Laboratorium 9 - Układy równań liniowych - metody iteracyjne

Piotr Karamon

12.05.2024r.

Treści zadań

Zadanie 1

Dany jest układ równań liniowych w postaci $Ax = b$ Macierz A o wymiarze $n \times n$ jest określona wzorem:

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

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:

$$\left\| x^{(t+1)} - x^{(t)} \right\| < \rho \quad \text{i} \quad \frac{\left\| Ax^{(t+1)} - b \right\|}{\|b\|} < \rho$$

Zadanie 2

Udowodnij, że proces iteracyjny dla poniższego układu równań jest zbieżny:

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

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

Zadanie 1

Kod został napisany w języku Python. Użyte biblioteki to Numpy oraz Scipy.

Na początku tworzymy funkcję tworząca macierz oraz funkcję generującą losowy wektor x

```
def generate_matrix(n):
    matrix = []
    for i in range(n):
        row = [Fraction(0) for _ in range(n)]
        if i == 0:
            row[0], row[1] = Fraction(1), Fraction(1,2)
        elif i == n-1:
            row[n-2], row[n-1] = Fraction(1,n), Fraction(1)
        else:
            row[i-1], row[i], row[i+1] = Fraction(1,i+1), Fraction(2), Fraction(1,i+2)
        matrix.append(row)
    return matrix


def generate_x(n):
    random.seed(42)
    return [Fraction(random.randint(-1, 0)) for _ in range(n)]
```

Dzięki tym funkcjom wyznaczamy b, przyjmuje $n = 100$.

```
n = 100
A, x_correct = generate_matrix(n), generate_x(n)

b = [0 for _ in range(n)]
for i in range(n):
    for j in range(n):
        b[i] += A[i][j] * x_correct[j]

x_correct = np.array(x_correct, dtype='float64')
A, b = np.array(A, dtype='float64'), np.array(b, dtype='float64')
```

Metoda Jacobiego

Metoda Jacobiego opiera się na rozkładzie macierzy systemu na trzy części: macierz diagonalną D, oraz macierze dolnotrójkątną L i górnotrójkątną U. Dla układu równań $Ax = b$, gdzie $A = D + L + U$ wykonujemy:

1. Przekształcenie układu do postaci $Dx = b - (L + U)x$.
2. Iteracyjne rozwiązywanie równania dla x, wykorzystując wcześniejsze przybliżenie $x^{(t)}$ do obliczenia nowego przybliżenia $x^{(t+1)}$.

Każda iteracja metody Jacobiego wykorzystuje prostą formułę aktualizacji:

$$x_i^{(t+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(t)} \right)$$

Tworzymy funkcję, która realizuję metodę Jacobiego. Przy okazji zbiera ona dane dotyczące warunku stopu.

```
def jacobi_method(A, b, p):
    n = A.shape[0]
    x = np.zeros(n)

    norm1, norm2 = float('inf'), float('inf')

    i = 1
    data = {'i': [], 'norm1': [], 'norm2':[]}
```

```
    while norm1 > p or norm2 > p:
        new_x = np.array(
            [1/A[i,i] * (b[i] - sum(A[i,j]*x[j] for j in range(n) if j!=i))
            for i in range(n)]
        )
        norm1 = np.linalg.norm(x - new_x)
        norm2 = np.linalg.norm(A @ new_x - b) / np.linalg.norm(b)
        x = new_x

        data['i'].append(i)
        data['norm1'].append(norm1)
        data['norm2'].append(norm2)
        i += 1
    return x, data
```

Tablica 1: Wyniki dla metody Jacobiego z precyzją 0.01. Całkowita liczba iteracji: 7.

i	$\|\|x^t - x^{t-1}\|\|$	$\|\|Ax^{t+1} - b\|\|/\|\|b\|\|$
1	7.808633588	0.099823630
2	0.940120275	0.034781102
3	0.324603428	0.012783008
4	0.135150563	0.005293125
5	0.050265407	0.001977497
6	0.021182318	0.000824941
7	0.007856511	0.000308357

Tablica 2: Wyniki dla metody Jacobiego z precyzją 0.0001. Całkowita liczba iteracji: 12.

i	$\|\|x^t - x^{t-1}\|\|$	$\|\|Ax^{t+1} - b\|\|/\|\|b\|\|$
1	7.808633588	0.099823630
2	0.940120275	0.034781102
3	0.324603428	0.012783008
4	0.135150563	0.005293125
5	0.050265407	0.001977497
6	0.021182318	0.000824941
7	0.007856511	0.000308357
8	0.003310392	0.000128756
9	0.001226878	0.000048130
10	0.000516908	0.000020100
11	0.000191546	0.000007514
12	0.000080701	0.000003138

Tablica 3: Wyniki dla metody Jacobiego z precyzją 1e-06. Całkowita liczba iteracji: 17.

i	$\|\|x^t - x^{t-1}\|\|$	$\|\|Ax^{t+1} - b\|\|/\|\|b\|\|$
1	7.808633588	0.099823630
2	0.940120275	0.034781102
3	0.324603428	0.012783008
4	0.135150563	0.005293125
5	0.050265407	0.001977497
6	0.021182318	0.000824941
7	0.007856511	0.000308357
8	0.003310392	0.000128756
9	0.001226878	0.000048130
10	0.000516908	0.000020100
11	0.000191546	0.000007514
12	0.000080701	0.000003138
13	0.000029904	0.000001173
14	0.000012599	0.000000490
15	0.000004668	0.000000183
16	0.000001967	0.000000076
17	0.000000729	0.000000029

Tablica 4: Wyniki dla metody Jacobiego z precyzją 1e-08. Całkowita liczba iteracji: 22.

i	$\|\|x^t - x^{t-1}\|\|$	$\|\|Ax^{t+1} - b\|\|/\|\|b\|\|$
1	7.808633588	0.099823630
2	0.940120275	0.034781102
3	0.324603428	0.012783008
4	0.135150563	0.005293125
5	0.050265407	0.001977497
6	0.021182318	0.000824941
7	0.007856511	0.000308357
8	0.003310392	0.000128756
9	0.001226878	0.000048130
10	0.000516908	0.000020100
11	0.000191546	0.000007514
12	0.000080701	0.000003138
13	0.000029904	0.000001173
14	0.000012599	0.000000490
15	0.000004668	0.000000183
16	0.000001967	0.000000076
17	0.000000729	0.000000029
18	0.000000307	0.000000012
19	0.000000114	0.000000004
20	0.000000048	0.000000002
21	0.000000018	0.000000001
22	0.000000007	0.000000000

Metoda Czebyszewa

Jest ona znana z szybszej zbieżności w porównaniu do tradycyjnych metod iteracyjnych, dzięki wykorzystaniu wielomianów Czebyszewa, które optymalizują proces iteracyjny przez redukcję maksymalnego błędu.

Proces iteracyjny w metodzie Czebyszewa obejmuje kilka kluczowych kroków:

1. **Inicjalizacja**: Ustalenie początkowego przybliżenia x_0 i obliczenie residuum $r_0 = b - Ax_0$, gdzie A to macierz systemu, a b to wektor prawych stron.
2. **Wyznaczenie parametrów** α_t i β_t: Parametry te są wyznaczane na podstawie wielomianów Czebyszewa, które minimalizują wpływ największych błędów w poprzednich iteracjach.
3. **Aktualizacja przybliżenia rozwiązania**: Nowe przybliżenie jest obliczane jako $x_{t+1} = x_t + \alpha_t p_t$, gdzie p_t to odpowiednio wybrany kierunek poszukiwań, zwykle oparty na poprzednich residuach i kierunkach poszukiwań.
4. **Obliczenie nowego residuum**: Residuum dla nowego przybliżenia jest obliczane jako $r_{t+1} = b - Ax_{t+1}$.

Tworzymy funkcję która realizuję metodę Czebyszewa. Parametry $l_{\min}$ i $l_{\max}$ oznaczają: najmniejszą i największą bezwzględną wartość wartości własnych. Tak samo jak w poprzedniej metodzie, zbieramy dane dotyczące warunku stopu.

```
def chebyshev_method(A, b, l_min, l_max, precision):
    d = (l_max + l_min) / 2
    c = (l_max - l_min)/ 2

    x = np.zeros(A.shape[0])
    r = b - A @ x
    norm1, norm2 = 2,2


    i = 1
    data = {'i': [], 'norm1': [], 'norm2':[]}
    while norm1 > precision or norm2 > precision:
        z = r

        if i == 1:
            p = z
            alpha = 1/d
        elif i == 2:
            beta = 1/2 * (c*alpha) * (c*alpha)
            alpha = 1/(d - beta/alpha)
            p = z + beta * p
        else:
            beta = (c*alpha/2) * (c*alpha/2)
            alpha = 1/(d - beta/alpha)
            p = z + beta * p

        new_x = x + alpha * p
        r = b - A @ new_x

        norm1 = np.linalg.norm(x - new_x)
        norm2 = np.linalg.norm(A @ new_x - b) / np.linalg.norm(b)

        x = new_x

        data['i'].append(i)
        data['norm1'].append(norm1)
```

```
        data['norm2'].append(norm2)

        i+= 1
    return x, data
```

Możemy skorzystać z tego, że nasza macierz jest trójdiagonalna i symetryczna i użyć funkcji `eigvalsh_tridiagonal` z biblioteki Scipy w celu wyliczenia $l_{\min}$ i $l_{\max}$.

```
eigvalues = sp.linalg.eigvalsh_tridiagonal(np.diag(A), np.diag(A, k=1))
l_min, l_max = np.min(eigvalues), np.max(eigvalues)
print(f'l_min = {l_min}, l_max = {l_max}')
```

```
l_min = 0.7781395783931968, l_max = 2.4946908392857066
```

Tablica 5: Wyniki dla metody Czebyszewa z precyzją 0.01. Całkowita liczba iteracji: 7.

i	$\|\|x^t - x^{t-1}\|\|$	$\|\|Ax^{t+1} - b\|\|/\|\|b\|\|$
1	9.410590989	0.280417554
2	1.695183688	0.094850576
3	1.015771898	0.042951955
4	0.358840775	0.006982264
5	0.051698988	0.002609285
6	0.025479556	0.000894928
7	0.007264400	0.000151942

Tablica 6: Wyniki dla metody Czebyszewa z precyzją 0.0001. Całkowita liczba iteracji: 11.

i	$\|\|x^t - x^{t-1}\|\|$	$\|\|Ax^{t+1} - b\|\|/\|\|b\|\|$
1	9.410590989	0.280417554
2	1.695183688	0.094850576
3	1.015771898	0.042951955
4	0.358840775	0.006982264
5	0.051698988	0.002609285
6	0.025479556	0.000894928
7	0.007264400	0.000151942
8	0.001296312	0.000065839
9	0.000612894	0.000019011
10	0.000151699	0.000003751
11	0.000033825	0.000001527

Tablica 7: Wyniki dla metody Czebyszewa z precyzją 1e-06. Całkowita liczba iteracji: 14.

i	$\|\|x^t - x^{t-1}\|\|$	$\|\|Ax^{t+1} - b\|\|/\|\|b\|\|$
1	9.410590989	0.280417554
2	1.695183688	0.094850576
3	1.015771898	0.042951955
4	0.358840775	0.006982264
5	0.051698988	0.002609285
6	0.025479556	0.000894928
7	0.007264400	0.000151942
8	0.001296312	0.000065839
9	0.000612894	0.000019011
10	0.000151699	0.000003751
11	0.000033825	0.000001527
12	0.000013790	0.000000395
13	0.000003147	0.000000095
14	0.000000882	0.000000036

Tablica 8: Wyniki dla metody Czebyszewa z precyzją 1e-08. Całkowita liczba iteracji: 18.

i	$\|\|x^t - x^{t-1}\|\|$	$\|\|Ax^{t+1} - b\|\|/\|\|b\|\|$
1	9.410590989	0.280417554
2	1.695183688	0.094850576
3	1.015771898	0.042951955
4	0.358840775	0.006982264
5	0.051698988	0.002609285
6	0.025479556	0.000894928
7	0.007264400	0.000151942
8	0.001296312	0.000065839
9	0.000612894	0.000019011
10	0.000151699	0.000003751
11	0.000033825	0.000001527
12	0.000013790	0.000000395
13	0.000003147	0.000000095
14	0.000000882	0.000000036
15	0.000000315	0.000000008
16	0.000000065	0.000000002
17	0.000000021	0.000000001
18	0.000000007	0.000000000

Wnioski

Używając metody Czebyszewa potrzebujemy mniejszej ilości iteracji niż w przypadku metody Jacobiego przy tej samej precyzji. Metoda Jacobiego jest bardzo łatwa w implementacji i dobrze radzi w szczególności z macierzami trójdiagonalnymi. Metoda Czebyszewa przyśpieszania zbieżność, ale ceną jest wymagana wiedza o wartościach własnych macierzy. Bardzo rzadko istnieje na nie wzór jawny, co utrudnia zastosowanie tej metody.

Zadanie 2

W metodzie Jacobiego macierz jest zbieżna, jeśli spełnia kryteria silnej dominacji diagonalnej:

- wierszowo

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|$$

- kolumnowo

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ji}|$$

Możemy łatwo to sprawdzić używając biblioteki Numpy. `np.sum(A, axis=1)` sumuje macierz po wierszach. `-np.abs(np.diag(A))` sprawia, że nie dodajemy elementów na przekątnej.

```
b = np.array([0, 5, -10, 15])

A = np.array([
    [10, -1, 2, -3],
    [1, 10, -1, 2],
    [2, 3, 20, -1],
    [3, 2, 1, 20]
])

abs_diag = np.abs(np.diag(A))
row_sums = np.sum(A, axis=1) - abs_diag

print(f'sumy wierszowe = {row_sums}')
print(f'A ma dominującą przekątną: {np.all(abs_diag > row_sums)}')
```

```
sumy wierszowe = [-2  2  4  6]
A ma dominującą przekątną: True
```

Z tego wniosek, że A ma dominującą przekątną, zatem proces iteracji jest zbieżny dla tego układu.

```
for precision in [10**(-3), 10**(-4), 10**(-5)]:
    x, data = jacobi_method(A, b, precision)
    print(f'precyzja = {precision:.0e}, liczba iteracji = {len(data["i"])}')
```

```
precyzja = 1e-03, liczba iteracji = 6
precyzja = 1e-04, liczba iteracji = 7
precyzja = 1e-05, liczba iteracji = 9
```

Tablica 9: Liczba iteracji w zależności od zadanej precyzji.

Precyzja	Liczba iteracji
10^{-3}	6
10^{-4}	7
10^{-5}	9

Wnioski: Sprawdzenie czy proces iteracji jest zbieżny dla podanego układu równań jest zadaniem prostym o ile zna się odpowiednie twierdzenie. Jeżeli macierz ma dominującą przekątną to nie musimy

liczyć wartości własnych w celu sprawdzenia zbieżności metody. Im lepszą chcemy mieć precyzję tym więcej musimy wykonać iteracji, jednakże liczba iteracji rośnie wolno. To oznacza, że uzyskanie dobrej precyzji nie wymaga ogromnego nakładu obliczeniowego.

Bibliografia

- Marian Bubak, Katarzyna Rycerz: *Metody Obliczeniowe w Nauce i Technice. Iteracyjne rozwiązywanie* $Ax = B$
- Chebyshev iteration method
- Chebyshev iteration