**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Інститут **ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

До лабораторної роботи №3

**На тему: “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-16

Кабачок Т. О.

**Прийняв:**

доц. каф. ПЗ

Мельник Н. Б.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2025 р.

Львів – 2025

**Тема роботи:** Розвʼязування систем лінійних алгебраїчних рівнянь методом Крамера та методом оберненої матриці.

**Мета роботи:** Ознайомлення на практиці з методом Крамера та методом оберненої матриці розвʼязування систем лінійних алгебраїчних рівнянь.

**Теоретичні відомості**

**Метод Крамера**

Метод Крамера — це аналітичний метод розв’язування систем лінійних алгебраїчних рівнянь (СЛАР) з n рівняннями і n невідомими, де визначник матриці коефіцієнтів не дорівнює нулю (det A ≠ 0). Метод базується на використанні визначників і дозволяє знаходити кожну змінну системи окремо за допомогою формул Крамера.

**Формули Крамера:**

де:

* — матриця коефіцієнтів СЛАР,
* — матриця, отримана з A шляхом заміни i-го стовпця на стовпець вільних членів.

**Геометричний зміст:** кожне невідоме розраховується як відношення визначника відповідної модифікованої матриці до визначника основної матриці.

**Обчислювальна складність:** для системи з n невідомими потрібно знайти n+1 визначників n-го порядку, що вимагає приблизно n! операцій. Через це метод Крамера не використовується для систем великої розмірності (n > 3–4) через надмірну обчислювальну складність.

**Метод оберненої матриці**

Метод оберненої матриці — це алгебраїчний спосіб розв’язування систем лінійних алгебраїчних рівнянь (СЛАР) вигляду **AX = B**, де A — квадратна матриця коефіцієнтів, X — стовпець невідомих, B — стовпець вільних членів. Метод можливий лише тоді, коли матриця A має обернену, тобто її визначник не дорівнює нулю (det A ≠ 0).

**Формула розв’язку:**

де — обернена матриця до .

**Алгоритм знаходження оберненої матриці:**

1. **Обчислення визначника** . Якщо він дорівнює нулю, оберненої матриці не існує.
2. **Обчислення алгебраїчних доповнень** для кожного елемента матриці .
3. **Формування матриці доповнень**.
4. **Транспонування** цієї матриці для отримання приєднаної.
5. **Обчислення оберненої матриці** за формулою:

де — приєднана матриця.

**Особливості:**

Метод оберненої матриці є ефективним для невеликих систем (наприклад, 2×2 або 3×3), однак для великих систем обчислення стають складними і ресурсоємними, оскільки потребують обчислення великої кількості визначників.

**Хід роботи**

Файл **main.rs**

mod functions;

fn main() {

let a: Vec<Vec<f32>> = vec![

vec![0.34, 0.71, 0.63],

vec![0.71, -0.65, -0.18],

vec![1.17, -2.35, 0.75],

];

let b: Vec<f32> = vec![2.08, 0.17, 1.28];

println!("Задана СЛАР:");

for i in 0..3 {

for j in 0..3 {

print!("{} x{} ", a[i][j],j+1);

}

println!("= {}", b[i]);

}

let (x1, x2, x3) = functions::kramer(&a, &b);

println!("Корені системи: x1 = {}, x2 = {}, x3 = {}", x1, x2, x3);

let (x1, x2, x3) = functions::obern\_matr(&a, &b);

println!("Корені системи: x1 = {}, x2 = {}, x3 = {}", x1, x2, x3);

}

Файл **functions.rs**

pub fn kramer(a: &Vec<Vec<f32>>, b: &Vec<f32>) -> (f32, f32, f32) {

println!("=====================");

println!("Метод Крамера:");

println!("=====================");

let det\_main = det(&a);

if det\_main.abs() < f32::EPSILON {

panic!("Система не має унікального розв'язку або має безліч розв'язків");

}

println!("\nОсновний детермінант: {}", det\_main);

println!("---------------------");

let mut a\_1 = a.clone();

for i in 0..3 {

a\_1[i][0] = b[i];

}

let det\_a\_1 = det(&a\_1);

println!("\nМатриця з заміненим першим стовпцем на b:");

for row in &a\_1 {

println!("{:?}", row);

}

println!("Детермінант для a\_1: {}", det\_a\_1);

println!("---------------------");

let mut a\_2 = a.clone();

for i in 0..3 {

a\_2[i][1] = b[i];

}

let det\_a\_2 = det(&a\_2);

println!("\nМатриця з заміненим другим стовпцем на b:");

for row in &a\_2 {

println!("{:?}", row);

}

println!("Детермінант для a\_2: {}", det\_a\_2);

println!("---------------------");

let mut a\_3 = a.clone();

for i in 0..3 {

a\_3[i][2] = b[i];

}

let det\_a\_3 = det(&a\_3);

println!("\nМатриця з заміненим третім стовпцем на b:");

for row in &a\_3 {

println!("{:?}", row);

}

println!("Детермінант для a\_3: {}", det\_a\_3);

println!("---------------------");

if det\_a\_1.abs() < f32::EPSILON || det\_a\_2.abs() < f32::EPSILON || det\_a\_3.abs() < f32::EPSILON {

panic!("Система не має унікального розв'язку або має безліч розв'язків");

}

let x1 = det\_a\_1 / det\_main;

let x2 = det\_a\_2 / det\_main;

let x3 = det\_a\_3 / det\_main;

println!("\nЗнайдені розв'язки:");

println!("x1 = {}", x1);

println!("x2 = {}", x2);

println!("x3 = {}", x3);

println!("=====================");

(x1, x2, x3)

}

pub fn obern\_matr(a: &Vec<Vec<f32>>, b: &Vec<f32>) -> (f32, f32, f32) {

println!("=====================");

println!("Метод оберненої матриці:");

println!("=====================");

let det\_main = det(a);

if det\_main.abs() < f32::EPSILON {

panic!("Система не має унікального розв'язку");

}

println!("\nДетермінант основної матриці: {}", det\_main);

println!("---------------------");

let mut cofactors = vec![vec![0.0; 3]; 3];

for i in 0..3 {

for j in 0..3 {

let sign = if (i + j) % 2 == 0 { 1.0 } else { -1.0 };

cofactors[i][j] = sign \* minor(a, i, j);

}

}

println!("\nМатриця алгебраїчних доповнень:");

for row in &cofactors {

println!("{:?}", row);

}

println!("---------------------");

let mut adjugate = vec![vec![0.0; 3]; 3];

for i in 0..3 {

for j in 0..3 {

adjugate[i][j] = cofactors[j][i];

}

}

println!("\nТранспонована матриця алгебраїчних доповнень:");

for row in &adjugate {

println!("{:?}", row);

}

println!("---------------------");

let mut inverse = vec![vec![0.0; 3]; 3];

for i in 0..3 {

for j in 0..3 {

inverse[i][j] = adjugate[i][j] / det\_main;

}

}

println!("\nОбернена матриця:");

for row in &inverse {

println!("{:?}", row);

}

println!("---------------------");

let mut result = vec![0.0; 3];

for i in 0..3 {

for j in 0..3 {

result[i] += inverse[i][j] \* b[j];

}

}

println!("\nЗнайдені розв'язки:");

println!("x1 = {}", result[0]);

println!("x2 = {}", result[1]);

println!("x3 = {}", result[2]);

println!("=====================");

(result[0], result[1], result[2])

}

fn minor(a: &Vec<Vec<f32>>, row: usize, col: usize) -> f32 {

let mut m = Vec::new();

for i in 0..3 {

if i == row {

continue;

}

let mut row\_vals = Vec::new();

for j in 0..3 {

if j == col {

continue;

}

row\_vals.push(a[i][j]);

}

m.push(row\_vals);

}

m[0][0] \* m[1][1] - m[0][1] \* m[1][0]

}

pub fn det(a: &Vec<Vec<f32>>) -> f32 {

let det\_value = a[0][0] \* (a[1][1] \* a[2][2] - a[1][2] \* a[2][1])

- a[0][1] \* (a[1][0] \* a[2][2] - a[1][2] \* a[2][0])

+ a[0][2] \* (a[1][0] \* a[2][1] - a[1][1] \* a[2][0]);

det\_value

}

**Результат виконання програми**

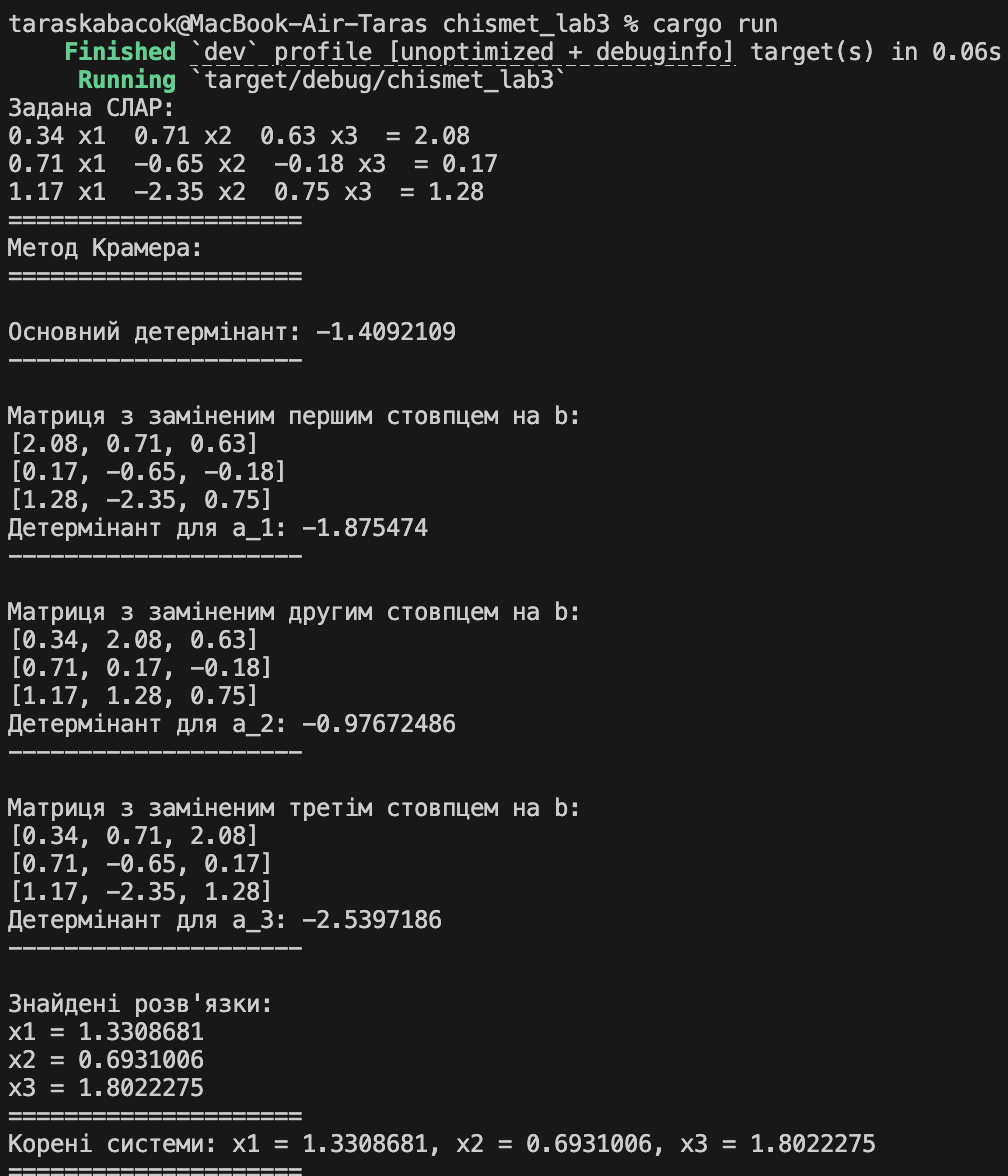
****

Рис. 1. Розвʼязок СЛАР методом Крамера.

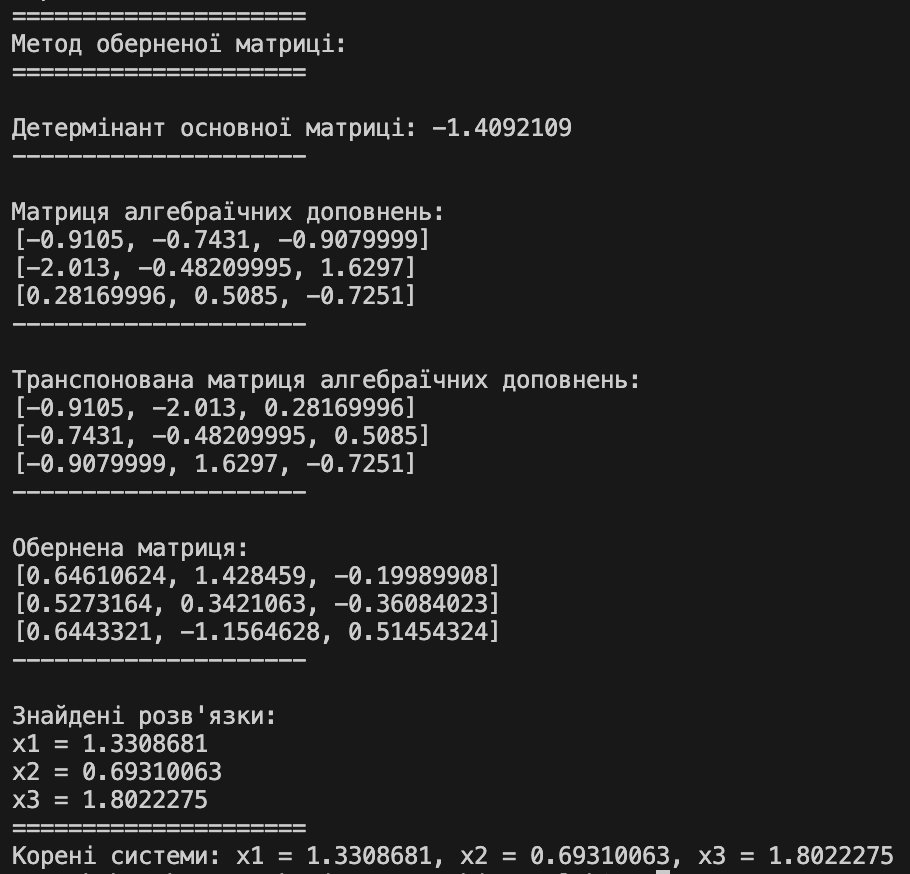


Рис. 2. Розвʼязок СЛАР методом оберненої матриці.

**Висновки**

Під час виконання даної лабораторної роботи я ознайомився з двома методами розв’язання систем лінійних алгебраїчних рівнянь: методом Крамера та методом оберненої матриці. Я вивчив теоретичні основи кожного з методів, умови їх застосування, а також особливості обчислень. З використанням мови програмування Rust я реалізував обидва методи, а також програму для поетапного виведення результатів розв’язку. Отримані значення змінних підтвердили правильність реалізації алгоритмів та відповідність результатів очікуваним. Обидва методи виявилися ефективними для задач з невеликою кількістю рівнянь, але малопридатними для систем високого порядку через складність обчислень.