Brandon Swatek - zadanie numeryczne nr 2

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


def thomas(a, b, c, d):
    nf = len(d)
    ac, bc, cc, dc = map(np.array, (a, b, c, d))
    for it in range(1, nf):
        mc = ac[it - 1] / bc[it - 1]
        bc[it] = bc[it] - mc * cc[it - 1]
        dc[it] = dc[it] - mc * dc[it - 1]
    xc = bc
    xc[-1] = dc[-1] / bc[-1]
    for il in range(nf - 2, -1, -1):
        xc[il] = (dc[il] - cc[il] * xc[il + 1]) / bc[il]
    return xc


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 = thomas(matrix_A.diagonal(-1), matrix_A.diagonal(), matri
x_A.diagonal(1), vector_b)
    vector_q = thomas(matrix_A.diagonal(-1), matrix_A.diagonal(), matri
x_A.diagonal(1), vector_u)

    return vector_z - vector_q * (np.dot(vector_vt, vector_z) / (1 + n
p.dot(vector_vt, vector_q)))
```

Do rozwiązania zadania wykorzystuję algorytm Shermana-Morrisona, przy wyliczaniu wektorów pomocniczych u oraz z wspomagam się algorytmem Thomasa. Obydwa algorytmy mają złożoność obliczeniową / czasową O(N), więc po działaniach wykonywanych powyżej mamy O(3N).

In [2]:
```
    b = np.array([1, 2, 3, 4, 5, 6, 7], float)

    A = np.array([[3, 1, 0, 0, 0, 0, 0],
                  [1, 4, 1, 0, 0, 0, 0],
                  [0, 1, 4, 1, 0, 0, 0],
                  [0, 0, 1, 4, 1, 0, 0],
                  [0, 0, 0, 1, 4, 1, 0],
                  [0, 0, 0, 0, 1, 4, 1],
                  [0, 0, 0, 0, 0, 1, 3]], float)
    v = np.array([[1, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0]], float)
    u = np.copy(v)

    print(A+(u@v.T))
```

```
[[4. 1. 0. 0. 0. 0. 1.]
 [1. 4. 1. 0. 0. 0. 0.]
 [0. 1. 4. 1. 0. 0. 0.]
 [0. 0. 1. 4. 1. 0. 0.]
 [0. 0. 0. 1. 4. 1. 0.]
 [0. 0. 0. 0. 1. 4. 1.]
 [1. 0. 0. 0. 0. 1. 4.]]
```

Jako dane wejściowe wprowadzam wektor b oraz macierz z polecenia. Macierz A to macierz z polecenia z odjętym produktem wektorów u i v, które mam wyizolowane ze względu na konieczność tego działania przy alg. Shermana-Morissona.

In [3]:
```
    x = sherman_morrison(A, u, v, b)
    print(x)
```

```
[-0.2601626   0.44715447  0.47154472  0.66666667  0.86178862  0.8861788
6
  1.59349593]
```

Podczas sprawdzania wyników korzystając z alternatywnego algorytmu (Eliminacji Gaussa o $O(N^3)$) otrzymuję te same wyniki.

In [4]:
```
def gauss(matrix_a, vector_d):

    vector_x = np.array([0, 0, 0, 0, 0, 0, 0], float)
    n = len(vector_x)
    for i in range(0, n):
        for j in range(i+1, n):
            c = matrix_a[j][i] / matrix_a[i][i]
            vector_d[j] = vector_d[j] - c * vector_d[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]
    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_x[j]
        vector_x[i] = (vector_d[i] - local_sum) / matrix_a[i][i]

    return vector_x
```

In [5]:
```
    x = gauss(A+(u@v.T), b)
    print(x)
```

```
[-0.2601626   0.44715447  0.47154472  0.66666667  0.86178862  0.8861788
6
  1.59349593]
```