод квадратного кореня**

Якщо матриця невироджена та �симетрична, то ми можемо розкласти її на добуток матриць �=����A = LDLТ, де L �llUISCHD — одинична нижня [трикутна матриця](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D0%BA%D1%83%D1%82%D0%BD%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F); D  �dD— [діагональна матриця](https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F).

Отримаємо систему:����⋅�=�L

LDLТ \* x = b

Розв'язок �отримаємо послідовно розв'язавши дві трикутні СЛАР:

LD \* y = b та

LТ \* x = y.

Порівняно з загальнішими методами ([метод Гауса](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%93%D0%B0%D1%83%D1%81%D0%B0) чи [LU-розклад матриці](https://uk.wikipedia.org/wiki/LU-%D1%80%D0%BE%D0%B7%D0%BA%D0%BB%D0%B0%D0%B4_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%96)) він стійкіший і потребує вдвічі менше арифметичних операцій.

**Результат виконання завдання**

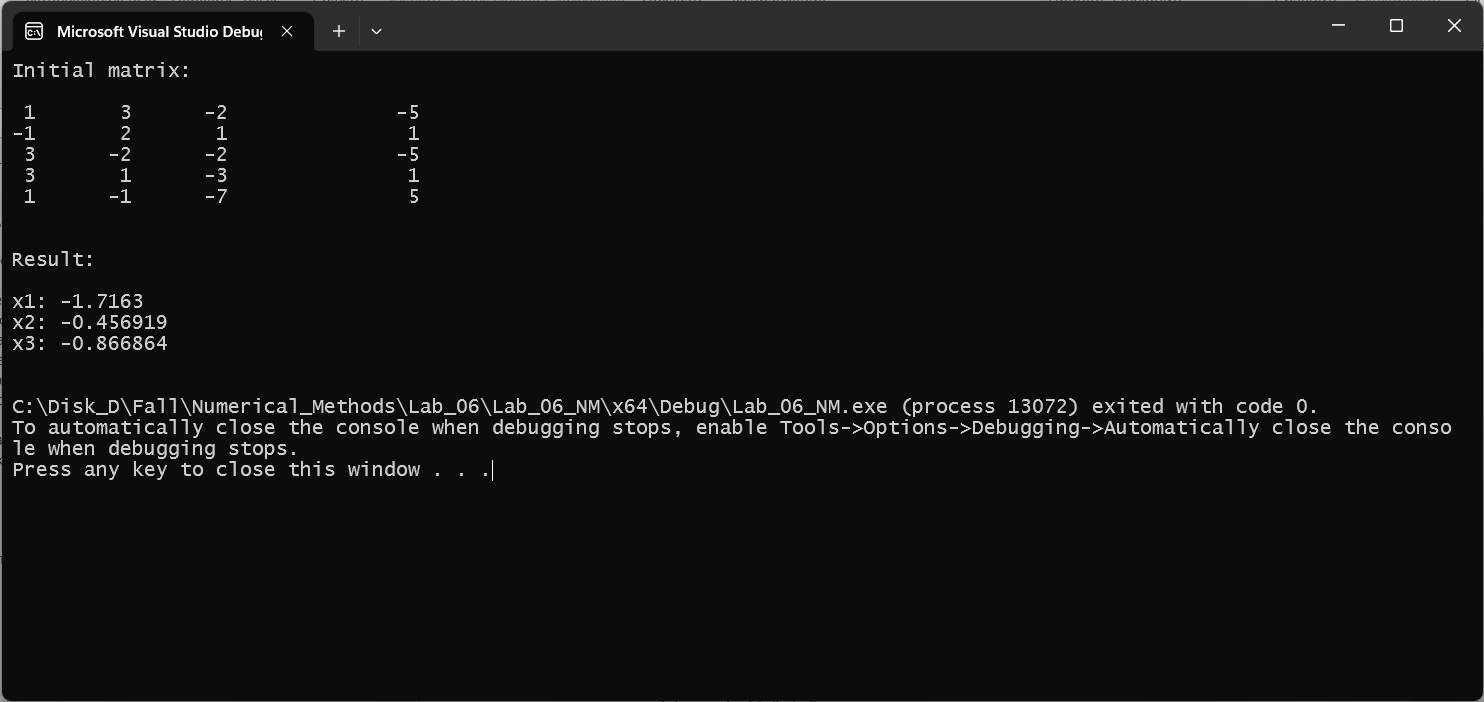
****

Рис. 12. Результат обчислення CЛАР методом найменших квадратів

**Висновки**

У результаті виконання лабораторної роботи створено обчислювальний

алгоритм, за допомогою мови С++, для розв’язування перевизначеної системи

лінійних алгебраїчних рівнянь методом найменших квадратів для заданої

системи лінійних алгебраїчних рівнянь.

**Додаток**

LeastSquares.h:

#pragma once

#include <iostream>

#include <vector>

using namespace std;

class LeastSquares

{

public:

LeastSquares() = delete;

template <class T>

LeastSquares(const T matrix, const size\_t num\_of\_variables, const size\_t num\_of\_equations, const vector<double> free\_terms);

vector<double> Find();

private:

vector<vector<double>> m\_matrix;

vector<double> m\_free\_terms;

size\_t m\_num\_of\_variables;

size\_t m\_num\_of\_equation;

template <typename T>

vector<vector<double>> CopyMatrix(const T matrix, const size\_t num\_of\_variables, const size\_t num\_of\_equasions) const;

vector<vector<double>> TranspondMatrix(const vector<vector<double>> matrix) const;

vector<vector<double>> MultiplyMatrixes(const vector<vector<double>> first\_matrix, const vector<vector<double>> second\_matrix) const;

vector<double> MultiplyMatrixAndColumn(const vector<vector<double>> matrix, const vector<double> column) const;

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

vector<vector<double>> SplitMatrixIntoToTransponded(const vector<vector<double>> matrix) const;

vector<double> GetY(const vector<vector<double>> matrix, const vector<double> new\_free\_terms) const;

vector<double> GetX(const vector<vector<double>> matrix, const vector<double> y) const;

};

template <typename T>

vector<vector<double>> LeastSquares::CopyMatrix(const T matrix, const size\_t num\_of\_variables, const size\_t num\_of\_equasions) const {

vector<vector<double>> new\_vector(num\_of\_equasions, vector<double>(num\_of\_variables));

do {

if (new\_vector.empty())

break;

for (int i = 0; i < num\_of\_equasions; i++) {

for (int j = 0; j < num\_of\_variables; j++) {

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

}

}

} while (false);

return new\_vector;

}

template <class T>

LeastSquares::LeastSquares(const T matrix, const size\_t num\_of\_variables, const size\_t num\_of\_equations, const vector<double> free\_terms) {

this->m\_num\_of\_equation = num\_of\_equations;

this->m\_num\_of\_variables = num\_of\_variables;

this->m\_matrix = CopyMatrix(matrix, num\_of\_variables, num\_of\_equations);

this->m\_free\_terms = free\_terms;

}

Lab\_06\_NM.cpp:

#include "LeastSquares.h"

int main()

{

const double result\_accuracy = 0.0001;

const size\_t num\_of\_equations{ 5 };

const size\_t num\_of\_variables{ 3 };

double matrix[num\_of\_equations][num\_of\_variables] ={{ 1, 3, -2},

{-1, 2, 1},

{ 3, -2, -2},

{ 3, 1, -3},

{ 1, -1, -7} };

vector<double> free\_terms{ -5, 1, -5, 1, 5 };

LeastSquares ls(matrix, num\_of\_variables, num\_of\_equations, free\_terms);

vector<double> result = ls.Find();

cout << "Initial matrix:\n\n";

for (int i = 0; i < num\_of\_equations; i++) {

for (int j = 0; j < num\_of\_variables; j++) {

if (matrix[i][j] >= 0)

cout << " ";

cout << matrix[i][j] << "\t";

}

cout << "\t";

if (free\_terms[i] >= 0)

cout << " ";

cout << free\_terms[i] << endl;

}

cout << "\n\n";

cout << "Result:\n\n";

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

cout << "x" << i + 1 << ": " << result[i] << "\n";

}

cout << endl;

}

LeastSquares.cpp:

#include "LeastSquares.h"

vector<vector<double>> LeastSquares::TranspondMatrix(const vector<vector<double>> matrix) const {

vector<vector<double>> transponded(matrix[0].size(), vector<double>(matrix.size()));

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

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

transponded[i][j] = matrix[j][i];

}

}

return transponded;

}

vector<double> LeastSquares::MultiplyMatrixAndColumn(const vector<vector<double>> matrix, const vector<double> column) const {

vector<double> local\_column(column.size());

double sum = 0;

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

for (int j = 0; j < matrix[0].size(); j++) {

local\_column[i] += matrix[i][j] \* column[j];

}

}

return local\_column;

}

double LeastSquares::FindDeterminant(const vector<vector<double>> matrix) const {

int index = 0;

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

if (matrix.size() == 1)

return matrix[0][0];

vector<vector<double>> smaller\_matrix(matrix\_size - 1, vector<double>(matrix\_size - 1));

double 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<vector<double>> LeastSquares::MultiplyMatrixes(const vector<vector<double>> first\_matrix, const vector<vector<double>> second\_matrix) const {

vector<vector<double>> multiplication(first\_matrix.size(), vector<double>(second\_matrix[0].size()));

for (int i = 0; i < first\_matrix.size(); i++) {

for (int j = 0; j < second\_matrix[0].size(); j++) {

multiplication[i][j] = 0;

for (int k = 0; k < first\_matrix[0].size(); k++) {

multiplication[i][j] += first\_matrix[i][k] \* second\_matrix[k][j];

}

}

}

return multiplication;

}

vector<vector<double>> LeastSquares::SplitMatrixIntoToTransponded(const vector<vector<double>> matrix) const {

vector<vector<double>> square\_matrix(matrix.size(), vector<double>(matrix.size()));

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

square\_matrix[i][i] = matrix[i][i];

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

square\_matrix[i][i] -= pow(square\_matrix[j][i], 2);

}

square\_matrix[i][i] = sqrt(square\_matrix[i][i]);

for (int j = i + 1; j < matrix.size(); j++) {

square\_matrix[i][j] = matrix[i][j];

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

square\_matrix[i][j] -= square\_matrix[k][i] \* square\_matrix[k][j];

}

square\_matrix[i][j] /= square\_matrix[i][i];

}

}

return square\_matrix;

}

vector<double> LeastSquares::GetY(const vector<vector<double>> matrix, const vector<double> new\_free\_terms) const {

vector<double> y(matrix.size());

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

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

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

y[i] = (new\_free\_terms[i] - y[i]) / matrix[i][i];

}

return y;

}

vector<double> LeastSquares::GetX(const vector<vector<double>> matrix, const vector<double> y) const {

vector<double> x(matrix.size());

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

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

x[i] += x[k] \* matrix[i][k];

x[i] = (y[i] - x[i]) / matrix[i][i];

}

return x;

}

vector<double> LeastSquares::Find() {

vector<vector<double>> transponded\_matrix = TranspondMatrix(m\_matrix);

vector<vector<double>> multiplied = MultiplyMatrixes(transponded\_matrix, m\_matrix);

vector<double> new\_free\_terms = MultiplyMatrixAndColumn(transponded\_matrix, m\_free\_terms);

if (!FindDeterminant(multiplied)) {

cout << "Determinant is equal zero, try another method";

return vector<double>(multiplied.size());

}

vector<vector<double>> splited\_upper = SplitMatrixIntoToTransponded(multiplied);

vector<vector<double>> splited\_lower = TranspondMatrix(splited\_upper);

vector<double> y = GetY(splited\_lower, new\_free\_terms);

vector<double> x = GetX(splited\_upper, y);

return x;

}