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

Лабораторна робота № 3

з дисципліни «Чисельні методи»

на тему

“Розв’язання систем лінійних алгебраїчних рівнянь (СЛАР) ітераційними методами. Метод простої ітерації. Метод Зейделя”

Виконав:
студент гр. ІП-93
Завальнюк Максим

Викладач:
доц. Рибачук Л.В.

Київ – 2021

Зміст

1 Постановка задачі

Розв’язати систему рівнянь методом простих ітерацій з кількістю значущих цифр m = 6. Якщо матриця не є матрицею із діагональною перевагою, привести систему до еквівалентної, у якій є діагональна перевага (письмово). Вивести проміжні результати та кінцевий результат, на кожній ітерації вектори нев’язки та результати перших трьох та останньої ітерацій методу. 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 і вивести вектор нев’язки для цього розв’язку.

2 Розв’язок

Початкова матриця системи A

```
a = [[2.12, 0.42, 1.34, 0.88],
     [0.42, 3.95, 1.87, 0.43],
     [1.34, 1.87, 2.98, 0.46],
     [0.88, 0.43, 0.46, 4.44]]
```

Вектор правої частини b

```
b = [11.172, 0.115, 0.009, 9.349]
```

Як видно, матриця A не є матрицею з діагональною перевагою, тому домножимо ліву і праву частину рівняння на транспоновану матрицю A.

$$A^T \times A \times x = A^T \times b$$

Результати:

```
a = [[4.4944, 0.1764, 1.7956, 0.7744],
     [0.1764, 15.6025, 3.4969, 0.1849],
     [1.7956, 3.4969, 8.8804, 0.2116],
     [0.7744, 0.1849, 0.2116, 19.7136]]
b = [31.97212, 9.18339, 19.51289, 51.39451]
```

Нижче приведені результати виконання програми.

Результати перших трьох ітерацій:

```
Temporary result: [7.113768244927021, 0.5885845217112642, 2.1972985451105806, 2.607058578849119]
Residual vector: [[ -6.06820174  -9.42064713 -15.38335707  -6.08267978]]
Temporary result: [5.763598758301179, -0.015206353599200084, 0.4650165454430484, 2.298506118665221]
Residual vector: [[3.45593729 6.35283817 4.60105034 1.52376305]]
Temporary result: [6.532541730483049, 0.3919616112760568, 0.9831295068674704, 2.375801136034834]
```

Вектори нев’язки на кожній ітерації:

```
Residual vector: [[-1.06200532 -1.9617226  -2.82089528 -0.7803875 ]]
Residual vector: [[0.62321426 1.15980653 0.87233796 0.27345035]]
Residual vector: [[-0.20023953 -0.37053211 -0.51186219 -0.14191221]]
Residual vector: [[0.11326142 0.21074993 0.16456809 0.05108951]]
Residual vector: [[-0.03766499 -0.06972774 -0.0930327  -0.02593413]]
Residual vector: [[0.02061813 0.03835572 0.03095395 0.00953288]]
Residual vector: [[-0.00706695 -0.01308761 -0.01693612 -0.00474468]]
Residual vector: [[0.0037588  0.00699093 0.00580756 0.00177631]]
Residual vector: [[-0.00132309 -0.00245108 -0.00308762 -0.00086888]]
Residual vector: [[0.00068615 0.00127591 0.00108727 0.00033059]]
Residual vector: [[-0.00024726 -0.00045818 -0.00056364 -0.00015925]]
Residual vector: [[0.0001254  0.00023315 0.00020318 0.00006146]]
Residual vector: [[-0.00004613 -0.00008551 -0.00010302 -0.00002921]]
Residual vector: [[0.00002294 0.00004265 0.00003791 0.00001142]]
Residual vector: [[-0.0000086  -0.00001594 -0.00001885 -0.00000536]]
Residual vector: [[0.0000042  0.00000781 0.00000706 0.00000212]]
```

Останній результат, к-сть ітерацій та вектор **x**:

```
Last result: [6.410854615931584, 0.3278297003946807, 0.7160811273879595, 2.344463045415071]
Iterations: 18
Our solution: [[6.41085462 0.3278297  0.71608113 2.34446305]]
```

Вектор нев’язки r = b – Ax:

```
Residual vector: [[-0.0000016  -0.00000297 -0.00000345 -0.00000098]]
```

З к-сті ітерацій випливає, що при великому числі невідомих метод Гаусса стає вельми складним у плані обчислювальних і тимчасових витрат. Тому іноді зручніше використовувати наближені (ітераційні) чисельні методи, зокрема метод Якобі.

3 Розв’язок за допомогою NumPy

Нижче наведено розв’язок системи у NumPy:

```
sol_np = np.linalg.solve(a, b)
print(f'NumPy solution: {sol_np}')
print(f'Residual vector for NumPy: {np.matrix(np.subtract(b, np.dot(a,
sol_np)))}')
```

```
NumPy solution: [6.41085439 0.32782958 0.71608083 2.34446301]
Residual vector for NumPy: [[ 0.  0.  0. -0.]]
```

Порівняння отриманого результату (п. 3) із результатом з NumPy за допомогою методу середньоквадратичної похибки:

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

```
Fault: 0.0
```

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

4 Лістинг програми

```
import numpy as np
from checker.fault import get_fault

np.set_printoptions(suppress=True)


def solve_jacobi(matrix_a: list, vector_b: list, epsilon=10 ** (-6)) -> list:
    """
    Solve by Jacobi method(simple iteration)
    :param matrix_a: start matrix
    :param vector_b: start vector
    :param epsilon: number for comparing
    :return: solution vector
    """
    solution_vector = [0 for _ in range(len(matrix_a[0]))]

    matrix_c = []
    vector_d = []
    # Create matrix C and vector D
    for i in range(len(matrix_a)):
        element_d = vector_b[i] / matrix_a[i][i]
        vector_d.append(element_d)

        element_c = []
        for j in range(len(matrix_a[0])):
            if i == j:
                element_c.append(0)
            else:
                element_c.append((-1) * matrix_a[i][j] / matrix_a[i][i])
        matrix_c.append(element_c)

    iterations = 0
    tmp = 0
    # Start the main algorithm
    while True:
        divs = []
        left_part = np.dot(matrix_c, solution_vector)
        for i in range(len(solution_vector)):
            x_next = left_part[i] + vector_d[i]
            divs.append(abs(x_next - solution_vector[i]))
            solution_vector[i] = x_next
        # Check if we need to stop
        if max(divs) < epsilon:
            print(f'Last result: {solution_vector}')
            # It's time to stop!
            break
        else:
            if tmp < 3:
                print(f'Temporary result: {solution_vector}')
                tmp += 1
            print(f'Residual vector: {np.matrix(np.subtract(vector_b,
np.dot(matrix_a, solution_vector)), float)}')
            iterations += 1
    print(f'Iterations: {iterations}')
    return solution_vector


a = [[4.4944, 0.1764, 1.7956, 0.7744],
     [0.1764, 15.6025, 3.4969, 0.1849],
     [1.7956, 3.4969, 8.8804, 0.2116],
     [0.7744, 0.1849, 0.2116, 19.7136]]
b = [31.97212, 9.18339, 19.51289, 51.39451]
print(f'Matrix A:', np.matrix(a))
```

```
print(f'Vector b:', b)
sol = solve_jacobi(a.copy(), b.copy())
sol_np = np.linalg.solve(a, b)
print(f'Our solution: {np.matrix(sol)}')
print(f'Residual vector: {np.matrix(np.subtract(b, np.dot(a, sol)))}')
print(f'NumPy solution: {sol_np}')
print(f'Residual vector for NumPy: {np.matrix(np.subtract(b, np.dot(a,
sol_np)))}')
print('Fault:', round(get_fault(sol, sol_np), 6))
```