Feuille de TP 2

Tableaux

1 Avant de commencer

Commencer par vous rendre, dans un terminal, dans le dossier consacré aux TPs M208. Y créer un répertoire spécifique à ce tp :

```
mkdir TP2_numpy
cd TP2_numpy
jupyter3-notebook
```

— La commande `mkdir nom_dossier` permet de créer un dossier dont le nom sera "nom_dossier"
— La commande `cd nom_dossier` permet d'aller dans le dossier "nom_dossier"
— La commande `jupyter3-notebook` permet de lancer les notebook pour python 3.

Les notebook sont une interface web pour le développement de code python. Cliquer sur `new` puis choisissez `python3` pour créer un nouveau notebook en python 3. Un notebook est constitué de cellules (appelées *cell*) qui peuvent contenir soit du code soit du texte (y compris des formules mathématiques, des schémas, etc). Le code peut être exécuté directement dans le notebook et le notebook peut être sauvegardé pour être réutilisé.

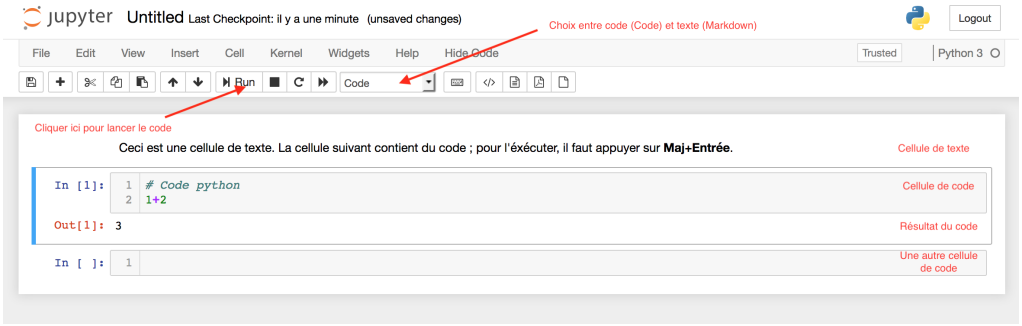

FIGURE 1 – Présentation de jupyter notebook

Enfin, notez que python est un logiciel gratuit et libre, installé par défaut sous mac et sous linux et qu'il est facile de l'installer sous windows. La suite "anaconda" (disponible sous linux, mac et windows) permet de facilement installer python 3, les notebooks et les librairies que nous allons utiliser.

Ces notebooks nous permettrons de vous donner des sujets à trou ; vous pourrez aussi noter (en texte) des remarques qui vous seront utiles ou répondre à des questions plus théoriques.

Exercice 2.1 : Les scripts

1. Afin de pouvoir exécuter plusieurs fois un même code et de pouvoir sauvegarder ses codes (d'un TP à l'autre par exemple), la façon la plus simple de faire est d'utiliser des scripts. Avec les notebooks, cela sera plus utilisé pour utiliser le même bout de code dans plusieurs notebook (par exemple s'ils se suivent).
2. Créer une cellule de code avec le contenu suivant :

```
%%writefile MonScript.py
# -*- coding: utf-8 -*-
print("Voici mon premier script")
```

La commande `%%writefile MonScript.py` enregistre le contenu de la cellule dans le fichier *MonScript.py*. Vous auriez pu créer ce script depuis n'importe quel éditeur de texte (notepad, gedit, emacs, vim, etc). Pour l'exécuter, tappez `%run MonScript.py` dans une autre cellule.

3. Créer un script qui demande à l'utilisateur de saisir deux nombres décimaux et qui affiche ensuite la somme de ces 2 nombres.

2 Tableaux

Nous avons vu précédemment que la commande `import` permettait de charger un module python. Il existe de nombreuses bibliothèques python permettant d'en étendre les fonctionnalités. Leurs possibilités dépassent largement le cadre de ce cours : développement web, application web, jeux vidéos, interface graphique, traitement d'image, traitement de son, musique par ordinateur, cryptographie, etc. Dans le cadre de ce cours, nous utiliserons surtout deux bibliothèques : `numpy` qui ajoute des fonctions mathématiques, des tableaux et des opérations sur les tableaux (dont les opérations matricielles usuelles) et la bibliothèque `matplotlib.pyplot` qui permet de tracer et/ou représenter graphiquement des fonctions et des courbes 2D ou 3D (pour être précis pyplot est une sous-bibliothèque de matplolib).

Python ne propose pas, de façon standard, de tableau. Par contre, il dispose de plusieurs ajouts qui en proposent. Le plus connu d'entre eux est numpy. Pour pouvoir utiliser numpy, il faut le charger à l'aide de la commande `import` :

```
import numpy as np
# Desormais les commandes numpy sont disponibles.
# Il faut juste les prefixer par "np."
A = np.array([1, 2]) # Crée un tableau de dimension 1 à 2 élements
                     # dont le premier vaut 1, le second 2.
```

Exercice 2.2 : Affectation et premières manipulations

Saisir les commandes suivantes

1. Allocation et affectation :

```
X, Y = np.zeros(3), np.ones((1,3)) # vecteurs de 0 ou de 1
print(X)
print(Y)

print(type(X))
T = [1, 2, 3]
print(type(T)) # T est une liste
X = np.array(T) # on peut convertir les types (sans faire de copie)
print(type(X))
print(X)

X = np.array([[1,2,3],[4,5,6]]) # tableau 2D saisi a la main
print(X)
Y = np.empty(X.shape) # tableau 2D vide pour reserver la place memoire
print(Y)
Y = np.sin(X)
print(Y)
```

2. Les opérations sont termes à termes :

```
print(X+1) # ajoute un à tous les elements
print(2*X) # multiplie par 2 tous les elements
print(X**2) # élève au carré tous les elements
```

```
print(np.zeros((2,3))) # matrices rectangulaires de 0
print(np.ones((2,3))) # matrices rectangulaires de 1
print(np.eye(3)) # matrice identite

Z = np.ones(X.shape)
print("Les opérations sont termes à termes : X+Y= ",X+Y,"\n et X*Y= ",X*Y)
```

Les tableaux peuvent être de dimension 1 à 27 ; en général, nous utiliserons des tableaux de dimension 1, de dimension 2 ou 3. Il est possible d'utiliser des masques pour travailler sur des sous tableaux : pour des tableaux de dimension 2, cela se fait via la commande `my_array[Idebut:Ifin,Jdebut:Jfin]`. Attention, les indices commencent à 0 et ici on prendra i allant de `Idebut` inclus à `Ifin` **exclu**.

```
import numpy as np
x = np.arange(0,20,2)
print(x)
print(x[1:])
print(x[2:5])
print(x[4:10:2]) #ici on va de 4 à 10 mais de 2 en 2.
print("dernier élément = ", x[-1]," et avant dernier = ",x[-2])
z = np.array([[1,2,3],[2,4,6]])
print(z[:,1:2])
```

Enfin, il faut faire attention : lorsque l'on souhaite copier un tableau, il faut le faire explicitement :

```
import numpy as np
# Ne pas utiliser le égal pour copier le contenu d'un tableau
A = np.arange(10)
print("A =",A)
B = A
B[0] = 2
print("A =", A, " et B =", B)
# Utiliser la méthode copie :
A = np.arange(10)
print(A)
B = A.copy()
B[0] = 2
print("A =", A, " et B =", B)
```

Si l'on avait fait `B = 1*A`, cela aurait fonctionné grâce à la présence d'une opération.

```
import numpy as np
# Ne pas utiliser le égal pour copier le contenu d'un tableau
A = np.arange(10)
print("A =",A)
B = 1*A
B[0] = 2
print("A =", A, " et B =", B)
```

Pour les informaticiens : pour `A=B`, python réalise une copie de la référence, A et B sont deux pointeurs pointant vers le même tableau.

Exercice 2.3 :

1. Importez les packages `numpy` comme `np`. Testez sur un tableau 2D `A` et un tableau 1D `x` représentant, respectivement une matrice et un vecteur, les méthodes `shape`, `size`, `ndim`, `dtype`, `sum` et `prod` (une méthode s'utilise sur une variable `B` à l'aide de la commande `B.methode()`).
2. Comment obtenir le nombre de lignes, de colonnes, le produit des coefficients sur les lignes, les colonnes de la matrice.
3. Pour les mêmes `A` et `x`, testez `A*x`, `A.dot(x)` et enfin `np.dot(A, x)`. Commentez les résultats.

Exercice 2.4 : Autour du produit matriciel

1. Déclarer les tableaux suivants à l'aide `np.array` (C et D doivent être de dimension 1).

$$A = \begin{bmatrix} 7 & 0 \\ -1 & 5 \\ -1 & 2 \end{bmatrix}, B = \begin{bmatrix} 1 & 4 \\ -4 & 0 \end{bmatrix}, C = \begin{bmatrix} 7 & 3 \end{bmatrix}, D = \begin{bmatrix} 8 & 2 \end{bmatrix}.$$

2. Calculer, avec python, les produit matriciels $AB, CA, CD, DC, DBC, A^TA, AA^T$. La commande `np.dot` permet de faire le produit matriciel (taper `help(np.dot)` pour en savoir plus). Au besoin on pourra utiliser `reshape` pour changer la dimension et le profil. Est-ce nécessaire ici ?

Exercice 2.5 : Affectation rapide

1. Définir de la façon la plus simple possible les vecteurs :

   $v1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \ldots, 16], v2 = [0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, \ldots, 2.0], v3 = [1, 2, 4, 8, 16, 32, 64].$

2. Définir de la façon la plus simple possible les tableaux :

$$A = \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^8 \\ 1 & 3 & 9 & \cdots & 3^8 \\ 1 & 5 & 25 & \cdots & 5^8 \end{bmatrix}, X = \begin{bmatrix} 0.0 & 0.2 & 0.4 & \cdots & 1. \\ 0.0 & 0.2 & 0.4 & \cdots & 1. \\ 0.0 & 0.2 & 0.4 & \cdots & 1. \end{bmatrix}, Y = \begin{bmatrix} 0.0 & 0.0 & 0.0 & \cdots & 0.0 \\ 0.5 & 0.5 & 0.5 & \cdots & 0.5 \\ 1.0 & 1.0 & 1.0 & \cdots & 1.0 \end{bmatrix},$$

3. Créer un tableau à 2 dimensions et 10×10 éléments dont tous les éléments sont égaux à 0 sauf ceux aux bords et sur la diagonale qui seront pris égaux à 1.
4. Créer un tableau à 2 dimensions et 10×10 éléments dont tous les éléments sont égaux à 0 sauf les éléments diagonaux qui seront égaux à 2 et les éléments immédiatement au-dessus ou en-dessous de la diagonale qui seront égaux à -1 :

$$L = \begin{bmatrix} 2 & -1 & 0 & 0 & \ldots & 0 & 0 \\ -1 & 2 & -1 & 0 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & \\ \vdots & \ldots & & \ddots & \ddots & \ddots & -1 \\ 0 & \ldots & & & 0 & -1 & 2 \end{bmatrix}.$$

5. Créer un tableau T à 2 dimensions tel que $T[i, j] = 1$ si $i + j$ est paire, et $T[i, j] = 0$ sinon.