Brève introduction à NumPy

On suppose qu'au début du programme, les bibliothèques suivantes sont chargées :

```
from math import *
import numpy as np
```

Au besoin, n'hésitez pas, après avoir tapé chaque instruction qui va suivre, à user sans réserve de l'instruction `print`, afin de visualiser les résultats et comprendre ce que fait exactement chaque ligne de code.

* La bibliothèque `NumPy` permet la création et la manipulation des tableaux :

```
v = np.array([1,2,3]) \# Definition d'un vecteur
len(v) \# Longueur
M = np.array([[0,1],[2,3],[4,5]]) \# Definition d'une matrice
np.ndim(M) \# Nombre de dimensions
np.size(M) \# Nombre d'elements
np.shape(M) \# Dimensions
```

Pour retrouver les elements du vecteur ou de la matrice, on utilise les crochets `[ ]` :

```
v[0] \# Le premier element
v[1] \# Le deuxi\`eme element
v[2] \# Le dernier element
v[3] \# Erreur, ce n'est pas un element  !
M[0,1] \# Element en ligne 1, colonne 2
```

Notez la différence avec les listes de listes de `Python` standard, où l'élément dans la même position serait noté `M[0][1]`

* Il est possible de générer des éléments particuliers :

```
np.ones(3) \# Vecteur de dimension 3 rempli de 1
np.zeros(3) \# Vecteur de dimension 3 rempli de 0
np.arange(0.,1.,0.1) \# De 0 (inclus) a 1 (exclu) par pas de 0.1
np.linspace(0,1,10) \# 10 points repartis uniformement entre 0 et 1
np.ones([3,4]) \# Matrice de dimension 3 x 4 remplie de 1
np.zeros([3,4]) \# La meme chose avec des zeros\dots
np.eye(3) \# Matrice identite 3 x 3
```

Ou encore, avec `from numpy.random import *`, des matrices aléatoires :

```
size = 10
mean = 2.
stdev = 0.3
normal(mean,stdev,size) \# Distribution normale
low = 0.
high = 5.
uniform(low,high,size) \# Distribution uniforme
randint(low,high,size) \# Distribution uniforme a valeurs entieres
```

* Plusieurs fonctions utiles existent pour les `array` :

```
M.min(), np.min(M) \# Valeur minimale du tableau
M.sum(), np.sum(M) \# Somme des elements du tableau
M.sum(axis = 0) \# Sommes des lignes du tableau
M.sum(axis = 1) \# Sommes des colonnes du tableau
np.mean(M) \# Moyenne des coefficients
```

Les opérations usuelles sur les matrices se font de la façon suivante :

```
M = np.array([[1,2,3],[4,5,6],[7,8,9]])
N = 3*M \# Multiplication de chaque terme
A = M+N \# Sommation terme a terme
np.dot(M,N) \# Produit matriciel (si celui-ci est bien defini)
mat*b \# Produit terme a terme
np.transpose(mat), mat.T \# Transposee d'une matrice
vec = np.array([1,2])
vec.T \# Transposee d'un vecteur ?
vec[np.newaxis].T \# c'est mieux  !
```

* Pour extraire des sous-parties d'un tableau `NumPy`, on peut utiliser l'indexation simple

```
t = np.array([1,2,3,4,5,6])
t[1:3] \# De t[1] a t[2]
t[:3] \# Du debut a t[2]
t[3:] \# De t[3] a la fin
t[:-1] \# Tout sauf le dernier element
t[1:-1] \# Tout sauf le premier et le dernier elements
N = np.array([[0,1,2],[2,3,4],[4,5,6]])
N[0:2,2] \# Deux premieres lignes et troisieme colonne de N
```

Ou alors on utilise un "masque", constitué d'un tableau de booléens :

```
a = np.arange(6)**2
a < 15 \# Le masque (tableau de booleens)
a[a < 15] \# Maniere compacte d'extraire les valeurs < 15
```

Il est possible de redimensionner les tableaux :

```
np.arange(6).reshape(3,2)  \# 3 lignes, 2 colonnes
np.arange(3,9).reshape(3,2) \# 3 lignes, 2 colonnes
```

Et également de les concaténer :

```
tab1 = np.array([1,2])
tab2 = np.array([3,4])
np.concatenate((tab1,tab2)) \# Noter les doubles parentheses  !
```

* **Attention à la copie des tableaux** (voir le début du TP 2)
Vous l'avez vu en fin de S2, pour les scalaires, pas de problème :

```
a = 1
b = a
print('b = ',b)
a = 0
print('b = ',b)
```

mais pour les tableaux, c'est une autre affaire :

```
a = np.zeros([2,2])
b = a
print('b = ',b)
a[1,1] = 10
print('b = ',b) \# Eh oui, petit probleme  !
```

Pour effectuer une copie des valeurs, il faut utiliser a méthode `.copy()` :

```
c = b.copy()
print('c = ',c)
b[1,1] = 0
print('b = ',b)
print('c = ',c) \# c'est mieux  !
```

* Il est également possible d'évaluer une fonction en plusieurs points :

```
def toto(var) :
        return(var-2)
n = 4
x = np.linspace(0,1,n)
y = toto(x) \# y vecteur de composantes yi = f(xi), 0 <= i <= n-1
```

Et mettre en œuvre les fonctions de manière vectorielle :

```
np.cos(x) * np.exp(-x**2) + 2*(x-1)**2
np.sqrt(np.log(x+1))
```

* La documentation `NumPy` est accessible à partir du menu "aide" de Jupyter, et à

https://docs.scipy.org/doc/numpy/reference/

mais en TP dans ce semestre, vous n'avez pas le droit d'utiliser les fonctions avancées ; vous devez vous contenter de celles présentées ici.

F. Goichot, septembre 2023