Brandon Swatek - zadanie numeryczne nr 3

2. Zadania numeryczne

3.

$$\mathbf{A} = \begin{bmatrix} -116.66654 & 583.33346 & -333.33308 & 100.00012 & 100.00012 \\ 583.33346 & -116.66654 & -333.33308 & 100.00012 & 100.00012 \\ -333.33308 & -333.33308 & 133.33383 & 200.00025 & 200.00025 \\ 100.00012 & 100.00012 & 200.00025 & 50.000125 & -649.99988 \\ 100.00012 & 100.00012 & 200.00025 & -649.99988 & 50.000125 \end{bmatrix} \tag{5}$$

$$\mathbf{b_1} = \begin{bmatrix} -0.33388066 \\ 1.08033290 \\ -0.98559856 \\ 1.31947922 \\ -0.09473435 \end{bmatrix} \tag{6}$$

$$\mathbf{b_2} = \begin{bmatrix} -0.33388066 \\ 1.0803329 \\ -0.98559855 \\ 1.32655028 \\ -0.10180541 \end{bmatrix} \tag{7}$$

$$\mathbf{b_3} = \begin{bmatrix} 0.72677951 \\ 0.72677951 \\ -0.27849178 \\ 0.96592583 \\ 0.96592583 \end{bmatrix} \tag{8}$$

$$\mathbf{b_4} = \begin{bmatrix} 0.73031505 \\ 0.73031505 \\ -0.27142071 \\ 0.96946136 \\ 0.96946136 \end{bmatrix} \tag{9}$$

$\mathbf{z_i} = \mathbf{A}^{-1}\mathbf{b_i}$ gdzie $i = 1, 2, 3, 4$. Obliczyć: $||\mathbf{b_1} - \mathbf{b_2}||$, $||\mathbf{b_3} - \mathbf{b_4}||$, $||\mathbf{z_1} - \mathbf{z_2}||/||\mathbf{b_1} - \mathbf{b_2}||$, $||\mathbf{z_3} - \mathbf{z_4}||/||\mathbf{b_3} - \mathbf{b_4}||$

Zinterpretować otrzymane wyniki.

Do rozwiązania zadania korzystam z algorytmu Eliminacji Gaussa, który znajduje się poniżej. Eliminacja Gaussa $O(N^3)$ + backsubstitution $O(N^2)$, dające łączną złożoność $O(N^3)$.

In [1]:

```
import numpy as np


def gauss_elimination(matrix_a, vector_b):
    n = len(vector_b)
    vector_z = np.zeros(n)

    # macierz trójkątna górna
    for i in range(0, n):
        for j in range(i+1, n):
            c = matrix_a[j][i] / matrix_a[i][i]
            vector_b[j] = vector_b[j] - c * vector_b[i]
            for k in range(i, n):
                if k == i:
                    matrix_a[j][k] = 0
                else:
                    matrix_a[j][k] = matrix_a[j][k] - c * matrix_a[i][k]

    # otrzymywanie wartości wektora x
    for i in range(n-1, -1, -1):
        local_sum = 0.0
        for j in range(i+1, n):
            if i != j:
                local_sum += matrix_a[i][j] * vector_z[j]
        vector_z[i] = (vector_b[i] - local_sum) / matrix_a[i][i]

    return vector_z
```

Input danych z polecenia - macierz A jest kopią tej z polecenia, z typem np.ndarray. Wektory b1, b2, b3, b4 dla wygody wprowadzam do jednej np.ndarray, jednak nie przeprowadzam na nich działań jak dla jednej macierzy, tylko zawsze izoluję pojedyńcze wektory które mnie interesują.

In [2]:

```
if __name__ == '__main__':
    A = np.array([[-116.66654, 583.33346, -333.33308, 100.00012, 100.00012],
                  [583.33346, -116.66654, -333.33308, 100.00012, 100.00012],
                  [-333.33308, -333.33308, 133.33383, 200.00025, 200.00025],
                  [100.00012, 100.00012, 200.00025, 50.000125, -649.99988],
                  [100.00012, 100.00012, 200.00025, -649.99988, 50.000125]])

    b = np.array([[-0.33388066, 1.08033290, -0.98559856, 1.31947922, -0.09473435],
                  [-0.33388066, 1.0803329, -0.98559855, 1.32655028, -0.10180541],
                  [0.72677951, 0.72677951, -0.27849178, 0.96592583, 0.96592583],
                  [0.73031505, 0.73031505, -0.27142071, 0.96946136, 0.96946136]])
```

Następnie, przechodzę do wykonywania właściwego polecenia. Korzystam z zależności że $z_i = A^{-1}b_i$ (dla i = 1,2,3,4) odpowiada $Az_i = b_i$. Szukając rozwiązania drugiego równania unikam konieczności jawnego wyznaczania macierzy odwrotnej, dzięki czemu złożoność czasowa rozwiązania jest mniejsza. Szukane wektory z_i otrzymuję wspomnianą wyżej metodą Eliminacji Gaussa.

In [3]:

```
    z = np.array([gauss_elimination(np.copy(A), np.copy(b[0])),
                  gauss_elimination(np.copy(A), np.copy(b[1])),
                  gauss_elimination(np.copy(A), np.copy(b[2])),
                  gauss_elimination(np.copy(A), np.copy(b[3]))])

    print("z1: ", z[0], "\n")
    print("z2: ", z[1], "\n")
    print("z3: ", z[2], "\n")
    print("z4: ", z[3], "\n")
```

```
z1:  [ 1.98285955e-03 -3.74455354e-05 -2.19649469e-04  2.40550981e-04
 -1.77975410e-03]

z2:  [ 1.98536896e-03 -3.49361303e-05 -2.14630646e-04  2.53161908e-04
 -1.78734621e-03]

z3:  [354.8851814  354.8851814  709.76819773 354.88343244 354.88343244]

z4:  [358.43402458 358.43402458 716.86588409 358.43227555 358.43227555]
```

Ostatnią częścią treści polecenia jest wyznaczenie odpowiednich norm. Normy wyznaczam korzystając z funkcji numpy.linalg.norm pochodzącej z pythonowego modułu numpy. Funkcja ta bez podawania argumentów innych niż pierwszy, w przypadku wektorów wykorzystuje normę euklidesową.

In [4]:

```
    norm1 = np.linalg.norm(b[0]-b[1])
    norm2 = np.linalg.norm(b[2]-b[3])
    norm3 = np.linalg.norm(z[0]-z[1]) / np.linalg.norm(b[0]-b[1])
    norm4 = np.linalg.norm(z[2]-z[3]) / np.linalg.norm(b[2]-b[3])

    print("||b1 - b2|| = ", norm1, "\n")
    print("||b3 - b4|| = ", norm2, "\n")
    print("||z1 - z2|| / ||b1 - b2|| = ", norm3, "\n")
    print("||z3 - z4|| / ||b3 - b4|| = ", norm4, "\n")
```

```
||b1 - b2|| =  0.009999988952358834

||b3 - b4|| =  0.010000003094494577

||z1 - z2|| / ||b1 - b2|| =  0.0015951766717637163

||z3 - z4|| / ||b3 - b4|| =  1003.7641154377923
```