Glossaire des commandes python 3

Notons que l'indentation est codante en python. Par exemple, un bloc de type boucle ou test doit impérativement être indenté ; le retour à l'indentation initiale signifie que le bloc est fini.

import

permet d'importer des modules. Les modules classiques pour le calcul scientifique sont

```
import numpy              as np
import numpy.linalg       as nlin
import numpy.random       as rd
import scipy              as sp
import scipy.linalg       as slin
import numpy.random       as rd
import matplotlib.pyplot  as plt
```

1 Syntaxe de base

print

permet d'afficher une variable ou une chaîne de caractères

```
print(0)
print(0.)
print("Bonjour")
```

liste

une liste permet de stocker plusieurs objets de type éventuellement différent. On accède aux éléments de la liste à l'aide des crochets. Les indices commencent à 0.

```
L = [0, 1., "toto", "titi"]
print(L[0])
print(L[-1])
L.append(2)
L[1] = 3
print(L)
```

tuple

un tuple permet de stocker plusieurs objets de type éventuellement différent. Il s'utilise comme une liste mais il n'est pas possible de modifier ses éléments.

```
L = (0, 1., "toto", "titi")
print(L[0])
print(L[-1])
print(L)
```

integer - double

le type des variables est déterminé de manière automatique. Il faut faire attention notamment à ne pas faire des calculs dans $\mathbb{R}$ avec une variable vue comme un entier.

```
x = 1
y = 1.
print(x/2)
print(x//2)
print(y/2)
```

format

permet de formater une chaîne de caractères. Les formats à connaître sont `d` pour les entiers, `f` et `e` pour les doubles en précisant le nombre de caractères à formater.

```
print("{0:d} = {0:3d} = {0:4d}".format(123))
print("pi = {0:f}".format(np.pi))
print("pi = {0:10.3e}".format(np.pi))
print("pi = {0:.25f}".format(np.pi))
print("{0:10.3e} = {1:2d}".format(123, 123))
```

for

permet de faire une boucle sur une liste. La plupart du temps, la liste est la suite des entiers de 0 à n mais il est possible de faire une boucle sur n'importe quelle liste.

```
for k in range(10):
    print(k)
for s in ["toto", "titi", "tata"]:
    print(s)
```

while

permet de faire une boucle tant qu'une certaine condition est respectée.

```
k = 0
while k<10:
    print(k)
    k += 1
```

if - elif - else

```
for n in range(10):
    if n == 0:
        print(n, "␣est␣nul")
    elif n%2:
        print(n, "␣est␣impair")
    else :
        print(n, "␣est␣pair")
```

2 Le package numpy

2.1 Création de tableau

Le package `numpy` est accessible sous le diminutif `np` après la commande `import numpy as np`.

zeros

Permet de créer un `numpy array` rempli de 0 de la forme désirée.

```
x = np.zeros(3)      # vecteur
y = np.zeros((3,1)) # matrice colonne
z = np.zeros((1,3)) # matrice ligne
A = np.zeros((3,3)) # matrice carree
```

ones

Permet de créer un `numpy array` rempli de 1 de la forme désirée.

```
x = np.ones(3)      # vecteur
y = np.ones((3,1)) # matrice colonne
z = np.ones((1,3)) # matrice ligne
A = np.ones((3,3)) # matrice carree
```

array

Permet de convertir une liste en `numpy array`.

```
x = np.array([0,1,2])
y = np.array(range(10))
A = np.array([[1,0],[0,1]])
```

arange

permet de créer un `numpy array` d'entiers selon une progression arithmétique.

```
print(np.arange(10))
print(np.arange(2, 10))
print(np.arange(2, 10, 2))
print(np.arange(10, 0, -1))
```

linspace

Permet de créer un `numpy array` contenant le maillage uniforme d'un segment avec nombre de points donné.

```
# maillage de [0,1] avec 100 points
print(np.linspace(0, 1, 100))
```

eye

Permet de créer un `numpy array` de l'identité.

```
Id1 = np.eye(5)          # shape = (5,5)
Id2 = np.eye(N=5, M=4) # shape=(5,4)
Id3 = np.eye(5, k=1)    # upper diagonal
```

diag

Si l'argument n'a qu'une seule dimension, construit un `numpy array` diagonal à partir de ce vecteur. Il est possible de préciser sur quelle diagonale on place le vecteur. Si l'argument possède deux dimensions, construit un `numpy array` avec une seule dimension qui contient la diagonal de l'argument.

```
v = np.linspace(0,1,5)
A = np.diag(v)
B = np.diag(v, k=1)
v = np.diag(B)
```

random_integers

Permet de créer un `numpy array` aléatoire. Les nombres retournés sont des entiers, il faut donc préciser les bornes souhaitées (qui sont inclues). Attention, cette commande diffère légèrement de `randint` pour laquelle la borne supérieure est exclue.

```
# entiers entre -10 et 10
A = rd.random_integers(-10, 10, size=(5,5))
```

random_sample

Permet de créer un `numpy array` aléatoire. Les nombres retournés sont des réels de l'intervalle $[0, 1[$.

```
# matrice de nombres entre
# 0 (inclu) et 1 (exclu)
A = rd.random_sample(size=(5,5))
# matrice de nombres entre
# a (inclu) et b (exclu)
a, b = -10, 10
B = a + (b-a) * rd.random_sample(size=(5,5))
```

2.2 Manipulation de tableau

shape

Retourne la forme du `numpy array`.

```
A = np.zeros(5,10)
print(A.shape)
```

[]

Permet d'extraire les valeurs d'un `numpy array`.

```
A = rd.random_sample((5,4))
print(A[0,0], A[0,-1])
print(A[:,1])
print(A[0:2,0:2])
```

fonctions

De nombreuses fonctions mathématiques peuvent d'appliquer directement sur un `numpy array`

```
A = rd.random_sample((4,5))
print(np.sin(A))
print(np.cos(A))
print(np.tan(A))
print(np.exp(A))
```

sum - prod

Permet de calculer la somme ou le produit des éléments d'un `numpy array`.

```
A = rd.random_sample((5,5))-.5
print(np.sum(A, 0))
print(np.sum(A, 1))
print(np.sum(A))
print(np.sum(abs(A)))
```

2.3 Algèbre linéaire

transpose

Permet de transposer un `numpy array` à deux dimensions

```
A = rd.random_sample((5,5))
print(A)
print(A.transpose())
```

dot

Permet de calculer le produit matriciel entre deux `numpy array` (produit entre deux matrices ou produit matrice-vecteur)

```
A = rd.random_sample((5,5))
b = rd.random_sample((5,))
print(np.dot(A, b))
print(np.dot(A, A))
```

outer

Permet de calculer la matrice `numpy array` $(u_i v_j)$ à partir des deux vecteurs `numpy array` u et v.

```
u = rd.random_sample((5,))
v = rd.random_sample((7,))
print(np.outer(u, v))
```

solve

Permet de résoudre un système linéaire.

```
A = rd.random_sample((5,5))
b = rd.random_sample((5,))
x = nlin.solve(A, b)
print(np.allclose(A.dot(x), b))
```

norm

Permet de calculer la norme d'un `numpy array`. Un argument permet de préciser quelle norme calculer :

ord	matrice	vecteur
None	Frobenius	$\|\cdot\|_2$
'fro'	Frobenius	
np.inf	$\|\cdot\|_\infty$	$\|\cdot\|_\infty$
1	$\|\cdot\|_1$	$\|\cdot\|_1$
2	$\|\cdot\|_2$	$\|\cdot\|_2$

det

Permet de calculer le déterminant d'une matrice.

```
A = rd.random_sample((5,5))
print(nlin.det(A))
```

inv

Permet de calculer l'inverse multiplicative d'une matrice.

```
A = rd.random_sample((5,5))
print(np.allclose(nlin.inv(A).dot(A),
                  np.eye(A.shape[0])))
```