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)
```

.....

```
np.zeros(4, dtype=int)
```

.....

```
np.ones(3)
```

.....

```
np.eye (3).
```

.....

— Création de tableaux aléatoires : essayez et expliquez `np.random.rand(6)`

.....

```
np.random.randint(2,6,20)
```

.....

— Création de listes régulières :
Essayez et expliquez `np.arange(0,3,0.1)`

.....

.....

et `np.linspace(0,10,51)`

.....

.....

— 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 ?

.....
.....

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 à la main puis vérifier avec `Python` $A = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 1 & 3 \\ 0 & 1 & 4 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 0 & 0 \\ 0 & 3 & 5 \end{pmatrix}$.

.....
.....
.....

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(....)
    resultat=A
    for k in range(n-1):
        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.min(A)

.....

np.max(A)

.....
np.cumsum(A)

.....
```

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

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

   (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(....)
  X=np.array(....)
  for k in range(n):
     X=np.dot(....)
  return X[0]
```

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

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