1 Tableaux unidimensionnels avec numpy

Le module `numpy` permet de définir et de manipuler des tableaux numériques dont l'usage est plus pratique pour le calcul scientifique que les listes de `python` : d'une part le module dispose de fonctions spécifiques permettant leur manipulation, d'autre part on peut leur appliquer des opérations et fonctions mathématiques terme à terme.

On commence par importer le module `numpy` :

```
import numpy as np
```

Tous les appels des fonctions de `numpy` devront alors être précédés du préfixe "`np.`".

1.1 Création de tableaux sous numpy

On peut définir simplement des tableaux unidimensionnels grâce aux fonctions :

• `array(Liste)` : elle prend en paramètre une liste et retourne un tableau `numpy` contenant les mêmes valeurs :

```
In[1] : np.array([2, 0.5, -3, 7.0]) # prend en paramètre une liste
Out[1] : array([2, 0.5, -3, 7.0])    # et retourne un tableau numpy
```

• `arange(a,b)` ou `arange(a, b, k)` : elle prend en paramètre deux/trois nombres à virgule `a`, `b` et `k` (par défaut `k` vaut 1) et retourne le tableau `numpy` des valeurs de `a` (inclus) à `b` (exclu) par pas de `k`. Son fonctionnement est analogue à la fonction `range()`, à ceci près que `arange()` accepte en paramètres des nombres à virgules :

```
In[2] : np.arange(0, 2, 0.5)    # de 0 à 2 (exclu) par pas de 0.5
Out[2] : array([0, 0.5, 1, 1.5])
```

• `linspace(a, b, n)` : elle prend en paramètres 2 nombres à virgule `a` et `b` et un entier positif `n` et retourne le tableau constitués de n valeurs régulièrement espacées de `a` à `b` (inclus) :

```
In[3] : np.linspace(0, 2.5, 6)
Out[3] : array([0, 0.5, 1, 1.5, 2, 2.5])
```

• `zeros(n)` : prend en paramètre un entier `n` et constitue un tableau contenant `n` zéros.

```
In[3] : np.zeros(5)
Out[3] : array([0, 0, 0, 0, 0])
```

1.2 Manipulation de tableaux

La manipulation des éléments d'un tableau `numpy` est identique à celle des liste `python` :

– On accède à la valeur d'un élément par son indice entre crochets : `tab[i]`,
– on peut appliquer du slicing pour extraire un sous-tableau : `tab[i:j]` ou `tab[i:j:k]`
– on parcourt un tableau `tab` grâce à une boucle `for` :

```
for x in tab:
```

– la fonction `len()` de `python` retourne son nombre d'éléments : `len(tab)`,
– la fonction `sum()` de `python` retourne la somme des éléments d'un tableau.

• Les opérations arithmétiques sur des tableaux `numpy` sont exécutées termes à terme :

```
In[4] : T1 = array([0, 2, -3, 5.5])
In[5] : 2 * T1 + 1
Out[5] : array([1, 5, -5, 12])
```

Exercice 1

1. Ecrire une fonction `suite(n)` prenant en paramètre un entier `n` et qui retourne le tableau `numpy` des termes u_0, u_1, $\dots u_n$ de la suite définie par :
$$\forall\, n \in \mathbb{N},\ u_n = \frac{1}{\sqrt{1+n^2}}$$
2. Calculer sans boucle `for`, à l'aide de la fonction `sum()` de `python` et d'un tableau `numpy` la somme suivante :
$$\sum_{k=1}^{100} \frac{(2.k+1)^3}{k}$$

• `numpy` contient aussi toutes les fonctions mathématiques usuelles (qui se passent de description) :

```
exp  log  sin  cos  tan  arcsin  arccos  arctan  cosh  sinh  tanh
```

mais aussi : `abs` (valeur absolue), `floor` (partie entière), etc...

Appliquées à des tableaux `numpy` elles s'appliquent terme à terme.

2 Tracés dans le plan avec pyplot

On commence par importer le sous-module `pyplot` de `matplotlib` :

```
import matplotlib.pyplot as plt
```

Tous les appels de fonctions de `pyplot()` devront être précédées du préfixe "`plt.`".

• `figure(n)` crée la figure n. Les fonctions de `pyplot` lui seront alors appliquées.
• `clf()` efface la figure courante.

• `title("Le titre")` pour donner un titre à la figure.
• `axis([xa, xb, ya, yb])` affichage pour abscisses $[xa, xb]$ et ordonnées $[ya, yb]$.
• `axhline(color='black')` trace l'axe des abscisses (en noir).
• `axvline(color='black')` trace l'axe des ordonnées (en noir).
• `show()` provoque l'affichage de la figure courante.

2.1 Tracé de la courbe représentative d'une fonction

• `plot(X,Y)` où `X` est une liste ou un tableau `numpy` des abscisses et `Y` est une liste ou un tableau `numpy` des ordonnées trace la courbe passant par les points de coordonnées (x, y) avec x décrivant `X` et y décrivant (simultanément !) `Y`.

Exercice 2

(1) Exécuter le code suivant pour comprendre son fonctionnement :

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-10,10,1000)
Y = np.cos(X)
plt.figure(1)
plt.plot(X,Y)
plt.title("Tracé de cos sur [-10;10]")
plt.show()
```

(2) Effectuer le tracé de la courbe représentative de la fonction exponentielle sur l'intervalle $[-5, 2]$.

2.2 Tracé d'un histogramme

Deux solutions pour tracer un histogramme : Soient `X` liste/tableau des abscisses et `Y` liste tableau des ordonnées.

• `vlines(X,0,Y,lw = 30)` trace des lignes verticales au dessus des abscisses dans `X` allant des ordonnées 0 aux valeurs dans `Y`, avec épaisseur des traits 30 (optionnel).

• `bar(X,Y)` trace l'histogramme au dessus des abcisses dans `X` des valeurs dans `Y`.
Pour que les barres soient centrées au dessus de leur abscisse il faudra décaler les valeurs dans `X` en leur retranchant 0.4 : `X = [x-0.4 for x in X]`.

Exercice 3

Simulation d'une loi de Bernoulli

Soit X une v.a.r. de Bernoulli de paramètre p ($0 < p < 1$). On rappelle que son univers image est $X(\Omega) = \{0, 1\}$ et sa loi est $\mathbb{P}(X = 1) = p$, $\mathbb{P}(X = 0) = 1 - p$.

1. Ecrire une fonction `simulBernoulli(p)` prenant en paramètre un nombre $p \in]0, 1[$ et qui simule le résultat d'une expérience aléatoire suivant une loi de Bernoulli de paramètre p. Elle retournera 0 ou 1 et utilisera la fonction `random` du module `random`.
2. Ecrire une fonction `loiBernoulli(p,k)` prenant en paramètres p et un entier k et qui retourne une estimation de $\mathbb{P}(X = k)$. Pour celà elle appellera 1000 fois la fonction `simulBernoulli(p)` et retournera la proportion des résultats k obtenues.
3. Effectuer le tracé de l'histogramme de la loi de Bernoulli à l'aide de cette simulation.