TP3 - La bibliothèque numpy

Le but de ce TP est d'introduire le concept de bibliothèques en `Python` et de commencer à en utiliser une particulière, la bibliothèque `numpy`.

1 Qu'est-ce qu'une bibliothèque ?

1.1 Calcul d'une racine

Nous avons évité jusqu'à présent le calcul de racine carrée. La raison en est très simple : c'est passablement compliqué. Mais c'est pour nous une bonne occasion de découvrir le concept de bibliothèque.

Exercice 1 ⋆

1. Recopier le code [1] suivant :

```
def sqrt(x):
  if x == 0:
    return 0
  steps = 10
  y = 1
  for i in range(steps):
    y = y - (y*y - x)/(2*y)
  return y
```

2. Vérifier les valeurs renvoyées par `sqrt(4)`, `sqrt(9)`, `sqrt(16)`, `sqrt(0.25)`, `sqrt(1)` et `sqrt(0)`.
3. Vérifier enfin le résultat de `sqrt(2)*sqrt(2)`, `sqrt(3)*sqrt(3)` et `sqrt(0.5)*sqrt(0.5)`.

1.2 Utilisation d'une bibliothèque

Il serait bon de ne pas avoir à écrire la fonction précédente dès que l'on veut calculer une racine carrée. C'est justement le but des bibliothèques. Une bibliothèque est un fichier qui contient du code `Python`, principalement des fonctions, que l'on peut *importer* pour les utiliser dans notre code. L'avantage des bibliothèques est donc dans la *réutilisation* du code. On peut même réutiliser le code fait par d'autres personnes.

Exercice 2 ⋆

Utiliser les différents codes suivants :

```
from math import sqrt
print(sqrt(4))
```

```
from math import *
print(sqrt(4))
```

```
import math as m
print(m.sqrt(4))
```

Ces codes se comportent-ils différement ? Remarquez-vous des différences dans l'utilisation des fonctions de la bibliothèque ? À votre avis que signifie l'étoile $*$ dans le deuxième code ?

1. Ce code utilise un algorithme, appelé la méthode de Newton, pour calculer une approximation de la racine carrée. Vous n'avez pas besoin de comprendre cet algorithme pour la suite.

Exercice 3 ⋆⋆

En utilisant la bibliothèque `math`, écrire les trois fonctions suivantes :

1. `def cote(a):` qui calcule le côté d'un carré d'aire a ;
2. `def surface(r):` qui calcule la surface d'un disque de rayon r ;
3. `def angle(a,h):` qui calcule l'angle (en degrés) dans un triangle rectangle d'hypoténuse h et de côté adjacent à l'angle de longueur a.

On pourra utiliser `math.acos`, `math.pi` *et* `math.sqrt`.

2 La bibliothèque Numpy

2.1 Introduction

Cette année, nous utiliserons principalement (et presque exclusivement) la bibliothèque `numpy`. Elle contient de nombreuses fonctions spécialisées dans le *calcul numérique*, c'est-à-dire qu'elle permet de faire des simulations numériques et des calculs d'approximations. La bibliothèque `numpy` implémente beaucoup de fonctions semblables à `math` mais avec un accent sur la rapidité et l'efficacité numérique.
Presque toute l'année, nous importerons la bibliothèque `numpy` de la manière suivante :

```
1 import numpy as np
```

Exercice 4 ⋆⋆

Réimplémenter les trois fonctions de l'exercice 3 en utilisant `numpy`.

On pourra utiliser `np.arccos`, `np.pi` *et* `np.sqrt`.

2.2 Les tableaux numpy

L'outil principal que nous utiliserons cette année est celui des *tableaux* `numpy`. Pour nous, il s'agit d'un nouveau de type de variables. Il y avait jusqu'à les nombres, les chaines de caractères, les booléens et désormais, il y aura les tableaux `numpy`.

Exercice 5 ⋆

1. Exécuter les lignes suivantes :

```
1 tab = np.zeros(5)
  print(tab)
```

   Que se passe-t-il ? Que se passe-t-il si on change le 5 en un autre nombre ? Quel est le rôle de la fonction `np.zeros` ?

2. Remplacer la fonction `np.zeros` par la fonction `np.ones`. Que se passe-t-il ?
3. Utiliser le code suivant :

```
1 tab = np.array([1,2,3])
  print(tab)
```

   Que fait la fonction `array` ?

4. Recopier le code suivant :

```
1 n = 10
  tab = np.zeros(n)
  for k in range(n):
    tab[k] = k*k
5 print(k)
```

   À votre avis, que peut signifier la syntaxe `tab[k]` ? Rappeler quel est l'ensemble des valeurs parcourues par `k` dans la boucle ? Quel est le numéro du premier élément du tableau `tab` ?

Exercice 6 ⋆⋆

Écrire une fonction `def fibonacci(n):` qui prend un entier $n \geqslant 1$ et renvoie un tableau `numpy` contenant les n premiers nombres de la suite de Fibonacci, qui est définie par $\begin{cases} u_0 = 1 \\ u_1 = 1 \\ \forall n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n \end{cases}$.

2.3 Manipulations élémentaires

Exercice 7 ⋆⋆

Exécutez le code suivant :

```
print(np.arange(7))
print(np.arange(2,6))
print(np.arange(1,9,2))
print(np.linspace(1,5,3))
print(np.linspace(1,5,5))
print(np.linspace(1,5,9))
```

Que font les fonctions `np.arange` et `np.linspace` ?

Exercice 8 ⋆

Exécutez le code suivant :

```
t1 = np.arange(1,11,1)
t2 = np.arange(11,1,-1)
resultat = t1 + t2
print(t1)
print(t2)
print(resultat)
```

Que se passe-t-il lorsque l'on additionne 2 tableaux `numpy` ?

De manière générale, on peut utiliser les opérations algébriques usuelles sur des tableaux `numpy` de même taille et cela va faire l'opération terme à terme.

Exercice 9 ⋆⋆

Sans écrire de boucles, écrire une fonction `def carres(n):` utilisant des tableaux `numpy` et renvoyant un tableau contenant les n premiers entiers carrés en commençant par 0.

Exercice 10 ⋆⋆

1. Préparer un tableau `numpy` `angles` contenant les valeurs 0, $\frac{\pi}{6}$, $\frac{\pi}{4}$, $\frac{\pi}{3}$ et $\frac{\pi}{2}$.
2. Exécuter `np.cos(angles)`. Que constate-t-on ?

Exercice 11 ⋆

Certaines fonctions sont au contraire prévues pour agir directement sur un tableau.

1. Construire un tableau `numpy` `A` contenant les nombres 1, 42, -2, 0.75 et 0.25 dans cet ordre.
2. Exécuter les fonctions suivantes une à une et déterminer leurs rôles :

```
np.min(A)
np.max(A)
np.mean(A)
np.sum(A)
np.prod(A)
```

3. Exécuter également les fonctions suivantes : `np.cumsum` et `np.cumprod`. Qu'est-ce qui les distinguent des fonctions précédentes ?

2.4 Utilisation comme matrices

Les tableaux `numpy` peuvent avoir plusieurs *dimensions*. Pour l'instant, nous les avons toujours présentés en tableau à 1 ligne et plusieurs colonnes, mais il est tout à fait possible de faire des tableaux à plusieurs lignes et plusieurs colonnes. Cela permet notamment de représenter des *matrices*.

Exercice 12 ⋆

1. Créer un tableau `numpy` `A` arbitraire. Puis exécuter `np.shape(A)`. Que fait la fonction `np.shape` ?
2. Exécuter maintenant :

```
A = np.eye(5)
print(np.shape(A))
```

Que s'affiche-t-il ? Comment l'interpréter ? Exécuter directement `print(np.eye(5))`. Que fait `np.eye` ?

3. Que fait `np.ones((2,3))` ?
4. Que fait `np.array([[1,2,3],[4,5,6]])` ?

Exercice 13 ⋆⋆

1. Afficher le résultat de :

```
np.ones((3,3))*np.ones((3,3))
```

Est-ce le résultat attendu ? Quel problème cela pose ?

2. Exécuter :

```
np.dot(np.ones((3,3)),np.ones((3,3)))
```

Que fait `np.dot` ?

3. Que font les fonctions `np.linalg.inv` et `np.linalg.det` ? Vérifiez-le sur quelques exemples.