Laboratorium IX

Układy równań liniowych – metody iteracyjne

Dominik Marek

23 kwietnia 2024

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

1.Zadania

1.1.

Dany jest układ równań liniowych $\mathbf{A}x=\mathbf{b}$.

Macierz $\mathbf{A}$ o wymiarze mxn jest określona wzorem:

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

Przyjmij wektor $\boldsymbol{x}$ 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 $\mathbf{A}\boldsymbol{x}=\mathbf{b}$ (przyjmując jako niewiadomą wektor $\boldsymbol{x}$).

W obu przypadkach oszacuj liczbę iteracji przyjmując test stopu:

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

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

1.2.

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}

2.Rozwiązania

2.1

Do rozwiązania tego zadania w znaczącym stopniu wykorzystałem narzędzia z biblioteki numpy z języka programowania Python. Dla każdej metody zmierzyłem ilość iteracji dla następujących precyzji 10^{-2}, 10^{-4} i 10^{-6}. Za wektor niewiadomych x początkowo przyjąłem wektor równy 0. Wyniki dla zadanych precyzji zostały przedstawione w poniższych tabelach.

a)Metoda Jacobiego:

Wyniki przy precyzji $\varepsilon = 10^{-2}$		
Iteracja	$\left\|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^{(t)}\right\|$	$\frac{1}{\|\boldsymbol{b}\|}\left\|\boldsymbol{A}\boldsymbol{x}^{(t+1)} - \boldsymbol{b}\right\|$
1	6.248755456088836	2.23606797749979
2	1.8118643316716279	0.8410228891058792
3	0.7803824073435589	0.29172138710827317
4	0.2754856303664639	0.12505121278663373
5	0.12241093800739225	0.044636266236187314
6	0.04311567752301798	0.01947083629560741

Wyniki przy precyzji $\varepsilon = 10^{-4}$		
Iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	6.821547808232381	2.2360679774997902
2	1.977948928771987	0.8410228891058793
3	0.8519161835994692	0.29172138710827333
4	0.30073802875334527	0.1250512127866337
5	0.1336317402298623	0.04463626623618734
6	0.04706787736764593	0.01947083629560738
7	0.02093177773024309	0.00695636916907400
8	0.0073613214540232145	0.0030419583145610512
9	0.003273458740636398	0.0010867613024429696
10	0.0011508032559900269	0.00047544872408419544
11	0.000511730829084996 5	0.00016985444207819197

Wyniki przy precyzji $\varepsilon = 10^{-6}$		
Iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	5.824969055711799	2.23606797749979
2	1.6889849089631472	0.8410228891058792
3	0.7274573963314147	0.2917213871082732
4	0.25680237984257825	0.1250512127866337
5	0.1141091103635561	0.044636266236187216
6	0.04019160121603238	0.019470836295607325
7	0.01787379653230027	0.006956369169074061
8	0.00628588567935909	0.0030419583145609996
9	0.002795230115714257	0.001086761302442883
10	0.0009826792311365881	0.00047544872408417614
11	0.0004369706594562282	0.00016985444207825103
12	0.00015360799834844238	7.431667871316526e-05
13	6.830491840160893e-05	2.6549586693540188e-05
14	2.4010809554824082e-05	1.161649954827878e-05
15	1.0676882422179498e-05	4.149983652886101e-06
16	3.753167149475629e-06	1.8157897755410572e-06

b)Metoda Czebyszewa

Wyniki przy precyzji $\varepsilon = 10^{-2}$		
Iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	5.163323870112893	1.0997371500152666
2	2.5370993477796993	0.32882151773229046
3	0.7279187610120931	0.0926077319411635
4	0.1967804493174055	0.02784838699449673
5	0.06155327973149402	0.00773213667882066

Wyniki przy precyzji $\varepsilon = 10^{-4}$		
Iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	5.636620104094822	1.0997371500152664
2	2.7696626339010257	0.3288215177322906
3	0.7946434555883957	0.09260773194116369
4	0.21481833497518935	0.027848386994496856
5	0.06719556292329333	0.0077321366788206764
6	0.01942178223436863	0.0021823365794919705
7	0.005467015676387295	0.0005816559942679519
8	0.001365620215048893	0.00017008233969926937
9	0.00040081528274612074	4.9683428245902476e-05

Wyniki przy precyzji $\varepsilon = 10^{-6}$		
Iteracja	$\|x^{(t+1)} - x^{(t)}\|$	$\frac{1}{\|b\|}\|Ax^{(t+1)} - b\|$
1	4.813150711269907	1.0997371500152664
2	2.3650349731134104	0.32882151773229057
3	0.678551799276434	0.0926077319411635
4	0.183434931339175	0.027848386994496772
5	0.057378777619494746	0.007732136678820496
6	0.01658440044727641	0.002182336579491942
7	0.004668324262657132	0.000581655994267887
8	0.0011661129875706901	0.00017008233969931629
9	0.0003422590715025221	4.968342824585108e-05
10	0.00010320830153307182	1.4222558484881944e-05
11	3.0721549170516475e-05	3.762935884422478e-06
12	7.875017562376205e-06	1.062924397614278e-06

Powyższe wyniki zostały uzyskane za pomocą poniższego programu napisanego w języku Python:

```
from random import randint
from typing import Final
import numpy as np.
from numpy import e

def generate_matrix(n:int):
    M = [[0.0]*n for _ in range(n)]
    V = [randint(-1, 0) for _ in range(n)]

    for i in range(n):
        for j in range(n):
            if j - i == 1:
                M[i][j] = round(1/(j+1), 3)
                M[j][i] = round(1/(j+1), 3)

            if i == j:
                if 0 < i < n-1:
                    M[i][i] = 2
```

```
                else:
                    M[i][i] = 1
    return M, V


def jacobi_iteration_method(matrix, vector, precision):
    x = np.zeros(len(matrix[0])).reshape(-1, 1)
    D =  np.diag(matrix).reshape(-1,1)
    L_U = matrix-np.diagflat(np.diag(matrix))
    results = []
    norm_vector = np.linalg.norm(vector)
    norm_one = 2
    norm_two = 2
    i = 1

    while norm_one > precision or norm_two > precision:
        next_x = (vector-L_U@x)/D
        norm_one = np.linalg.norm(abs(x-next_x))
        norm_two = np.linalg.norm(matrix@x-vector)/norm_vector
        results.append((i, norm_one, norm_two))
        x =  next_x
        i += 1
    return x, results


def chebyshev_iteration_method(matrix, vector, precision):
    x_prior = np.zeros(len(matrix[0])).reshape(-1, 1)
    t = []
    results = []
    eigs = np.linalg.eig(matrix)[0]
    p, q = np.min(np.abs(eigs)) , np.max(np.abs(eigs))
    r = vector-matrix@x_prior
    x_posterior = x_prior + 2*r/(p+q)
    r = vector - matrix @ x_posterior
    t.append(1)
    t.append(-(p+q)/(q-p))
    beta = -4/(q-p)
    i = 1
    norm_one = 2
    norm_two = 2
    norm_vector = np.linalg.norm(vector)

    while norm_one > precision or norm_two > precision:
        norm_one = np.linalg.norm(abs(x_posterior-x_prior))
        norm_two = np.linalg.norm(matrix@x_posterior-vector)/norm_vector
        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=vector-matrix@x_posterior

    return x_posterior, results


def main() -> None:
    N:Final = 5
```

```
    matrix, vector = generate_matrix(N)
    A = np.array(matrix)
    x = np.array(vector)
    b = A @ x
    print(f'{x=}')
    solves1, res1 = jacobi_iteration_method(A, b, 10e-6)
    solves2, res2 = chebyshev_iteration_method(A, b, 10e-6)
    print("Matrix A:")
    print(A)
    print("Vector b:")
    print(b)
    print("Solutions vector:")
    print(f'{solves1=}')
    print(f'{solves2=}')
    print("--------------Jacobi_iteration_method--------------")
    for t in res1:
        print(t[0], " ", t[1], " ", t[2])
    print("--------------Chebyshev_iteration_method--------------")
    for t in res2:
        print(t[0], " ", t[1], " ", t[2])


if __name__ == '__main__':
    main()
```

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 stukrotnym zwiększeniu precyzji rośnie prawie dwukrotnie, zaś przy metodzie Czebyszewa około półtorakrotnie. To co przemawia za metodą Jacobiego to z pewnością łatwość implementacji.

2.2

W celu dowiedzenia zbieżności metody iteracyjnej wykorzystane zostało twierdzenie o zbieżności metody iteracyjnej Jacobiego , które mówi, że jeżeli macierz A ma dominującą przekątną, czyli gdy:

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

Przedstawmy powyższe zależności dla naszego przykładu w formie tabeli:

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

Można zauważyć, że nasza macierz A posiada dominującą przekątną więc z powyższego twierdzenia otrzymujemy, że metoda iteracyjna jest zbieżna.

Przechodząc do drugiej części zadania w celu znalezienia odpowiedzi musimy wykonać iteracje Jacobiego przy następujących precyzjach $10^{-3}, 10^{-3}, 10^{-5}$.

Otrzymuje następujące wyniki:

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

Powyższe wyniki zostały uzyskane po wykonaniu poniższe programu w Pythonie:

```
import numpy as np


def jacobi_iteration_method(matrix, vector, precision):
    x = np.zeros(len(matrix[0])).reshape(-1, 1)
    D =  np.diag(matrix).reshape(-1,1)
    L_U = matrix-np.diagflat(np.diag(matrix))
    results = []
    norm_vector = np.linalg.norm(vector)
    norm_one = 2
    norm_two = 2
    i = 1

    while norm_one > precision or norm_two > precision:
        next_x = (vector-L_U@x)/D
        norm_one = np.linalg.norm(abs(x-next_x))
        norm_two = np.linalg.norm(matrix@x-vector)/norm_vector
        results.append((i, norm_one, norm_two))
        x =  next_x
        i += 1
    return x, results


def iteration_test(precision):
    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_method(A, b, precision)
    print(f'{res=}')


def main() -> None:
    iteration_test(10e-4)
    iteration_test(10e-5)
    iteration_test(10e-6)


if __name__ == '__main__':
    main()
```

Wnioski:

Po przeprowadzeniu analizy zbieżności metody iteracyjnej z pomocą odpowiedniego twierdzenia i sprowadzeniu problemu do oceny dominującej przekątnej, stwierdzenie zbieżności okazało się być zadaniem trywialnym. Jednakże, gdy przechodzimy do oceny ilości iteracji potrzebnych do osiągnięcia zadanej dokładności, nasuwa się refleksja, że wraz z kolejnymi iteracjami wzrost liczby iteracji zwiększa się niemalże proporcjonalnie o 1, maksymalnie 2. Niemniej jednak, brak dostatecznej ilości danych uniemożliwia dokładne określenie, jak liczba iteracji skaluje się przy dokładności rzędu 10^n. Pozostaje jedynie przypuszczać o jej charakterze.

Warto również zauważyć, że mimo niewielkiego wzrostu liczby iteracji w zależności od żądanej precyzji, używanie wyższej precyzji może okazać się korzystne. Choć to zwiększa koszt obliczeniowy, jest to szczególnie istotne w obszarach, takich jak informatyka medyczna czy projektowanie bardzo małych układów cyfrowych.

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/