Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

Про виконання лабораторної роботи №3

**На тему:**

«Розв’язування систем лінійних алгебраїчних рівнянь

методом Крамера та методом оберненої матриці»

з дисципліни «Чисельні методи»

**Лекторка:**

доцент каф. ПЗ

Мельник Н. Б.

**Виконав:**

ст. гр. ПЗ-11

Солтисюк Д.А.

**Прийняла:**

доцент каф. ПЗ

Мельник Н. Б.

« \_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑ = \_\_\_\_\_ \_\_\_\_\_\_\_\_\_ .

Львів – 2022

**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.

**Теоретичні відомості**

**Метод Крамера** – використовується для розв’язання СЛАР яка містить n рівнянь та n невідомих, причому визначник матриці коефіцієнтів А не дорівнює нулю. Для знаходження коренів застосовують формулу , де це матриця A, в якій *i*-тий стовпець замінений стовпцем вільних членів (матриці B).

**Метод оберненої матриці** – полягає в отриманні рівняння з рівняння шляхом домноження його на . Для знаходження оберненої матриці потрібно транспонувати матрицю алгебраїчних доповнень та поділити її на визначник матриці А. Для знаходження алгебраїчних доповнень скористаємось формулою , де – це мінор, який отримують з матриці А викреслюванням *i*-го рядка та *j*-го стовпця.

**Індивідуальне завдання**

*Варіант 24*

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

**Код функцій**

**import** textwrap

**from** colorama **import** Fore, Style

Row = list[float]

Matrix = list[Row]

Values = list[float]

MethodComputationResult = list[float]

**def** matrix\_is\_square(matrix: Matrix) -> bool:

row\_should\_have: int = len(matrix)

**for** row **in** matrix:

**if** len(row) != row\_should\_have:

**return** False

**return** True

**def** det(m: Matrix, n: int) -> float:

**if** n == 1:

**return** m[0][0]

z = 0

**for** r **in** range(n):

k = m[:]

**del** k[r]

z += m[r][0] \* (-1)\*\*r \* det([p[1:] **for** p **in** k], n - 1)

**return** z

**def** cramers\_method(m: Matrix, v: Values) -> MethodComputationResult:

w = len(v)

d = det(m, w)

**if** d == 0:

**return** []

r = [

det([r[0:i] + [s] + r[i + 1:] **for** r, s **in** zip(m, v)], w) / d

**for** i **in** range(w)

]

**return** r

**def** multiply\_2d\_to\_1d\_matrices(m2d, m1d) -> MethodComputationResult:

m2d\_length = len(m2d)

m1d\_length = len(m1d)

**if** (m2d\_length != m1d\_length):

**raise** ValueError(

"One of the matrices is not eligible for multiplication")

width = m1d\_length

result: MethodComputationResult = [0 **for** \_ **in** range(width)]

**for** i **in** range(m2d\_length):

**for** j **in** range(len(m2d[0])):

*# resulted matrix*

result[i] += m2d[i][j] \* m1d[j]

**return** result

**def** matrix\_method(eq\_matrix: Matrix,

values: Values) -> MethodComputationResult:

width = len(values)

determinant = det(eq\_matrix, width)

**if** determinant == 0:

**return** []

ac\_matrix: Matrix = []

**for** row **in** range(width):

ac\_row = []

**for** column **in** range(width):

*# deep copy without refference*

intermediate = eq\_matrix.copy()

*# delete row from matrix*

intermediate.pop(row)

*# delete column from matrix*

intermediate = [

list(x) **for** x **in** zip(\*[

d **for** i, d **in** enumerate(zip(\*intermediate)) **if** i != column

])

]

*# minor out of intermediate array*

minor = det(intermediate, width - 1)

*# algebraic complement*

ac = pow(-1, row + column) \* minor

ac\_row.append(ac)

ac\_matrix.append(ac\_row)

ac\_transposed\_matrix = [list(x) **for** x **in** zip(\*ac\_matrix)]

ac\_inverted\_matrix = [[z / determinant **for** z **in** y]

**for** y **in** ac\_transposed\_matrix]

**return** multiply\_2d\_to\_1d\_matrices(ac\_inverted\_matrix, values)

**def** run():

matrix: Matrix = [

[0.13, -0.14, -2.00],

[0.75, 0.18, -0.77],

[0.28, -0.17, 0.39],

]

values: Values = [0.15, 0.11, 0.12]

**if** **not** matrix\_is\_square(matrix):

**raise** ValueError("Please, provide valid square matrix")

**def** method\_result\_description(title: str, result: MethodComputationResult):

**return** textwrap.dedent(f"""

{Fore.CYAN}{title}{Style.RESET\_ALL}

{Fore.RED}Result:{Style.RESET\_ALL} {result}

""")

output = [("Cramers method", cramers\_method(matrix, values)),

("Matrix method", matrix\_method(matrix, values))]

**for** j, k **in** output:

**print**(method\_result\_description(j, k))

**if** \_\_name\_\_ == "\_\_main\_\_":

run()

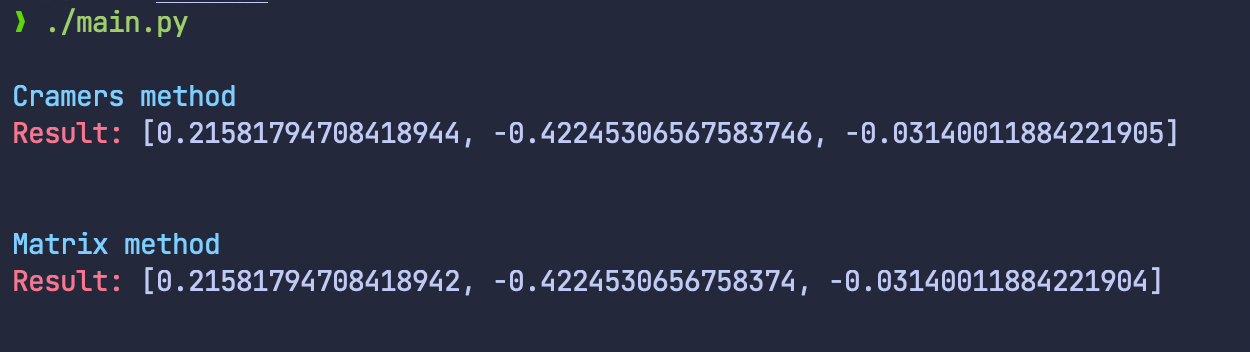
**Протокол роботи**

Рис.1. Робота програми

**Висновки**

Виконуючи лабораторну роботу №3, я навчився розв’язувати СЛАР методами Крамера та оберненої матриці, а також склав програму, яка їх розв’язує автоматично.