TD n° 1

Introduction aux outils numériques pour la Physique

Ce TD a pour but de mettre en place les outils qu'on utilisera au cours de l'année pour étudier toutes sortes de systèmes physiques. Le langage utilisé sera Python.

L'environnement de programmation

Pour écrire et exécuter un programme Python, il faut un environnement adapté. Cette année, nous en utiliserons Pyzo ou Spyder, ou Capytale.

Pyzo ou Spyder sont des IDE (Integrated Development Environment), une application qui permet d'entrer un programme Python et de l'exécuter.

Dans la fenêtre qui s'ouvre, on voit une partie destinée à écrire le programme, et une autre où se trouve lancé l'interpréteur Python, qui attend des commandes.

Remarque : On va souvent tracer des courbes avec la bibliothèque Matplotlib. Avec Pyzo, les courbes s'ouvrent dans une nouvelle fenêtre ; avec Spyder, par défaut, ce n'est pas le cas, et alors on ne dispose pas de commandes permettant de zoomer. Pour que les courbes s'ouvrent dans une nouvelle fenêtre, il faut aller dans `Outils -> Préférences -> Console IPython -> Graphiques -> Backend` et mettre à `Automatique`.

Capytale est une interface qui permet de travailler sur des Notebook («carnets» d'extension `.ipynb`) Python ; ce sont des fichiers qui contiennent des morceaux de programmes Python, mais aussi des commentaires, des images,… Les Notebooks sont des fichiers de plus en plus utilisés pour les échanges entre scientifiques. Le logiciel Jupyter permet d'en éditer localement, mais Capytale est un outil mettant à disposition une interface très pratique.

Pour accéder à Capytale, on va sur le site capytale2.ac-paris.fr, on se connecte par l'ENT Rhône-Alpes avec les identifiants Educonnect. Ensuite, soit on crée une activité de type Notebook, soit on entre un code de partage donné par le professeur. Dans le notebook, chaque cellule peut être de type Code (par défaut) ou Markdown (commentaires). Pour exécuter une cellule et passer à la suivante, on utilise Maj+Entrée.

La bibliothèque Numpy

Python possède des listes qui permettent de stocker plusieurs valeurs : `liste=[1,2,3,5.2,"quatre"]` ; elles sont souples, permettant de mélanger toutes sortes d'objets dans la même liste, et d'ajouter/enlever des éléments. Mais elles ne sont pas performantes pour les calculs.

On va alors utiliser la bibliothèque Numpy qui implémente des tableaux de longueur fixe, contenant un seul type de valeurs numériques (entiers, réels ou complexes), et permettant d'effectuer dessus des calculs vectoriels très performants. Les exemples suivant illustrent le fonctionnement des tableaux Numpy, entrez les lignes dans l'interpréteur Pyzo ou dans une cellule de Notebook et exécutez-les.

Avant toutes choses, on importe la bibliothèque, et souvent, on raccourcit son nom pour gagner du temps : `import numpy as np`.

Création d'un tableau

Pour créer un tableau, on a essentiellement deux solutions :

— soit on le crée à partir d'une liste (tableau 1D) ou d'une liste de listes (tableau 2D) :

```
1 a=np.array([1,2,4,6,3])
2 print(a)
3 b=np.array([[1,2,3],[6,5,4]])
4 print(b)
```

— soit on crée un tableau de zéros en donnant ses dimensions :

```
c=np.zeros(20)
print(c)
d=np.zeros( (2,3) ) # 2 lignes, 3 colonnes
print(d)
```

— enfin, il existe une fonction très pratique, `np.linspace`, qui crée un tableau 1D de valeurs régulièrement espacées entre deux bornes :

```
e=np.linspace(1,5,20) # 20 points entre 1 et 5
print(e)
```

Accès à une case

Les cases d'un tableau sont repérées par un index, la première case étant repérée par l'index 0. Pour un tableau 2D, on doit donner l'index de la ligne, puis de la colonne :

```
print(a[2])
print(b[1,0])
```

On peut s'en servir pour modifier une valeur :

```
d[1,1]=3
print(d)
```

Si on a besoin de connaître les dimensions d'un tableau, il y a 2 possibilités :

— pour un tableau 1D, avec `len(tableau)`

— pour tous les tableaux, avec `tableau.shape` qui renvoie une liste (un tuple en fait…) des dimensions

```
print(len(a))
print(b.shape)
```

Opérations vectorielles

Avec Numpy, toutes les opérations ou presque sont vectorielles, c'est-à-dire qu'elles s'appliquent terme à terme :

— ajout/soustraction d'un nombre, multiplication/division par un nombre

```
print(a)
print(a+1)
print(b)
print(2*b)
```

— addition/soustraction/multiplication/division terme à terme de deux tableaux **de mêmes dimensions** :

```
print(b)
print(d)
print(b+d)
```

sinon on a un message d'erreur

```
print(a)
print(c)
print(a+c)
```

— application d'une fonction mathématique à tous les termes du tableau grâce aux fonctions de la bibliothèque Numpy :

```
print(np.cos(a))
print(np.exp(b))
```

La bibliothèque Matplotlib

Matplotlib est une bibliothèque de tracé de courbes à partir de tableaux ou de listes. On commence par l'importer avec `import matplotlib.pyplot as plt` pour faire court.

Tracé de courbes simples

Matplotlib ne trace pas des courbes de fonctions, mais des points dont on lui donne les coordonnées sous forme de deux tableaux. La syntaxe de base est constituée de 3 instructions : `plt.figure()` qui crée une nouvelle figure (facultatif si on ne veut faire qu'une seule figure), `plt.plot(x,y)` qui trace la courbe mais ne l'affiche pas, et `plt.show()` qui affiche le résultat (facultatif sous Jupyterlab, mais conseillé quand même) :

```
a=np.array([1,2,4,5])
b=np.array([2,3,2,4])
plt.figure()
plt.plot(a,b)
plt.show()
```

Pour tracer la courbe représentant une fonction, on procèdera toujours en créant d'abord le tableau des abscisses avec la commande `np.linspace` puis en calculant les ordonnées grâce au calcul vectoriel :

```
tab_x=np.linspace(0,2*np.pi,20) # 20 points entre 0 et 2 pi
tab_y=np.cos(tab_x) # fonction cosinus
plt.figure()
plt.plot(tab_x,tab_y)
plt.show()
```

Ajout de labels, modification du style

Testez, sur l'exemple précédent, les possibilités suivantes :

— entre le `plot` et le `show` on peut ajouter des instructions qui ajoutent des détails ; en voici quelques-uns :
  * `plt.xlabel("texte")` ou `plt.ylabel("texte")` qui ajoutent des labels sur les axes
  * `plt.title("texte")` qui ajoute un titre
  * `plt.xlim( [xmin,xmax] )` ou `plt.ylim( [ymin,ymax] )` qui modifient l'échelle

— dans la commande `plot`, après les tableaux d'abscisse et d'ordonnée, on peut ajouter des options qui modifient l'aspect du tracé de la courbe ; en voici quelques-uns (cf. l'aide sur internet pour voir toutes les valeurs possibles de ces options) :
  * `color="red"` trace la courbe en rouge
  * `marker="+"` ajoute des croix aux points de données (ou bien `"x"`, `"*"`,…)
  * `linestyle="--"` trace la courbe en tiretés (ou bien `":"` en pointillés ; si on met une chaîne vide `""`, la courbe disparaît)
  * `linewidth=3` trace une courbe de 3 pixels de large
  * `markersize=20` trace de gros marqueurs de 20 pixels

Tracé de plusieurs courbes

Pour tracer plusieurs courbes, il suffit de mettre plusieurs `plot` avant le `show`. Pour les distinguer, on peut ajouter l'option `label="texte"` à chaque courbe, et ajouter une légende avec `plt.legend()` qui peut prendre un paramètre `loc` pour changer sa position :

```
tab_x=np.linspace(0,2*np.pi,100) # 100 points entre 0 et 2 pi
plt.figure()
plt.plot(tab_x,np.cos(tab_x),color="red",label="cosinus")
plt.plot(tab_x,np.sin(tab_x),color="green",label="sinus")
plt.legend(loc=3)
plt.show()
```

Ajout de texte ou de traits

On peut ajouter du texte sur le tracé avec `plt.text(x,y,texte)`.
On peut tracer une ligne horizontale avec `plt.axhline(y)` ou verticale avec `plt.axvline(x)`.

Exercices simples

1. Tracez la courbe de la fonction $x \mapsto x^2 + 1$ entre -2 et 2.
2. Tracez les courbes de $x \mapsto e^x$, $x \mapsto e^{x+0,5}$ et $x \mapsto e^{x+1}$ entre 0 et 2 en ajoutant une légende.
3. Le calcul vectoriel (sur un tableau) ne fonctionne pas s'il y a des branchements conditionnels, par exemple. Dans ce cas, le tableau des ordonnée doit être calculé avec une boucle. On commence par créer un tableau de N zéros avec la commande `tab_y=np.zeros(N)` puis on rempli les cases `tab_y[i]` dans une boucle `for i in range(len(tab_x))`.

   Tracez par exemple la courbe de la fonction $x \mapsto \begin{cases} 0 \text{ si } x < 0 \\ x \text{ si } 0 \leq x \leq 1 \\ 1 \text{ si } x > 1 \end{cases}$ entre -2 et 2.