Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки
Кафедра обчислювальної техніки

Лабораторна робота № 2

з дисципліни «Чисельні методи»

на тему

“ Розв’язання систем лінійних алгебраїчних рівнянь (СЛАР) прямими методами. Звичайний метод Гауса та метод квадратних коренів”

Виконав:
студент гр. ІП-93
Завальнюк Максим

Викладач:
доц. Рибачук Л.В.

Київ – 2021

Зміст

1 Постановка задачі

Розв’язати систему рівнянь методом квадратних коренів з кількістю значущих цифр m = 6. Вивести матрицю T, вектор y та розв’язок системи. Навести результат перевірки: вектор нев’язки r = b – Ax, де x - отриманий розв’язок.

Порівняти корені рівнянь, отримані у NumPy, із власними результатами за допомогою методу середньоквадратичної похибки.

2 Розв’язок

Початкова матриця системи А

```
a = [[1.0, 0.42, 0.54, 0.66],
     [0.42, 1.0, 0.32, 0.44],
     [0.54, 0.32, 1.0, 0.22],
     [0.66, 0.44, 0.22, 1.0]]
```

Вектор правої частини b

```
b = [0.3, 0.5, 0.7, 0.9]
```

Нижче приведена наведені результати виконання програми.

Проміжні результати (матриця Т та вектор y):

```
matrix T: [[ 1.          0.          0.          0.        ]
 [ 0.42        0.9075241   0.          0.        ]
 [ 0.54        0.102697    0.83537616  0.        ]
 [ 0.66        0.17938917 -0.18533295  0.7055999 ]]
matrix T-transpose: [[ 1.          0.42        0.54        0.66      ]
 [ 0.          0.9075241   0.102697    0.17938917]
 [ 0.          0.          0.83537616 -0.18533295]
 [ 0.          0.          0.          0.7055999 ]]
Vector y: [0.3        0.41211027 0.59335846 1.04597627]
```

Вектор коренів рівнянь **x**:

```
Solution vector: [-1.25779375  0.0434873   1.03916625  1.48239288]
```

Вектор нев’язки r = b – Ax:

```
Residual vector [[0 0 0 0]]
```

З вигляду матриці незв’язки випливає, що метод квадратного кореня є абсолютно точним для розв’язання систем з симетричною матрицею А. Така точність забезпечується в першу чергу тим, що сам метод заснований на строгих математичних перетвореннях.

3 Розв'язок за допомогою NumPy

Нижче наведено розв'язок системи у NumPy:

```
xm = np.linalg.solve(a, b)
print('NumPy solution:', xm)
print('Residual vector', np.matrix(np.subtract(b, np.dot(a, np.linalg.solve(a, b))), int))
```

```
NumPy solution: [-1.25779375  0.0434873   1.03916625  1.48239288]
Residual vector [[0 0 0 0]]
```

Порівняння отриманого результату (п. 2) із результатом з NumPy за допомогою методу середньоквадратичної похибки:

```
print('Fault:', round(get_fault(x, xm)))
```

```
Fault: 0
```

Це дуже добре, оскільки результат вважають гарним, якщо відносна похибка не перевищує 0.1 %. У даному випадку взагалі 0 %.

4 Лістинг програми

```python
import numpy as np
from math import sqrt


def get_transpose(matrix: list) -> list:
    """
    Function for getting transpose matrix
    :param matrix: input matrix
    :return: transpose input matrix
    """
    matrix_t = matrix.copy()
    n = len(matrix_t)
    for i in range(n):
        for j in range(i, n):
            matrix_t[i][j], matrix_t[j][i] = matrix_t[j][i], matrix_t[i][j]
    return matrix_t


def subtract(matrix_a: list, matrix_b: list) -> list:
    """
    Function for substracting two matrices
    :param matrix_a: the first input matrix
    :param matrix_b: the second input matrix
    :return: the result matrix
    """
    result = matrix_a.copy()

    for i in range(len(matrix_a)):
        for j in range(len(matrix_a[0])):
            result[i][j] = matrix_a[i][j] - matrix_b[i][j]

    return result


def multiply(matrix_a: list, matrix_b: list) -> list:
    """
    Function for multiplying two matrices
    :param matrix_a: the first input matrix A[i][j]
    :param matrix_b: the second input matrix B[m][n]
    :return: the result matrix C[i][n]
    """
    result = []
    # Creating result by sizes
    for _ in range(len(matrix_a)):
        array = [0] * len(matrix_b[0])
        result.append(array)

    if len(matrix_a[0]) == len(matrix_b):  # j == m
        for i in range(len(matrix_a)):
            for j in range(len(matrix_b[0])):
                for k in range(len(matrix_b)):
                    result[i][j] += matrix_a[i][k] * matrix_b[k][j]
    else:
        raise ValueError('j != m')

    return result


def factorization(matrix_a: list) -> list:
    """
    Cholesky decomposition
    :param matrix_a: start matrix
    :return: Lower-triangular matrix
```

```
    """
    t = np.zeros_like(matrix_a)
    n = len(matrix_a)
    for j in range(n):
        for i in range(j, n):
            if i == j:
                sum_k = 0
                for k in range(j):
                    sum_k += t[i][k] ** 2
                t[i][j] = sqrt(matrix_a[i][j] - sum_k)
            else:
                sum_k = 0
                for k in range(j):
                    sum_k += t[i][k] * t[j][k]
                t[i][j] = (matrix_a[i][j] - sum_k) / t[j][j]
    return t


def solve(lower_matrix: list, upper_matrix: list, vector_b: list) -> list:
    """
    The solve main function
    :param lower_matrix: Lower-triangular matrix
    :param upper_matrix: Upper-triangular matrix
    :param vector_b: vector B
    :return: vector x, the solution
    """
    n = len(lower_matrix)
    y = np.zeros(n)
    vector_x = np.zeros(n)

    # forward substitution
    for i in range(n):
        sum_j = 0
        for j in range(i):
            sum_j += lower_matrix[i][j] * y[j]
        y[i] = (vector_b[i] - sum_j) / lower_matrix[i][i]
    print('Vector y:', y)
    # backward substitution
    for i in range(n - 1, -1, -1):
        sum_j = 0
        for j in range(i + 1, n):
            sum_j += upper_matrix[i][j] * vector_x[j]
        vector_x[i] = (y[i] - sum_j) / upper_matrix[i][i]

    return vector_x


def get_fault(x: list, xm: list) -> float:
    """
    Function for finding get_fault
    :param x: my solution vector
    :param xm: NumPy solution vector
    :return: fault
    """
    sum_k = 0
    n = len(x)
    for k in range(1, n):
        sum_k += (x[k] - xm[k]) ** 2
    result = sqrt(sum_k / n)
    return result


a = [[1.0, 0.42, 0.54, 0.66],
     [0.42, 1.0, 0.32, 0.44],
     [0.54, 0.32, 1.0, 0.22],
     [0.66, 0.44, 0.22, 1.0]]
```

```
b = [0.3, 0.5, 0.7, 0.9]
T = factorization(a)
print('matrix T:', T)
U = get_transpose(T)
print('matrix T-transpose:', U)
x = solve(T, U, b)
xm = np.linalg.solve(a, b)
print('Solution vector:', x)
print('Residual vector', np.matrix(np.subtract(b, np.dot(a, x)), int))
print('NumPy solution:', xm)
print('Residual vector', np.matrix(np.subtract(b, np.dot(a, np.linalg.solve(a,
b))), int))
print('Fault:', round(get_fault(x, xm)))
```