INTRODUCTION À NUMPY

À la fin de ce chapitre, je sais :

- ☞ importer la bibliothèque Numpy
- ☞ utiliser un tableau Numpy mono et multidimensionnel
- ☞ écrire des opérations élément par élément

A Pourquoi Numpy? --→ HORS PROGRAMME

Le cur du langage Python ne dispose pas d'un type tableau **statique**, c'est-à-dire de taille fixe. Les concepteurs de ce langage ont créé un type hybride qui se comporte à la fois comme une liste et un tableau : la liste Python. L'implémentation est réalisée grâce à un tableau **dynamique**, c'est-à-dire un tableau dont la taille évolue en fonction du besoin au cours du programme. Néanmoins, cette évolution de taille a un impact sur la rapidité des opérations. La liste Python est très souple et (trop) pratique : on peut la tordre pour faire, au moins en apparence, à peu près ce que l'on veut, même si cela n'est pas toujours efficace. Cependant, elle n'est pas adaptée au calcul numérique : les listes imbriquées ne sont pas des tableaux multidimensionnels et trop de flexibilité rend le code propice à des bogues à répétitions difficiles à cerner.

ă Pour le calcul numérique, il est important de pouvoir utiliser des tableaux statiques pour représenter des vecteurs, desămatrices ou des tableaux à n dimensions car les opérations qu'on souhaite réaliser se font soit élément par élément (calcul vectoriel) soit par matriciellement. Or, ces calculs vectoriels ou matriciels ne sont cohérents que si les tailles des opérandes correspondent. La taille fixe permet de garantir la cohérence des calculs effectués :ăon ne peut pas par inadvertance ajouter un élément à un tableau statique. D'un point de vue machine, **l'espace mémoire alloué peut rester le même tout au long du calcul**, le chargement en mémoire est donc plus facile à gérer, les modifications et les accès plus simples et plus rapides.

Les plus-values de Numpy, dans ce contexte, sont les suivantes :ă

1. le type `array` est un tableau statique multidimensionnel. En machine, tous les éléments sont stockés de manière contigüe, à plat : l'aspect multidimensionnel n'est qu'une vision du tableau procurée par le génie logiciel, du sucre syntaxique. C'est pourquoi il est très facile de changer de redimensionner un tableau Numpy car il n'est pas modifié en mémoire par cette opération.ăă

2. les types simples Numpy sont plus riches que les types de base Python : on peut par exemple utiliser des types entiers non signés (np.uint8) pour stocker une information sur un octet plutôt que d'utiliser un entier signé sur 64 bits, ou utiliser des flottants simple précision plutôt que double. Gain de place, gain de temps également dans les opérations.ă
3. les calculs vectoriel (élément par élément) et matriciel proposés par Numpy présentent deux avantages :
   - les opérations vectorielles `+,-,*,/,...` sont pré-compilées en langage machine et donc le calcul est exécuté très rapidement par l'interpréteur Python car il n'est pas interprété. Les codes compilés utilisent les instructions vectorielles des processeurs, des instructions capables de réaliser plusieurs opérations du même type (addition par exemple) sur des opérandes différentes en un seul cycle d'horloge.
   - la syntaxe proposée via la surcharge d'opérateur permet de s'affranchir de l'utilisation des boucles et d'écrire une formule vectorielle avec la même syntaxe qu'une formule scalaire. C'est à cette condition d'écriture d'ailleurs que les calculs peuvent être accélérés par les opérations pré-compilées.ă

P Numpy est un bibliothèque logicielle Python dédiée au calcul numérique. Elle n'est pas explicitement au programme. Certaines épreuves de concours en interdisent son usage quand d'autres l'exigent explicitement. Il faut rester vigilant à la lecture de l'épreuve.

STOP Le paragraphe précédent est important pour l'épreuve d'informatique!

Numpy est très utilisé dans le monde entier. Les raisons principales sont les suivantes :

1. Numpy est open source,
2. Numpy est optimisée avec un cœur compilé en langage C (compilé, pas interprété donc plus rapide),
3. Numpy s'interface facilement avec d'autres langage de calcul scientifique (C et Fortran notamment),
4. Numpy procure les `array`, une structure de données de type tableau multidimensionnel de dimensions fixes. Cela complète habilement le langage Python qui ne propose que des listes qui sont implémentées par des tableaux dynamiques.
5. Numpy propose des opérations sur les tableaux élément par élément. Cela permet à la fois d'éviter d'écrire des boucles et de paralléliser les opérations.
6. Numpy permet l'écriture de calculs matriciels.
7. Ces deux derniers points font que les formules mathématiques apparaissent écrites quasi-naturellement dans le code.

B Importation

On peut importer comme on le désire la bibliothèque Numpy. Cependant une convention très utilisée fait qu'on l'importe souvent comme ceci :

```
1  import numpy as np
2
3  t = np.zeros(42)
```

C Créer des vecteurs, des matrices ou des tableaux

■ **Définition 1 — Vecteur numpy.** Le terme *vecteur* désigne des `array numpy` de dimension 1.

■ **Définition 2 — Matrice numpy .** Le terme *matrice* désigne les `array` numpy de dimension deux.

Lorsque le tableau possède plus de deux dimensions, on parle de tableau.

On peut créer, comme le code 1 le montre, un vecteur, une matrice ou un tablau numpy à partir :

- du constructeur np.array(),
- d'une liste ou d'une liste imbriquée,
- de fonctions spéciales en précisant les dimensions du tableau et la valeur d'initialisation des cases du tableau,
- d'un tableau existant : la tableau créé aura les mêmes dimensions. On précise la valeur à laquelle on veut initialiser les cases.

Code 1 – Créer des tableaux Numpy

```
1  import numpy as np
2
3  v1 = np.array([1, 2, 3])
4  print(v1.shape, v1)  # (3,) [1 2 3]
5  v2 = np.array([[10], [20], [30]])
6  print(v2.shape, v2)  # (3, 1) [[10] [20] [30]]
7
8  v3 = np.zeros((1, 3))
9  print(v3.shape, v3)  # (1, 3) [[0. 0. 0.]]
10 v4 = np.ones((1, 7))
11 print(v4.shape, v4)  # (1, 7) [[1. 1. 1. 1. 1. 1. 1.]]
12
13 m1 = np.zeros((5, 5))
14 print(m1.shape, m1)
15 # (5, 5)
16 # [[0. 0. 0. 0. 0.]
17 # [0. 0. 0. 0. 0.]
18 # [0. 0. 0. 0. 0.]
19 # [0. 0. 0. 0. 0.]
20 # [0. 0. 0. 0. 0.]]
21
22 m1 = np.ones((2, 3))
23 print(m1.shape, m1)
24 # (2, 3)
```

```
# [[1. 1. 1.]
# [1. 1. 1.]]

t0 = np.zeros((42, 21, 84, 3))
print(t0.shape)  # (42, 21, 84, 3)

t1 = np.zeros_like(v1)
print(t1.shape, t1)  # (3,) [0 0 0]
t2 = np.ones_like(v3)
print(t1.shape, t1)  # (1, 3) [[1. 1. 1.]]
t3 = np.full_like(m1, 42)
print(t1.shape, t1)
# (2, 3)
#  [[42. 42. 42.]
#  [42. 42. 42.]]

s = np.zeros((4, 4, 500)).shape
print(s[0], s[1], s[2])  # 4 4 500

r = np.arange(0, 1, 0.1) # like range but for float !
print(r)  # [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
time = np.linspace(0, 1, 11)
print(time) # [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

P Comme le montre la fin du code 1, la fonction `shape` renvoie un tuple qui spécifie les dimensions du tableau. Comme c'est un tuple, on peut accéder à la taille de chaque dimension par l'opérateur `[]`. Pour un matrice, par exemple, on obtient le nombre de lignes et le nombre de colonnes.

D Accéder aux éléments d'un tableau

Les cases d'un `array` numpy sont numérotées à partir de zéro, comme pour les listes Python. Le tronçonnage ainsi que l'indexage négatif sont également disponibles comme le montre le code 2.

P Cependant, la syntaxe de la manipulation des tableaux multidimensionnels est différente de celle des listes imbriquées : une virgule sépare les indices des dimensions différentes. Il faut rester vigilant et ne pas confondre ces syntaxes.

STOP Le paragraphe précédent est important pour l'épreuve d'informatique!

Code 2 – Accéder aux éléments d'un tableau numpy

```
import numpy as np

m = np.array([[1, 2], [3, 4]])
print(m)
```

```
# [[1 2]
#  [3 4]]
print(m[0, 0], m[0, 1], m[1, 0], m[1, 1])  # 1 2 3 4
# slicing
print(m[:, 0])  # [1 3]
print(m[:, 1])  # [2 4]
print(m[0, :])  # [1 2]
print(m[1, :])  # [3 4]

# negative indexing
print(m[-1, -1])
# reversing
print(m[::-1])
# assignment
m[0, :] = 42
print(m)
# [[42 42]
#  [ 3  4]]

# resizing
m = m.reshape((1, 4))
print(m)  # [[1 2 3 4]]
```

P Comme on peut le voir à la fin du code 2, on peut facilement redimensionner un tableau, avec ses éléments dedans. Cela vient du fait que numpy stocke toujours les éléments d'un tableau de manière contiguë en mémoire et ce, quelle que soit les dimensions du tableau.

La bibliothèque stocke aussi avec le tableau ses dimensions apparentes pour l'utilisateur. L'accès aux éléments au travers ses dimensions `t[3,2]` n'est donc qu'une commodité (du sucre syntaxique) fournie par le génie logiciel de la bibliothèque. Un indice réel est calculé par numpy à partir de [3,2] pour accéder à la case l'élément dans la zone contiguë en mémoire.

Vocabulary 1 — Syntactic sugar ⇜⇝ Sucre syntaxique. Élément de la syntaxe d'un langage visant à faciliter l'utilisation, l'écriture et la lecture associées à un concept. Cela adoucit le travail humain.

E Opérations élément par élément

On a très souvent besoin d'appliquer les mêmes formules à tous les éléments d'un tableau, à toute une série de données. Par exemple, pour filtrer un signal, le moyenner, pour changer de couleur tous les pixels d'une image ou tout simplement pour calculer tous le prix TTC de tous les éléments d'une liste de prix HT.

Le succès de numpy repose principalement sur la capacité à opérer des calculs sur un tableau comme si on les effectuait sur une variable scalaire (cf. code 3). **Il devient alors inutile de faire une boucle pour traiter tous les éléments du tableau** et l'écriture et la lecture des formules physiques et mathématiques s'en trouvent facilitées :

Code 3 – Opérer élément par élément

```
import numpy as np

x = np.ones(4)
print(x + x)  # [2. 2. 2. 2.]
print(x - x)  # [0. 0. 0. 0.]
print(4 * x * x / 5)  # [0.8 0.8 0.8 0.8]
print(x / x)  # [1. 1. 1. 1.]

r = np.arange(0, 1, 0.3)
print(r)  # [0.  0.3 0.6 0.9]
s = np.pi * r ** 2
print(s)  # [0. 0.28274334 1.13097336 2.54469005]

time = np.linspace(0, 1, 11)
print(time)  # [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
signal = 5 * np.cos(2 * np.pi * 7 * time)
print(signal)
# [ 5.         -1.54508497 -4.04508497  4.04508497  1.54508497 -5.
#  1.54508497  4.04508497 -4.04508497 -1.54508497  5.        ]
```

P Les opérateur arithmétiques `+,-,*,/,*` utilisés dans le code 3 sont des opérateurs qui agissent sur des tableaux numpy. En informatique, on dit alors qu'on a surchargé les opérateurs `+,-,*,/,*` pour qu'ils puissent agir sur d'autres données que les `int` ou les `float`.

F Opérations matricielles

Numpy possède également la multiplication matricielle comme le montre le code 4. C'est l'opérateur `@` qui permet de réaliser cette opération. Il est également très facile de faire de l'algèbre linéaire [1] avec le module `numpy.linalg`.

Code 4 – Calcul matriciel

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
X = np.array([[.2], [.1]])
B = np.array([[.1, .2, .3], [.4, .5, .6]])
U = np.array([[0.1], [0.2], [0.3]])

Xp = A @ X + B @ U
print(Xp.shape)  # (2, 1)
print(Xp)
# [[0.54]
# [1.32]]
```

1. Calculer une matrice inverse par exemple.

G Types de données

La bibliothèque numpy fournit également des types de données supplémentaires à Python. Ce sont en fait des types utilisés par le langage C qui sous-tend les calculs. On y trouve notamment les types `int` non signés désignés par `uint` ou les `float` sur 32 bits.

De nombreuses données concrètes sont représentées par des entiers non signés. C'est le cas par exemple des pixels d'une image dont on code généralement l'intensité sur huit bits, c'est à dire par une valeur comprise entre 0 et $2^8 - 1 = 255$. Numpy propose le type `np.uint8` pour représenter ce type de données.

Code 5 – Types de données numpy

```
import numpy as np

unsigned_integer = np.uint8(42)
print(unsigned_integer) # 42
image = np.zeros((1024, 512), dtype=np.uint8)
image[:] = 237
print(image)
#[[237 237 237 ... 237 237 237]
# [237 237 237 ... 237 237 237]
# [237 237 237 ... 237 237 237]
# ...
# [237 237 237 ... 237 237 237]
# [237 237 237 ... 237 237 237]
# [237 237 237 ... 237 237 237]]
```

R Utiliser le type de données le plus adapté à la nature de la donnée est très important : cela permet d'économiser radicalement l'espace mémoire et d'accélérer les calculs.

H Autres fonctions

Numpy regorge de fonctionnalités. N'hésitez pas à consulter la documentation en ligne. Cette bibliothèque est complétée par `scipy`, bibliothèque scientifique.

- fonctions numpy standards à connaître : `min`, `max`, `std`, `mean`, `sum`,
- `unique` : permet de ne conserver qu'un seul exemplaire de chaque valeur dans un tableau,
- `argmax`, `argmin` : permet de récupérer l'indice du maximum ou du minimum d'un tableau,
- `concatenate` agrège deux tableaux,
- le module `numpy.random` et notamment `shuffle` pour mélanger un tableau.

■ **Exemple 1 — Normalisation d'un ensemble de données par colonne.** On suppose qu'on dispose d'un ensemble de données sous la forme d'un tableau numpy de dimention `(n,m)`. Chaque colonne représente un paramètre que l'on souhaite normaliser : centrer en zéro et d'écart type égal à un.

Pour chaque échantillon x_{ij} de la ligne i et colonne j, on veut :

$$x_{ij}^{norm} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

où μ_j et σ_j sont les moyennes et écart-type de la colonne j.

On peut réaliser ceci en une seule instruction :

```
1  import numpy as np
2
3  def normalized(data):
4      return (data — np.mean(data, axis=0)) / np.std(data, axis=0)
5
6
7  data = np.random.random((100, 5))
8  data = normalized(data)
9
10 print(np.max(data, axis=0), np.min(data, axis=0), np.mean(data, axis=0), np.std(data,
       axis=0))
```