Zadanie numeryczne NUM4

1 Wstęp

Zadanie polegało na obliczeniu równania $Ay = b$ dla

$$A = \begin{pmatrix} 10 & 8 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 10 & 8 & 1 & \dots & 1 & 1 & 1 & 1 \\ 1 & 1 & 10 & 8 & \dots & 1 & 1 & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 & \dots & 1 & 10 & 8 & 1 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 10 & 8 \\ 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 10 \end{pmatrix}$$

oraz $b \equiv (5, \dots, 5)^T$, gdzie macierz A jest wymiarów 50×50. Należało zaimplementować odpowiedni algorytm, tak, aby program wykonał się w czasie liniowym.

2 Rozwiązanie

Do rozwiązania równiania skorzystałem ze wzoru Shermana-Morrisona. By móc z niego skorzystać należało zapisać macierz A jako $A' + uv^T$. W tym przpadku:

$$A' = \begin{pmatrix} 9 & 7 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 9 & 7 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 7 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & 9 & 7 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 9 & 7 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 9 \end{pmatrix}$$

$u \equiv (1, \dots, 1)^T$ oraz $v \equiv (1, \dots, 1)$. Otrzymana macierz jest wstęgowa i górnotrójkątna. Teraz można skorzystać ze wzoru Shermana-Morrisona:

$$\begin{gathered} y = A^{-1}b \\ y = (A' + uv^T)^{-1}b \\ y = (A'^{-1} - \tfrac{A'^{-1}uv^TA'^{-1}}{1+v^TA'^{-1}u})b \\ y = A'^{-1}b - \tfrac{A'^{-1}uv^TA'^{-1}b}{1+v^TA'^{-1}u} \end{gathered}$$

niech $z = A'^{-1}b$ oraz $q = A'^{-1}u$

$$y = z - \tfrac{v^Tz}{1+v^Tq}q$$

By obliczyć y należało rozwiązać równania:

$$\begin{gathered} A'z = b \\ A'q = u \end{gathered}$$

3 Implementacja

Opisany algorytm zaimplementowałem w języku python. Ponieważ macierz A' jest macierzą wstęgową, przechowuję tylko elementy diagonali i naddiagonali

```
N = 50
A1 = [7 for n in range(N - 1)]  # elementy nad diagonala
A2 = [9 for n in range(N)]      # diagonala
Aprim = [A1, A2]
b = [5 for n in range(N)]       # wektor b
u = [1 for n in range(N)]       # wektor u
v = u.copy()                    # wektor v
```

Macierz A' jest trójkątna górna, więc równania $A'z = b$ i $A'q = u$ rozwiązuję metodą *backward substitution*

```
def solve(vector):
    result = [vector[N - 1] / Aprim[1][N - 1]]
    for n in range(N - 2, -1, -1):
        element = (vector[n + 1] - Aprim[0][n] * result[N - n - 2])
     / Aprim[1][n]
        result.append(element)
    result.reverse()
    return result
```

Iloczyny $v^T z$ oraz $v^T q$ to liczby, do których obliczenia służy funkcja *mul*

```
def mul(vector):
    result = 0
    for n in range(N):
        result += v[n] * vector[n]
    return result
```

4 Wynik

Wynikiem równania macierzowego $Ay = b$ jest wektor

$$y = (0.07525844089350037, 0.07525904117533852, 0.07525826938440369, 0.07525926168703423, 0.07525798586936636, 0.07525962620636797, 0.07525751720165161, 0.07526022877914401, 0.07525674246522518, 0.07526122486883524, 0.07525546177847939, 0.07526287146607977, 0.07525334472487927, 0.07526559339213704, 0.07524984510566277, 0.07527009290255826 0.07524406002083556, 0.07527753086876468, 0.0752344969214272, 0.0752898262822897 2, 0.07521868853260927, 0.07531015135362709, 0.07519255629803279, 0.07534374994093965, 0.07514935811434514, 0.07539929046282379, 0.07507794887192268, 0.07549110234593842, 0.07495990502220382, 0.07564287300986267, 0.07476477131144413, 0.0758937592094108, 0.07444220334059656, 0.07630848945764337, 0.07390897873572605, 0.07699406394961972, 0.07302752581747077, 0.0781273605588052, 0.07157043017708939, 0.08000076923929544, 0.0691617618736019, 0.08309762848663654, 0.06518008569844908, 0.08821692642611872, 0.058598131204829124, 0.09667943934648726, 0.04771775745006959, 0.11066849131689238, 0.029731833488120224, 0.13379325069654147)^T$$

Do sprawdzenia wyniku skorzystałem z biblioteki numerycznej numpy.

```
def np_solution():
    A = np.ones((N, N))
    b = np.ones(N)
    b.fill(5)
    for n in range(N):
        A[n][n] = 10
        if n != (N - 1):
            A[n][n + 1] = 8
    result = np.linalg.solve(A, b)
    return result.tolist()


class TestNUM4(unittest.TestCase):

    def test_result(self):
        np_y = np_solution()
        for n in range(N):
            self.assertAlmostEqual(y[n], np_y[n])
```

Wynik jest zgodny z rozwiązaniem zaimplementowanego algorytmu.