5
Les tableaux de numpy

1. On importe le module `numpy` pour tous les exemples de code qui suivent.

```
import numpy as np
```

2. Le module `numpy` définit la structure de **tableau** : ce sont des données rassemblées dans un ensemble et identifiées par un certain nombre d'indices.
2.1 Les éléments d'un **tableau unidimensionnel** sont identifiés par un seul indice entier. Un tel tableau peut être imaginé comme une liste.
2.2 Les éléments d'un **tableau bidimensionnel** sont identifiés par un couple d'indices entiers. Un tel tableau peut être imaginé comme une matrice. En particulier, on parlera des *lignes* et des *colonnes* d'un tableau bidimensionnel.
2.3 Si on en a besoin, on peut utiliser des tableaux de dimension 3 (où les éléments sont identifiés par un triplet d'indices) ou plus encore.

3. Comme les listes, les tableaux sont des **variables mutables**. On peut modifier la valeur de chaque élément et si un tableau `T` passé en argument à une fonction `f` est modifié à l'intérieur de la fonction `f`, ces modifications sont encore effectives après l'exécution de `f`.

4. Le **format** d'un tableau est un tuple d'entiers.
– Le nombre d'éléments de ce tuple donne la dimension du tableau.
– Les entiers qui constituent ce tuple donnent le nombre d'éléments du tableau pour chaque dimension.
Le format du tableau `T` est donné par `T.shape`.

5. La **taille** d'un tableau est le nombre total d'éléments qui constituent ce tableau. Donnée par `T.size`, la taille du tableau `T` est égale au produit des entiers du tuple `T.shape`.

6. Arithmétique modulaire
Contrairement aux listes dont les éléments sont de type quelconque, les tableaux ne regroupent que des données *de même type*.
6.1 Dans le module `numpy`, les entiers (long_scalars) sont normalement codés sur 4 octets (soit 32 bits) et les flottants sur 8 octets (soit 64 bits).
6.2 On peut vérifier le type de données utilisé dans un tableau avec l'attribut `dtype`.

```
A = np.arange(20)
A.dtype # 'int32'
A = np.arange(0., 1., 0.05)
A.dtype # 'float64'
```

6.3 Lorsque les entiers sont trop grands pour être représentés sur 4 octets, ils sont représentés sur 8 octets (longlong_scalars), voire convertis en flottants ou en objets (qui n'est pas un type numérique interne à `numpy`).

```
A = np.arange(0, 2**40, 2**35)
A.dtype # 'int64'
A = np.arange(0, 2**63, 2**60)
A.dtype # 'float64'
A = np.arange(0, 2**64, 2**60)
A.dtype # 'O' (pour objet)
```

6.4 Lorsque les entiers sont traités comme un type numérique intégré et représentés par un nombre fixe d'octets, on ne travaille pas dans l'anneau intègre $\mathbb{Z}$ mais dans un anneau *non intègre* de la forme $\mathbb{Z}/2^d\mathbb{Z}$, chaque classe étant représentée par un entier compris entre -2^{d-1} et $2^{d-1}-1$ (au sens large).
Lorsque les règles de calcul appliquées diffèrent des règles de calcul dans $\mathbb{Z}$, un message d'avertissement est émis.

```
A = np.arange(0, 2**30, 2**28) # 'int32'
for i in range(20):
  print(i*A[1])
# RuntimeWarning:
# overflow encountered in long_scalars

A = np.arange(0, 2**62, 2**60) # 'int64'
for i in range(0, 2500, 25):
  print(i*A[1])
# RuntimeWarning:
# overflow encountered in longlong_scalars
```

On voit sur ces exemples qu'un produit d'entiers strictement positifs peut être négatif ou négatif — c'est impossible dans $\mathbb{Z}$, mais normal dans $\mathbb{Z}/2^d\mathbb{Z}$.

I
Création d'un tableau

I.1 Tableaux paramétrables

7. Tableaux constants
Les méthodes `zeros` et `ones` du module `numpy` retournent des tableaux entièrement constitués de 0 ou de 1.
7.1 L'argument de ces méthodes est un tuple qui définit le format du tableau construit.
On construit un tableau de 0 de format $(2,3)$ avec l'instruction suivante.

```
np.zeros((2,3))
```

Ce tableau peut être considéré comme une matrice ayant 2 lignes et 3 colonnes.
7.2 Pour construire un tableau unidimensionnel de d éléments, on peut remplacer le tuple `(d,)` par l'entier `d`.
Les instructions `np.ones((3,))` et `np.ones(3)` produisent toutes deux un tableau unidimensionnel de 1, qu'on peut considérer comme une liste.
7.3 Lignes et colonnes
Les **lignes** et les **colonnes** doivent être des tableaux bidimensionnels (pour être considérées commes des *matrices* et non comme des *listes*). Les instructions suivantes produisent respectivement une colonne de 1 et une ligne de 1.

```
np.ones((3,1))  # colonne de 1
np.ones((1,3))  # ligne de 1
```

8. Tableaux aléatoires
Le sous-module `np.random` possède deux méthodes pour définir des tableaux de nombres au hasard : `random` et `randint`.

8.1 Flottants aléatoires
Étant donné un tuple `t`, on construit un tableau de format `t` contenant des valeurs indépendamment et uniformément réparties sur $[0,1]$ avec

`np.random.random(t)`.

8.2 Entiers aléatoires
Étant donnés un tuple `t` et deux entiers `a` et `b`, on construit un tableau de format `t` contenant des valeurs entières indépendamment et uniformément réparties sur $\{a,\dots,(b-1)\}$ avec

`np.random.randint(a, b, t)`.

Par défaut, la valeur de `a` est 0. L'instruction

`np.random.randint(b, t)`

retourne un tableau de format `t` qui contient des valeurs entières indépendamment et uniformément réparties entre 0 (inclus) et b (exclu).
8.3 On peut simuler 10 000 lancers de trois dés équilibrés par la commande suivante.

```
A = np.random.randint(1, 7, (10000, 3))
```

On définit ainsi un tableau `A` de format $(10\,000,3)$.
Chaque ligne de `A` représente le résultat d'un lancer : trois entiers au hasard entre 1 et 6.
Chaque colonne de `A` regroupe les résultats amenés par un dé au cours des 10 000 lancers effectués.

9. Matrices diagonales
9.1 L'instruction `np.identity(n)` produit un tableau *bidimensionnel* de format (n,n) constitué de 1 sur la diagonale et de 0 hors de la diagonale.
9.2 Si `A` est une liste de nombres ou un tableau unidimensionnel de longueur n, l'instruction `np.diag(A)` construit une matrice diagonale (un tableau bidimensionnel de format (n,n)) dont les coefficients diagonaux sont les éléments de `A`.

```
np.diag([1,7,8,9])
np.diag(np.arange(5))
```

10. Tableaux en progression arithmétique
La méthode `arange` du module `numpy` produit des tableaux *unidimensionnels* dont les éléments sont en progression arithmétique. Les éléments de ce tableau peuvent être entiers ou flottants.
10.1 L'instruction `np.arange(a, b, dx)` produit un tableau unidimensionnel avec tous les nombres de la forme $a+k\cdot \mathrm{d}x$ pour tout entier $k\in\mathbb{N}$ tel que

$$a \leqslant a+k\cdot \mathrm{d}x < b$$

lorsque $\mathrm{d}x>0$ ou tel que

$$b < a+k\cdot \mathrm{d}x \leqslant a$$

lorsque $\mathrm{d}x<0$.
10.2 Par défaut, la valeur initiale a est égale à 0 et le pas $\mathrm{d}x$ est égal à 1, ces valeurs pouvant être des entiers ou des flottants. Ainsi, les instructions

`np.arange(10)` et `np.arange(0, 10, 1)`

sont équivalentes, de même que les instructions

`np.arange(10.)` et `np.arange(0., 10., 1.)`.

I.2 Conversion de listes

11. La méthode `np.array` permet de convertir une liste en tableau.
11.1 Une liste d'entiers est convertie en tableau d'entiers.

```
L = [1, 2, 3]
np.array(L)
```

11.2 Dès qu'une liste contient au moins un flottants, elle est convertie en tableau de flottants.

```
L = [1, 2, 3.]
np.array(L)
```

12. La dimension du tableau obtenu par conversion d'une liste dépend de l'existence de sous-listes.
12.1 Une liste de nombres est convertie en tableau unidimensionnel.

```
L = [1, 2, 3]
A = np.array(L)
A.size==len(L)
A.shape==(len(L),)
```

12.2 Une liste de listes de nombres est convertie en tableau bidimensionnel.

```
nl, nc = 4, 3
L = [[i+j for j in range(nc)] for i in range(nl)]
A = np.array(L)
A.size==nl*nc
A.shape==(nl, nc)
```

I.3 Changement de format

13. La classe `array` possède la méthode `transpose`.
13.1 Appliquée à un tableau bidimensionnel, la méthode `transpose` produit le même effet que la tranposition des matrices.

```
L = [range(3*i, 3*(i+1)) for i in range(4)]
A = np.array(L)
A.transpose()
```

13.2 Appliquée à un tableau de dimension $d\geqslant 3$, elle transforme le tableau de format $(n_1,n_2,\dots,n_d)$ et de terme général

$$a_{i_1,i_2,\dots,i_d}$$

en un tableau de format $(n_d,\dots,n_2,n_1)$ et de terme général

$$a_{i_d,\dots,i_2,i_1}.$$

13.3 Un tableau unidimensionnel est invariant par la méthode `transpose`, raison pour laquelle un tableau unidimensionnel ne doit pas être considéré comme une ligne (qui est changée en colonne par transposition) mais comme une liste.

```
A = np.arange(10) # tableau unidimensionnel
A, A.transpose()  # invariant par transposition

A = np.array([[i] for i in range(10)])
A.shape           # colonne
B = A.transpose()
B.shape           # ligne
```

14. La classe `array` possède aussi la méthode `reshape` qui permet de modifier le format d'un tableau sans en modifier la taille.

14.1 Un tableau de taille $s \in \mathbb{N}^*$ peut se voir attribuer tous les formats $(n_1, \dots, n_d) \in (\mathbb{N}^*)^d$ tels que

$$s = n_1 \cdot n_2 \cdots n_d.$$

En particulier, on peut ainsi modifier la dimension d'un tableau.

14.2 Si A est un tableau unidimensionnel de taille s, alors on obtient
- une ligne de taille s avec l'instruction `A.reshape(1,s)`
- une colonne de taille s avec l'instruction `A.reshape(s,1)` ou encore avec `A.reshape(s,)`.

14.3 On peut ainsi convertir un tableau unidimensionnel de taille $s = n \cdot p$ en un tableau bidimensionnel de format (n, p) ou de format (p, n), mais aussi en un tableau tridimensionnel de format $(n, 1, p)$ (même si l'usage d'un tel tableau semble douteux).

```
A = np.arange(10)
B = A.reshape(2,5)
C = A.reshape(5,2)
```

On notera que les tableaux `B.transpose()` et `C` ont même format mais sont différents.
D'autre part, les trois tableaux `A`, `B` et `C` partagent les mêmes valeurs : ce sont trois **vues** différentes d'un même tableau. En changeant la valeur d'un élément d'un de ces trois tableaux, on change en même temps la valeur de l'élément correspondant dans les deux autres tableaux.

II

Sous-tableaux

15. Accès à un élément
Chaque élément d'un tableau est identifié par un tuple.

15.1 Si A est un tableau unidimensionnel de format $(n,)$, on accède à la valeur des éléments de ce tableau par `A[(i,)]` ou plus simplement par `A[i]` pour tout entier $0 \leqslant i < n$.

15.2 Si A est un tableau bidimensionnel de format (n, p), on accède à la valeur des éléments de ce tableau par `A[(i,j)]` ou par `A[i,j]` pour $0 \leqslant i < n$ et $0 \leqslant j < p$.

16. Diagonales d'une matrice
Soit $A = (a_{i,j})_{0 \leqslant i < n, 0 \leqslant j < p}$, une matrice représentée par le tableau bidimensionnel `A`.

16.1 La **diagonale** de A est l'ensemble des éléments de la forme $a_{i,i}$. L'instruction `np.diag(A)` renvoie un tableau unidimensionnel avec les éléments de la diagonale de `A`.

16.2 Pour tout entier k, l'instruction `np.diag(A, k)` renvoie un tableau unidimensionnel, éventuellement vide, constitué des éléments de A de la forme $a_{i,i+k}$, c'est-à-dire d'une **sur-diagonale** de A pour $k \geqslant 1$ ou d'une **sous-diagonale** de A pour $k \leqslant -1$.

Tranches

17. Pour les tableaux comme pour les listes, on peut définir des tranches (*slices*). La syntaxe est la même pour les tableaux et pour les listes.

18. Lignes et colonnes
Soit $A = (a_{i,j})_{0 \leqslant i < n, 0 \leqslant j < p}$, un tableau bidimensionnel.

18.1 La i-ème ligne du tableau A est constituée des éléments $a_{i,j}$ pour $0 \leqslant j < p$. On accède donc à la i-ème ligne du tableau `A` par `A[i,:]`.
La j-ème colonne de A est quant à elle constituée des éléments $a_{i,j}$ pour $0 \leqslant i < n$. On y accède par `A[:,j]`.

18.2 Les lignes et les colonnes du tableau bidimensionnel `A` sont des tableaux *uni*dimensionnels. Ce ne sont donc pas des lignes et des colonnes au sens de [7.3].

18.3 Tant qu'elles ne sont pas explicitement modifiées, ces tranches restent des *références* sur des parties du tableau `A` : toute modification du tableau `A` modifie les tranches en conséquence. En revanche, une modification de tranche ne modifie pas le tableau `A`.

```
A = np.arange(12).reshape(4,3)
L = A[1,:]   # Deuxième ligne
L.shape
C = A[:,2]   # Troisième colonne
C.shape
# Modification de A, de L et de C
A[1,:] = np.ones(3)
A, L, C
# Modification de L mais pas de A, ni de C
L = np.zeros(3)
A, L, C
```

18.4 Pour supprimer le lien entre le tableau `A` et l'une de ses tranches, il faut réaliser une *copie*.

```
A = np.arange(12).reshape(3,4)
L1 = A[1,:]          # référence
L2 = A[1,:].copy()   # copie
# Modification de A et de L1 mais pas de L2
A[:,2] = np.zeros(3)
A, L1, L2
```

18.5 Permutation de lignes et de colonnes
Cette suppression de la référence par copie est nécessaire pour échanger deux lignes ou deux colonnes d'un tableau bidimensionnel.

```
# Permutation L1 ↔ L2
A[1,:], A[2,:] = A[2,:].copy(), A[1,:].copy()
# Permutation C0 ↔ C1
A[:,1], A[:,0] = A[:,0].copy(), A[:,1].copy()
```

19. On peut varier les tranches à loisir — l'utilité des commandes suivantes reste à démontrer.

```
A[:2, :]   # Les deux premières lignes
A[:, -2:]  # Les deux dernières colonnes
A[1::2, :] # Les lignes de rang impair
A[:, ::2]  # Les colonnes de rang pair
```

Indexation avancée

20. On parle d'**indexation avancée** (*advanced indexing*) quand l'indice n'est pas un tuple mais une liste ou un tableau.

21. Soit $A = (a_{i,j})_{0 \leqslant i < n, 0 \leqslant j < p}$, un tableau bidimensionnel.

21.1 On obtient la i-ème ligne de `A` avec l'instruction `A[[i],:]`. Il s'agit cette fois d'un tableau bidimensionnel (comme `A`), *copie* de la i-ème ligne de `A` (et non plus référence sur une partie de `A` comme au [18]).

21.2 De même, on obtient la j-ème colonne de `A` avec l'instruction `A[:,[j]]`. Là encore, il s'agit d'un tableau bidimensionnel, copie de la j-ème colonne de `A`.

22. Permutation de lignes et de colonnes
L'indexation avancée permet de réaliser facilement les opérations de pivot

$$L_i \leftrightarrow L_j \quad \text{et} \quad C_i \leftrightarrow C_j.$$

```
# Échange de deux lignes
A[[0],:], A[[2],:] = A[[2],:], A[[0],:]
# Variante
A[[0,2],:] = A[[2,0],:]
```

```
# Échange de deux colonnes
A[:,[1]], A[:,[2]] = A[:,[2]], A[:,[1]]
# Variante
A[:,[1,2]] = A[:,[2,1]]
```

III

Opérations

23. L'intérêt de la structure de tableau tient au grand nombre d'opérations qu'on peut effectuer facilement sur des tableaux — en particulier, les opérations de pivot sur les lignes et les colonnes des tableaux bidimensionnels : **[22]**, **[36]**, **[37]**.

III.1 Application d'une fonction vectorialisée

24. Les fonctions du module `numpy` sont vectorialisées : en appliquant une telle fonction `f` à un tableau `A`, on construit un tableau `fA` de même format que `A`. Le terme général du tableau `fA` est `f(a)`, où `a` est le terme général du tableau `A`.

```
A = np.arange(0., 1., 0.032).reshape(8,4)
np.exp(A)
np.sqrt(A)
np.log(A)  # Message d'avertissement pour ℓn 0
```

25. Le module `numpy` possède la méthode `vectorize` pour vectorialiser une fonction définie par l'utilisateur.

```
def f(x):
  if (x<.5):
    return 2*x
  else:
    return 3*x+1

f_vec = np.vectorize(f)
f_vec(A)
```

Si une fonction est définie en n'utilisant que des opérations du module `numpy`, elle est déjà vectorialisée. →**[43]**

III.2 Fonctions d'agrégation

26. La classe `array` possède des méthodes qu'on peut utiliser comme des fonctions d'agrégations.
Pour un tableau bidimensionnel `A` de format (n,p), on peut appliquer ces méthodes
– colonne par colonne et obtenir un tableau unidimensionnel de taille p ;
– ligne par ligne et obtenir un tableau unidimensionnel de taille n.
Il suffit pour cela de préciser l'**axe** le long duquel le calcul est effectué.
– Avec `axis=0`, on applique la méthode au sous-tableau des $a_{i_0,j}$ pour $0 \leqslant i_0 < n$ et on obtient un tableau de b_j pour $0 \leqslant j < p$.
– Avec `axis=1`, on applique la méthode au sous-tableau des a_{i,j_1} pour $0 \leqslant j_1 < p$ et on obtient un tableau de c_i pour $0 \leqslant i < n$.

27. Les méthodes `max` et `min` retournent le plus grand élément et le plus petit élément d'un tableau.

```
A = np.random.random((5,3))
A.max()       # Maximum global
A.max(axis=0) # Tableau des max par colonne
A.max(axis=1) # Tableau des max par ligne
```

28. Les méthodes `argmax` et `argmin` retournent la position du plus grand élément et celle du plus petit élément du tableau.

28.1 Par défaut, le tableau $(a_{i,j})_{0\leqslant i<n,0\leqslant j<p}$ est transformé en tableau unidimensionnel $(a_k)_{0\leqslant k<np}$ et la position donnée est l'indice correspondant dans ce tableau unidimensionnel. C'est peu pratique !
On retrouve la position avec une division euclidienne puisque le tableau unidimensionnel est obtenu en concaténant les lignes :

$$\forall\, 0 \leqslant i < n,\ \forall\, 0 \leqslant j < p, \quad k = pi + j.$$

```
A = np.random.random((5,3))
idx = A.max()
n,p = A.shape
idx_lig = idx//p  # quotient
idx_col = idx%p   # reste
```

28.2 Plus généralement, la méthode `ravel` enchevêtre les éléments $a_{i_0,\dots,i_{d-1}}$ d'un tableau de dimension d et de format $(n_0,\dots,n_{d-1})$ en un tableau unidimensionnel (a_ℓ) selon la règle suivante :

$$\ell = \sum_{0\leqslant j<d} i_j \prod_{j<k<d} n_k = i_0 n_1 \cdots n_{d-1} + \cdots + i_{d-2} n_{d-1} + i_{d-1}.$$

On peut alors récupérer les indices $i_0, \dots, i_{d-1}$ par des divisions euclidiennes successives.
28.3 Pour un tableau `A` de format (n,p), si un axe est précisé, la méthode est appliquée le long de cet axe et le format du tableau retourné est le format (n,p) privé de la dimension de l'axe choisi.

```
A = np.random.random((5,3))
# Dans quelle colonne se trouve le max du tableau ?
MaxCol = A.max(axis=0)
PosMaxCol = MaxCol.argmax()
# Où se trouve le max de cette colonne ?
PosMaxLig = A[:,PosMaxCol].argmax()
# On a localisé le max du tableau.
A[PosMaxLig,PosMaxCol] == A.max()
```

Calculs de sommes et de produits

29. **Avertissement : `no error is raised on overflow`**
Les méthodes suivantes calculent des sommes et des produits sur les éléments d'un tableau.
Lorsque les éléments de ce tableau sont des entiers, les règles de calcul ne sont pas celles de $\mathbb{Z}$, mais celles d'un anneau de la forme $\mathbb{Z}/2^d\mathbb{Z}$ et contrairement aux habitudes **[6]**, aucun avertissement n'est émis lorsque les entiers calculés sont supérieurs à 2^{d-1} en valeur absolue.

30. La méthode `sum` retourne la somme de tous les éléments d'un tableau.
30.1 Pour un tableau de dimension $d \geqslant 2$ et de format

$$n = (n_0,\dots,n_{d-1}),$$

l'option `axis=i` calcule les sommes le long de l'axe i :

$$s_{k_0,\dots,k_{i-1},k_{i+1},\dots,j_{d-1}} = \sum_{0\leqslant k_i<n_i} a_{k_0,\dots,k_{i-1},k_i,k_{i+1},\dots,k_{d-1}}$$

et le tableau retourné a pour format

$$(n_0,\dots,n_{i-1},n_{i+1},\dots,n_{d-1}).$$

```
A = np.random.random((6,4))
A.sum()       # Somme totale
A.sum(axis=0) # Sommes partielles par colonnes
A.sum(axis=1) # Sommes partielles par lignes
```

30.2 Soit (X, Y), un couple de variables aléatoires discrètes. La **loi jointe** de ce couple est la famille de terme général

$$a_{i,j} = \mathbf{P}\big([X = i] \cap [Y = j]\big).$$

Lorsque le tableau bidimensionnel `A` de format (n, p) contient la loi du couple (X, Y), on calcule les lois marginales de X et Y avec la méthode `sum` en choisissant un axe de sommation.
– L'instruction `A.sum(axis=0)` calcule les sommes sur la dimension de rang 0, c'est-à-dire

$$\forall\, 0 \leqslant j < p, \quad s_j = \sum_{0 \leqslant i < n} a_{i,j} = \sum_{0 \leqslant i < n} \mathbf{P}\big([X = i] \cap [Y = j]\big) = \mathbf{P}(Y = j)$$

et donne la loi de Y.
– L'instruction `A.sum(axis=1)` calcule les sommes sur la dimension de rang 1, c'est-à-dire

$$\forall\, 0 \leqslant i < n, \quad \sigma_i = \sum_{0 \leqslant j < p} a_{i,j} = \mathbf{P}(X = i)$$

et donne la loi de X.

31. Fonction de répartition
Si $X : \Omega \to \mathbb{N}$ est une variable aléatoire discrète, sa loi est la famille de terme général

$$p_i = \mathbf{P}(X = i)$$

et sa **fonction de répartition** est la famille de terme général

$$f_k = \mathbf{P}(X \leqslant k) = \sum_{i=0}^{k} \mathbf{P}(X = i) = \sum_{i=0}^{k} p_i.$$

32. La méthode `cumsum` retourne les sommes cumulées d'un tableau.
32.1 Lorsque le tableau unidimensionnel `A` contient la loi de la variable aléatoire X, le tableau `F = A.cumsum()` est de même format que `A` et contient la fonction de répartition de X.
32.2 Lorsque le tableau `A` contient la loi jointe d'un couple (X, Y) de variables aléatoires discrètes, les instructions

`A.cumsum(axis=0)` et `A.cumsum(axis=1)`

calculent respectivement les sommes sur la dimension de rang 0 et sur la dimension de rang 1, c'est-à-dire

$$s_{i,j} = \mathbf{P}\big([X \leqslant i] \cap [Y = j]\big) \quad \text{et} \quad \sigma_{i,j} = \mathbf{P}\big([X = i] \cap [Y \leqslant j]\big).$$

33. Les méthodes `prod` et `cumprod` sont les analogues de `sum` et `cumsum`, en remplaçant les sommes par des produits.

34. La méthode `mean` calcule la moyenne arithmétique d'un tableau.
Pour un tableau d'entiers, le calcul de la moyenne est effectué en convertissant les données en flottants codés sur 8 octets (`float64`).

```
A = np.random.random((6,4))
A.mean()       # Moyenne générale
A.mean(axis=0) # Moyenne par colonne
A.mean(axis=1) # Moyenne par ligne
```

III.3 Opérations algébriques

35. Pour un tableau `A` dont les éléments sont des nombres complexes, les instructions

`A.real()` `A.imag()` et `A.conj()`

retournent des tableaux de même format que `A`, obtenus en calculant respectivement la partie réelle, la partie imaginaire et le conjugué de chaque élément de `A`.

36. Multiplication par un scalaire
Comme une matrice, un tableau peut être multiplié par un scalaire λ : chaque élément du tableau est alors multiplié par λ.
36.1 Le tableau défini par `7*np.arange(10)` est le tableau unidimensionnel qui contient 10 fois le nombre 7.
36.2 Le tableau défini par `2*np.identity(5)` est un tableau bidimensionnel qu'on peut assimiler à la matrice $2I_5$.
36.3 On peut effectuer simplement les opérations de pivot

$$L_i \leftarrow \alpha L_i \quad \text{et} \quad C_j \leftarrow \beta C_j$$

sur les lignes et les colonnes d'un tableau bidimensionnel.

```
A = np.random.randint(0, 10, (5,3))
A[2,:] = 3*A[2,:]     # L_2 ← 3L_2
A[:,1] = -A[:,1]      # C_1 ← −C_1
```

37. Addition terme à terme
Deux tableaux de même format peuvent être sommés terme à terme (comme des matrices).
37.1 Si `A` est un tableau et `x`, un nombre, alors le tableau `A+x` est de même format que `A` et est obtenu en ajoutant `x` à chaque élément de `A`.
Si `x` est un flottant, alors l'expression `A+x` est un raccourci pour `A+x*np.ones(A.shape)`.

```
A = np.arange(36).reshape(9,4)
B = np.random.random(A.shape)
A+B
A-2
```

37.2 On peut effectuer simplement les opérations de pivot

$$L_i \leftarrow L_i + \alpha L_k \quad \text{et} \quad C_j \leftarrow C_j + \beta C_k$$

sur les lignes et les colonnes d'un tableau bidimensionnel.

```
A[3,:] = A[3,:]+3*A[0,:]  # L_3 ← L_3 + 3L_0
A[:,1] = A[:,1]-2*A[:,3]  # C_1 ← C_1 − 2C_3
```

38. Multiplication terme à terme
Utilisé avec les tableaux bidimensionnels, l'opérateur `*` doit être distingué de la multiplication matricielle.
38.1 Si `A` et `B` sont deux tableaux de même format (n, p), alors le produit `A*B` est un tableau de même format, obtenu en multipliant terme à terme les éléments de `A` et de `B`.

$$\forall\, 0 \leqslant i < n,\ \forall\, 0 \leqslant j < p, \quad c_{i,j} = a_{i,j} b_{i,j}$$

```
A = np.random.randint(0, 10, 5)
B = np.random.randint(0, 10, 5)
A*B
```

38.2 Sous certaines conditions, le produit `A*B` peut être défini même quand les tableaux n'ont pas le même format. Voir les règles générales du *broadcasting* dans la documentation de `SciPy`.

39. Application
On suppose connu un échantillon $[(x_i, y_i)]_{0 \leqslant i < N}$ de points du graphe d'une fonction f. On peut approcher l'intégrale de f par la méthode des rectangles ou par la méthode des trapèzes :

$$\int_a^b f(t)\,\mathrm{d}t \approx \sum_{1 \leqslant i < N} (x_i - x_{i-1}) y_i \approx \sum_{1 \leqslant i < N} (x_i - x_{i-1}) \cdot \frac{y_{i-1} + y_i}{2}.$$

Si l'échantillon de points est donnés par les tableaux `x` et `y`, l'approximation de l'intégrale par la méthode des rectangles est donnée par

```
np.sum((x[1:]-x[:-1])*y[1:])
```

et l'approximation par la méthode des trapèzes est donnée par

```
np.sum(0.5*(x[1:]-x[:-1])*(y[:-1]+y[1:]))
```

40. Multiplication matricielle

Les tableaux unidimensionnels et bidimensionnels peuvent être traités comme des vecteurs et des matrices : on a vu qu'on peut effectuer des additions **[37]**, des multiplications par un scalaire **[36]** et donc des combinaisons linéaires.

40.1 Il y a deux manières possibles de calculer le produit matriciel AB :

`np.dot(A,B)` et `A.dot(B)`.

Dans les deux cas, les formats de `A` et `B` doivent être compatibles.

– Si `A` et `B` sont bidimensionnels, leurs formats respectifs doivent être de la forme (n,p) et (p,q).
Le format du produit `np.dot(A,B)` sera alors (n,q).

```
A = np.arange(12).reshape(3,4)
B = np.arange(20).reshape(4,5)
A.dot(B)
```

– Si le premier facteur est un tableau unidimensionnel de taille n, il est converti en tableau bidimensionnel de format $(1,n)$.
Le format du second facteur doit alors être (n,q) et le produit est un tableau unidimensionnel de taille q.

```
A = np.arange(3)
B = np.arange(6).reshape(3,2)
np.dot(A,B)
```

– Si le second facteur est un tableau unidimensionnel de taille p, il est converti en tableau bidimensionnel de format $(p,1)$.
Le format du premier facteur doit alors être (n,p) et le produit est un tableau unidimensionnel de taille n.

```
A = np.arange(6).reshape(2,3)
B = np.arange(3)
np.dot(A,B)
```

Si le produit matriciel n'a pas de sens, un message d'erreur apparaît :

```
ValueError: matrices are not aligned
```

40.2 Produits de deux vecteurs

Soient `A` et `B`, deux tableaux unidimensionnels de même taille n. On les identifie ici à deux matrices colonnes A et B de $\mathfrak{M}_{n,1}(\mathbb{K})$.
Le produit scalaire canonique de ces deux vecteurs :

$$ {}^tAB = \sum_{0\leqslant i<n} a_i b_i $$

peut être obtenu par `np.dot(A,B)` et par `np.inner(A,B)`.
Le produit hermitien canonique, lorsque ces vecteurs ont des coordonnées complexes :

$$ \overline{{}^tA}B = \sum_{0\leqslant i<n} \overline{a_i}\, b_i $$

est obtenu par `np.vdot(A,B)`.
La matrice carrée

$$ A^tB = (a_i b_j)_{0\leqslant i,j<n} \in \mathfrak{M}_n(\mathbb{K}) $$

est obtenue avec `np.outer(A,B)`. L'exemple suivant permet de réviser la table de Pythagore.

```
A = np.arange(1,11)
np.outer(A,A)
```

III.4 Tableaux booléens

41. On peut comparer les éléments d'un tableau `A` à une valeur scalaire `x` et créer un tableau de même format que `A` dont les éléments sont les booléens qui résultent de la comparaison des éléments de `A` à `x`.

```
A = np.arange(36).reshape(9,4)
B1 = (A>20)
B2 = (A<=25)
```

42. Opérations sur les tableaux booléens

Les opérations sur les tableaux booléens ont des syntaxes spécifiques.

42.1 Complémentaire

Si `B` est un tableau booléen, le tableau `-B` est le tableau de même format que `B`, obtenu en appliquant l'opérateur `not` (négation) à chaque élément de `B`.

42.2 Union

Si `B1` et `B2` sont deux tableaux booléens de même format, le tableau `B1+B2` est le tableau de même format, obtenu en appliquant l'opérateur `or` (disjonction inclusive) terme à terme.

42.3 Intersection

Le tableau `B1*B2` est le tableau obtenu en appliquant l'opérateur `and` (conjonction) terme à terme.

42.4 Différence symétrique

Les tableaux `B1-B2` et `B2-B1` sont obtenus en appliquant l'opérateur `xor` (disjonction exclusive) terme à terme. Ces deux tableaux sont donc égaux.

43. Fonctions en escalier

En multipliant un tableau booléen `B` par un scalaire λ, on obtient un tableau numérique `A` de même format que `B`. Pour chaque élément de `B` égal à `True` (resp. à `False`), on obtient un élément de `A` égal à λ (resp. à 0).
La fonction suivante est une version améliorée de l'exemple vu en **[25]** : construite à l'aide des règles de calcul de `numpy`, cette fonction est vectorialisée par définition.

```
def f(x):
  B = (x<.5)
  return (2*x)*B+(3*x+1)*(-B)
```

44. Masques

En utilisant simultanément les tableaux booléens et l'indexation avancée **[20]**, on définit un **masque** qui permet d'extraire d'un tableau les valeurs qui remplissent une condition donnée (présentées sous la forme d'un tableau unidimensionnel).

```
A = np.random.random((6,4))
condition = (A>.5)
# Éléments de A strictement supérieurs à 0.5
A[condition]
```