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: [[7.113768 0.588585 2.197299 2.607059]]
Residual vector: [[ -6.068202  -9.420647 -15.383357  -6.08268 ]]
Temporary result: [[ 5.763599 -0.015206  0.465017  2.298506]]
Residual vector: [[3.455937 6.352838 4.60105  1.523763]]
Temporary result: [[6.532542 0.391962 0.98313  2.375801]]
Residual vector: [[-1.062005 -1.961723 -2.820895 -0.780387]]
```

Вектори нев’язки на кожній ітерації:

```
Residual vector: [[-1.062005 -1.961723 -2.820895 -0.780387]]
Residual vector: [[0.623214 1.159807 0.872338 0.27345 ]]
Residual vector: [[-0.20024  -0.370532 -0.511862 -0.141912]]
Residual vector: [[0.113261 0.21075  0.164568 0.05109 ]]
Residual vector: [[-0.037665 -0.069728 -0.093033 -0.025934]]
Residual vector: [[0.020618 0.038356 0.030954 0.009533]]
Residual vector: [[-0.007067 -0.013088 -0.016936 -0.004745]]
Residual vector: [[0.003759 0.006991 0.005808 0.001776]]
Residual vector: [[-0.001323 -0.002451 -0.003088 -0.000869]]
Residual vector: [[0.000686 0.001276 0.001087 0.000331]]
Residual vector: [[-0.000247 -0.000458 -0.000564 -0.000159]]
Residual vector: [[0.000125 0.000233 0.000203 0.000061]]
Residual vector: [[-0.000046 -0.000086 -0.000103 -0.000029]]
Residual vector: [[0.000023 0.000043 0.000038 0.000011]]
Residual vector: [[-0.000009 -0.000016 -0.000019 -0.000005]]
Residual vector: [[0.000004 0.000008 0.000007 0.000002]]
```

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

```
Last result: [[6.410855 0.32783  0.716081 2.344463]]
Iterations: 18
Our solution: [[6.410855 0.32783  0.716081 2.344463]]
```

Вектор нев’язки r = b – Ax:

```
Residual vector: [[-0.000002 -0.000003 -0.000003 -0.000001]]
```

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))
```