 Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

До лабораторної роботи №3

**На теми:**  “Розв’язування систем лінійних алгебраїчних рівнянь методом Крамера та методом оберненої матриці”

**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

**Тема**: розв’язування систем лінійних алгебраїчних рівнянь методом Крамера та методом оберненої матриці.

**Мета**: ознайомлення на практиці з методом Крамера та методом оберненої матриці розв’язування систем лінійних алгебраїчних рівнянь. Код програмної реалізації подано у додатку.

**Завдання**

Скласти програму розв’язування системи лінійних алгебраїчних рівнянь методом оберненої матриці та методом Крамера

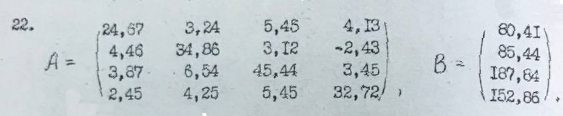


Рис. 1. Система лінійних алгебраїчних рівнянь

**Метод оберненої матриці (матричний метод)**

У лінійній алгебрі часто використовують матричний метод розв’язування систем лінійних алгебраїчних рівнянь. Цей метод ґрунтується на обчисленні оберненої матриці A-1, яка існує лише при умові, коли визначник матриці A відмінний від нуля detA ≠ 0. Якщо обидві частини матричного рівняння зліва домножити на матрицю A-1, то отримаємо співвідношення . Враховуючи, що добуток оберненої матриці на саму матрицю дає одиничну матрицю, а результатом добутку одиничної матриці E на матрицю-стовпець X є матриця-стовпець X, тобто EX  X , одержимо матричний розв'язок системи лінійних алгебраїчних рівнянь у вигляді .

**Метод Крамера**

Розглянемо СЛАР, яка містить n рівнянь та n невідомих, причому визначник її не дорівнює нулеві. Для знаходження невідомих i x застосовують формули Крамера: , , де detA - визначник матриці A, detAi - визначник матриці Ai, яку отримують з матриці A шляхом заміни її i -го стовпця стовпцем вільних членів. Щоб розв’язати СЛАР з n невідомими потрібно обчислити (n1) визначників n -го порядку, що призводить до виконання nn! операцій. Через громіздкість обчислень визначників метод Крамера не застосовують на практиці для великої розмірності матриці коефіцієнтів СЛАР.

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

1. внесення даних в конструкторі FunctionHolder() (рис. 2);
2. виклик метода MatrixMethod() (рис. 3), що реалізує знаходження коренів системи лінійних рівнянь матричним методом;
3. знаходження оберненої матриці реалізоване за допомогою методу FindReversedMatrix() (рис. 4), який в свою чергу використовує метод FindDeterminant() (рис. 5) для знаходження визначника заданої матриці та перевантаження цього методу – FindDeterminant() (рис. 6) для знаходження алгебраїчних доповнень;
4. знаходження добутку оберненої матриці та вільних членів у методі FindMultiplicationOfMatrixAndColumn() (рис. 7);
5. вивід результату виконання в консоль (рис. 8);
6. перевірка точності отриманого розв’язку системи лінійних рівнянь (рис. 11).

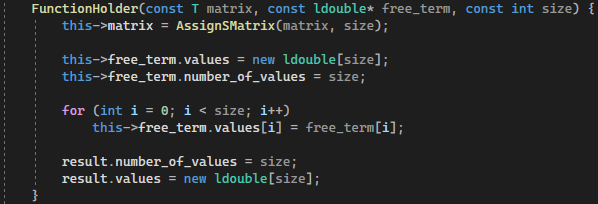


Рис. 2. Конструктор FunctionHolder()

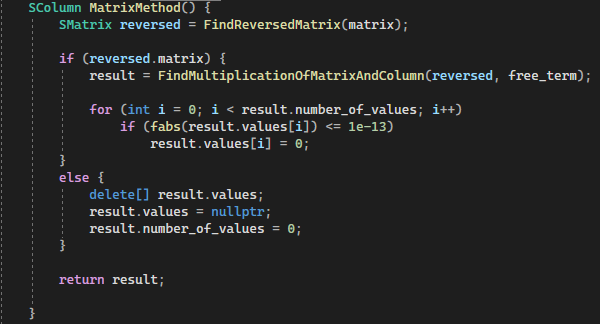


Рис. 3. Метод MatrixMethod()

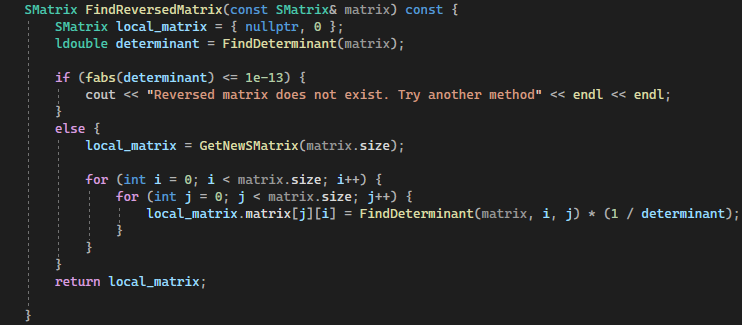


Рис. 4. Метод FindReversedMatrix()

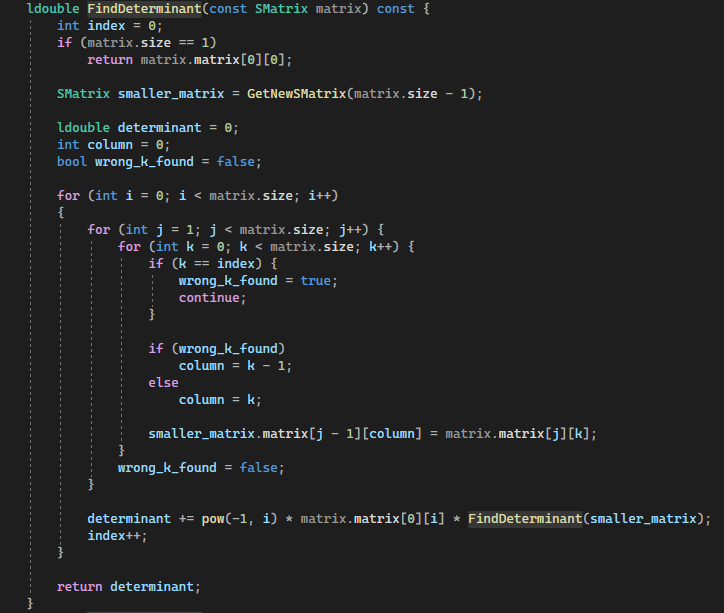


Рис. 5. Метод FindDeterminant()

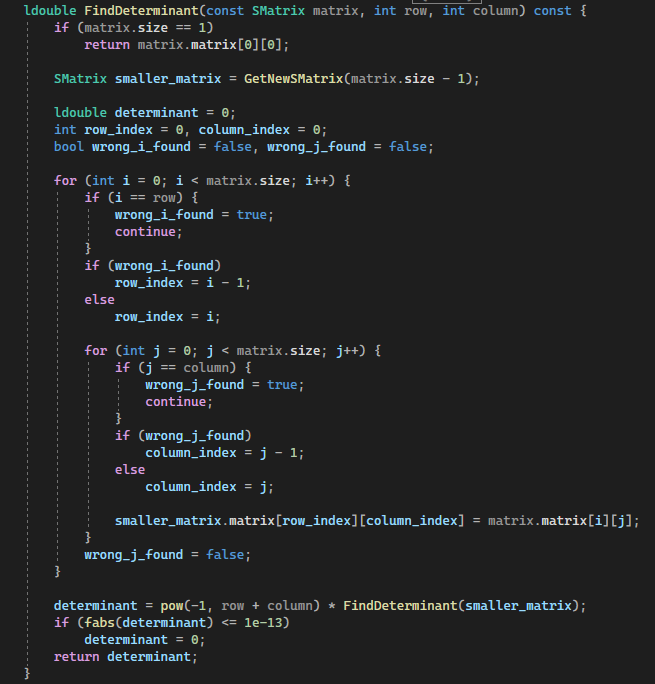


Рис. 6. Метод FindDeterminant()

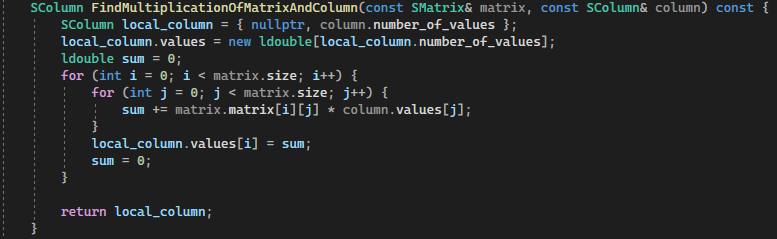


Рис. 7. Метод FindMultiplicationOfMatrixAndColumn()

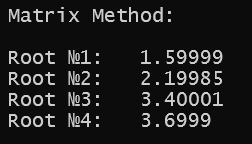


Рис. 8. Результат виконання програмної реалізації матричного методу

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

1. внесення даних в конструкторі FunctionHolder() (рис. 2);
2. виклик метода Kramer() (рис. 3), що реалізує знаходження коренів системи лінійних рівнянь методом Крамера;
3. знаходження визначника за допомогою методу FindDeterminant() (рис. 5);
4. знаходження визначників для матриць, отриманих послідовною заміною кожного стовпця заданої матриці (рис. 1) методом FindDeterminantKramer() (рис. 9);
5. вивід результату виконання в консоль (рис. 10);
6. перевірка точності отриманого розв’язку системи лінійних рівнянь (рис. 11).

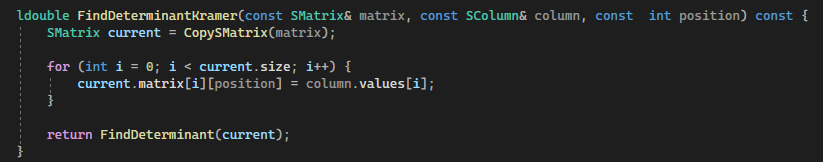


Рис. 9. Метод FindDeterminantKramer()

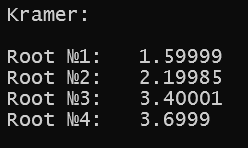


Рис. 10. Результат виконання програмної реалізації методу Крамера

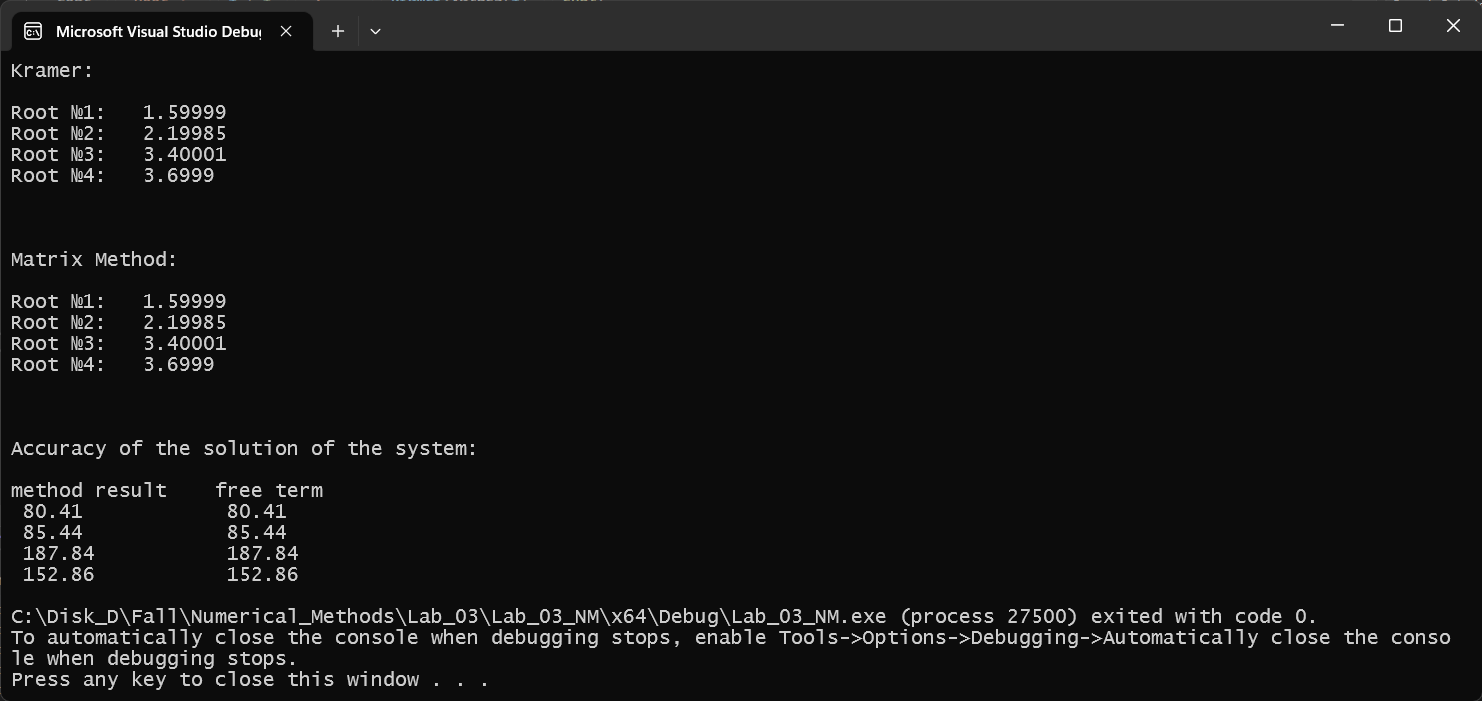


Рис. 11. Перевірка точності отриманого розв’язку системи лінійних рівнянь

**Висновки**

У результаті виконання лабораторної роботи я розробив програму розв’язування системи лінійних алгебраїчних рівнянь методом оберненої матриці та методом Крамера для заданої системи лінійних алгебраїчних рівнянь.

**Додаток**

**MethodsHeader.h:**

#pragma once

#include <cmath>

using namespace std;

typedef long double ldouble;

struct SMatrix {

ldouble\*\* matrix;

int size;

};

struct SColumn {

ldouble\* values;

int number\_of\_values;

};

template <class T>

class FunctionHolder {

private:

SMatrix matrix;

SColumn free\_term;

SColumn result;

ldouble FindDeterminant(const SMatrix matrix) const {

int index = 0;

if (matrix.size == 1)

return matrix.matrix[0][0];

SMatrix smaller\_matrix = GetNewSMatrix(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.matrix[j - 1][column] = matrix.matrix[j][k];

}

wrong\_k\_found = false;

}

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

index++;

}

return determinant;

}

ldouble FindDeterminant(const SMatrix matrix, int row, int column) const {

if (matrix.size == 1)

return matrix.matrix[0][0];

SMatrix smaller\_matrix = GetNewSMatrix(matrix.size - 1);

ldouble determinant = 0;

int row\_index = 0, column\_index = 0;

bool wrong\_i\_found = false, wrong\_j\_found = false;

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

if (i == row) {

wrong\_i\_found = true;

continue;

}

if (wrong\_i\_found)

row\_index = i - 1;

else

row\_index = i;

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

if (j == column) {

wrong\_j\_found = true;

continue;

}

if (wrong\_j\_found)

column\_index = j - 1;

else

column\_index = j;

smaller\_matrix.matrix[row\_index][column\_index] = matrix.matrix[i][j];

}

wrong\_j\_found = false;

}

determinant = pow(-1, row + column) \* FindDeterminant(smaller\_matrix);

if (fabs(determinant) <= 1e-13)

determinant = 0;

return determinant;

}

ldouble FindDeterminantKramer(const SMatrix& matrix, const SColumn& column, const int position) const {

SMatrix current = CopySMatrix(matrix);

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

current.matrix[i][position] = column.values[i];

}

return FindDeterminant(current);

}

SMatrix FindReversedMatrix(const SMatrix& matrix) const {

SMatrix local\_matrix = { nullptr, 0 };

ldouble determinant = FindDeterminant(matrix);

if (fabs(determinant) <= 1e-13) {

cout << "Reversed matrix does not exist. Try another method" << endl << endl;

}

else {

local\_matrix = GetNewSMatrix(matrix.size);

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

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

local\_matrix.matrix[j][i] = FindDeterminant(matrix, i, j) \* (1 / determinant);

}

}

}

return local\_matrix;

}

SColumn FindMultiplicationOfMatrixAndColumn(const SMatrix& matrix, const SColumn& column) const {

SColumn local\_column = { nullptr, column.number\_of\_values };

local\_column.values = new ldouble[local\_column.number\_of\_values];

ldouble sum = 0;

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

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

sum += matrix.matrix[i][j] \* column.values[j];

}

local\_column.values[i] = sum;

sum = 0;

}

return local\_column;

}

SMatrix AssignSMatrix(const T matrix, int size) {

SMatrix values = { nullptr, 0 };

bool is\_allocated = true;

ldouble\*\* local\_matrix = new ldouble \* [size];

if (local\_matrix) {

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

if (local\_matrix + i) {

local\_matrix[i] = new ldouble[size];

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

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

}

else {

is\_allocated = false;

}

}

}

else {

is\_allocated = false;

}

if (is\_allocated) {

values.matrix = local\_matrix;

values.size = size;

}

return values;

}

SMatrix GetNewSMatrix(const int size) const {

SMatrix values = { nullptr, 0 };

bool is\_allocated = true;

ldouble\*\* matrix = new ldouble \* [size];

if (matrix) {

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

if (matrix + i) {

matrix[i] = new ldouble[size];

}

else {

is\_allocated = false;

}

}

}

else {

is\_allocated = false;

}

if (is\_allocated) {

values.matrix = matrix;

values.size = size;

}

return values;

}

SMatrix CopySMatrix(const SMatrix& arr) const {

SMatrix values = { nullptr, 0 };

bool is\_allocated = true;

ldouble\*\* matrix = new ldouble \* [arr.size];

if (matrix) {

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

if (matrix + i) {

matrix[i] = new ldouble[arr.size];

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

matrix[i][j] = arr.matrix[i][j];

}

else {

is\_allocated = false;

}

}

}

else {

is\_allocated = false;

}

if (is\_allocated) {

values.matrix = matrix;

values.size = arr.size;

}

return values;

}

public:

FunctionHolder(const T matrix, const ldouble\* free\_term, const int size) {

this->matrix = AssignSMatrix(matrix, size);

this->free\_term.values = new ldouble[size];

this->free\_term.number\_of\_values = size;

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

this->free\_term.values[i] = free\_term[i];

result.number\_of\_values = size;

result.values = new ldouble[size];

}

FunctionHolder(const FunctionHolder& other) {

this->matrix = AssignSMatrix(other.matrix.matrix, other.matrix.size);

this->free\_term.number\_of\_values = other.free\_term.number\_of\_values;

this->free\_term.values = new ldouble[this->free\_term.number\_of\_values];

for (int i = 0; i < this->free\_term.number\_of\_values; i++)

this->free\_term.values[i] = other.free\_term.values[i];

result.number\_of\_values = this->matrix.size;

result.values = new ldouble[size];

}

~FunctionHolder() {

delete[] free\_term.values;

delete[] result.values;

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

delete[] matrix.matrix[i];

delete[] matrix.matrix;

}

void SetSMatrix(const T matrix, const int size) {

FunctionHolder(matrix, size);

}

SMatrix GetSMatrix() const {

return matrix;

}

SColumn Kramer() {

ldouble main\_determinant = 0;

ldouble\* var\_determinants = new ldouble[free\_term.number\_of\_values];

if (!matrix.matrix || !var\_determinants) {

cout << "Set a matrix!" << endl << endl;

}

else {

main\_determinant = FindDeterminant(matrix);

for (int i = 0; i < free\_term.number\_of\_values; i++) {

var\_determinants[i] = FindDeterminantKramer(matrix, free\_term, i);

}

if (main\_determinant == 0) {

bool var\_are\_all\_zeros = true;

for (int i = 0; i < free\_term.number\_of\_values; i++) {

if (var\_determinants[i] != 0) {

var\_are\_all\_zeros = false;

break;

}

}

if (var\_are\_all\_zeros) {

cout << "System is undefined. Try to use other method" << endl << endl;

result.number\_of\_values = 0;

delete[] result.values;

result.values = nullptr;

}

else {

cout << "System is incompatible." << endl << endl;

result.number\_of\_values = 0;

delete[] result.values;

result.values = nullptr;

}

}

else {

for (int i = 0; i < free\_term.number\_of\_values; i++)

result.values[i] = var\_determinants[i] / main\_determinant;

}

return result;

}

}

SColumn MatrixMethod() {

SMatrix reversed = FindReversedMatrix(matrix);

if (reversed.matrix) {

result = FindMultiplicationOfMatrixAndColumn(reversed, free\_term);

for (int i = 0; i < result.number\_of\_values; i++)

if (fabs(result.values[i]) <= 1e-13)

result.values[i] = 0;

}

else {

delete[] result.values;

result.values = nullptr;

result.number\_of\_values = 0;

}

return result;

}

};

**Lab\_03\_NM.cpp:**

#include <iostream>

#include "..\Methods\_Lib\MethodsHeader.h"

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}};

ldouble free\_terms[size] = {80.41, 85.44, 187.84, 152.86};

FunctionHolder<ldouble(\*)[size]> Fh(matrix, free\_terms, size);

SColumn kramer = Fh.Kramer();

SColumn matrix\_method = Fh.MatrixMethod();

cout << "Kramer: " << endl << endl;

for (int i = 0; i < kramer.number\_of\_values; i++) {

cout << "Root №" << i + 1 << ": " << kramer.values[i] << endl;

}

cout << endl << endl << endl;

cout << "Matrix Method: " << endl << endl;

for (int i = 0; i < matrix\_method.number\_of\_values; i++) {

cout << "Root №" << i + 1 << ": " << matrix\_method.values[i] << endl;

}

cout << endl << endl << endl;

cout << "Accuracy of the solution of the system: " << endl << endl << "method result \t free term" << endl;

ldouble sum{ 0 };

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

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

sum += matrix[i][j] \* kramer.values[j];

}

cout << " " << sum << "\t\t " << free\_terms[i] << endl;

sum = 0;

}

return 0;

}