MO_02	Romaniak Hubert	Informatyka niestacjonarna II rok	Semestr letni 2023/24

Zadanie 1

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

```python
def lower_upper_decomposition(A):
    n = A.shape[0]
    a = A.copy()
    for k in range(n - 1):
        akk = a[k][k]
        for i in range(k + 1, n):
            aux = a[i][k] / akk if akk else 0
            for j in range(k + 1, n):
                a[i][j] -= a[k][j] * aux
            a[i][k] = aux
    U = np.triu(a)
    L = a - U
    return L, U

def eliminate_forward(L, B):
    n = L.shape[0]
    b = B.copy()
    for k in range(n - 1):
        for i in range(k + 1, n):
            b[i] -= b[k] * L[i][k]
    return b

def substitute_backward(U, Y):
    n = U.shape[0]
    y = Y.copy()
    y[n-1] /= U[n-1][n-1]
    for i in range(n-2, -1, -1):
        s = 0
        for j in range(i+1, n):
            s += U[i][j] * y[j]
        y[i] -= s
        y[i] /= U[i][i]
    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

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

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*

```python
def jacobi(A, B):
    D = np.diag(np.diag(A))
    D_inv = nla.inv(D)
    M = (- D_inv) @ (A - D)
    print(f'\t{M = }')
    print(f'\t||M|| = {norm_inf(M):.6f}')
    C = D_inv @ B
    x = np.zeros(B.size)
    errors = []
    while True:
        new_x = M @ x + C
        errors.append(error := norm_inf(x - new_x))
        if error <= 1e-7:
            return new_x, np.array(errors)
        x = new_x

def gauss_seidel(A, B):
    L = np.tril(A) - np.diag(np.diag(A))
    DU_inv = nla.inv(A - L)
    M = (- DU_inv) @ L
    print(f'\t{M = }')
    print(f'\t||M|| = {norm_inf(M):.6f}')
    C = DU_inv @ B
    x = np.zeros(B.size)
    errors = []
    while True:
        new_x = M @ x + C
        errors.append(error := norm_inf(x - new_x))
        if error <= 1e-7:
            return new_x, np.array(errors)
        x = new_x

def norm_inf(a):
    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 przerwania:

$$\|\overrightarrow{x_{n+1}} - \overrightarrow{x_n}\|_\infty \leq 10^{-7}$$

Rozwiązanie i jego dokładność dla każdego elementu

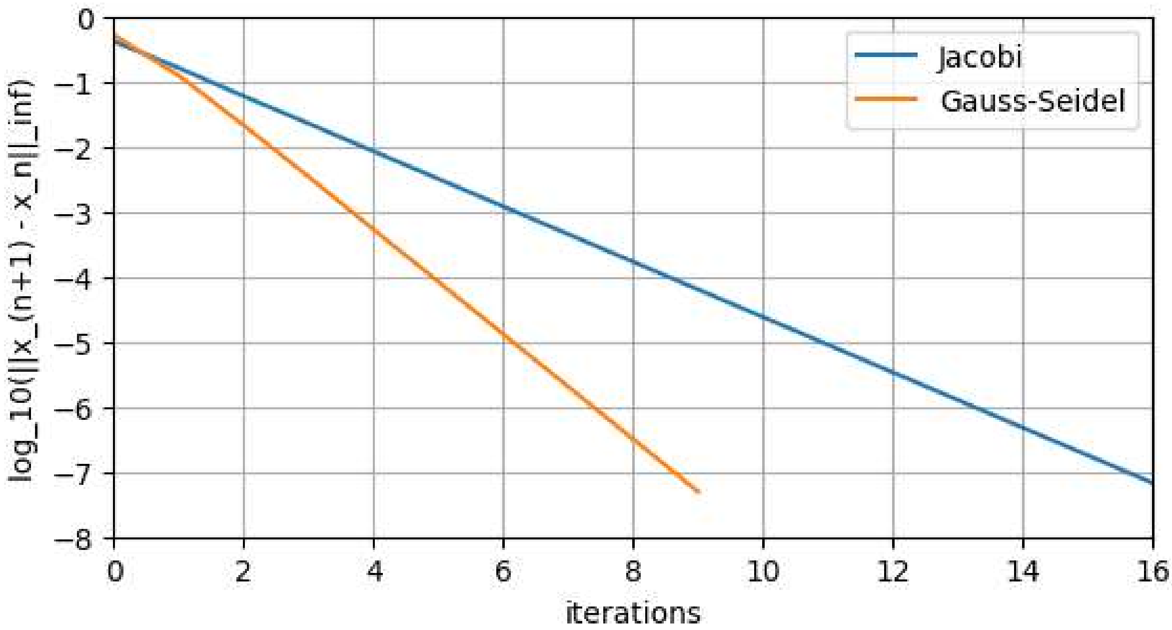
Metoda Jacobiego

$$\vec{x} \approx \begin{bmatrix} -0{,}113108 \\ 0{,}123168 \\ -0{,}293279 \\ 0{,}143770 \end{bmatrix} \qquad \Delta\vec{x} \approx \begin{bmatrix} -1{,}096239 \cdot 10^{-7} \\ 1{,}423434 \cdot 10^{-7} \\ -1{,}423432 \cdot 10^{-7} \\ 1{,}096239 \cdot 10^{-7} \end{bmatrix}$$

Metoda Gaussa-Seidela

$$\vec{x} \approx \begin{bmatrix} -0{,}113108 \\ 0{,}123168 \\ -0{,}293279 \\ 0{,}143770 \end{bmatrix} \qquad \Delta\vec{x} \approx \begin{bmatrix} 0 \\ 1{,}090945 \cdot 10^{-8} \\ -3{,}590197 \cdot 10^{-8} \\ 6{,}972243 \cdot 10^{-8} \end{bmatrix}$$

Wykresy zbieżności metod $f(n) = \log_{10}\|\overrightarrow{x_{n+1}} - \overrightarrow{x_n}\|_\infty$



Rząd zbieżności metod p

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

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

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

Metoda Jacobiego

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

Metoda Gaussa-Seidela

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

Appendix

ex_1.py

```python
import numpy as np

np.set_printoptions(precision=6, floatmode='fixed')

def print_array(name, array):
    array_string = str(array).replace('\n', '\n' + ' ' * (len(name) + 3))
    print(name, '=', array_string)

def lower_upper_decomposition(A):
    n = A.shape[0]
    a = A.copy()
    for k in range(n - 1):
        akk = a[k][k]
        for i in range(k + 1, n):
            aux = a[i][k] / akk if akk else 0
            for j in range(k + 1, n):
                a[i][j] -= a[k][j] * aux
            a[i][k] = aux
    U = np.triu(a)
    L = a - U
    return L, U

def eliminate_forward(L, B):
    n = L.shape[0]
    b = B.copy()
    for k in range(n - 1):
        for i in range(k + 1, n):
            b[i] -= b[k] * L[i][k]
    return b

def substitute_backward(U, Y):
    n = U.shape[0]
    y = Y.copy()
    y[n-1] /= U[n-1][n-1]
    for i in range(n-2, -1, -1):
        s = 0
        for j in range(i+1, n):
            s += U[i][j] * y[j]
        y[i] -= s
        y[i] /= U[i][i]
    return y

if __name__ == '__main__':
    data = np.loadtxt('data.txt', dtype=np.float64)
    A = data[:-1]
    B = data[-1]

    L, U = lower_upper_decomposition(A)
```

```python
    L += np.eye(A.shape[0])

    print_array('L', L)
    print_array('U', U)
    print(f'{np.allclose(A, L @ U, atol=1e-6) = }')

    y = eliminate_forward(L, B)
    x = substitute_backward(U, y)
    print(f'solution = {x}')

    B_check = A @ x
    print_array('B_check', B_check)
    print_array('B_orgin', B)

    error = B - B_check
    print(f'error = {error}')
```

ex_2.py

```python
import matplotlib.pyplot as plt
import numpy as np
import numpy.linalg as nla

np.set_printoptions(precision=6, floatmode='fixed')

def print_array(name, array):
    array_string = str(array).replace('\n', '\n' + ' ' * (len(name) + 3))
    print(name, '=', array_string)

def jacobi(A, B):
    D = np.diag(np.diag(A))
    D_inv = nla.inv(D)
    M = (- D_inv) @ (A - D)
    print_array('    M', M)
    print(f'\t||M|| = {norm_inf(M):.6f}')
    C = D_inv @ B
    x = np.zeros(B.size)
    errors = []
    while True:
        new_x = M @ x + C
        errors.append(error := norm_inf(x - new_x))
        if error <= 1e-7:
            return new_x, np.array(errors)
        x = new_x

def gauss_seidel(A, B):
    L = np.tril(A) - np.diag(np.diag(A))
    DU_inv = nla.inv(A - L)
    M = (- DU_inv) @ L
    print_array('    M', M)
    print(f'\t||M|| = {norm_inf(M):.6f}')
    C = DU_inv @ B
    x = np.zeros(B.size)
    errors = []
    while True:
        new_x = M @ x + C
        errors.append(error := norm_inf(x - new_x))
        if error <= 1e-7:
            return new_x, np.array(errors)
        x = new_x

def norm_inf(a):
    return np.abs(a).sum(axis=len(a.shape)-1).max()

if __name__ == '__main__':
    data = np.loadtxt('data.txt', dtype=np.float64)
    A = data[:-1]
    B = data[-1]

    print('JACOBI')
```

```python
    x, jacobi_errors = jacobi(A, B)
    jacobi_errors = np.log10(jacobi_errors)
    errors_slope = (jacobi_errors[-1] - jacobi_errors[0]) / jacobi_errors.size
    print(f'\torder of convergence = {errors_slope:.6f}')
    B_check = A @ x
    B_error = B - B_check
    print_array('\tx', x)
    print_array('\tB check', B_check)
    print_array('\tB orgin', B)
    print_array('\tB error', B_error)
    print()

    print('GAUSS-SEIDEL')
    x, gauss_seidel_errors = gauss_seidel(A, B)
    gauss_seidel_errors = np.log10(gauss_seidel_errors)
    errors_slope = (gauss_seidel_errors[-1] - gauss_seidel_errors[0]) / gauss_seidel_errors.size
    print(f'\torder of convergence = {errors_slope:.6f}')
    B_check = A @ x
    B_error = B - B_check
    print_array('\tx', x)
    print_array('\tB check', B_check)
    print_array('\tB orgin', B)
    print_array('\tB error', B_error)
    print()

    fig = plt.figure()
    ax = fig.add_subplot()
    ax.set_xlabel('iterations')
    ax.set_ylabel('log_10(||x_(n+1) - x_n||_inf)')
    ax.set_xlim(0, 16)
    ax.set_ylim(-8, 0)
    ax.plot(range(jacobi_errors.size), jacobi_errors, label = 'Jacobi')
    ax.plot(range(gauss_seidel_errors.size), gauss_seidel_errors, label = 'Gauss-Seidel')
    ax.grid()
    ax.legend()
    ax.set_aspect('equal')
    fig.show()
```