Tema3

October 18, 2022

1 Laborator 3 - tema 3

- **Termen de predare: 26 octombrie 2022, ora 20:00**

Se vor folosi type annotations pentru variabile, parametrii tuturor funcțiilor, tipuri de retur. Se vor folosi docstrings pentru toate funcțiile. Neîndeplinirea acestei cerințe duce la înjumătățirea punctajului pentru exercițiul în cauză.

Se acordă doua puncte din oficiu. Fișierul va fi denumit tema3_ia_nume_prenume.ipynb. Verificați înainte de trimitere faptul ca execuția celulelor de sus în jos funcționează corespunzător. Aserțiunile sunt obligatoriu a fi indeplinite. Suplimentar, puteti face verificari si pentru alte valori.

1. (1p) Pentru un vector de numere, care sunt toate pozitiile pe care apare valoarea minima? Folositi functii NumPy.

[ ]:
```
import numpy as np
v = np.array([1, 2, 3, 4, 4, 3, 1] * 2, dtype=np.float32)  # de adnotat
...
positions_min = ...

assert np.all(positions_min == [ 0,  6,  7, 13])
```

2. (1p) Construiti o functie care, primind o matrice, determina pe ce pozitii (linii, coloane) se afla valorile in intervalele [a, b] sau [c, d] date ca parametri. Functia va returna un tuplu de doi vectori: primul e cu indicii de linie, al doilea cu indicii de coloana unde se afla elementele cautate.

Exemplu:

[ ]:
```
def pos_values... # de adaugat typing annotations pentru parametri si tip de␣
 ↪retur
    ...

mat = np.arange(12).reshape(3, 4)
rows, cols = pos_values(mat, 2, 4, 6, 8)
assert np.all(rows == [0, 0, 1, 1, 1, 2])
assert np.all(cols == [2, 3, 0, 2, 3, 0])

mat = -np.arange(12).reshape(3, 4)
rows, cols = pos_values(mat, 2, 4, 6, 8)
```

```
assert np.all(rows == [])
assert np.all(cols == [])
```

3. (1p) Se da un vector cu numere intregi nenule. Sa se determine toti indicii `i` unde urmeaza o schimbare de semn. Codul se va scrie intr-o functie adnotata.

Exemplu:

```
v = [1, 2, -1, 2, 3, -1, 3, -10, -10, -10]
```

Trebuie sa rezulte vectorul de indici: `[1, 2, 4, 5, 6]`.

De luat in considerare ca: 1. Magnitudinea valorilor nu conteaza, doar semnele lor 1. Functia `where` gaseste indicii unde anumite proprietati sunt indeplinite

[ ]:
```
def jumping(v):  # de adnotat, doctstring
    assert np.all( v != 0), 'All values should be non-zero'
    ...
    return ...

# test 1
v = np.array([1, 2, -1, 2, 3, -1, 3, -10, -10, -10])
assert np.all(jumping(v) == [1, 2, 4, 5, 6])

# test 2: random vector, naive counting
v = np.random.randint(-100, 100, 1000)
# patching 0 values
v[v==0] = -1
# naive way of counting jumps:
positions = []
for i in range(len(v)-1):
    if v[i] * v[i+1] < 0:
        positions.append(i)

assert np.all(positions == jumping(v))
```

4. (1p) Generati o matrice aleatoare de dimensiune m*n care are m valori nan (not-an-number, https://numpy.org/doc/stable/reference/constants.html), la pozitii aleatoare. Determinati apoi suma elementelor non-nan din matrice in doua feluri (construiti doua functii). Comparati sumele returnate.

Exemplu:

```
a = get_matrix(3, 4)
# sa presupunem ca a este
a =
    1.5 2.2 np.nan
    4.3 np.nan np.nan
    10.1 5.0 -3.1
```

Suma elementelor este 20.0. Indicatii: folositi functia `np.random.choice` cu parametri adecvati pentru a alege indici in mod aleator, fara repetare.

[ ]:
```python
def get_matrix(m, n):  # de adnotat, docstring
    ...
    assert np.sum(np.isnan(a)) == m  # verificare: avem exact m nan-uri
    return a

def sum_1(a):  # de adnotat, docstring
    ...

def sum_2(a): # de adnotat, docstring
    ...

def naive_sum(a: np.ndarray) -> float:
    s = 0
    m, n = a.shape
    for i in range(m):
        for j in range(n):
            s += a[i, j] if not np.isnan(a[i, j]) else 0
    return s

m, n = 3, 4
a = get_matrix(m, n)

assert np.allclose(sum_1(a), naive_sum(a))
assert np.allclose(sum_2(a), naive_sum(a))

# pentru studenti: de ce e nevoie de allclose si nu se face comparatie cu == ?
```

5. (1p) Folosnd functia `np.convolve`, sa se determine media alunecatoare (moving average) pentru un vector de 10000 de numere aleatoare; lungimea ferestrei alunecatore se da ca parametru al functiei construite de voi. Comparati rezultatele si viteza de executie cu a functiaie `naive_moving_average`.

Referinte: 1. Moving average 1. np.convolve: How to Use Numpy convolve() Method

[ ]:
```python
def moving_average_fast(x, w):  # de adnotat, docstring
    assert x.ndim == 1, 'One dimension numpy array'
    assert w > 1, 'Average of at least two values'
    ...
    return ...

def moving_average_slow(x, w): # de adnotat, docstring
    assert x.ndim == 1, 'One dimension numpy array'
    assert w > 1, 'Average of at least two values'
    result = np.zeros(x.shape[0]-w+1)
    for i in range(x.shape[0]-w+1):
        result[i] = np.mean(x[i:i+w])
    return result
```

```
len_vec, w = 10000, 10
x = np.random.rand(len_vec)

a = moving_average_fast(x, w)
b = moving_average_slow(x, w)

assert len(a) == len(b), 'The two methods should return vectors of the same␣
 ↪lengths'
assert np.allclose(a, b), 'The two methods should return close vectors'

%timeit moving_average_fast(x, w)
%timeit moving_average_slow(x, w)
```

6. (1p) Se dau vectorii de valori $x = [x_0 \dots x_{n-1}]$, $t = [t_0, \dots t_n - 1]$, cu $t[0] < t[1] < ....$ Sa se calculeze intr-o maniera vectorizata vectorul v ale carui componente sunt:

$$v[i] = \frac{x[i+1] - x[i]}{t[i+1] - t[i]}$$

In prealabil sa se verifice (vectorizat sau functii NumPy) ca vectorul t este format doar din valori strict crescatoare. Comparati rezultatele si viteza de executie cu o implementare care construieste vectorul v pas cu pas.

Optional: incercati si o implementare folosind Numba.

[ ]:
```
import numpy as np

n = 100000
x = np.random.rand(n)
t = np.random.randint(1, 10, n)
t = t.cumsum()

def is_increasing(t) -> bool: # de adnotat, docstring
    ...
    return ...

assert is_increasing(t), 't should be an increasing vector'

def fast_v(x, t): # de adnotat, docstring
    assert is_increasing(t), 'non increasing values'
    ...
    return ...

def slow_v(x, t): # de adnotat, docstring
    assert is_increasing(t), 'non increasing values'
    ...
    return ...

v1 = fast_v(x, t)
```

```
v2 = slow_v(x, t)

assert v1.shape == v2.shape
assert np.allclose(v1, v2)

%timeit fast_v(x, t)
%timeit slow_v(x, t)
```

7. (2p) Se da o matrice a cu numere reale.
   1. Sa se calculeze o alta matrice b care are valorile calculate astfel:

$$b[i,j] = \frac{a[i,j] - MIN}{MAX - MIN}$$

      unde MAX si MIN sunt valoarile minima, respectiv maxima din a. Tranformarea se face printr-o functie care preia o matrice si returneaza un tuplu compus din matricea b si valorile MIN si MAX
   2. Verificati ca in urma acestei transformari matricea b are valorile in intervalul $(-epsilon, 1 + epsilon)$, unde $epsilon = 1e - 5$.
   3. Daca se dau b, MIN, MAX, care este transfomarea inversa prin care obtinem matricea a originara? Verificati ca valorile lui a si $a_{reconstituit}$ sunt aproximativ egale.

[ ]:
```
from typing import Tuple

import numpy as np
m, n = 100, 200

a = np.random.randn(m, n)

# rezolvare pct A
def min_max_scale(x): # de adnotat, docstring
    ...
    return ..., ..., ...

# rezolvare pct B
def check_scaling(y: np.ndarray, eps = 1e-5): # de adnotat, docstring
    assert eps > 0
    return ...

assert check_scaling(min_max_scale(a)[0])

# rezolvare pct C
def inverse_min_max_scale(y: np.ndarray, MIN: float, MAX: float) # de adnotat,␣
 ↪docstring
    assert MAX > MIN
    return ...

b, MIN, MAX = min_max_scale(a)
assert np.allclose(a, inverse_min_max_scale(b, MIN, MAX))
```