Laboratorium 9

Układy równań liniowych – metody iteracyjne

Bartłomiej Szubiak

7.05.2024

Zad1:

Dany jest układ równań liniowych Ax=b. Macierz A o wymiarze nxn jest określona wzorem:

$$A = \begin{bmatrix} 1 & \frac{1}{2} & 0 & \dots & \dots & 0 \\ \frac{1}{2} & 2 & \frac{1}{3} & 0 & \dots & 0 \\ 0 & \frac{1}{3} & 2 & \frac{1}{4} & 0 \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \frac{1}{n-1} & 2 & \frac{1}{n} \\ 0 & \dots & \dots & 0 & \frac{1}{n} & 1 \end{bmatrix}$$

Przyjmij wektor x jako dowolną n-elementową permutację ze zbioru {-1, 0} i oblicz wektor b (operując na wartościach wymiernych).
Metodą Jacobiego oraz metodą Czebyszewa rozwiąż układ równań liniowych Ax=b (przyjmując jako niewiadomą wektor x).
W obu przypadkach oszacuj liczbę iteracji przyjmując test stopu:

$$\|x^{(t+1)} - x^{(t)}\| < \rho$$

$$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\| < \rho.$$

Kod Python z użyciem m.in. biblioteki numpy:

1. Generowanie danych:

```
def generate_b(A, x):
    return np.dot(A, x)

def generate_random_x(n):
    return np.random.choice([-1, 0], size=n)

def create_matrix(n:int):
    A = [[0 for _ in range(n)] for __ in range(n)]
    # przekatna glowna
    for i in range(n):
        if i in [0,n-1]:
            A[i][i] = 1
        else: A[i][i] = 2

    #przekatne po bokach
    for i in range(n):
        try:
```

```
            A[i-1][i] = 1/(i+2)
            A[i+1][i] = 1/(i+3)
        except:
            None

    return A
```

2. Metoda Jacobiego:

```
def jacobi_method(A, b, q, max_iterations=100): #-> x_new , iter
    n = len(A)
    x = np.zeros(n)
    D = np.diag(np.diag(A))
    R = A - D
    D_inv = np.linalg.inv(np.diag(np.diag(D)))

    for iter in range(max_iterations):
        x_new = np.dot(D_inv, b - np.dot(R, x))
        if np.linalg.norm(x_new - x) < q or np.linalg.norm(b - np.dot(A, x_new)) / np.linalg.norm(b) < q:
            return x_new , iter

        # distance = np.linalg.norm(x - x_new)
        # print("Odległość między x i new_x:", distance)
        x = x_new
    return x , iter
```

3. Metoda Czebyszewa:

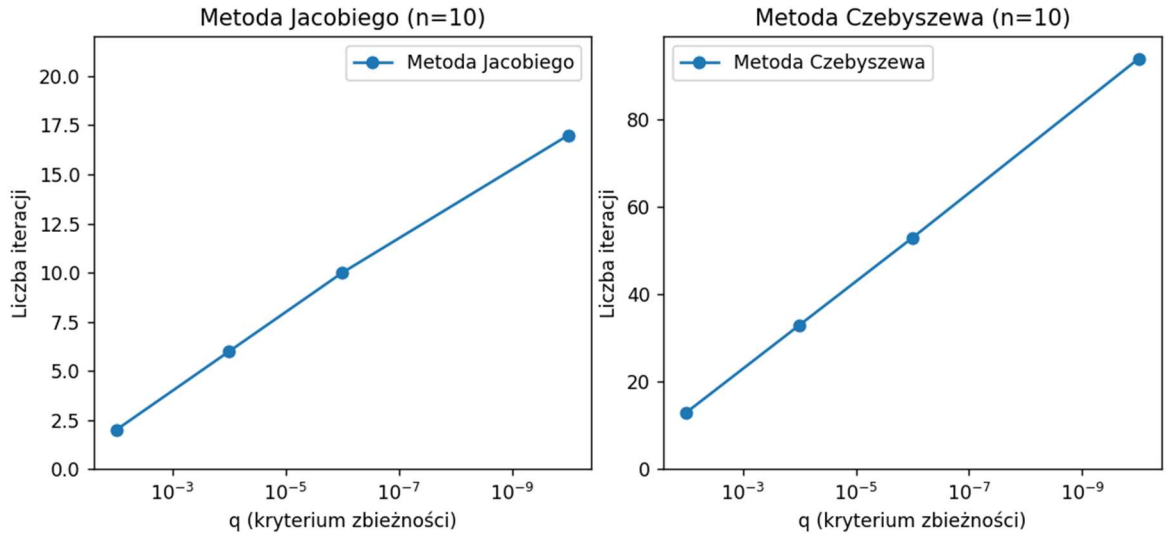
```
def chebyshev_method(A, b, q, max_iterations=100):
    eigenvalues = np.linalg.eigvals(A)
    lambda_min = np.min(eigenvalues)
    lambda_max = np.max(eigenvalues)
    alpha = (lambda_max - lambda_min) / (lambda_max + lambda_min)
    beta = 2 / (lambda_max + lambda_min)

    n = len(A)
    x = np.zeros(n)
    D = np.diag(np.diag(A))
    R = A - D
    D_inv = np.linalg.inv(np.diag(np.diag(D)))

    for iter in range(max_iterations):
        x_new = x + beta * np.dot(D_inv, b - np.dot(A, x))
        if np.linalg.norm(x_new - x) < q or np.linalg.norm(b - np.dot(A, x_new)) / np.linalg.norm(b) < q:
            return x_new , iter
        # old_x = x
        x = alpha * x_new + (1 - alpha) * x
        # distance = np.linalg.norm(old_x - x_new)
        # print("Odległość między x i new_x:", distance)
    return x , iter
```

4. Przykładowe uruchomienie dla n==100:

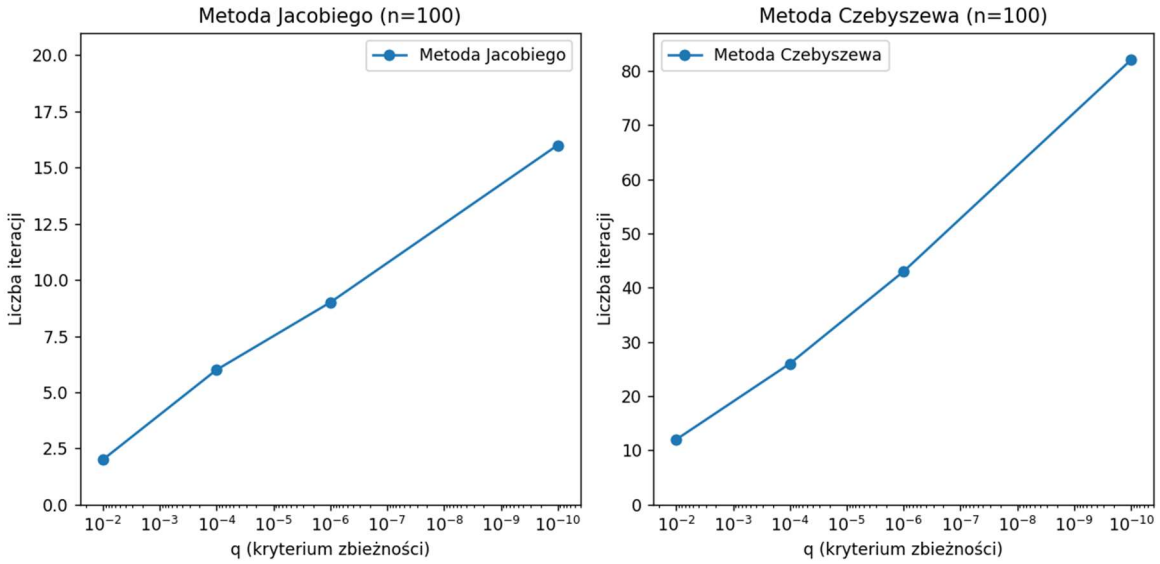
```
def main2(n):
    import matplotlib.pyplot as plt
    A = create_matrix(n)
    # x = [-1,-1,-1,-1,-1]
    x = generate_random_x(n)
    b = generate_b(A, x)

    res_jacobi = []
    res_chebyshev = []

    for q in [1e-2, 1e-4, 1e-6, 1e-10]:
        print(f"q = {q}")
        x_jacobi , iter_jacobi = jacobi_method(A, b, q)
        x_chebyshev, iter_chebyshev = chebyshev_method(A, b, q)

        print("x_jacobi:")
        print(x_jacobi)
        print(f"po iteracjach: {iter_jacobi}")
        print("x_chebyshev:")
        print(x_chebyshev)
        print(f"po iteracjach: {iter_chebyshev}")
        print('\n')

        res_jacobi.append((q, iter_jacobi))
        res_chebyshev.append((q, iter_chebyshev))

    import matplotlib.pyplot as plt
    qs_jacobi, iterations_jacobi = zip(*res_jacobi)

    # Rozpakowanie danych dla metody Czebyszewa
    qs_chebyshev, iterations_chebyshev = zip(*res_chebyshev)

    # Tworzenie wykresów
    plt.figure(figsize=(10, 5))

    # Wykres dla metody Jacobiego
    plt.subplot(1, 2, 1)
    plt.plot(qs_jacobi, iterations_jacobi, marker='o', label='Metoda Jacobiego')
    plt.xscale('log')
    plt.yscale('linear')  # Zmiana na skalę liniową
    plt.title('Metoda Jacobiego')
    plt.xlabel('q (kryterium zbieżności)')
    plt.ylabel('Liczba iteracji')
    # plt.xlim(qs_jacobi[-1], qs_jacobi[0])
    plt.ylim(0, max(iterations_jacobi) + 5)  # Ustawienie granic dla osi Y
    plt.gca().invert_xaxis()  # Odwrócenie osi Y
    plt.legend()

    # Wykres dla metody Czebyszewa
    plt.subplot(1, 2, 2)
```

```
plt.plot(qs_chebyshev, iterations_chebyshev, marker='o', label='Metoda Czebyszewa')
plt.xscale('log')
plt.yscale('linear')  # Zmiana na skalę liniową
plt.title('Metoda Czebyszewa')
plt.xlabel('q (kryterium zbieżności)')
plt.ylabel('Liczba iteracji')
# plt.xlim(qs_chebyshev[-1], qs_chebyshev[0])
plt.ylim(0, max(iterations_chebyshev) + 5)  # Ustawienie granic dla osi Y
plt.gca().invert_xaxis()  # Odwrócenie osi Y
plt.legend()

plt.tight_layout()
plt.show()
```

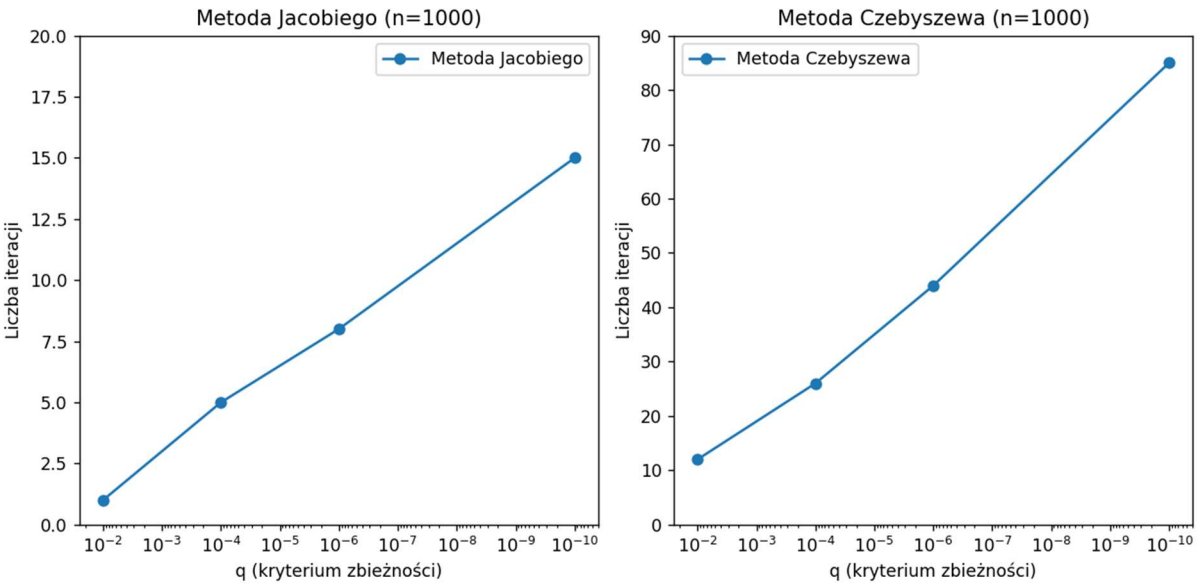
5. **Porównanie metod dla zadanych dokładności przy różnych rozmiarach macierzy:**



Rys 1.1: Zależność ilości iteracji dla zadanej dokładności przy rozmiarze macierzy n=10 dla obu metod



Rys 1.2: Zależność ilości iteracji dla zadanej dokładności przy rozmiarze macierzy n=100 dla obu metod



Rys 1.3: Zależność ilości iteracji dla zadanej dokładności przy rozmiarze macierzy n=1000 dla obu metod

Wnioski:

Pomimo drastycznych różnic w wymiarach macierzy obie metody dość szybko znajdują wynik lecz metoda Jacobiego znajduje go przy 4 razy mniejszej ilości iteracji.

Metoda Czebyszewa, choć może wymagać czterokrotnie większej liczby iteracji niż metoda Jacobiego, może być skuteczniejsza w stabilnym rozwiązywaniu niektórych macierzy o skomplikowanej strukturze lub silnie skorelowanych wartościach własnych. Jednakże, zwiększona złożoność obliczeniowa i konieczność odpowiedniego doboru parametrów mogą być istotnymi czynnikami przy wyborze odpowiedniej metody dla danego problemu.

Zad2:

Dowieść, że proces iteracji dla układu równań:

$$10x_1 - x_2 + 2x_3 - 3x_4 = 0$$

$$x_1 + 10x_2 - x_3 + 2x_4 = 5$$

$$2x_1 + 3x_2 + 20x_3 - x_4 = -10$$

$$3x_1 + 2x_2 + x_3 + 20x_4 = 15$$

jest zbieżny.

Ile iteracji należy wykonać, żeby znaleźć pierwiastki układu z dokładnością do 10^{-3}, 10^{-4}, 10^{-5} ?

Przedstawmy układ równań jako:

$$\begin{pmatrix} 10 & -1 & 2 & -3 \\ 1 & 10 & -1 & 2 \\ 2 & 3 & 20 & -1 \\ 3 & 2 & 1 & 20 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 5 \\ -10 \\ 15 \end{pmatrix}$$

Aby udowodnić, że metoda iteracyjna Jacobiego jest zbieżna, będziemy korzystać z twierdzenia mówiącego o zbieżności tej metody. Twierdzenie to mówi, że jeśli macierz *A* ma dominującą przekątną, czyli gdy elementy na przekątnej | $a_{i,i}$ | są większe od sumy pozostałych elementów w danym wierszu $\sum_{j\neq i}|a_{i,j}|$ dla każdego $i = 1,\ldots,N$ to metoda iteracyjna Jacobiego jest zbieżna.

wiersz	| $a_{i,i}$ |	$\sum_{j\neq i}\|a_{i,j}\|$
1	10	-2
2	10	2
3	20	4
4	20	6

Tabela 2.1: Analiza stwierdzenia czy macierz posiada dominująca przekątną

Jak możemy zauważyć macierz posiada dominującą przekątną więc metoda iteracyjna jest zbieżna.

Używając poprzedniego kodu dla zadanej precyzji znajdę liczbę iteracji (wynik programu):

q = 1e-03
x: [0.3445875 0.27212812 -0.54015937 0.69842969]
po iteracjach: 4

q = 1e-04
x: [0.34477359 0.27183937 -0.54035648 0.69810703]
po iteracjach: 5

q = 1e-05
x: [0.3446915 0.2718729 -0.54034268 0.69812774]
po iteracjach: 7

Wnioski:
Sprawdzenie warunku na zbieżność jest bardzo proste i może zostać wykonane maszynowo.
Dla tego przypadku metoda nie wymagała wielu iteracji

Bibliografia:

wykład dr inż. Katarzyna Rycerz
https://home.agh.edu.pl/~funika/mownit/lab9/12_iteracyjne.pdf