Aide-mémoire Numpy pour les activités expérimentales en Physique-Chimie

À quoi sert le module Numpy ?

Le module `numpy` permet de créer et de manipuler facilement des tableaux homogènes, également appelés *arrays*, à une, deux ou trois dimensions. En travaux pratiques, il est donc particulièrement efficace pour stocker et traiter des collections de données expérimentales, sans nécessiter l'écriture systématique de boucles (contrairement aux listes), et se substitue aisément à un tableur.

Importations

Dans toute la fiche, on supposera que le module `numpy` et le sous-module `numpy.random` (utile pour générer des nombres aléatoires) ont été importés comme suit :

```
» import numpy as np
» import numpy.random as rd
```

Création d'un tableau à 1D

Pour créer un tableau qui correspond à la série de valeurs

notes	14.75	18.25	11.5	13	15.75	12

on saisit l'instruction

```
» notes = np.array([14.75, 18.25, 11.5, 13, 15.75, 12])
```

Nombre d'éléments d'un tableau

```
» len(notes)
```
(à 1D uniquement !)

Création de tableaux prédéfinis

Tableau qui ne contient que des 0 (10 dans le cas ci-après) :

```
» t0 = np.zeros(10)
```

La même chose avec que des 1 :

```
» t1 = np.ones(10)
```

Tableau qui contient des valeurs régulièrement espacées, par pas de 0.2, entre 0 (**inclus**) et 10 (**exclu**) :

```
» t2 = np.arange(0, 10, 0.2)
```

Tableau qui contient 100 valeurs régulièrement espacées, entre 0 (**inclus**) et 10 (**inclus**) :

```
» t3 = np.linspace(0, 10, 100)
```

Tableaux de nombres aléatoires

Exemple de tableau qui contient 1000 réalisations d'une VA gaussienne d'espérance 0 et d'écart-type égal à 1 :

```
» t4 = rd.normal(0, 1, 1000)
```

Exemple de tableau qui contient 10 réalisations d'une VA uniforme sur $[-1, 1[$:

```
» t5 = rd.uniform(-1, 1, 10)
```

Fonctions « statistiques » utiles

Valeur min.	`» np.min(t4)`
Valeur max.	`» np.max(t4)`
Somme des éléments	`» np.sum(t4)`
Moyenne	`» np.mean(t4)`
Écart-type expérimental	`» np.std(t4, ddof = 1)`

Indexation des éléments

Chaque élément d'un tableau est repéré par un indice de position. L'indexation commence à 0. Par exemple :

éléments de `notes` :	14.75	18.25	11.5	13	15.75	12
indices de position :	0	1	2	3	4	5

Instructions de base :

`» notes[2]`	renvoie l'élément d'indice 2
`» notes[1:4]`	renvoie le sous-tableau contenant les éléments d'indices 1 (**inclus**) à 4 (**exclu**)
`» notes[1::3]`	renvoie le sous-tableau contenant tous les éléments depuis l'indice 1, par pas de 3

On peut aussi utiliser des indices négatifs ! Le dernier élément possède alors l'indice -1.

éléments de `notes` :	14.75	18.25	11.5	13	15.75	12
indices de position :	-6	-5	-4	-3	-2	-1

Opérations arithmétiques

Les opérateurs arithmétiques +, -, *, / permettent de réaliser des opérations termes à termes sur un ou plusieurs tableaux. Par exemple, l'instruction

```
» notes + 1.25
```

permet d'ajouter le nombre 1.25 à tous les éléments du tableau `notes` (sans modifier son contenu).

Les instructions

```
» coef = np.array([4, 5, 5, 4, 3, 2])
» notes * coef
```

permettent d'obtenir un tableau dont chaque élément résulte du produit entre les éléments des tableaux `notes` et `coef` de même position.

Fonctions mathématiques

Toutes les fonctions mathématiques usuelles sont disponibles dans le module `numpy` et s'appliquent à l'ensemble des termes du tableau passé en argument.

Quelques exemples :

`» np.sqrt(t2)`	fonction racine carrée
`» np.log(notes)`	fonction logarithme népérien
`» np.cos(t4)`	fonction cosinus
`» t5**3`	élévation à la puissance 3

Copie profonde d'un tableau

Pour copier le contenu du tableau `t5` dans un tableau nommé `t6`, il faut **impérativement** utiliser l'expression

```
» t6 = np.copy(t5)
```

On peut ensuite modifier le contenu de `t5` sans que cela affecte `t6` (et réciproquement !).

Auteur : Émilie Frémont
Contact : fremont.emilie@gmail.com