Tutorial - lucrul cu matrice și sisteme de ecuații liniare în Python

Borș David

Octombrie, 2020

Introducere

La sfârșitul acestui tutorial veți ști cum să lucrați cu matrice și sisteme de ecuații liniare în Python. Pentru parcurgerea tutorialului aveți nevoie de Python și de bibliotecile NumPy si SymPy.

Pentru utilizatorii pip și git instalarea bibliotecilor se realizează cu ajutorul comenzilor:

```
pip install numpy
git clone git://github.com/sympy/sympy.git
```

Mai multe detalii despre procesul de instalare pot fi găsite pe siteul oficial al bibiliotecii NumPy dar și al bibliotecii SymPy.

NumPy si SymPy ne vor permite să realizăm operații cu matrice foarte ușor, prin simple comenzi, dar și foarte eficient. Fără ajutorul acestor biblioteci procesul ar deveni unul ce solicită mai mult timp atât din partea programatorului cât și din partea calculatorului.

Pentru a ne folosi de funcțiile bibliotecilor odată ce le-am instalat va trebui ca la începutul codului sursă să le "importăm".

```
import numpy as np
import sympy as sp
```

- import numpy comunică interpreterului că vom folosi această bibliotecă
- as np îi transmite că ne vom referi la aceasta drept np si nu numpy. Facem acest lucru doar pentru o scriere mai ușoară a codului.
- Aceleași lucruri sunt valabile și pentru importarea bibliotecii SymPy.

Introducerea unei matrice

Primul pas pentru a lucra cu matrice este să le introducem în program. Acest tutorial prezintă două metode în care acest lucru se poate realiza. Prima metodă este introducerea manuală *"hard coded"*, prin care programatorul va introduce componentele matricei direct în codul sursă al programului. A doua metodă este mai elegantă și permite folosirea programului și de necunoscători folosindu-se de fișiere și o citire automată a acestora pentru introducerea matricei.

Metoda 1

Presupunem că dorim să introducem următoarea matrice:

$$\begin{matrix} 1 & 2 \\ 4 & 5 \end{matrix}$$

Pentru a o introduce și stoca într-o variabilă vom scrie următoarea linie de cod:

```
x = np.array([[1, 2], [4, 5]])
```

- x este numele variabilei în care vom stoca matricea, numele poate fi oricare, dar trebuie să respecte sintaxa Python;
- np face referire la biblioteca numpy, astfel interpreterul va ști că urmează să folosim o funcție din această librărie;
- **array()** este comanda prin care introducem vectori sau matrice cu ajutorul numpy.

Pentru a introduce matricea $\begin{matrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{matrix}$ vom folosi următoarea linie de cod:

```
y = np.array([[1, 3, 5], [2, 4, 6]])
```

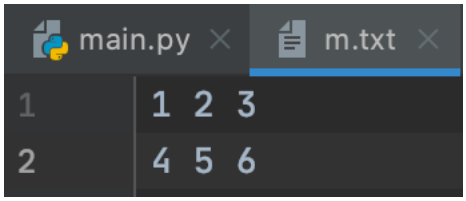
Metoda 2

Această metodă folosește același cod sursă indiferent de matricea introdusă.

```
z = np.loadtxt('m.txt')
```

- Observăm o linie de cod similară cu cea de la metoda 1 cu o singură diferență, funcția folosită acum este **loadtxt()**. Aceasta va citi fișierul m.txt și va extrage de acolo numerele.

Fișierul m.txt conține numerele pe care le dorim introduse în matrice cu spații între ele. Fiecare linie din fișier corespunde liniilor matricei, iar fiecare coloană coloanelor matricei.



Dacă vom afișa în consolă matricea introdusă cu ajutorul comenzii **print().**

```
z = np.loadtxt('m.txt')
print(z)
```

Vom primi următorul rezultat:

```
[[1. 2. 3.]
 [4. 5. 6.]]
```

Punctele apar deoarece numpy tratează numerele citite drept numere reale, astfel **1** este citit drept **1.0** și scris prescurtat **1.**. Deși această citire nu afectează calculele putem adăuga un argument funcției pentru a citi numerele drept numere întregi, **dtype=int**.

```
z = np.loadtxt('m.txt', dtype=int)
print(z)
```

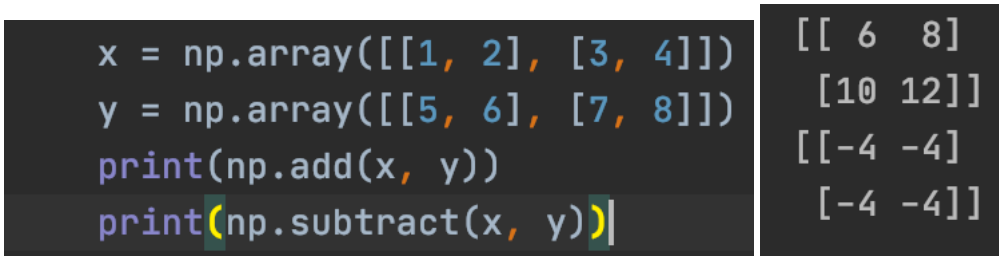
Acum se va afișa:

```
[[1 2 3]
 [4 5 6]]
```

Operații cu matrice

Operațiile cu matrice sunt predefinite în NumPy, astfel fiind foarte ușor de realizat.

Adunarea și scăderea



```
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])
print(np.add(x, y))
print(np.subtract(x, y))
```

```
[[ 6  8]
 [10 12]]
[[-4 -4]
 [-4 -4]]
```

- add() realizează adunarea matricelor
- substract() realizează scăderea matricelor

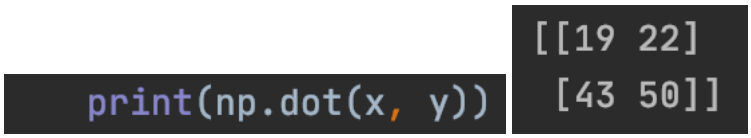
Produsul scalar

```
print(np.multiply(2, x))
```

```
[[2 4]
 [6 8]]
```

Produsul scalar se realizează prin folosirea funcției **multiply()**. Unul dintre argumente va fi un scalar, iar celălalt o matrice.

Înmulțirea a două matrice

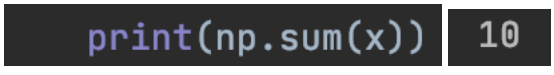


```
print(np.dot(x, y))
```

```
[[19 22]
 [43 50]]
```

Înmulțirea a două matrice se realizează cu ajutorul funcției **dot().** Pentru aceasta ambele argumente vor fi matrice.

Suma tuturor elementelor



```
print(np.sum(x))
```

```
10
```

Suma tuturor elementelor unei matrice se realizează prin funcția **sum()**, care primește ca argument o matrice.

Sume pe linii sau coloane

```
print(np.sum(x, axis=0))
print(np.sum(x, axis=1))
```

```
[4 6]
[3 7]
```

Tot funcția **sum()**, dar având încă un argument poate realiza sumele pe linii sau coloane ale unei matrice.
axis=0 va realiza sumele pe coloane. Coloana 1 conține elementele 1 și 3, 1 + 3 = 4. Coloana 2 conține elementele 2 și 4, 2 + 4 = 6.
axis=1 va realiza suma pe linii.

Transpusa unei matrice

```
print(x.T)
```

```
[[1 3]
 [2 4]]
```

Transpusa unei matrice va fi obținută prin **x.T**.

Inversa unei matrice

```
print(np.linalg.inv(x))
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Cu ajutorul funcției **np.lingalg.inv()** ce primește un argument, o matrice, putem afla inversa unei matrice.

Adjuncta unei matrice

```
print(np.matrix(x).H)
```

```
[[1 3]
 [2 4]]
```

Pentru a afla adjuncta unei matrice vom folosi funcția **matrix()** ce returnează un obiect al cărui atribut H conține chiar matricea adjunctă a matricei introduse ca argument pentru funcția **matrix()**.

Rangul unei matrice

```
print(np.linalg.matrix_rank(x))
```

```
2
```

Pentru a afla rangul unei matrice vom folosi funcția **matrix_rank()**. Aceasta se află în pachetul linalg al bibliotecii numpy și primește ca argument o matrice.

Determinantul unei matrice

Pentru a afla determinantul unei matrice vom folosi funcția **det()** care primește ca argument o matrice.

Rezolvarea sistemelor de ecuații liniare

Rezolvarea sistemelor compatibile determinate

Vom folosi drept exemplu următorul sistem de ecuații:

$$\begin{cases} 4x + 3y + 2x = 25 \\ -2x + 2y + 3x = -10 \\ 3x - 5y + 2x = -4 \end{cases}$$

Pentru a-l rezolva vom avea nevoie de două matrice:

$$A = \begin{matrix} 4 & 3 & 2 \\ -2 & 2 & 3 \\ 3 & -5 & 2 \end{matrix} \text{ și } B = \begin{matrix} 25 \\ -10 \\ -4 \end{matrix}$$

Mai întâi aceste două matrice trebuie introduse în program. Le vom introduce prin prima metodă folosită.

```
A = np.array([[4, 3, 2], [-2, 2, 3], [3, -5, 2]])
B = np.array([25, -10, -4])
```

Rezolvarea acestui sistem cu ajutorul NumPy este extrem de simplă și se realizează printr-o singură linie de cod.

```
print(np.linalg.solve(A,B))
```

Vom folosi funcția **solve()** ce primește ca argumente cele două matrice și va rezolva sistemul dat. Rezultatul obținut este:

```
[ 5.  3. -2.]
```

.

Rezolvarea sistemelor nedeterminate

Pentru rezolvarea acestor sisteme vom utiliza biblioteca SymPy. Vom folosi drept exemplu următorul sistem:

$$\begin{cases} x + y + z = 0 \\ x + 4y + 10z = 3 \\ 2x + 3y + 5z = 1 \end{cases}$$

```
x, y, z = sp.symbols('x y z')
res = sp.solve([sp.Eq(1 * x + 1 * y + 1 * z, 0),
                sp.Eq(1 * x + 4 * y + 10 * z, 3),
                sp.Eq(2 * x + 3 * y + 5 * z, 1)],
               [x, y, z])
print(res)
```

- prima linie de cod setează x, y, z ca simboluri, astfel șirul de caractere ‘x’ va fi o referire către variabila x
- **solve()** va rezolva o ecuație dată, primul argument este o listă ce conține ecuațiile sistemului, iar al doilea o listă ce conține necunoscutele
- **sp.Eq()** este o funcție ce ia ca argumente o ecuație, scrisă sub forma matematică, însă având rezultatul trecut ca un al doilea argument.

Codul de mai sus va afișa:

```
{y: 1 - 3*z, x: 2*z - 1}
```

.

Acest mod de rezolvare va rezolva și sistemele incompatibile. Vom folosi drept exemplu un sistem evident incompatibil:

$$\begin{cases} x - 4y = 3 \\ x - 4y = 0 \end{cases}$$

```
x, y = sp.symbols('x y')
res = sp.solve([sp.Eq(1 * x - 4 * y, 0),
                sp.Eq(1 * x - 4 * y, 3)],
               [x, y])
print(res)
```

Aplicând același algoritm de rezolvare se va afișa:

```
[]
```

.

Astfel se indică incompatibilitatea sistemului.