Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторної роботи №4

На тему:

«Розв’язування систем лінійних алгебраїчних рівнянь методом Гауса та методом LU-розкладу»
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 Гауса та методом LU-розкладу.

Мета: Ознайомлення на практиці з методом Гауса та методом LU-розкладу розв’язування систем лінійних алгебраїчних рівнянь.

Теоретичні відомості

Метод Гауса – поділяється на два етапи – прямий і зворотній хід. Під час прямого ходу СЛАР зводиться до східчастого вигляду. Для того щоб не втрачати точність при діленні на діагональний елемент, існує модифікація цього методу – з вибором головного елемента. Це означає, що на кожному етапі зведення системи до східчастого вигляду рівняння переставляють таким чином, щоб на діагоналі знаходився найбільший за модулем елемент цього стовпця. Під час зворотного ходу застосовують формулу $x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{k=i+1}^{n} a_{ik}x_k \right), \quad i = \overline{1,n}$ для знаходження коренів.

Метод LU-розкладу – полягає в розкладанні матриці коефіцієнтів A на добуток нижньої трикутної матриці L, елементи головної діагоналі якої не дорівнюють нулеві та верхньої трикутної U, на головній діагоналі якої містяться одиниці. Для знаходження оберненої матриці потрібно транспонувати матрицю алгебраїчних доповнень та поділити її на визначник матриці A. Звідси отримаємо рівняння LUX=B, введемо допоміжний вектор Y=UX і визначимо його з рівняння LY=B за формулою $y_i = \frac{1}{l_{ii}}\left(b_i - \sum_{m=1}^{i-1} l_{im}y_m \right), \quad i = \overline{1,n}$, що буде прямим ходом цього методу. Під час зворотного ходу визначаємо корені за формулою

$$x_i = y_i - \sum_{m=i+1}^{n} u_{im}x_m, \quad i \le n.$$

Для знаходження матриць L і U скористаємось формулами (метод Краута):

$$l_{ik} = a_{ik} - \sum_{m=1}^{k-1} l_{im}u_{mk}, \quad u_{ii} = 1,\ u_{kj} = \frac{1}{l_{kk}}\left(a_{kj} - \sum_{m=1}^{k-1} l_{km}u_{mj} \right),$$

$$i = \overline{k,n}, \quad j = \overline{k+1,n}$$

Індивідуальне завдання

Варіант 15

Скласти програму розв'язування системи лінійних алгебраїчних рівнянь методом Гауса та LU-розкладу:

$$9.\begin{cases} 0{,}13x_1 - 0{,}14x_2 - 2{,}00x_3 = 0{,}15 \\ 0{,}75x_1 + 0{,}18x_2 - 0{,}77x_3 = 0{,}11 \\ 0{,}28x_1 - 0{,}17x_2 + 0{,}39x_3 = 0{,}12 \end{cases}$$

Код функцій

```
from math import fabs

import numpy as np
from common.utils import print_header, print_matrix, print_method_introduction

Row = list[float]
Matrix = list[Row]


class Method:

    def compile(self):
        raise NotImplementedError

    def execute_method(self):
        raise NotImplementedError


class MatrixOrientedMethod(Method):
    A = np.array([])
    B = np.array([])

    method_name: str = "Unknown"

    def __init__(self, A: Matrix, B: Row) -> None:
        super().__init__()

        self.A = np.array(A)
        self.B = np.array(B)

    def execute_method(self):
        raise NotImplementedError

    def compile(self):

        print_method_introduction(self.method_name)

        print_header("Initial values")
        print_matrix(self.A, "A")
        print_matrix([self.B], "B")
```

```
        result = self.execute_method()

        print("\n")
        print_header("Resulting vectors")
        print(result)

        print("\n")
        print_header("Results verification")
        print(f"AX - B = {np.dot(self.A, result) - self.B}")


class GaussianEliminationMethod(MatrixOrientedMethod):

    def __init__(self, A: Matrix, B: Row) -> None:
        super().__init__(A, B)

        self.method_name = "Gaussian elimination"

    def execute_method(self):
        n = len(self.B)

        # Elimination phase
        for k in range(n - 1):
            # find the best raw
            for i in range(k + 1, n):
                if fabs(self.A[i, k]) > fabs(self.A[k, k]):
                    self.A[[k, i]] = self.A[[i, k]]
                    self.B[[k, i]] = self.B[[i, k]]
                    break

            for i in range(k + 1, n):
                if self.A[i, k] != 0.0:
                    cof = self.A[i, k] / self.A[k, k]
                    # calculate the new row
                    self.A[i] = self.A[i] - cof * self.A[k]
                    # update vector b
                    self.B[i] = self.B[i] - cof * self.B[k]

            print(f"\nIteration {k}:")
            print_matrix(self.A, "A")
            print_matrix([self.B], "B")

        # backward substitution
        x = np.zeros(n)
        for k in range(n - 1, -1, -1):
            x[k] = (self.B[k] -
                    np.dot(self.A[k, k + 1:n], x[k + 1:n])) / self.A[k, k]

        return x


class LUDecompositionMethod(MatrixOrientedMethod):

    def __init__(self, A: Matrix, B: Row) -> None:
        super().__init__(A, B)

        self.method_name = "LU decomposition"

    def lu_decompose(self):
        """
```

```
    Decomposes a matrix A by PA=LU
    """

    def pivotize(m):
        """
        Creates the pivoting matrix for m.
        """
        n = len(m)
        ID = [[float(i == j) for i in range(n)] for j in range(n)]
        for j in range(n):
            row = max(range(j, n), key=lambda i: abs(m[i][j]))
            if j != row:
                ID[j], ID[row] = ID[row], ID[j]
        return ID

    n = len(self.A)

    L = np.zeros((n, n))
    U = np.zeros((n, n))
    P = np.asarray(pivotize(self.A))
    PA = np.dot(P, self.A)

    for j in range(n):
        L[j][j] = 1.0
        for i in range(j + 1):
            s1 = sum(U[k][j] * L[i][k] for k in range(i))
            U[i][j] = PA[i][j] - s1
        for i in range(j, n):
            s2 = sum(U[k][j] * L[i][k] for k in range(j))
            L[i][j] = (PA[i][j] - s2) / U[j][j]

    return L, U, P

def lu_solve(self, L, U, P, B):

    def forward_sub(L, b):
        """solution to Lx = b
        L must be a lower-triangular matrix
        b must be a vector of the same leading dimension as L
        """
        n = len(L)
        x = np.zeros(n)
        for i in range(n):
            sum = 0
            for j in range(i):
                sum += L[i, j] * x[j]
            x[i] = (b[i] - sum) / L[i, i]
        return x

    def back_sub(U, b):
        """solution to Ux = b
        U must be an upper-triangular matrix
        b must be a vector of the same leading dimension as U
        """
        n = len(U)
        x = np.zeros(n)
        for i in range(n - 1, -1, -1):
            sum = 0
            for j in range(n - 1, i, -1):
                sum += U[i, j] * x[j]
            x[i] = (b[i] - sum) / U[i, i]
```

```
            return x

        y = forward_sub(L, np.dot(P, B))
        x = back_sub(U, y)

        return x

    def execute_method(self):
        L, U, P = self.lu_decompose()

        print_matrix(L, "L")
        print_matrix(U, "U")
        print_matrix(P, "P")
        print("\nIs LU = PA:", np.array_equal(np.dot(L, U), np.dot(P, self.A)))

        return self.lu_solve(L, U, P, self.B.copy())
```

Протокол роботи

```
>>> Gaussian elimination method

Initial values

A:
0.83    1.41    0.58
1.23    0.83    1.17
1.43    1.58    0.83

B:
2.71    5.26    1.03

Iteration 0

A:
1.23    0.83                    1.17
0.0     0.8499186991869918      -0.20951219512195118
0.0     0.6150406504065041      -0.5302439024390243

B:
5.26    -0.839430894308943      -5.085284552845528

Iteration 1

A:
1.23    0.83                    1.17
0.0     0.8499186991869918      -0.20951219512195118
0.0     0.0                     -0.3786311459728333

B:
5.26    -0.839430894308943      -4.4778333652190545


Resulting vectors
[-8.27382074  1.92764153 11.82637354]


Results verification
AX - B = [-8.88178420e-16  1.11022302e-16  0.00000000e+00]
```

```
>>> LU decomposition method

Initial values

A:
0.83    1.41    0.58
1.23    0.83    1.17
1.43    1.58    0.83

B:
2.71    5.26    1.03

L:
1.0                     0.0                     0.0
0.5804195804195804      1.0                     0.0
0.8601398601398602      -1.0732018726060442     1.0

U:
1.43    1.58                    0.83
0.0     0.49293706293706285     0.09825174825174826
0.0     0.0                     0.5615278762945098

P:
0.0     0.0     1.0
1.0     0.0     0.0
0.0     1.0     0.0

Is LU = PA: True


Resulting vectors
[-8.27382074  1.92764153 11.82637354]


Results verification
AX - B = [0.00000000e+00 8.88178420e-16 2.22044605e-16]
```

Рис.1. Робота програми

Висновки

Виконуючи лабораторну роботу №4, я навчився розв’язувати СЛАР методами Гауса та LU-розкладу, а також склав програму, яка їх розв’язує автоматично.