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 A не дорівнює нулю. Для знаходження коренів застосовують формулу $x_i = \frac{\det \mathrm{A_i}}{\det \mathrm{A}}$, де A_i це матриця A, в якій *i*-тий стовпець замінений стовпцем вільних членів (матриці B).

Метод оберненої матриці – полягає в отриманні рівняння $X = A^{-1}B$ з рівняння $AX = B$ шляхом домноження його на A^{-1}. Для знаходження оберненої матриці потрібно транспонувати матрицю алгебраїчних доповнень та поділити її на визначник матриці A. Для знаходження алгебраїчних доповнень скористаємось формулою $\bar{A}_{ij} = (-1)^{i+j} M_{ij}$, де M_{ij} – це мінор, який отримують з матриці A 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: $\begin{cases} 0{,}13x_1 - 0{,}14x_2 - 2{,}00x_3 = 0{,}15 \\ 0{,}75x_1 + 0{,}18x_2 - 0{,}77x_3 = 0{,}11 \\ 0{,}28x_1 - 0{,}17x_2 + 0{,}39x_3 = 0{,}12 \end{cases}$

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

```
❯ ./main.py

Cramers method
Result: [0.21581794708418944, -0.42245306567583746, -0.03140011884221905]


Matrix method
Result: [0.21581794708418942, -0.4224530656758374, -0.03140011884221904]
```

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.