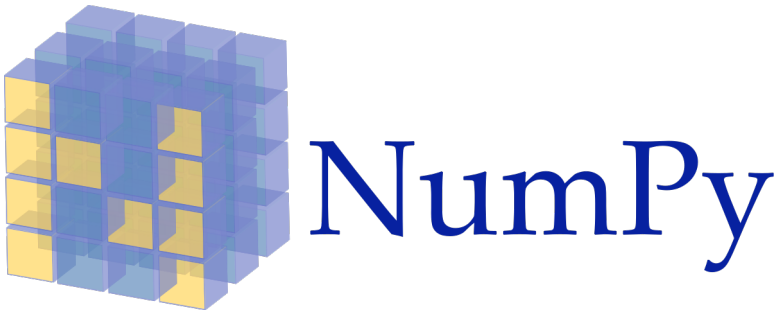
Patrick Nsukami

1/12/24

Contents



Numpy

NumPy, *Numerical Python*, est une bibliothèque Python libre et open source créée pour le calcul numérique et adoptée par la communauté scientifique. En effet, c'est une bibliothèque qui propose de nombreux outils pour la création et la manipulation de vecteurs, de polynômes et de tableaux multidimentionnels. Mais surtout, c'est une bibliiothèque qui est à la base de de la construction de nombreuses autres bibliothèques.

Numpy est développé par plusieurs developpeurs à travers la planète. Il existe plusieurs canaux pour se connecter, apprendre et partager avec les autres membres de cette communauté. Vous êtes vivement encouragé à faire partie de cette communauté.

Pourquoi NumPy?

1. Consommation mémoire

Un tableau Numpy consomme bien moins de mémoire qu'une liste:

```
import numpy as np
import sys

rng= range(1000)
print("Size of each element inside the list in bytes: ",sys.getsizeof(rng))
print("Size of the entire list in bytes: ",sys.getsizeof(rng)*len(rng))


arr= np.arange(1000)
print("Size of each element inside the Numpy array in bytes: ",arr.itemsize)
print("Size of the entire Numpy array in bytes: ",arr.size*arr.itemsize)
```

```
Size of each element inside the list in bytes:  48
Size of the entire list in bytes:  48000
Size of each element inside the Numpy array in bytes:  8
Size of the entire Numpy array in bytes:  8000
```

2. Vitesse d'exécution:

Parce que Numpy consomme moins de ressources mémoire, parce que les types manipulés dans les tableaux sont prédéfinis, le logiciel est optimité pour s'exécuter rapidement:

```
import numpy
import time

list1 = range(1000000)
list2 = range(1000000, 2000000)

array1 = numpy.arange(1000000)
array2 = numpy.arange(1000000, 2000000)


start = time.time()
print(sum([(a * b) for a, b in zip(list1, list2)]))
end =  time.time()
print(f"Temps d'execution pour le produit scalaire: {end - start} seconds")

start = time.time()
print(np.dot(array1, array2))  # vectorized computation, no need to write an explicit for loop
end = time.time()
print(f"Temps d'execution pour le produit scalaire: {end - start} seconds")
```

```
833332333333500000
Temps d'execution pour le produit scalaire: 0.11943793296813965 seconds
833332333333500000
Temps d'execution pour le produit scalaire: 0.0015277862548828125 seconds
```

3. Vectorisation des opérations:

On appelle vectorisation, le principe consistant à remplacer des boucles par des calculs vectoriels:

```
import numpy
import time

arr = np.random.randint(1000, size=10**6)

def loop():
  array = [element + 1 for element in arr]

def vectorization():
  array = arr + 1
```

Mesurons le temps d'execution des deux fonctions définies plus haut:

```
Temps d'execution utilisant la boucle: 0.135242701
Temps d'execution utilisant la vectorization: 0.001270771
```

Installation de Numpy:

Il existe plusieurs manières d'installer Numpy. Dans le doute, toujours vous référer à la documentation officielle.

Qu'est ce que le ndarray?

Numpy apporte cette structure de données nommée ndarray. Nous retiendrons que cette structure de données n'a rien à voir avec la classe array.array de la bibliothèque standard. Contrairement aux listes, cette structure de données ne peut changer de taille, et le type de ses éléments doit être homogène.

Voici les attributs important d'un objet de type ndarray:

ndarray.ndim	*le nombre d'axes (dimensions).*
ndarray.shape	*un tuple d'entiers indiquant la taille de chaque dimension*
ndarray.size	*le nombre total d'éléments.*
ndarray.dtype	*le type des éléments.*
ndarray.itemsize	*la taille en octets de chaque élément.*
ndarry.data	*les éléments.*

Illustration:

```python
import numpy as np
arr = np.arange(15).reshape(3, 5)
print(arr)
print("forme:", arr.shape)
print("dimensions:", arr.ndim)
print("type des données:", arr.dtype)
print("taille totale du tableau:", arr.itemsize)
print("nombre élements", arr.size)
print("type objet:", type(arr))
```

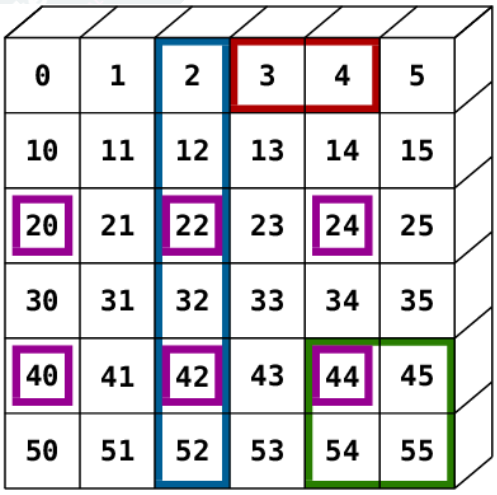
Création:

Il existe une infinité d'options permettant de créer des tableaux Numpy. Nous ne pourrons pas toutes les explorer ici. Voici quelques exemples:

```
>>> # CREATE AN ARRAY FILLED WITH ARBITRARY VALUES
>>> np.empty((2, 2, 4), int)

>>> # CREATE ARRAY FILLED WITH ZEROS
>>> np.zeros((3,3,3), dtype=np.int8)

>>> # CREATE ARRAY FILLED WITH ONES
>>> np.ones((2,3))

>>> # CREATE ARRAY WITH THE DIAGONAL FILLED WITH ONES
>>> np.eye(5)

>>> # CREATE ARRAY FILLED WITH 4 EVERYWHERE
>>> np.full((5,6), 4)

>>> # CREATE AN ARRAY CONTAINING 48 ELEMENTS
>>> # CHANGE THE SHAPE, WE WANT 2 DIMENSIONS
>>> np.arange(48).reshape(8,6)

>>> # CREATE ARRAY CONTAINING 20 ELEMENTS BETWEEN 10 AND 20, EVENLY SPACED
>>> np.linspace(10, 20, 20)
```

Indexing et slicing

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



Le Slicing est le mécanisme grâce auquel vous pouvez extraire un sous-ensemble d'un ensemble. En Python, l'opérateur slicing attend 3 paramètres, dont 2 optionnels. La syntaxe est: variable[début:fin:pas]. Le slicing des tableaux Numpy fonctionne de la même manière qu'avec listes en Python.

```
>>> # CREATE AN ARRAY OF 2 DIMENSIONS: (3,5)
>>> arr = np.array([[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15]])
```

```
>>> arr
Out >>>
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])

>>> # PRINT THE FIRST COLUMN
>>> arr[:,0]
Out >>> array([ 1,  6, 11])

>>> # PRINT THE FIRST LINE
>>> arr[0,:]
Out >>> array([1, 2, 3, 4, 5])

>>> # PRINT THE FIRST LINE (SIMPLIFIED)
>>> arr[0]
Out >>> array([1, 2, 3, 4, 5])

>>> # PRINT THE 3RD ELT OF THE 3RD COLUMN
>>> arr[:,2][2]
Out >>> 13

>>> # PRINT THE 3RD COLUMN AND THE FOURTH COLUMN
>>> arr[:,2:4]
Out >>>
array([[ 3,  4],
       [ 8,  9],
       [13, 14]])

>>> # PRINT THE 2ND & 3RD LINES - 3RD & 4TH COLUMNS
>>> arr[1:3,2:4]
Out >>>
array([[ 8,  9],
       [13, 14]])
```

Filtering:

Une opération que nous aurons souvent à réaliser est de créer un tableau t2, en s'appuyant sur un déjà existant t1, juste en selectionnant les éléments de t1 qui respectent un certain critère. Parmi les nombreuses méthodes existantes permettant de filtrer un tableau Numpy, nous pouvons regarder le *boolean mask slicing* et la méthode `where()`.

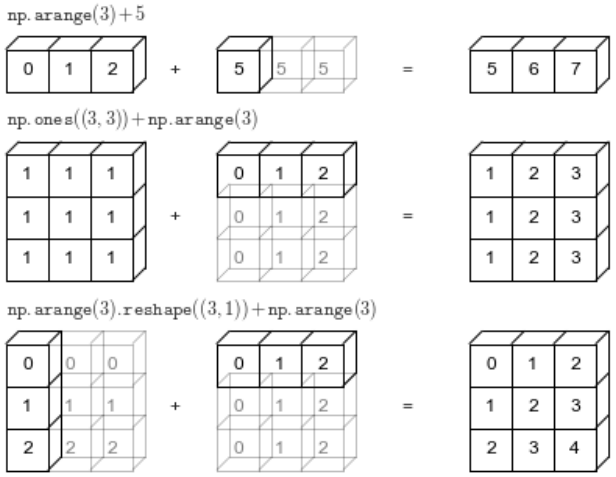
```
import numpy as np

arr = np.array([1, 4, 2, 7, 9, 3, 5, 8])
clause = arr < 5
print(arr[clause]) # conserver uniquement les elts du tableau inf à 5
```

Même exemple que précédent, cette fois ci avec la fonction where:

```
[1 2 3]
```

```
[11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
 35 36 37 38 39]
[15 17 19 21 23 25 27 29 31 33 35 37 39]
```

Broadcasting:



Le broadcasting est le mécanisme utilisé par Numpy pour manipuler ensemble, des tableaux n'ayant pas la même forme. Le tableau ayant le plus petit nombre de dimension sera transformé de manière à avoir la même forme que le second tableau.

```
[5 6 7]
```

```
[[5 6 7]
 [6 7 8]
 [7 8 9]]
```

Pour aller plus loin:

Le sujet est extrêmement vaste. Il serait prétentieux d'espérer le maitriser en quelques heures. Pour en découvrir d'avantage, je vous recommande et vous remercie d'explorer les liens suivants:

- Tutoriel officielle
- Ndarray
- Slicing, reshaping et indexation avancée
- Vectoriasation
- Broadcasting
- Slicing NumPy Arrays like an expert
- Introduction to Numerical Computing With NumPy - Logan Thomas | SciPy 2022