

|       |                 |                                      |                       |
|-------|-----------------|--------------------------------------|-----------------------|
| MO_02 | Romaniak Hubert | Informatyka<br>niestacjonarna II rok | Semestr letni 2023/24 |
|-------|-----------------|--------------------------------------|-----------------------|

## Zadanie 1

Dekompozycja LU, eliminacja w przód i podstawianie wstecz w języku Python, używając biblioteki *numpy*

```

1. def lower_upper_decomposition(A):
2.     n = A.shape[0]
3.     a = A.copy()
4.     for k in range(n - 1):
5.         akk = a[k][k]
6.         for i in range(k + 1, n):
7.             aux = a[i][k] / akk if akk else 0
8.             for j in range(k + 1, n):
9.                 a[i][j] -= a[k][j] * aux
10.            a[i][k] = aux
11.     U = np.triu(a)
12.     L = a - U
13.     return L, U
14.
15. def eliminate_forward(L, B):
16.     n = L.shape[0]
17.     b = B.copy()
18.     for k in range(n - 1):
19.         for i in range(k + 1, n):
20.             b[i] -= b[k] * L[i][k]
21.     return b
22.
23. def substitute_backward(U, Y):
24.     n = U.shape[0]
25.     y = Y.copy()
26.     y[n-1] /= U[n-1][n-1]
27.     for i in range(n-2, -1, -1):
28.         s = 0
29.         for j in range(i+1, n):
30.             s += U[i][j] * y[j]
31.         y[i] -= s
32.         y[i] /= U[i][i]
33.     return y

```

Układ równań do rozwiązania

$$\begin{bmatrix} 20 & 3,17 & -2,47 & 0,19 \\ 3,17 & 20 & 3,17 & -2,47 \\ -2,47 & 3,17 & 20 & 3,17 \\ 0,19 & -2,47 & 3,17 & 20 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1,12 \\ 0,82 \\ -4,74 \\ 1,62 \end{bmatrix}$$

Szukane:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = ?$$

Rozwiązanie i jego dokładność dla każdego elementu

$$\vec{x} \approx \begin{bmatrix} -0,113108 \\ 0,123168 \\ -0,293279 \\ 0,143770 \end{bmatrix} \quad \Delta \vec{x} \approx \begin{bmatrix} 0 \\ -3,330669 \cdot 10^{-16} \\ 0 \\ -8,881784 \cdot 10^{-16} \end{bmatrix}$$

## Macierze L i U

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0,1585 & 1 & 0 & 0 \\ -0,1235 & 0,182664 & 1 & 0 \\ 0,0095 & -0,128227 & 0,191665 & 1 \end{bmatrix}$$
$$U = \begin{bmatrix} 20 & 3,170 & -2,47 & 0,19 \\ 0 & 19,497555 & 3,561495 & -2,50115 \\ 0 & 0 & 19,044399 & 3,650145 \\ 0 & 0 & 0 & 18,978007 \end{bmatrix}$$

Po pomnożeniu macierzy  $L \cdot U$  otrzymujemy w rozwiązaniu macierz A.

## Zadanie 2

Metody iteracyjne Jacobiego i Gaussa-Seidela w języku Python, używając biblioteki *numpy*

```
1. def jacobi(A, B):
2.     D = np.diag(np.diag(A))
3.     D_inv = nla.inv(D)
4.     M = (- D_inv) @ (A - D)
5.     print(f'\t{M = }')
6.     print(f'\t||M|| = {norm_inf(M):.6f}')
7.     C = D_inv @ B
8.     x = np.zeros(B.size)
9.     errors = []
10.    while True:
11.        new_x = M @ x + C
12.        errors.append(error := norm_inf(x - new_x))
13.        if error <= 1e-7:
14.            return new_x, np.array(errors)
15.        x = new_x
16.
17. def gauss_seidel(A, B):
18.     L = np.tril(A) - np.diag(np.diag(A))
19.     DU_inv = nla.inv(A - L)
20.     M = (- DU_inv) @ L
21.     print(f'\t{M = }')
22.     print(f'\t||M|| = {norm_inf(M):.6f}')
23.     C = DU_inv @ B
24.     x = np.zeros(B.size)
25.     errors = []
26.     while True:
27.         new_x = M @ x + C
28.         errors.append(error := norm_inf(x - new_x))
29.         if error <= 1e-7:
30.             return new_x, np.array(errors)
31.         x = new_x
32.
33. def norm_inf(a):
34.     return np.abs(a).sum(axis=len(a.shape)-1).max()
```

Układ równań do rozwiązania

$$\begin{bmatrix} 20 & 3,17 & -2,47 & 0,19 \\ 3,17 & 20 & 3,17 & -2,47 \\ -2,47 & 3,17 & 20 & 3,17 \\ 0,19 & -2,47 & 3,17 & 20 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1,12 \\ 0,82 \\ -4,74 \\ 1,62 \end{bmatrix}$$

Szukane:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = ?$$

Warunek przzerwania:

$$\|\vec{x}_{n+1} - \vec{x}_n\|_{\infty} \leq 10^{-7}$$

Rozwiązanie i jego dokładność dla każdego elementu

Metoda Jacobiego

$$\vec{x} \approx \begin{bmatrix} -0,113108 \\ 0,123168 \\ -0,293279 \\ 0,143770 \end{bmatrix}$$

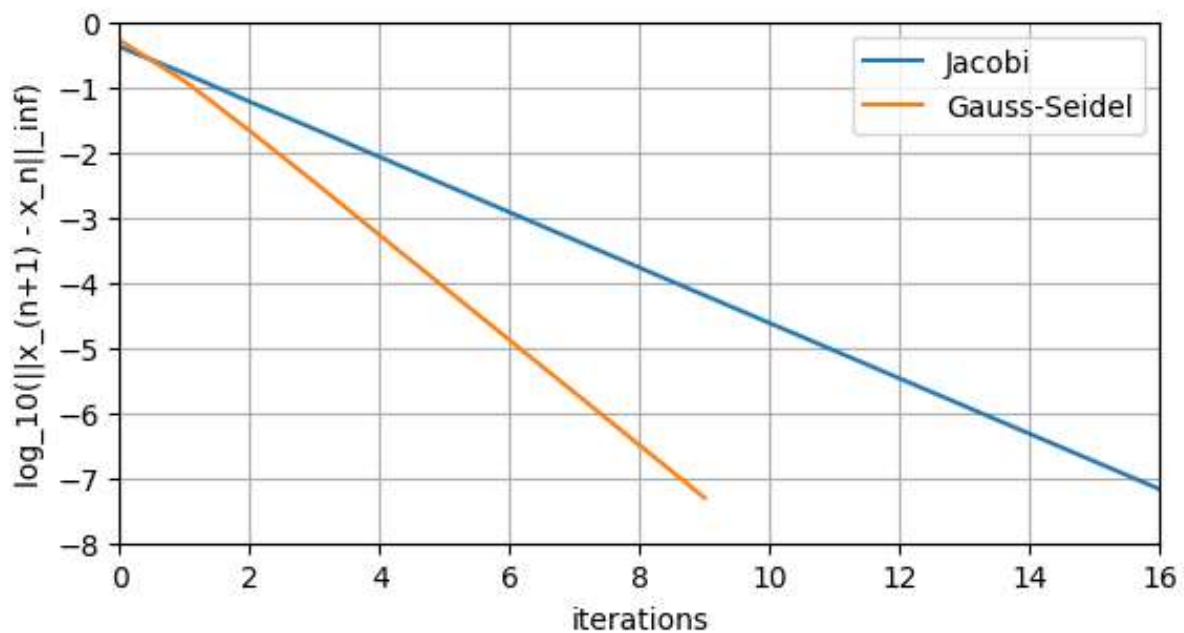
$$\Delta \vec{x} \approx \begin{bmatrix} -1,096239 \cdot 10^{-7} \\ 1,423434 \cdot 10^{-7} \\ -1,423432 \cdot 10^{-7} \\ 1,096239 \cdot 10^{-7} \end{bmatrix}$$

Metoda Gaussa-Seidela

$$\vec{x} \approx \begin{bmatrix} -0,113108 \\ 0,123168 \\ -0,293279 \\ 0,143770 \end{bmatrix}$$

$$\Delta \vec{x} \approx \begin{bmatrix} 0 \\ 1,090945 \cdot 10^{-8} \\ -3,590197 \cdot 10^{-8} \\ 6,972243 \cdot 10^{-8} \end{bmatrix}$$

Wykresy zbieżności metod  $f(n) = \log_{10} \|\vec{x}_{n+1} - \vec{x}_n\|_{\infty}$



Rząd zbieżności metod  $p$

$$p_{\text{jacobi}} = -0,399501$$

$$p_{\text{gauss-seidel}} = -0,702433$$

## Macierze M oraz ich normy zgodne $\|M\|_\infty$

### Metoda Jacobiego

$$M = \begin{bmatrix} 0 & -0,1585 & 0,1235 & -0,0095 \\ -0,1585 & 0 & -0,1585 & 0,1235 \\ 0,1235 & -0,1585 & 0 & -0,1585 \\ -0,0095 & 0,1235 & -0,1585 & 0 \end{bmatrix} \quad \|M\|_\infty = 0,4405$$

### Metoda Gaussa-Seidela

$$M = \begin{bmatrix} 0,043977 & -0,030057 & 0,008342 & 0 \\ -0,179487 & 0,043977 & -0,023557 & 0 \\ 0,125006 & -0,178075 & 0,025122 & 0 \\ -0,0095 & 0,1235 & -0,1585 & 0 \end{bmatrix} \quad \|M\|_\infty = 0,328203$$

## Appendix

### ex\_1.py

```
1. import numpy as np
2.
3. np.set_printoptions(precision=6, floatmode='fixed')
4.
5. def print_array(name, array):
6.     array_string = str(array).replace('\n', '\n' + ' ' * (len(name) + 3))
7.     print(name, '=', array_string)
8.
9. def lower_upper_decomposition(A):
10.     n = A.shape[0]
11.     a = A.copy()
12.     for k in range(n - 1):
13.         akk = a[k][k]
14.         for i in range(k + 1, n):
15.             aux = a[i][k] / akk if akk else 0
16.             for j in range(k + 1, n):
17.                 a[i][j] -= a[k][j] * aux
18.             a[i][k] = aux
19.     U = np.triu(a)
20.     L = a - U
21.     return L, U
22.
23. def eliminate_forward(L, B):
24.     n = L.shape[0]
25.     b = B.copy()
26.     for k in range(n - 1):
27.         for i in range(k + 1, n):
28.             b[i] -= b[k] * L[i][k]
29.     return b
30.
31. def substitute_backward(U, Y):
32.     n = U.shape[0]
33.     y = Y.copy()
34.     y[n-1] /= U[n-1][n-1]
35.     for i in range(n-2, -1, -1):
36.         s = 0
37.         for j in range(i+1, n):
38.             s += U[i][j] * y[j]
39.         y[i] -= s
40.         y[i] /= U[i][i]
41.     return y
42.
43. if __name__ == '__main__':
44.     data = np.loadtxt('data.txt', dtype=np.float64)
45.     A = data[:-1]
46.     B = data[-1]
47.
48.     L, U = lower_upper_decomposition(A)
```

```

49.     L += np.eye(A.shape[0])
50.
51.     print_array('L', L)
52.     print_array('U', U)
53.     print(f'{np.allclose(A, L @ U, atol=1e-6) = }')
54.
55.     y = eliminate_forward(L, B)
56.     x = substitute_backward(U, y)
57.     print(f'solution = {x}')
58.
59.     B_check = A @ x
60.     print_array('B_check', B_check)
61.     print_array('B_origin', B)
62.
63.     error = B - B_check
64.     print(f'error = {error}')

```

ex\_2.py

```

1. import matplotlib.pyplot as plt
2. import numpy as np
3. import numpy.linalg as nla
4.
5. np.set_printoptions(precision=6, floatmode='fixed')
6.
7. def print_array(name, array):
8.     array_string = str(array).replace('\n', '\n' + ' ' * (len(name) + 3))
9.     print(name, '=', array_string)
10.
11. def jacobi(A, B):
12.     D = np.diag(np.diag(A))
13.     D_inv = nla.inv(D)
14.     M = (- D_inv) @ (A - D)
15.     print_array('M', M)
16.     print(f'\t||M|| = {norm_inf(M):.6f}')
17.     C = D_inv @ B
18.     x = np.zeros(B.size)
19.     errors = []
20.     while True:
21.         new_x = M @ x + C
22.         errors.append(error := norm_inf(x - new_x))
23.         if error <= 1e-7:
24.             return new_x, np.array(errors)
25.         x = new_x
26.
27. def gauss_seidel(A, B):
28.     L = np.tril(A) - np.diag(np.diag(A))
29.     DU_inv = nla.inv(A - L)
30.     M = (- DU_inv) @ L
31.     print_array('M', M)
32.     print(f'\t||M|| = {norm_inf(M):.6f}')
33.     C = DU_inv @ B
34.     x = np.zeros(B.size)
35.     errors = []
36.     while True:
37.         new_x = M @ x + C
38.         errors.append(error := norm_inf(x - new_x))
39.         if error <= 1e-7:
40.             return new_x, np.array(errors)
41.         x = new_x
42.
43. def norm_inf(a):
44.     return np.abs(a).sum(axis=len(a.shape)-1).max()
45.
46. if __name__ == '__main__':
47.     data = np.loadtxt('data.txt', dtype=np.float64)
48.     A = data[:-1]
49.     B = data[-1]
50.
51.     print('JACOBI')

```

```

52. x, jacobi_errors = jacobi(A, B)
53. jacobi_errors = np.log10(jacobi_errors)
54. errors_slope = (jacobi_errors[-1] - jacobi_errors[0]) / jacobi_errors.size
55. print(f'\torder of convergence = {errors_slope:.6f}')
56. B_check = A @ x
57. B_error = B - B_check
58. print_array('\tx', x)
59. print_array('\tB check', B_check)
60. print_array('\tB orgin', B)
61. print_array('\tB error', B_error)
62. print()
63.
64. print('GAUSS-SEIDEL')
65. x, gauss_seidel_errors = gauss_seidel(A, B)
66. gauss_seidel_errors = np.log10(gauss_seidel_errors)
67. errors_slope = (gauss_seidel_errors[-1] - gauss_seidel_errors[0]) /
gauss_seidel_errors.size
68. print(f'\torder of convergence = {errors_slope:.6f}')
69. B_check = A @ x
70. B_error = B - B_check
71. print_array('\tx', x)
72. print_array('\tB check', B_check)
73. print_array('\tB orgin', B)
74. print_array('\tB error', B_error)
75. print()
76.
77. fig = plt.figure()
78. ax = fig.add_subplot()
79. ax.set_xlabel('iterations')
80. ax.set_ylabel('log10(||x(n+1) - xn||inf)')
81. ax.set_xlim(0, 16)
82. ax.set_ylim(-8, 0)
83. ax.plot(range(jacobi_errors.size), jacobi_errors, label = 'Jacobi')
84. ax.plot(range(gauss_seidel_errors.size), gauss_seidel_errors, label = 'Gauss-Seidel')
85. ax.grid()
86. ax.legend()
87. ax.set_aspect('equal')
88. fig.show()

```