PYTHON : INTRODUCTION À NUMPY

Le but de ce TD est de travailler le cours en ligne sur le site www.molin-mathematiques.fr. Ne pas hésiter à taper les exemples du cours pour mieux comprendre.

```
import numpy as np
```

EXERCICE 1 (Mise en perspecive)

On se propose de comparer 4 algorithmes qui réalisent le même travail, mais programmés différemment. Pour cela, on utilisera la fonction `time` du module du même nom, et on mesurera le temps mis par chaque algorithme pour s'exécuter.

Le but de cette exercice est de sensibiliser à l'efficacité de NumPy pour réaliser certaines tâches.

```
from time import time
N = 1000000
debut = time()
# écrire l'algorithme
print('durée = ', time()-debut)
```

Voici ce que l'on veut faire de 4 méthodes différentes (lire la totalité de l'énoncé de l'exercice avant de commencer à coder).

- Créer un tableau (ou une liste) `a` qui contient tous les entiers de 0 à $N-1$ (dans l'ordre).
- Créer un deuxième tableau (ou liste) `b` de taille N qui ne contient que des 1.
- Créer un troisième tableau (ou liste) `c` égal à la somme (terme à terme) de `a` et de `b`.
  Ainsi, dans notre exemple, `c` contiendra tous les entiers de 1 à N.

On pourra commencer par une petite valeur pour N.

1) Programmer l'algorithme en utilisant des boucles `for` pour construire des listes, et les opérations de concaténation du type `a+=[i]`.

2) Programmer le même algorithme, mais en remplaçant les opérations de concaténation par l'utilisation de la méthode `append` pour les trois listes.

3) Programmer le même algorithme, mais en utilisant la syntaxe par compréhension.

4) Programmer le même algorithme avec NumPy :

```
a = np.arange(N)
b = np.ones(N)
c = a+b
```

Mesurer et comparer le temps mis par chacun des trois algorithmes.

EXERCICE 2 (Création de matrices)

1) Créer un tableau comprenant les entiers de 1 à 10.
Le faire à partir d'une liste, puis à partir de la commande numpy `arange`.

2) Créer la matrice 0 de $\mathcal{M}_{4,5}(\mathbb{R})$.
Le faire à partir d'une liste, puis à partir de la commande numpy `zeros`.

3) Créer une matrice de $\mathcal{M}_{4,5}(\mathbb{R})$ dont les coefficients sont des nombres aléatoires pris dans $[\![0,99]\!]$.

4) Créer la matrice identité de taille 5.
Avec une liste, puis avec la commande `eye` puis avec la commande `diag`.

5) Créer une matrice triangulaire supérieure avec des 1 sur et au dessus de la diagonale.

6) Créer une matrice triangulaire supérieure avec des 1 sur la diagonale, des 2 sur la "deuxième" diagonale au dessus, des 3 sur la diagonale suivante...

7) Créer une matrice symétrique avec des 1 sur la diagonale et des -1 ailleurs.

8) Créer la matrice $M = \left(a_{i,j}\right) \in \mathcal{M}_n(\mathbb{R})$, telle que $\forall (i,j) \in [\![1,n]\!]^2, a_{i,j} = \frac{1}{i+j}$
Le faire avec des listes par compréhension, puis avec la commande `fromfunction`.
Attention : dans Python, les indices commencent à 0, il faut donc les décaler.

EXERCICE 3 (Opérations sur les tableaux)

1) Créer le tableau contenant les carrés de 0 à 100.
Le faire en utilisant les listes par compréhension, ou à partir de la fonction `arange` et des opérations entre tableaux.

2) Créer une fonction `estNulle(M)` qui teste si une matrice à deux dimensions est nulle (on ne connaît pas les dimensions de la matrice).

3) Créer une fonction `estEgal(M,N)` qui teste si deux matrices sont égales (pour simplifier, on pourra les supposer de même dimension).

4) Reprogrammer le produit matriciel entre deux matrices carrées de même taille.
Tester si le résultat est le même que celui donné par numpy en essayant avec des matrices carrées remplies de nombres entiers aléatoires pris entre 0 et 9.

5) Programmer une fonction `moyenne(M)` qui donne la moyenne des coefficients d'une matrice `M`.
Tester le programme en comparant le résultat avec celui donné par la fonction dédiée de `numpy`.

6) Faire de même avec l'écart-type.

EXERCICE 4 (Opérations matricielles)

1) Résoudre matriciellement avec numpy le système d'équations

$$\begin{cases} 5x+3y-2z+3t &= 1 \\ x-y+z &= 0 \\ 2x+2y+3z-t &= 4 \\ x+y+z+t &= 2 \end{cases}$$

2) Faire la même résolution d'équations en utilisant l'inverse d'une matrice bien choisie (c'est un système de Cramer).

3) Vérifier les solutions obtenues en réalisant un produit matriciel.

EXERCICE 5 (Probabilités)

De façon "classique", et comme nous l'avons vu en classe :

1) **(a)** Programmer une fonction qui effectue un tirage dans une urne contenant p boules blanches et q boules rouges et qui renvoie 1 si la boule est blanche et 0 sinon.

(b) En réalisant l'expérience un grand nombre de fois, vérifier que la fréquence s'approche de la probabilité de Bernoulli.

2) **(a)** Programmer une fonction qui effectue n tirages successifs avec remise dans une urne contenant p boules blanches et q boules rouges et qui renvoie le nombre de boules blanches obtenues.

(b) En réalisant l'expérience un grand nombre de fois, vérifier que la fréquence s'approche de la probabilité binomiale.

3) Faire de même pour la loi hypergéométrique.

Faire de même avec les fonctions dédiées numpy.