Kacper Korta,

23.05.2023

Laboratorium 10

Układy równań liniowych – metody iteracyjne

1. Treść zadań

1. Dany jest układ równań liniowych Ax=b.

Macierz A o wymiarze nxn 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 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:

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

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

Dowieść, że proces iteracji dla układu równań:

10x1 – x2 + 2x3 – 3x4 = 0

x1 + 10x2 – x3 + 2x4 = 5

2x1 + 3x2 + 20x3 – x4 = -10

3x1 + 2x2 + x3 + 20x4 = 15

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

2. Rozwiązania zadań

Zadanie 1.

a) metoda Jacobiego

kod użyty do rozwiązania zadania

```
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))
    norm_b = np.linalg.norm(b)
    res = []
    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
        res.append((i, norm_one, norm_two))
        x = next_x
        i += 1

    return x, res
```

Wyniki przy precyzji 10^{-2}

```
iteracja ->  1 ||x^(t+1)-x^t|| -> 2.1412937418506806 (1/||b||) *||Ax^(t+1) - b|| -> 1.0 \\
iteracja ->  2 ||x^(t+1)-x^t|| -> 0.8073973019298369 (1/||b||) *||Ax^(t+1) - b|| -> 0.2865829768378003 \\
iteracja ->  3 ||x^(t+1)-x^t|| -> 0.28528460201990186 (1/||b||) *||Ax^(t+1) - b|| -> 0.11591915637036816 \\
iteracja ->  4 ||x^(t+1)-x^t|| -> 0.12395025062187699 (1/||b||) *||Ax^(t+1) - b|| -> 0.041749569973320216 \\
iteracja ->  5 ||x^(t+1)-x^t|| -> 0.044515386975211436 (1/||b||) *||Ax^(t+1) - b|| -> 0.017858706958605725 \\
```

Lliczba iteracji 5

Wyniki przy precyzji 10^{-4}

```
iteracja ->  1 ||x^(t+1)-x^t|| -> 1.6121500481586004 (1/||b||) *||Ax^(t+1) - b|| -> 1.0 \\
iteracja ->  2 ||x^(t+1)-x^t|| -> 0.383256546757803 (1/||b||) *||Ax^(t+1) - b|| -> 0.20091981550700772 \\
iteracja ->  3 ||x^(t+1)-x^t|| -> 0.12939684139183566 (1/||b||) *||Ax^(t+1) - b|| -> 0.060417750875852425 \\
iteracja ->  4 ||x^(t+1)-x^t|| -> 0.04443738088920277 (1/||b||) *||Ax^(t+1) - b|| -> 0.023916013560296687 \\
iteracja ->  5 ||x^(t+1)-x^t|| -> 0.019470433520561225 (1/||b||) *||Ax^(t+1) - b|| -> 0.008472433183748872 \\
iteracja ->  6 ||x^(t+1)-x^t|| -> 0.006904559935389525 (1/||b||) *||Ax^(t+1) - b|| -> 0.003654607353710237 \\
iteracja ->  7 ||x^(t+1)-x^t|| -> 0.0030515156554574937 (1/||b||) *||Ax^(t+1) - b|| -> 0.001312665768656256 \\
iteracja ->  8 ||x^(t+1)-x^t|| -> 0.0010818159756296078 (1/||b||) *||Ax^(t+1) - b|| -> 0.0005700282032082552 \\
iteracja ->  9 ||x^(t+1)-x^t|| -> 0.0004776007470187107 3 (1/||b||) *||Ax^(t+1) - b|| -> 0.0002050106126455025 \\
```

Lliczba iteracji 9

Wyniki przy precyzji 10^{-6}

```
iteracja ->  1 ||x^(t+1)-x^t|| -> 1.4796067420470587 (1/||b||) *||Ax^(t+1) - b|| -> 1.0 \\
iteracja ->  2 ||x^(t+1)-x^t|| -> 0.4904087063531552 (1/||b||) *||Ax^(t+1) - b|| -> 0.3796837023699187 \\
iteracja ->  3 ||x^(t+1)-x^t|| -> 0.2362788139456721 (1/||b||) *||Ax^(t+1) - b|| -> 0.1258828532416492 \\
iteracja ->  4 ||x^(t+1)-x^t|| -> 0.07634497716243106 (1/||b||) *||Ax^(t+1) - b|| -> 0.05894204910493912 \\
iteracja ->  5 ||x^(t+1)-x^t|| -> 0.03710765354973177 (1/||b||) *||Ax^(t+1) - b|| -> 0.019523090225321556 \\
iteracja ->  6 ||x^(t+1)-x^t|| -> 0.011938573598580107 (1/||b||) *||Ax^(t+1) - b|| -> 0.009212910565360926 \\
iteracja ->  7 ||x^(t+1)-x^t|| -> 0.00580896055959463 (1/||b||) *||Ax^(t+1) - b|| -> 0.0030482989998818198 \\
iteracja ->  8 ||x^(t+1)-x^t|| -> 0.0018669469136947218 (1/||b||) *||Ax^(t+1) - b|| -> 0.0014405805414359327 \\
iteracja ->  9 ||x^(t+1)-x^t|| -> 0.0009085718820812495 (1/||b||) *||Ax^(t+1) - b|| -> 0.0004765270614776723 7 \\
iteracja ->  10 ||x^(t+1)-x^t|| -> 0.0002919419690576445 (1/||b||) *||Ax^(t+1) - b|| -> 0.0002252652729169134 \\
iteracja ->  11 ||x^(t+1)-x^t|| -> 0.00014208229074084112 (1/||b||) *||Ax^(t+1) - b|| -> 7.451109233264834e-05 \\
iteracja ->  12 ||x^(t+1)-x^t|| -> 4.565174040391666e-05 (1/||b||) *||Ax^(t+1) - b|| -> 3.522519818522416e-05 \\
iteracja ->  13 ||x^(t+1)-x^t|| -> 2.221795227159037e-05 (1/||b||) *||Ax^(t+1) - b|| -> 1.1651325115645562e-05 \\
iteracja ->  14 ||x^(t+1)-x^t|| -> 7.138670659638569e-06 (1/||b||) *||Ax^(t+1) - b|| -> 5.508243411807643e-06 \\
```

Lliczba iteracji 14

b) metoda Czebyszewa

kod użyty do rozwiązania zadania

```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 Chebyshev_method(A: np.array, b: np.array, precision):
    x_prior = np.zeros(len(A[0])).reshape(-1, 1)
    t = []
    res = []
    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 = 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
        res.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, res
```

Wyniki przy precyzji 10^{-2}

```
iteracja ->  1 ||x^(t+1)-x^t|| -> 1.4124704472380403 (1/||b||) *||Ax^(t+1) - b|| -> 0.4298400651148423 \\
iteracja ->  2 ||x^(t+1)-x^t|| -> 0.611034759562798 (1/||b||) *||Ax^(t+1) - b|| -> 0.13692202764744013 \\
iteracja ->  3 ||x^(t+1)-x^t|| -> 0.18325576514047073 (1/||b||) *||Ax^(t+1) - b|| -> 0.036645513036898246 \\
iteracja ->  4 ||x^(t+1)-x^t|| -> 0.044004354850793376 (1/||b||) *||Ax^(t+1) - b|| -> 0.012087863080302266 \\
```

Lliczba iteracji 4

Wyniki przy precyzji 10^{-4}

```
iteracja ->  1 ||x^(t+1)-x^t|| -> 1.4702138375548857 (1/||b||) *||Ax^(t+1) - b|| -> 0.4931255008069957 \\
iteracja ->  2 ||x^(t+1)-x^t|| -> 0.6926887422112868 (1/||b||) *||Ax^(t+1) - b|| -> 0.15843745808702228 \\
iteracja ->  3 ||x^(t+1)-x^t|| -> 0.22302986659469912 (1/||b||) *||Ax^(t+1) - b|| -> 0.042551506340266367 \\
iteracja ->  4 ||x^(t+1)-x^t|| -> 0.05937321699105551 (1/||b||) *||Ax^(t+1) - b|| -> 0.012704646426686101 \\
iteracja ->  5 ||x^(t+1)-x^t|| -> 0.01793624219309045 (1/||b||) *||Ax^(t+1) - b|| -> 0.0034046969674517403 \\
iteracja ->  6 ||x^(t+1)-x^t|| -> 0.0047341495938963895 (1/||b||) *||Ax^(t+1) - b|| -> 0.0010113144654913535 \\
iteracja ->  7 ||x^(t+1)-x^t|| -> 0.001432299745236391 (1/||b||) *||Ax^(t+1) - b|| -> 0.00027285548931404205 \\
iteracja ->  8 ||x^(t+1)-x^t|| -> 0.0003781240873594038 (1/||b||) *||Ax^(t+1) - b|| -> 8.03569897845297e-05 \\
```

Lliczba iteracji 8

Wyniki przy precyzji 10^{-6}

```
iteracja ->  1 ||x^(t+1)-x^t|| -> 0.9272614994099495 (1/||b||) *||Ax^(t+1) - b|| -> 0.42197498917367954 \\
iteracja ->  2 ||x^(t+1)-x^t|| -> 0.5253192151068748 (1/||b||) *||Ax^(t+1) - b|| -> 0.12767157234457 \\
iteracja ->  3 ||x^(t+1)-x^t|| -> 0.14702693380965384 (1/||b||) *||Ax^(t+1) - b|| -> 0.032038697420531885 \\
iteracja ->  4 ||x^(t+1)-x^t|| -> 0.03499957409271684 (1/||b||) *||Ax^(t+1) - b|| -> 0.012110862275031286 \\
iteracja ->  5 ||x^(t+1)-x^t|| -> 0.013214348192082734 (1/||b||) *||Ax^(t+1) - b|| -> 0.0031840604830639806 \\
iteracja ->  6 ||x^(t+1)-x^t|| -> 0.003822992712216337 (1/||b||) *||Ax^(t+1) - b|| -> 0.0008989778522055866 \\
iteracja ->  7 ||x^(t+1)-x^t|| -> 0.0010824032743458867 (1/||b||) *||Ax^(t+1) - b|| -> 0.00019536445826656926 \\
iteracja ->  8 ||x^(t+1)-x^t|| -> 0.00022580937443493832 (1/||b||) *||Ax^(t+1) - b|| -> 6.816368719176591e-05 \\
iteracja ->  9 ||x^(t+1)-x^t|| -> 7.356151494682105e-05 (1/||b||) *||Ax^(t+1) - b|| -> 2.0517086999771974e-05 \\
iteracja ->  10 ||x^(t+1)-x^t|| -> 2.2946410823445138e-05 (1/||b||) *||Ax^(t+1) - b|| -> 6.156362448925619e-06 \\
iteracja ->  11 ||x^(t+1)-x^t|| -> 7.376274011351274e-06 (1/||b||) *||Ax^(t+1) - b|| -> 1.3958193497688423e-06 \\
```

Lliczba iteracji 11

Wnioski: Można łatwo zauważyć, że metoda Czebyszewa wymaga mniejszej liczby iteracji (najlepiej widać to ze wzrostem precyzji). Przy 1000krotnym zwiększeniu precyzji ilość iteracji przy metodzie Jacobiego wzrośnie 3krotnie a Czebyszewa 2krotnie. Na plus metody Jacobiego jest łatwość implementacji.

Zadanie 2.

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

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

Na podstawie powyższego twierdzenia konstruujemy tabele

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

Łatwo wywnioskować że nasza macierz A posiada dominującą przekątną
Więc z powyższego twierdzenia otrzymujemy, że metoda iteracyjna jest zbieżna

Korzystając z programu

```python
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))
    norm_b = np.linalg.norm(b)
    res = []
    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
```

```
        res.append((i, norm_one, norm_two))
        x = next_x
        i += 1

    return x, res



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

b = np.array([0, 5, -10, 15])
solves, res = jacobi_iteration(A, b, 10e-5)
print(len(res))
```

Otrzymane wyniki z programu:

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

Wnioski: po skorzystaniu z twierdzenia sprawdzenie zależności było łatwym zadaniem. Widzimy, że przy zwiększaniu precyzji liczba iteracji nie zmienia się znacząco. Jednakże mamy za mało danych żeby sprawdzić jak zmieni się liczba iteracji dla 10^n. Można zauważyć, że przy małym wzroście iteracji opłaca się zwiększać precyzję, ponieważ nie jest to kosztowne obliczeniowo, a dla pewnych badań ta różnica może być kluczowa.

3. Bibliografia

Wykład

https://home.agh.edu.pl/~funika/mownit/lab10/12_iteracyjne.pdf

https://en.wikipedia.org/wiki/Jacobi_method

https://en.wikipedia.org/wiki/Chebyshev_iteration

https://numpy.org/