Laboratorium 8

Układy równań liniowych – metody iteracyjne

Jan Rajczyk

10 maja 2021

1 Treści zadań

1. Dany jest układ równań liniowych $Ax = b$. Macierz A o wymiarze $n \times n$ 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 $\{0, 1\}$ 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 stop:

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

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

2. Dowieść, że proces iteracji dla układu równań:

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

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

2 Rozwiązania

2.1 Zadanie 1

Obliczenia były wykonywane dla $n = 5$.

1. **Metoda Jacobiego**
   Skorzystałem z programów napisanych w języku *Python*:

```
    from random import randint
import numpy as np

n = 5


def generate_matrices():
    M = [[0.0 for _ in range(n)] for _ in range(n)]
    X = [randint(0, 1) for _ in range(n)]
    for i in range(n):
        for j in range(n):
            if i == j:
                if i == 0 or i == n - 1:
                    M[i][j] = 1
                else:
                    M[i][j] = 2
            elif i == j + 1:
                M[i][j] = 1 / (i + 1)
            elif j == i + 1:
                M[i][j] = 1 / (i + 2)
    return M, X


def jacobi_iteration(A: np.array, b: np.array, precision: float):
    x = np.zeros(len(A[0])).reshape(-1, 1)
    D = np.diag(A).reshape(-1, 1)
    L_U = A - np.diagflat(np.diag(A))
    results = []
    norm_b = np.linalg.norm(b)
    norm_one = 2
    norm_two = 2
    i = 1
    while norm_one > precision or norm_two > precision:
        next_x = (b - L_U @ x) / D
        norm_one = np.linalg.norm(abs(x - next_x))
        norm_two = np.linalg.norm(A @ x - b) / norm_b
        results.append((
```

```
            i,
            norm_one,
            norm_two
        ))
        x = next_x
        i += 1

    return x, results


def run():
    mA, mX = generate_matrices()
    A = np.array(mA)
    x = np.array(mX).reshape(-1, 1)
    b = A @ x
    solves, res = jacobi_iteration(A, b, 10e-7)
    print("Macierz A:")
    print(A)
    print("Wektor b:")
    print(b)
    print("Wektor x b d cy rozwi zaniem: ")
    print(solves)
    for t in res:
        print(t[0], "&", t[1], "&", t[2], "\\\\")


run()
```

Głównym narzędziem, które posłużyło mi przy rozwiązywaniu problemu była biblioteka *numpy*. Dla trzech precyzji: $10^{-2}, 10^{-4}, 10^{-6}$ zmierzyłem ilość iteracji jakie wykonała metoda. Jako wektor początkowy niewiadomych x przyjąłem wektor równy $\mathbf{0}$. Wyniki dla zadanych trzech precyzji przedstawiają tabele.

iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	1.9351643628154978	1.0
2	0.5961238041648523	0.30995626086184325
3	0.25979711551122026	0.10197592870128352
4	0.08696304667990903	0.04524338520386641
5	0.04051845432057678	0.0154910504327531
6	0.013618662750820415	0.007032486806424461

Tablica 1: Wyniki przy precyzji 10^{-2}.Liczba iteracji: 6.

iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	1.4396180048887968	1.0
2	0.30242193666759326	0.3725054503344475
3	0.1437297974598689	0.1228447714200682
4	0.04657106698886923	0.057035579498559544
5	0.02260778409678107	0.018905554369093438
6	0.007280429027815327	0.008908760326845724
7	0.003541778536834186	0.002948344045977062
8	0.001138532729539925	0.0013929543216747967
9	0.0005540607282854025	0.000460796181763099
10	0.00017803829920808078	0.000217816590569739 2
11	8.664704103896087e-05	7.204804518223717e-05

Tablica 2: Wyniki przy precyzji 10^{-4}.Liczba iteracji: 11.

iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	2.6890467125400077	1.0
2	1.0371336414371184	0.3463469830947061
3	0.4067701944427899	0.1340974511864947
4	0.16164401212239515	0.05249730987057719
5	0.06366132339346978	0.020806341359257532
6	0.025347268658867432	0.008177257642664996
7	0.009961793631655431	0.00325063693112076
8	0.003966443449118163	0.001277820154655031 7
9	0.0015580044105057717	0.000508232011979832 7
10	0.0006203363223596209	0.000199788546607413 34
11	0.00024363713986527417	7.94710213188885e-05
12	9.700646256269641e-05	3.124053043363581e-05
13	3.809836898807989e-05	1.2426990032660061e-05
14	1.5169220693777465e-05	4.8851257453414014e-06
15	5.957537950640904e-06	1.943234486657044e-06
16	2.3720489337231936e-06	7.63897423155836e-07

Tablica 3: Wyniki przy precyzji 10^{-6}.Liczba iteracji: 16.

2. **Metoda Czebyszewa**
Napisałem również program w języku *Python* dla iteracji Czebyszewa. Również wykorzystałem użyłem trzech rodzajów precyzji: $10^{-2}, 10^{-4}, 10^{-6}$. Kod programu oraz tabele przedstawiające wyniki eksperymentu znajdują się poniżej:

```
from random import randint
import numpy as np

n = 5


def generate_matrices():
    M = [[0.0 for _ in range(n)] for _ in range(n)]
    X = [randint(0, 1) for _ in range(n)]
    for i in range(n):
        for j in range(n):
            if i == j:
                if i == 0 or i == n - 1:
                    M[i][j] = 1
                else:
                    M[i][j] = 2
            elif i == j + 1:
                M[i][j] = 1 / (i + 1)
            elif j == i + 1:
                M[i][j] = 1 / (i + 2)
    return M, X


def Chebyshev_method(A: np.array, b: np.array, precision):
    x_prior = np.zeros(len(A[0])).reshape(-1, 1)
    t = []
    results = []
    eigs = np.linalg.eig(A)[0]
    p, q = np.min(np.abs(eigs)), np.max(np.abs(eigs))
    r = b - A @ x_prior
    x_posterior = x_prior + 2 * r / (p + q)
    r = b - A @ x_posterior
    t.append(1)
    t.append(-(p + q) / (q - p))
    beta = -4 / (q - p)
    i: int = 1
    norm_one = 2
    norm_two = 2
    norm_b = np.linalg.norm(b)
```

```
    while norm_one > precision or norm_two > precision:
        norm_one = np.linalg.norm(abs(x_posterior - x_prior))
        norm_two = np.linalg.norm(A @ x_posterior - b) / norm_b
        results.append((
            i,
            norm_one,
            norm_two
        ))
        i += 1
        t.append(2 * t[1] * t[-1] - t[-2])
        alpha = t[-3] / t[-1]
        old_prior, old_posterior = x_prior, x_posterior
        x_prior = old_posterior
        x_posterior = (1 + alpha) * old_posterior -
        alpha * old_prior + (beta * t[-2] / t[-1]) * r
        r = b - A @ x_posterior

    return x_posterior, results


def run():
    mA, mX = generate_matrices()
    A = np.array(mA)
    x = np.array(mX).reshape(-1, 1)
    b = A @ x
    solves, res = Chebyshev_method(A, b, 10e-3)
    print("Macierz A:")
    print(A)
    print("Wektor b:")
    print(b)
    print("Wektor x b d cy rozwi zaniem: ")
    print(solves)
    for t in res:
        print(t[0], "&", t[1], "&", t[2], "\\\\")


run()
```

iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	2.1526157103063044	0.5095900948288071
2	0.9941492619112802	0.15176431969784396
3	0.28318765238118376	0.043254053498560686
4	0.07860589629030064	0.012583503488152991
5	0.023599404307988472	0.003551568640025496

Tablica 4: Wyniki dla metody Czebyszewa przy precyzji 10^{-2}.
Liczba iteracji: 5.

iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	1.609210473922433	0.496527972250641
2	0.8117654501718635	0.1480533020825101
3	0.23201469839916497	0.04147743970515215
4	0.06261223237830313	0.012520014724932594
5	0.01962832191695594	0.003502363864471705
6	0.005701080996519378	0.0009767159166338634
7	0.001587340492188428	0.00026141035792711883
8	0.0004007217226866876	7.684385448682809e-05

Tablica 5: Wyniki dla metody Czebyszewa przy precyzji 10^{-4}.
Liczba iteracji: 8.

iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	0.6248865202898515	0.39299708922737125
2	0.3690670935913977	0.05729764028060735
3	0.06287485486655253	0.021979054511949567
4	0.010655458444122608	0.011153850071500297
5	0.007902494653985683	0.0033021137516776673
6	0.002822255780122259	0.0007302521363297456
7	0.0006867300178678448	8.019943132981218e-05
8	7.32071197525157e-05	4.638177786644325e-05
9	2.6912557148308848e-05	2.121325388749205e-05
10	1.5907171173690895e-05	6.031239494719148e-06
11	5.175592109809045e-06	9.413794243155084e-07

Tablica 6: Wyniki dla metody Czebyszewa przy precyzji 10^{-4}.
Liczba iteracji: 11.

Wnioski: Możemy łatwo zauważyć, że mniejszą ilość iteracji wykonuje metoda Czebyszewa (szczególnie widać, że iteracje zmniejszają się wraz ze wzrostem precyzji). Przy metodzie Jacobiego ilość iteracji przy tysiąckrotnym zwiększeniu precyzji rośnie prawie trzykrotnie, zaś przy metodzie Czebyszewa dwukrotnie. To co przemawia za metodą Jacobiego to z pewnością łatwość implementacji.

2.2 Zadanie 2

Aby dowieść zbieżności metody iteracyjnej skorzystamy z twierdzenia mówiącego o *zbieżności metody iteracyjnej Jacobiego*, które mówi, że jeżeli macierz A ma dominującą przekątną, czyli gdy:

$$|a_{i,i}| > \sum_{j \neq i} |a_{i,j}|, \forall i = 1, \ldots, N.$$

Przedstawmy powyższe zależności dla naszego przykładu w formie tabeli:

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

Tablica 7: Analiza przeprowadzona w celu rozstrzygnięcia, czy macierz ma dominującą przekątną

Łatwo wywnioskować, że nasza macierz A posiada dominująca przekątną, więc z powyższego twierdzenia otrzymujemy, że metoda iteracyjna jest zbieżna.
W celu udzielenia odpowiedzi na drugie pytanie musimy wykonać iteracje Jacobiego odpowiednio przy precyzjach: $10^{-3}, 10^{-4}, 10^{-5}$. Wykorzystujemy w tym celu nieco zmodyfikowany program rozwiązujący zadanie 1:

```
from random import randint
import numpy as np

def jacobi_iteration(A: np.array, b: np.array, precision: float):
    x = np.zeros(len(A[0])).reshape(-1, 1)
    D = np.diag(A).reshape(-1, 1)
    L_U = A - np.diagflat(np.diag(A))
    results = []
    norm_b = np.linalg.norm(b)
    norm_one = 2
    norm_two = 2
    i = 1
```

```
    while norm_one > precision or norm_two > precision:
        next_x = (b - L_U @ x) / D
        norm_one = np.linalg.norm(abs(x - next_x))
        norm_two = np.linalg.norm(A @ x - b) / norm_b
        results.append((
            i,
            norm_one,
            norm_two
        ))
        x = next_x
        i += 1

    return x, results


def run_test(prec):
    A = np.array([[10, -1, 2, -3],
                  [1, 10, -1, 2],
                  [2, 3, 20, -4],
                  [3, 2, 1, 20]])
    b = np.array([0, 5, -10, 15]).reshape(-1, 1)
    solves, res = jacobi_iteration(A, b, prec)
    print(len(res))


run_test(10e-4)
run_test(10e-5)
run_test(10e-6)
```

Otrzymane wyniki przedstawia tabelka:

Precyzja	liczba iteracji
10^{-3}	6
10^{-4}	8
10^{-5}	9

Tablica 8: Liczba operacji w zależności od przyjętej precyzji w metodzie Jacobiego

Wnioski: Samo sprawdzenie zbieżności metody iteracyjnej okazało się być trywialnym po wykorzystaniu twierdzenia i sprowadzeniu tego problemu do rozstrzygnięcia, czy posiada ona dominującą przekątną. co do ilości iteracji, które wykonujemy dla zadanego układu równań to możemy zauważyć, że przy kolejnych iteracjach zwiększanych dziesięciokrotnie liczba iteracji zwiększa się o 1, maksymalnie 2, jednakże mamy za mało danych, aby wyrokować o tym jaka liczba iteracji będzie wykonana przy precyzji rzędu 10^n i możemy jedynie snuć pewne domysły co do jej postaci.
Widzimy również, że przy tak małym wzroście liczby iteracji w zależności od precyzji warto pokusić się o używanie dużej precyzji, ponieważ nie jest to kosztowne obliczeniowo, a często może być kluczowe w pewnych projektach jak np. informatyka medyczna czy bardzo małe układy cyfrowe.

3 Bibliografia

- `http://wazniak.mimuw.edu.pl/index.php?title=MN08`
- Marian Bubak *PhD*
- `https://en.wikipedia.org/wiki/Jacobi_method`
- `https://en.wikipedia.org/wiki/Chebyshev_iteration#cite_note-1`
- `https://www.quantstart.com/articles/Jacobi-Method-in-Python-and-NumPy/`
- `https://numpy.org/doc/`
- `https://www.qualitetch.com/importance-precision/`