Brandon Swatek - zadanie numeryczne nr 5

1. Zadania numeryczne

5. Macierz $\mathbf{A} \in \mathbb{R}^{64\times 64}$ ma strukturę:

$$\begin{bmatrix} 5 & 2 & 1 & 1 & 2 & 1 & 1 & \dots \\ 2 & 5 & 2 & 1 & 1 & 2 & 1 & \dots \\ 1 & 2 & 5 & 2 & 1 & 1 & 2 & \dots \\ 1 & 1 & 2 & 5 & 2 & 1 & 1 & \dots \\ 2 & 1 & 1 & 2 & 5 & 2 & 1 & \dots \\ 1 & 2 & 1 & 1 & 2 & 5 & 2 & \dots \\ 1 & 1 & 2 & 1 & 1 & 2 & 5 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \tag{1}$$

Za pomocą odpowiedniego algorytmu rozwiązać układ równań $\mathbf{Ax} = \mathbf{e}$ gdzie $\mathbf{e}$ jest wektorem którego wszystkie składowe są równe 1 ($e_i = 1, i = 1, .., 64$).

Do rozwiązania zadania numerycznego wybieram metodę Shermana-Morrisona.

In [1]:
```
import numpy as np


def sherman_morrison(matrix_a, martix_u, matrix_v, vector_b):
    vector_u = np.array(martix_u[:, :1])
    vector_vt = np.array(matrix_v.T[0])
    vector_z = gauss_seidel(np.copy(matrix_a), np.zeros(len(vector_v
t)), np.copy(vector_b), 42)
    vector_q = gauss_seidel(np.copy(matrix_a), np.zeros(len(vector_v
t)), np.copy(vector_u.T[0]), 42)
    return vector_z - vector_q * (np.dot(vector_vt, vector_z) / (1 + n
p.dot(vector_vt, vector_q)))
```

Metoda ta ma złożoność $O(N)$.

In [2]:
```
def gauss_seidel(matrix_a, vector_x, vector_b, iterations):
    n = len(matrix_a)
    for k in range(iterations):
        for i in range(n):
            temp = vector_b[i]
            for j in range(n):
                if i != j and matrix_a[i][j] != 0:
                    temp -= matrix_a[i][j] * vector_x[j]
            vector_x[i] = temp / matrix_a[i][i]
    return vector_x
```

Właściwy algorytm wymaga też metody pomocniczej, którą jest tutaj metoda Gauss-Seidela. Jest ona zoptymalizowana dla macierzy rzadkich, tj. nie wykonuje ona mnożenia przez zera. Dzięki temu osiągamy złożoność czasową $O(kN)$. Parametr k czyli liczbę iteracji ustawiam na 42. Algorytm wykonuję w programie dwa razy. Podsumowywując, korzystając z powyższych metod rozwiązujemy zadanie numeryczne ze złożonością proporcjonalną do liniowej.

In [3]:
```
def create_a(n):
    matrix_a = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if i == j:
                matrix_a[i][j] = 4
            elif (i - j) == 1 or (j - i) == 1 or (j - i) == 4 or (i -
j) == 4:
                matrix_a[i][j] = 1
            else:
                matrix_a[i][j] = 0
    return matrix_a


e = np.ones(64)
A = create_a(64)
u = np.zeros((64, 64))
u[:, 0] = 1
v = np.copy(u)

print(sherman_morrison(A, u, v, e))

A = np.array(A+(u@v.T))
print(np.linalg.solve(A, e))
```

```
[0.0211007  0.01422043 0.0159436  0.0176289  0.00998803 0.01468523
 0.01298753 0.01216405 0.01524433 0.01267437 0.01386826 0.01409431
 0.0128075  0.01411595 0.01338201 0.01339422 0.01392666 0.01328358
 0.01370479 0.01363143 0.01341932 0.01372955 0.0134951  0.01356724
 0.01364634 0.01349509 0.01362929 0.01356946 0.01354428 0.01362574
 0.01353447 0.01359261 0.01359261 0.01353447 0.01362574 0.01354428
 0.01356946 0.01362929 0.01349509 0.01364634 0.01356723 0.0134951
 0.01372954 0.01341932 0.01363143 0.01370478 0.01328358 0.01392666
 0.01339421 0.01338202 0.01411595 0.01280751 0.01409431 0.01386826
 0.01267438 0.01524433 0.01216405 0.01298753 0.01468522 0.00998803
 0.0176289  0.0159436  0.01422044 0.0211007 ]
[0.0211007  0.01422044 0.0159436  0.0176289  0.00998803 0.01468522
 0.01298754 0.01216405 0.01524432 0.01267438 0.01386826 0.01409431
 0.01280751 0.01411594 0.01338202 0.01339421 0.01392666 0.01328359
 0.01370478 0.01363143 0.01341933 0.01372954 0.01349511 0.01356723
 0.01364633 0.0134951  0.01362928 0.01356947 0.01354428 0.01362573
 0.01353448 0.01359261 0.01359261 0.01353448 0.01362573 0.01354428
 0.01356947 0.01362928 0.0134951  0.01364633 0.01356723 0.01349511
 0.01372954 0.01341933 0.01363143 0.01370478 0.01328359 0.01392666
 0.01339421 0.01338202 0.01411594 0.01280751 0.01409431 0.01386826
 0.01267438 0.01524432 0.01216405 0.01298754 0.01468522 0.00998803
 0.0176289  0.0159436  0.01422044 0.0211007 ]
```

Dla porównania zamieszczam też wyniki uzyskane przy pomocy numpy.linalg.solve