Faktoryzacja LU

Agnieszka Ćwiękowska

Spis treści

1 Opis problemu

Napisany przeze mnie program realizuje rozkład LU dla dowolnej macierzy kwadratowej.

2 Listing programu

```
import numpy as np
import sys


def pivot(matrix):
    n = np.shape(matrix)
    ID = np.identity(n[0])
    for i in range(n[0]):
        maxm = abs(matrix[i][i])
        row = i
        for j in range(i, n[0]):
            if abs(matrix[j][i]) > maxm:
                maxm = abs(matrix[j][i])
                row = j
        if i != row:
            tmp = np.copy(ID[i])
            ID[i] = ID[row]
            ID[row] = tmp
    return ID


def facto_lu(matrix):
    size_of_matrix = np.shape(matrix)
    n = size_of_matrix[0]
    U = np.zeros((n, n))
    L = np.identity(n)
    P = np.identity(n)
    for i in range(n):
        j = i
        while j < n:
            sum = 0
            for k in range(i):
                sum += (L[i][k] * U[k][j])
            U[i][j] = matrix[i][j] - sum
            if i == j and U[i][j] == 0.0:
                tmp_P = pivot(matrix)
                P = np.dot(P, tmp_P)
                matrix = np.dot(tmp_P, matrix)
            else:
                j += 1
        for j in range(i + 1, n):
            if i != j:
                sum = 0
                for k in range(i):
                    sum += (L[j][k] * U[k][i])
```

```python
            L[j][i] = (matrix[j][i] - sum) / U[i][i]
    return L, U, P


def check_if(element, method):
    try:
        method(element)
    except ValueError:
        return False
    return True


def read_int():
    while True:
        size = input("Prosze podac rozmiar macierzy\n")
        if check_if(size, int):
            if int(size) > 0:
                return size
            else:
                print("Podana liczba jest mniejsza lub rowna zero")
        else:
            print("Podana wartosc nie jest liczba calkowita.")


def read_matrix(n):
    M = np.zeros((n, n))
    x = 0
    while x < n:
        print("Element w wierszu", x)
        m = input()
        if len(m.split()) != n:
            print("Zla liczba elementow w wierszu")
            x -= 1
        else:
            i = 0
            for y in m.split():
                if check_if(y, float):
                    M[x][i] = y
                    i += 1
                else:
                    print("Ktorys z elementow z wiersza nie jest liczba")
                    x -= 1
        x += 1
    if np.linalg.det(M) != 0:
        return M
    else:
        print("Nie da sie dokonac faktoryzacji dla tej macierzy, bo wyznacznik jest rowny zero (jest osobliwa)")
        sys.exit()
```

```python
def start():
    while True:
        print("Wybierz sposob wprowadzania danych:")
        print("1: Wczytaj z pliku (Podaj sciezke)")
        print("2: Wpisz do terminala")
        print("0: Wyjdz")
        n = int(input())
        if n == 0:
            break
        if n == 1:
            path = input("Podaj nazwe pliku:  ")
            f = open(path, 'r')
            l = [[int(num) for num in line.split()] for line in f]
            print("Macierz z pliku")
            print(l)
            L, U, P = facto_lu(l)
            print("L:\n", L, "\nU:\n", U, "\nP:\n", P, "\n")
            print("Sprawdzenie PxLxU: \n", np.dot(np.dot(P, L), U))
            print("Poczatkowa macierz:\n", l, "\n")
        if n == 2:
            n = int(read_int())
            print("Prosze podac kolejno elementy macierzy odzielajac 
               je spacja (np 0 1 2)")
            print("(w przypadku liczb dziesietnych uzywac kropki)")
            M = read_matrix(n)
            print("Wpisana macierz:\n", M, "\n")
            lu = facto_lu(M)
            L, U, P = lu
            print("L:\n", L, "\nU:\n", U, "\nP:\n", P, "\n")
            print("Sprawdzenie PxLxU: \n", np.dot(np.dot(P, L), U))
            print("Poczatkowa macierz:\n", M, "\n")


if __name__ == "__main__":
    start()

import unittest
import numpy as np
import faktoryzacja


class Test_lu(unittest.TestCase):
    def test1(self):
        matrix = np.array([[4, 4, 4], [1, 1, 18], [2, 10, 4]])
        lu = faktoryzacja.facto_lu(matrix)
        L, U, P = lu
        ll = np.array([[1., 0., 0.], [0.25, 1., 0.], [0.5,
            -0.11111111, 1.]])
        uu = np.array([[4., 4., 4.], [0., 9., 3.], [0., 0.,
            16.33333333]])
```

```
        pp = np.array([[1., 0., 0.], [0., 0., 1.], [0., 1., 0.]])
        self.assertEqual(L.all(), ll.all())
        self.assertEqual(U.all(), uu.all())
        self.assertEqual(P.all(), pp.all())
        self.assertEqual((np.dot(np.dot(P, L), U)).all(), matrix.all
            ())

    def test2(self):
        matrix = np.array([[5, 3, 2], [1, 2, 0], [3, 0, 4]])
        lu = faktoryzacja.facto_lu(matrix)
        L, U, P = lu
        ll = np.array([[1., 0., 0.], [0.2, 1., 0.], [0.6,
            -1.28571429, 1.]])
        uu = np.array([[5., 3., 2.], [0., 1.4, -0.4], [0., 0.,
            2.28571429]])
        pp = np.array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
        self.assertEqual(L.all(), ll.all())
        self.assertEqual(U.all(), uu.all())
        self.assertEqual(P.all(), pp.all())
        self.assertEqual((np.dot(np.dot(P, L), U)).all(), matrix.all
            ())

    def test3(self):
        matrix = np.array(
            [[6, 1, 0, 0, 0, 0], [1, 6, 1, 0, 0, 0], [0, 1, 6, 1, 0,
                0], [0, 0, 1, 6, 1, 0], [0, 0, 0, 1, 6, 1],
             [0, 0, 0, 0, 1, 6]])
        lu = faktoryzacja.facto_lu(matrix)
        L, U, P = lu
        ll = np.array([[1., 0., 0., 0., 0., 0.], [0.16666667, 1., 0.,
            0., 0., 0.], [0., 0.17142857, 1., 0., 0., 0.],
                        [0., 0., 0.17156863, 1., 0., 0.], [0., 0., 0.,
                            0.17157275, 1., 0.],
                        [0., 0., 0., 0., 0.17157287, 1.]])
        uu = np.array([[6., 1., 0., 0., 0., 0.], [0., 5.83333333, 1.,
            0., 0., 0.], [0., 0., 5.82857143, 1., 0., 0.],
                        [0., 0., 0., 5.82843137, 1., 0.], [0., 0., 0.,
                            0., 5.82842725, 1.],
                        [0., 0., 0., 0., 0., 5.82842713]])
        pp = np.array(
            [[1., 0., 0., 0., 0., 0.], [0., 1., 0., 0., 0., 0.], [0.,
                0., 1., 0., 0., 0.], [0., 0., 0., 1., 0., 0.],
             [0., 0., 0., 0., 1., 0.], [0., 0., 0., 0., 0., 1.]])
        self.assertEqual(L.all(), ll.all())
        self.assertEqual(U.all(), uu.all())
        self.assertEqual(P.all(), pp.all())
        self.assertEqual((np.dot(np.dot(P, L), U)).all(), matrix.all
            ())

    def test4(self):
        matrix = np.array([[0, 1, 0], [0, 0, 1], [1, 0, 0]])
```

```python
        lu = faktoryzacja.facto_lu(matrix)
        L, U, P = lu
        ll = np.array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
        uu = np.array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
        pp = np.array([[1., 0., 0.], [0., 1., 0.], [1., 0., 0.]])
        self.assertEqual(L.all(), ll.all())
        self.assertEqual(U.all(), uu.all())
        self.assertEqual(P.all(), pp.all())
        self.assertEqual((np.dot(np.dot(P, L), U)).all(), matrix.all
            ())

    def test5(self):
        matrix = np.array([[2, 1, 1], [4, -6, 0], [-2, 7, 2]])
        lu = faktoryzacja.facto_lu(matrix)
        L, U, P = lu
        ll = np.array([[1., 0., 0.], [2., 1., 0.], [-1., -1., 1.]])
        uu = np.array([[2., 1., 1.], [0., -8., -2.], [0., 0., 1.]])
        pp = np.array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
        self.assertEqual(L.all(), ll.all())
        self.assertEqual(U.all(), uu.all())
        self.assertEqual(P.all(), pp.all())
        self.assertEqual((np.dot(np.dot(P, L), U)).all(), matrix.all
            ())

    def test6(self):
        matrix = np.array([[0.2425, 0, 0.9701], [0, 0.2425, 0.9701],
            [0.2357, 0.2357, 0.9428]])
        lu = faktoryzacja.facto_lu(matrix)
        L, U, P = lu
        ll = np.array([[1., 0., 0.], [0., 1., 0.], [0.97195876,
            0.97195876, 1.]])
        uu = np.array([[0.2425, 0., 0.9701], [0., 0.2425, 0.9701],
            [0., 0., -0.94299439]])
        pp = np.array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
        self.assertEqual(L.all(), ll.all())
        self.assertEqual(U.all(), uu.all())
        self.assertEqual(P.all(), pp.all())
        self.assertEqual((np.dot(np.dot(P, L), U)).all(), matrix.all
            ())

    def test7(self):
        matrix = np.array([[0.2425, 0, 0.9701], [0, 0.2425, 0.9701],
            [0.2357, 0.2357, 0.9428]])
        lu = faktoryzacja.facto_lu(matrix)
        L, U, P = lu
        ll = np.array([[1., 0., 0.], [0., 1., 0.], [0.97195876,
            0.97195876, 1.]])
        uu = np.array([[0.2425, 0., 0.9701], [0., 0.2425, 0.9701],
            [0., 0., -0.94299439]])
        pp = np.array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
        self.assertEqual(L.all(), ll.all())
```

```
        self.assertEqual(U.all(), uu.all())
        self.assertEqual(P.all(), pp.all())
        self.assertEqual((np.dot(np.dot(P, L), U)).all(), matrix.all
            ())


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

3 Rozwiązanie problemu

Problem rozwiązałam implementując metodę Doolittle'a służącą do rozkładu macierzy na dwie macierze trójkątne L i U. Algorytm ten działa dla dowolnej macierzy kwadratowej n na n (w których nie dochodzi do dzielenia przez zero). Aby rozwiązać problem dzielenia przez zero wykorzystałam w swoim rozwiązaniu wybór elementu podstawowego (funkcja pivot).
Poniżej przedstawiam krótki opis algorytmu (wykorzystany z wikipedii)

Metoda Doolittle'a [edytuj | edytuj kod]

W metodzie tej równość $\mathbf{A} = \mathbf{LU}$ traktuje się jako układ n^2 równań z n^2 niewiadomymi[5]. Te niewiadome to elementy l_{ij} dla $i > j$ (elementy poniżej przekątnej), oraz u_{ij} dla $j \geqslant i$ (elementy na i powyżej przekątnej), przy założeniu, że na diagonali macierzy $\mathbf{L}$ znajdują się 1:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}.$$

Wyznaczanie kolejnych elementów macierzy $\mathbf{L}$ i $\mathbf{U}$ robi się naprzemiennie, tj. raz wyznacza wiersz macierzy $\mathbf{U}$, raz kolumnę macierzy $\mathbf{L}$.

Wzory ogólne na poszczególne elementy macierzy rozkładu przedstawiają się następująco:

dla wszystkich $i \in \{1, 2, \ldots, n\}$:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \text{ dla } j \in \{i,\ i+1, \ldots,\ n\},$$

$$l_{ji} = \frac{1}{u_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} u_{ki} \right) \text{ dla } j \in \{i+1,\ i+2, \ldots,\ n\}.$$

Z ostatniego równania wynika, że metoda nie zadziała, gdy $u_{ii} = 0$.

Liczba działań potrzebna do rozkładu[5]:

- mnożenia: $\frac{1}{3}n^3 - \frac{1}{3}n,$
- dodawania: $\frac{1}{3}n^3 - \frac{1}{2}n^2 + \frac{1}{6}n.$

Mój program zawiera również interfejs użytkownika. Można wpisać macierz, na której chcemy dokonać faktoryzacji do terminala jak i również wczytać ją z pliku.

4 Moje przykładowe wyniki

```
/usr/bin/python3 /home/aga/IdeaProjects/FaktoryzacjaLU/faktoryzacja.py
Wybierz sposob wprowadzania danych:
1: Wczytaj z pliku (Podaj sciezke)
2: Wpisz do terminala
0: Wyjdź
1
Podaj nazwę pliku:  file1.txt
Macierz z pliku
[[6, 1, 0, 0, 0, 0], [1, 6, 1, 0, 0, 0], [0, 1, 6, 1, 0, 0], [0, 0, 1, 6, 1, 0], [0, 0, 0, 1, 6, 1], [0, 0, 0, 0, 1, 6]]
L:
 [[1.         0.         0.         0.         0.         0.        ]
 [0.16666667 1.         0.         0.         0.         0.        ]
 [0.         0.17142857 1.         0.         0.         0.        ]
 [0.         0.         0.17156863 1.         0.         0.        ]
 [0.         0.         0.         0.17157275 1.         0.        ]
 [0.         0.         0.         0.         0.17157287 1.        ]]
U:
 [[6.         1.         0.         0.         0.         0.        ]
 [0.         5.83333333 1.         0.         0.         0.        ]
 [0.         0.         5.82857143 1.         0.         0.        ]
 [0.         0.         0.         5.82843137 1.         0.        ]
 [0.         0.         0.         0.         5.82842725 1.        ]
 [0.         0.         0.         0.         0.         5.82842713]]
P:
 [[1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]]
```

```
Wpisana macierz:
 [[2. 4. 6. 8.]
 [3. 5. 1. 7.]
 [2. 0. 9. 4.]
 [1. 4. 2. 8.]]

L:
 [[ 1.          0.          0.          0.        ]
 [ 1.5         1.          0.          0.        ]
 [ 1.          4.          1.          0.        ]
 [ 0.5        -2.         -0.48571429  1.        ]]
U:
 [[ 2.          4.          6.          8.        ]
 [ 0.         -1.         -8.         -5.        ]
 [ 0.          0.         35.         16.        ]
 [ 0.          0.          0.          1.77142857]]
P:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

Sprawdzenie PxLxU:
 [[2. 4. 6. 8.]
 [3. 5. 1. 7.]
 [2. 0. 9. 4.]
 [1. 4. 2. 8.]]
Poczatkowa macierz:
 [[2. 4. 6. 8.]
 [3. 5. 1. 7.]
 [2. 0. 9. 4.]
 [1. 4. 2. 8.]]
```

```
Element w wierszu 2
1 0 0
Wpisana macierz:
 [[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]

L:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
U:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
P:
 [[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]


Sprawdzenie PxLxU:
 [[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
Poczatkowa macierz:
 [[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]

Process finished with exit code 0
```
6: TODO   Terminal   9: Version Control

5 Bibliografia

- *Analiza numeryczna* Kincaid D.
- *Zaawansowany Python. Jasne, zwięzłe i efektywne programowanie* L.Ramalho
- https://www.wikipedia.org/