Matrice et Numpy

Importez le module `numpy`. L'usage est de l'importer avec l'alias `np`, entrez donc `import numpy as np`.

1 Notion de tableau

Les objets manipulés par `numpy` sont des tableaux (*array*) multidimensionnels. Un tableau à une dimension ressemble très fortement à une liste (mais ne se comportera pas toujours de la même façon), un tableau à deux dimensions ressemble à une matrice.

Un tableau est un objet `itérable`, nous pouvons donc déterminer sa longueur et accéder à ses éléments de la manière habituelle : `T[i]`.
Un tableau est un objet `mutable`, c'est-à-dire que nous pouvons modifier un élément par une affectation directe : entrez `T[2]=5` et vérifiez ce qu'est devenu `T`.

2 Tableaux particuliers

— Tableaux constants : essayez et expliquez le résultat
`np.zeros(5)`
Nous obtenons un tableau `array([0.,0.,0.,0.,0.])` donc `np.zeros(n)` renvoie une liste de n flottants 0.
`np.zeros(4, dtype=int)`
Nous obtenons un tableau `array([0,0,0,0])` donc `np.zeros(n)` renvoie une liste de n entiers 0
`np.ones(3)`
Nous obtenons un tableau `array([1.,1.,1.,1.])` donc `np.zeros(n)` renvoie une liste de n flottants 1.
`np.eye (3).`
nous obtenons :

```
array([[1.,  0.,  0.],
       [0.,  1.,  0.],
       [0.,  0.,  1.]])
```

`np.eye(n)` renvoie la matrice carrée $n \times n$ identité I_n.

— Création de tableaux aléatoires : essayez et expliquez `np.random.rand(6)`
Nous obtenons un tableau de longueur 6 de nombres aléatoires dans $[0,1[$. Donc `np.random.rand(n)` renvoie un tableau de longueur n de nombres aléatoires dans $[0,1[$.
`np.random.randint(2,6,20)`
Nous obtenons un tableau de longueur 20 de nombres aléatoires dans $[\![2,5]\!]$. Donc `np.random.randint(a,b,n)` renvoie un tableau de longueur n de nombres aléatoires dans $[\![a,b-1]\!]$.

— Création de listes régulières :
Essayez et expliquez `np.arange(0,3,0.1)`
Nous obtenons le tableau des nombres de 0 à 3 exclus de pas 0.1. Donc `np.arange(a,b,p)` renvoie le tableau des nombre de $[a,b[$ commençant à a et s'arrêtant à $b-p$.
et `np.linspace(0,10,51)`
Nous obtenons un tableau de 51 nombres de 0 à 10 répartis uniformément. Donc `np.linspace(a,b,n)` renvoie n nombres entre a et b, le premier étant a, le dernier b et l'écart entre deux nombres est $\frac{b-a}{n-1}$.

— Pour définir une matrice en Python, nous utilisons la commande `np.array`. Par exemple, pour définir la matrice $A = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 1 & 3 \\ 0 & 1 & 4 \end{pmatrix}$, il faut saisir la commande suivante : `np.array ([[1,2,0],[-1,1,3],[0,1,4]])`

— Dans cet exemple, le tableau unidimensionnel arr est remodelé en une matrice 2x6 à l'aide de np.reshape()

```
import numpy as np

# Créer un tableau 1D avec 12 éléments
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

# Remodeler le tableau en une matrice 2x6
reshaped_arr = np.reshape(arr, (2, 6))
```

Nous obtenons :

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
```

2.1 Opérations sur les tableaux

Définissez les tableaux `T=np.array([3,-5,4,2,8,7])` et `U=np.array([2,8,0,1,4,4])`. Essayez les commandes suivantes : `T+3` `4*T` `1/T` `T**2` `T+U` `T*U`
Les tableaux se comportent-ils comme les listes du point de vue des opérations ?
Oui mais attention le produit $*$ ne se comporte pas comme une multiplication de matrices mais multiplie les coefficient de même position ensemble.
Pour multiplier deux matrices entre elle en Python, nous utilisons la commande `np.dot(mat(A),mat(B))`.

Exercice 1 :

Calculer le produit des deux matrices : $A = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 1 & 3 \\ 0 & 1 & 4 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 0 & 0 \\ 0 & 3 & 5 \end{pmatrix}$

```
A=np.reshape(np.array([1,2,0,-1,1,3,0,1,4]),(3,3))

B=np.reshape(np.array([1,2,0,-1,0,0,0,3,5]),(3,3))

>>>np.dot(A,B)

array([[-1,  2,  0],
       [-2,  7, 15],
       [-1, 12, 20]])
```

Exercice 2 : (Concours)

Considérons la matrice

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 2 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

Compléter la fonction Python suivante pour qu'elle calcule et affiche la matrice A^n pour une valeur de l'entier naturel n donnée

```
import numpy as np
def puissanceA(n):
    A=np.array([[1,2,1],[2,2,2],[
1,2,1]])
    resultat=A
    for k in range(n-1):
        resultat=np.dot(A,resultat)
    return resultat
```

Exercice 3 : Soit `A=np.array([[1,2,0],[-1,1,3],[0,1,4]])`.

Tester et expliquer ce que fait chaque méthode :

```
np.sum(A)
np.int64(11)

# C'est la somme des coefficients de la matrice.

np.min(A)
np.int64(-1)

# C'est le plus petit des coefficients de la matrice.

np.max(A)
np.int64(4)

# C'est le plus grand des coefficients de la matrice.

np.cumsum(A)

array([ 1,3,3,2,3,6,6,7,11])


# C'est la somme des coefficients le premier, puis le premier et le deuxième,
# puis le premier et le deuxième et le troisième et ainsi de suite.
```

Nous pouvons aussi définir :

```
np.mean(A) # moyenne des termes de la matrice A
np.median # médiane des termes de la matrice A
np.var # variance des termes de la matrice A
np.std # écart-type des termes de la matrice A
```

Exercice 4 : (concours)

Considérons la matrice $A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{pmatrix}$. Considérons les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ définies par $a_0 = 2$, $b_0 = 0$ et pour tout $n \in \mathbb{N}$:

$$\begin{cases} a_{n+1} = 2a_n + 3^n \\ b_{n+1} = 3b_n + 3^n \end{cases}$$

1. Montrer par récurrence que pour tout entier naturel n, nous avons :

$$A^n = \begin{pmatrix} 2^n & 0 & 3^n - 2^n \\ 0 & 3^n & n3^{n-1} \\ 0 & 0 & 3^n \end{pmatrix}.$$

2. Je raisonne par récurrence sur $n \in \mathbb{N}$.

   Pour tout $n \in \mathbb{N}$, nous posons $\mathscr{P}_n$: « $A^n = \begin{pmatrix} 2^n & 0 & 3^n - 2^n \\ 0 & 3^n & n3^{n-1} \\ 0 & 0 & 3^n \end{pmatrix}$ ».

   Initialisation : Nous montrons $\mathscr{P}_0$.

   $A^0 = I_3$ et

$$\begin{pmatrix} 2^0 & 0 & 3^0 - 2^0 \\ 0 & 3^0 & 0 \times 3^{0-1} \\ 0 & 0 & 3^0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1-1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I_3.$$

Donc $A^0 = \begin{pmatrix} 2^0 & 0 & 3^0 - 2^0 \\ 0 & 3^0 & 0 \times 3^{0-1} \\ 0 & 0 & 3^0 \end{pmatrix}$.

$\mathscr{P}_0$ est vraie.

Hérédité : Fixons $n \in \mathbb{N}$ et nous supposons $\mathscr{P}_n$ vraie. Nous avons donc $A^n = \begin{pmatrix} 2^n & 0 & 3^n - 2^n \\ 0 & 3^n & n3^{n-1} \\ 0 & 0 & 3^n \end{pmatrix}$

et nous voulons montrer : $A^{n+1} = \begin{pmatrix} 2^{n+1} & 0 & 3^{n+1} - 2^{n+1} \\ 0 & 3^{n+1} & (n+1)3^n \\ 0 & 0 & 3^{n+1} \end{pmatrix}$.

Nous avons : $A^{n+1} = AA^n$. Par hypothèse de récurrence, $A^n = \begin{pmatrix} 2^n & 0 & 3^n - 2^n \\ 0 & 3^n & n3^{n-1} \\ 0 & 0 & 3^n \end{pmatrix}$. D'où

$$A^{n+1} = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{pmatrix} \times \begin{pmatrix} 2^n & 0 & 3^n - 2^n \\ 0 & 3^n & n3^{n-1} \\ 0 & 0 & 3^n \end{pmatrix} = \begin{pmatrix} 2^{n+1} & 0 & 2 \times 3^n - 2^{n+1} + 3^n \\ 0 & 3^{n+1} & n3^n + 3^n \\ 0 & 0 & 3^{n+1} \end{pmatrix}$$

Car $2 \times 2^n = 2^{n+1}$ et $3 \times 3^n = 3^{n+1}$.

De plus $2 \times 3^n - 2^{n+1} + 3^n = 3^n(2+1) - 2^{n+1} = 3^{n+1} - 2^{n+1}$

Et $n3^n + 3^n = (n+1)3^n$. Donc $A^{n+1} = \begin{pmatrix} 2^{n+1} & 0 & 3^{n+1} - 2^{n+1} \\ 0 & 3^{n+1} & (n+1)3^n \\ 0 & 0 & 3^{n+1} \end{pmatrix}$.

Ainsi $\mathscr{P}_n$ est héréditaire.

Conclusion : D'après le principe de récurrence $\mathscr{P}_n$ est vraie pour tout $n \geqslant 0$, soit

$$\boxed{A^n = \begin{pmatrix} 2^n & 0 & 3^n - 2^n \\ 0 & 3^n & n3^{n-1} \\ 0 & 0 & 3^n \end{pmatrix}}.$$

3. Pour tout n entier posons $X_n = \begin{pmatrix} a_n \\ b_n \\ 3^n \end{pmatrix}$

4. (a) Montrer que pour tout entier naturel n, nous avons $X_{n+1} = AX_n$.

$$AX_n = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} a_n \\ b_n \\ 3^n \end{pmatrix} = \begin{pmatrix} 2a_n + 3^n \\ 3b_n + 3^n \\ 3^{n+1} \end{pmatrix} = X_{n+1}.$$

(b) Recopier et compléter la fonction Python suivante afin qu'elle affiche la valeur de a_n, pour un entier n donné.

```
import numpy as np
def calculbisa(n):
    A=np.array([[2,0,1],[0,3,1],[0,0,3]])
    X=np.array([[2],[0],[1]])
    for k in range(n):
        X=np.dot(A,X)
    return X[0]
```

(c) Établir que pour tout $n \in \mathbb{N}$, $X_n = A^n X_0$.

Je raisonne par récurrence sur $n \in \mathbb{N}$.

Pour tout $n \in \mathbb{N}$, nous posons $\mathscr{P}_n$: « $X_n = A^n X_0$ ».

Initialisation : Nous montrons $\mathscr{P}_0$.

$A^0 = I_3$ et $A^0 X_0 = X_0$. $\mathscr{P}_0$ est vraie.

Hérédité : Fixons $n \in \mathbb{N}$ et nous supposons $\mathscr{P}_n$ vraie. Nous avons donc $X_n = A^n X_0$ et nous voulons montrer : $X_{n+1} = A^{n+1} X_0$

Nous avons, $X_{n+1} = AX_n$ par la question précédente et par hypothèse de récurrence :

$$X_{n+1} = AX_n = A \times A^n X_0 = A^{n+1} X_0$$

Ainsi $\mathscr{P}_n$ est héréditaire.

Conclusion : D'après le principe de récurrence $\mathscr{P}_n$ est vraie pour tout $n \in \mathbb{N}$, soit $\boxed{X_n = A^n X_0}$.

5. En déduire, en utilisant le résultat de la question 1 que nous avons : $a_n = 2^n + 3^n$ et $b_n = n3^{n-1}$

Nous avons

$$A^n X_0 = \begin{pmatrix} 2^{n+1} + 3^n - 2^n \\ n3^{n-1} \\ 3^n \end{pmatrix}$$

Donc $a_n = 2^{n+1} + 3^n - 2^n = 2^n \times 2 + 3^n - 2^n = 2^n - 3^n$ et $b_n = n3^{n-1}$.