Laboratorium 10

Układy równań – metody iteracyjne

Krzysztof Solecki

22 Maja 2023

1 Treści zadań

1. Dany jest układ równań liniowych Ax = b.

   Macierz A o wymiarze n x n jest określona wzorem:

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

   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:

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

$$\frac{1}{||b||}||Ax^{t+1} - b|| < p$$

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 zadań

2.1 Zadanie 1:

Obliczenia były wykonywane dla n=5

1. **Metoda Jacobiego:** Programy zostały napisane 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 będący rozwiązaniem: ")
    print(solves)
    for t in res:
        print(t[0], "&", t[1], "&", t[2], "\\\\")

run()
```

Do rozwiązania tego problemu wykorzystałem bibliotekę numpy. Dla trzech precyzji: 10^2, 10^4, 10^6 zmierzyłem ilość iteracji jakie wykonała metoda. Jako wektor początkowy niewiadomych x przyjąłem wektor równy 0. Wyniki dla zadanych trzech precyzji przedstawiają poniższe tabele.

iteracja	$\|\|x^{(t+1)} - x^{(t)}\|\| < p$	$\frac{1}{\|\|b\|\|}\|\|Ax^{t+1} - b\|\| < p$
1	2.6890467125400077	1.0
2	1.0371336414371182	0.34634698309470596
3	0.40677019444278995	0.1340974511864947
4	0.1616440121223953	0.05249730987057718
5	0.06366132339346983	0.020806341359257616

Table 1: Wyniki przy precyzji 10^{-2} Liczba iteracji: 5

iteracja	$\|\|x^{(t+1)} - x^{(t)}\|\| < p$	$\frac{1}{\|\|b\|\|}\|\|Ax^{t+1} - b\|\| < p$
1	2.005703672585316	1.0
2	0.7445258331597931	0.31056916548778096
3	0.26360366975012284	0.1254295956200472
4	0.11289459212018003	0.045685346490808534
5	0.04122359834200346	0.01928113277247783
6	0.01763220188220559	0.007123550964675059
7	0.006460398135860875 5	0.0030079182282454374
8	0.00275711610627993	0.001113764578304602
9	0.001010772648656847	0.0004701507147625337
10	0.0004311382926157623	0.00017415997747767306

Table 2: Wyniki przy precyzji 10^{-4} Liczba iteracji: 10

iteracja	$\|\|x^{(t+1)} - x^{(t)}\|\| < p$	$\frac{1}{\|\|b\|\|}\|\|Ax^{t+1} - b\|\| < p$
1	1.5411260313304829	1.0
2	0.6390600955688415	0.29745281050185207
3	0.2056368787580749	0.13013642929609098
4	0.09877823147364358	0.04311175637902368
5	0.03194323986244671	0.02014459672062545
6	0.015521485844769279	0.00667382233944876
7	0.004994924733972802	0.003148230513318812
8	0.0024302699973011415	0.0010417253915007286
9	0.0007811083111612775	0.0004922688747339936
10	0.00038013219329430665	0.00016283884963403845
11	0.00012214528246874427	7.697654824917496e-05
12	5.9445546102630874e-05	2.5461634620982723e-05
13	1.9100190397552135e-05	1.2036979323172483e-05
14	9.295745490733139e-06	3.981434993056681e-06

Table 3: Wyniki przy precyzji 10^{-6} Liczba iteracji: 14

2. **Metoda Czebyszewa:** Dla programu implementującego metodę Czebyszewa również 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 będący rozwiązaniem: ")
    print(solves)
    for t in res:
        print(t[0], "&", t[1], "&", t[2], "\\\\")

run()
```

iteracja	$\|\|x^{(t+1)} - x^{(t)}\|\| < p$	$\frac{1}{\|\|b\|\|}\|\|Ax^{t+1} - b\|\| < p$
1	2.0202485690464753	0.4335009125047346
2	0.8018889158296976	0.15003638756445328
3	0.27888517151414965	0.038005390574777885
4	0.06596165147744575	0.01237621475171096

Table 4: Wyniki przy precyzji 10^{-2} Liczba iteracji: 4

iteracja	$\|\|x^{(t+1)} - x^{(t)}\|\| < p$	$\frac{1}{\|\|b\|\|}\|\|Ax^{t+1} - b\|\| < p$
1	2.3089799577112506	0.4712442511719165
2	0.9895542374034594	0.13812761260016096
3	0.2979835610123919	0.0440482973248522
4	0.09194622860561251	0.011825625106019087
5	0.024209487077004335	0.003150128222297431
6	0.0068796485182498705	0.000991035557286659
7	0.0021354650794973822	0.0002675738738626793
8	0.0005424866243563961	6.936919745157333e-05

Table 5: Wyniki przy precyzji 10^{-4} Liczba iteracji: 8

iteracja	$\|\|x^{(t+1)} - x^{(t)}\|\| < p$	$\frac{1}{\|\|b\|\|}\|\|Ax^{t+1} - b\|\| < p$
1	2.199110749055527	0.44811185842776646
2	0.8694370293364297	0.1580843424318497
3	0.31367522863133396	0.03870380970911001
4	0.07272936239678658	0.012660592676128088
5	0.025317127143329347	0.0031201833334730694
6	0.005823987690537954	0.0010027860745345613
7	0.002022252284556236	0.00025107591871561977
8	0.00046301941755837916	7.924827403066105e-05
9	0.00010648352469752618	2.035030868896849e-05
10	3.740182084996565e-05	6.258286151086795e-06
11	1.2768347143945973e-05	1.645973189484307e-06
12	3.017384014750027e-06	4.917795801763545e-07

Table 6: Wyniki przy precyzji 10^{-6} Liczba iteracji: 12

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:

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

Przedstawmy powyższe zależności dla naszego przykładu w formie tabeli:

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

Table 7: Analiza rozstrzygająca, czy macierz posiada 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: 1^3, 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^{-2}	6
10^{-4}	8
10^{-6}	9

Table 8: Liczba operacji w zależności od użytej 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 10n 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

1. Katarzyna Rycerz: Materiały wykładowe z przedmiotu Metody Obliczeniowe w Nauce i Technice
2. Włodzimierz Funika: Materiały ze strony
3. https://wazniak.mimuw.edu.pl/index.php?title=MN08
4. https://en.wikipedia.org/wiki/Jacobi_method
5. https://en.wikipedia.org/wiki/Chebyshev_iterationcite_note-1
6. https://www.quantstart.com/articles/Jacobi-Method-in-Python-and-NumPy/