M1 IM/MPA : TP EDP et Différences Finies.

Feuille 0 : Introduction à numpy.

Ce TP a pour vocation de vous familiariser avec les commandes de base de *Python* et en particulier du module *NumPy*.

L'apprentissage d'un langage de programmation repose fortement sur l'expérimentation et sur la consultation de ressources. Les quelques commandes réunies ici seront évidemment insuffisantes. Si vous souhaitez de l'aide sur une commande en particulier (par exemple sur la commande "*range*", vous pouvez tapez directement " *? range*".

Attention : si vous copiez/coller directement depuis ce pdf, certains caractères de police peuvent ne pas être reconnus dans Python et générer des messages d'erreur. Il est donc plus sûr d'éviter les copier/coller.

Quelques astuces :

Commenter : tout ce que vous écrivez après le symbole # n'est pas compilé, cela vous permet de commenter votre code, ce qui est très utile lorsque vous rouvrez un vieux fichier, ou que vous partagez votre code.

Interrompre : pour interrompre une compilation en cours (par exemple si vous êtes coincés dans une boucle "while" infinie, ou trop longue), placez vous dans la console et faites **Ctrl+C** (ou appuyez sur le carré rouge en haut à droite).

Exécuter la sélection : dans l'éditeur de script, vous pouvez exécuter uniquement les lignes sélectionnées grâce à **F9**)

Diviser son code en cellules : dans l'éditeur de script, vous pouvez créer une nouvelle *cellule* en écrivant *#%%*. Celà vous permet de structurer votre code de façon plus lisible mais aussi d'exécuter chaque cellule indépendamment (F5 pour exécuter tout le script, Maj+Entrée pour exécuter la cellule courante)

Interface Spyder L'interface de Spyder (personnalisable) se divise en plusieurs fenêtres : l'éditeur de script, l'arborescence des fichiers, la console, la liste des variables définies, l'historique des commandes. Vous pouvez taper des instructions simples dans la console pour les exécuter directement. En revanche, dès que vous souhaitez écrire (et sauvegarder) un programme un minimum complexe, il est indispensable de le faire dans l'éditeur de script.

Exercice 1 *Quelques commandes de base*

- *Dans l'arborescence : placez vous dans un répertoire adapté (par exemple, créez un répertoire EDPetDF/TP0)*
- *Enregistrez votre ficher (format .py).*
- *importer le module Numpy avec l'alias np :* `import numpy as np`
- *Tester les commandes suivantes et comprendre leur effet (lorsque ce n'est pas indiqué). Pour observer le résultat, taper le nom de la variable dans la console ou* `print(nom_variable)` *dans l'éditeur de script.*

Manipulation de tableaux 1D

```
x = np.arange(-2,2.0,0.5)   # De -2 (inclus) à 2 (exclus) par pas de 0.1
y = np.linspace(-2,2,10)   # De -2 (inclus) à 2 (inclus) en 10 points
np.size(x) # Sa taille
x[0] # Le premier élément
x[1] # Le deuxième élément
x[-1]  # Le dernier élément, équivalent à x[7] ici
x[20] # indice supérieur à la taille du tableau
z=np.abs(x) #valeur absolue
np.sqrt(z)#racine carée
z**(1/7) # racine 7-ème

a = np.array ([[1,2,3], [4,5,6], [7,8,9]])
b = 2 * a  # Multiplication de chaque terme
c = a + b  # Sommation terme à terme
d = a**2
np.dot(a, b) # Produit de matrices
a * b      # Produit terme à terme
a.T # Transposée
np.linalg.det(a)
# Attention ! La méthode de calcul du déterminant
souffre de petites erreurs d'arrondis. Normalement det(a)=0
np.linalg.inv(a) # inverse (en théorie, a est non inversible)
a.shape
a.size
```

Accès aux éléments / coupes d'un tableau 2D

```
a = np.array ([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
a[1,2]
a[1,2]=8.45 ; print(a) # a a été créé avec uniquement des entiers,
on ne peut pas y insérer un float.
type(A[1,2])
# Pour utiliser des float, 3 options :
a1=a.astype(float)
a1[1,2] = 8.45
a2 = np.array ([[1.,2,3,4], [5,6,7,8], [9,10,11,12]]) # il suffit
d'un seul float (ici a2[0,0]) pour définir  a2  comme tableau de floats.
a3 = np.array ([[1,2,3,4], [5,6,7,8], [9,10,11,12]], dtype=float)

a[:,-1]
a[1,:]
a[:2,1]
a[2:,1]
a[:2,1]=[100,200]
a[0,0]=a[0,0]*8
m=a[:,2]
```

Copie d'un tableau. *Attention ! Pour un tableau (comme pour les listes) on ne copie que l'adresse du tableau (pointeur) pas son contenu (les deux noms correspondent alors aux mêmes adresses en mémoire).*

```
A = np.array ([[1,2,3], [4,5,6], [7,8,9]], dtype=float)
B = A
id(A)==id(B) # A et B désignent le même objet
A[0,0]=15
print(A,B)
C = A[0,:]
A[0,0]=150
print(A,C) # contrairement aux listes, cette manipulation ne crée
pas deux objets indépendantes.


# Deux manières de créer une copie (indépendante) d'un tableau :
D= 1*A
A[0,0]=4.5
print(A,D)

E = A.copy()
A[0,0]=6.7
print(A,D)
```

Résolution d'un système linéaire.

```
A = np.array ([[1,2,3], [4,5,6], [7,8,6]], dtype=float)
b = np.array([2,3,1])
x = np.linalg.solve(A,b)
print(np.dot(A,x))
```

Utilisation de fonctions sur les tableaux

```
def f(x):
        return x**3 , x*2
n = 4
x = np.linspace(0, 1, n)
y,z = f(x)
```

Remarque importante : *Lorsque c'est possible, il est beaucoup plus avantageux d'utiliser des vecteurs plutôt que des boucles ! Ce sera très important pour résoudre efficacement des équations différentielles.*

```
import time
n = 4000
x = np.linspace(0, 1, n)
tic = time.time() # nombres de secondes écoulées depuis
                                    # le 1er janvier 1970 (Heure Unix)
y = f(x)
elapsed = time.time() - tic
print(elapsed)

tic = time.time()
y2 = np.zeros(n) # Génère un vecteur remplis de 0
for i in range(n):
        y2[i] = f(x[i])
elapsed2 = time.time() - tic
print(elapsed2)
```

Matrices particulières

```
A1 = np.zeros([3,4])
A2 = np.ones([3,4])
A3 = np.eye(5)
d=[1,2,3,4]
A4 = np.diag(d)
A5 = np.eye(5,k=2)
A6 = np.eye(5,k=-2)
A7 = np.diag (d,k =2)
```

Boucles et tests booléens

```
t = np.arange(6)**2
t == 16
t>=9
t[t>=16]
```

Exercice 2 *Créer une fonction qui Suite prend comme paramètre n et m et qui renvoie un tableau de forme $n \times m$ dont la i-ème ligne (pour $i = 0, 1, ...n-1$) contient les m premiers termes de la suite arithmétique de terme initial 1 et de raison i. Cette fonction ne doit faire intervenir qu'une seule boucle* `for`.

Exercice 3 *Créer une fonction Norme qui prend comme paramètre x (un tableau NumPy) et p ($\in \mathbb{N}^*$) et qui renvoie la norme p de t*

$$||x||_p = \sqrt[p]{\sum |x_i|^p}.$$

Exercice 4 *Un exemple d'EDO linéaire* **scalaire**

On considère l'équation différentielle ordinaire donnée par le problème de Cauchy suivant

$$\begin{cases} y'(t) = -ky(t) + r, \\ y(t_0) = y_0. \end{cases} \tag{1}$$

sur l'intervalle de temps $[t_0, t_f]$, avec $t_0 \in \mathbb{R}$ et $t_f > t_0$, $k > 0$, $r \in \mathbb{R}$, $y_0 \in \mathbb{R}$.

1. *Programmation de la méthode d'Euler explicite.*

(a) *(i) Ecrire (sur feuille) le schéma d'Euler explicite pour cette équation. Exprimer y_{n+1} en fonction de y_n, puis en fonction de y_0 (reconnaitre une suite arithmético-géométrique).*

(ii) Coder une fonction qui à k, r, t_0, t_f, y_0 et N, renvoie la suite $(y_n)_{n\in[0,N]}$ des valeurs approchées de la solution y de (1) aux temps $(t_n)_{n\in[0,N]}$ par la méthode d'Euler explicite.

(b) Validation.

On considère le problème de Cauchy suivant :

$$\begin{cases} y'(t) = -10y(t), \\ y(0) = 1, \end{cases}$$

avec $t_0 = 0$ *et* $t_f = 5$*, et donc* $k = 10$*,* $r = 0$*,* $y_0 = 1$*.*

Pour commencer, on choisira une discrétisation en $N = 1000$ *points.*

(i) Tracer la courbe de la solution exacte en fonction du temps et, sur le même graphe, la solution approchée. Pour cela, importer le package matplotlib.pyplot et utiliser la commande plot de ce package.

```
import matplotlib.pyplot as plt # package pour faire des graphes
plt.plot(tableau_abscisses,tableau_ordonnées, (options))
```

(ii) Tracer la courbe représentative de la valeur absolue de l'erreur $|y(t_n) - y_n|$ *entre la solution exacte et la solution approchée au cours du temps.*

(iii) Recommencer avec $N = 100$*, puis* $N = 10$*. Commenter.*