TP 1 - Représentation de données en Python

Dans l'optique des concours, la maîtrise du langage Python est un attendu du programme de CPGE. Nous allons donc l'utiliser couramment au cours de l'année, et certaines des méthodes que nous verrons sont à connaitre (il y a aussi des fonctions que vous n'avez pas à connaitre et dont les spécifications vous seront fournies, mais il ne faut pas espérer s'en sortir sans les avoir travaillées au préalable.

Capacités numériques à acquérir dans ce TP :
- Représentation graphique d'un nuage de points : Utiliser les fonctions de base de la bibliothèque *matplotlib* pour représenter un nuage de points.
- Représentation graphique d'une fonction : Utiliser les fonctions de base de la bibliothèque *matplotlib* pour tracer la courbe représentative d'une fonction.

I Les bibliothèques

Python est un langage de programmation qui n'était pas initialement conçu pour faire du calcul numérique. des scientifiques se sont vite aperçus du potentiel qu'il offrait, et afin de faciliter son usage, des *bibliothèques* (*libraries* en anglais) ont été créées : il s'agit d'un paquet de fonctions, cachées à l'utilisateur, qui permettent de simplifier l'écriture.

Parmi les bibliothèques que vous allez être devoir savoir utiliser se trouvent *numpy* (*NUMerical PYthon expansiions*) pour le calcul numérique, et *matplotlib* (*MATlab PLOT LIBrary*) pour la visualisation de données, en particulier son module *pyplot*.

La liste d'instructions que vous allez écrire doit donc commencer par l'importation des bibliothèques, et vu que qu'il faudra spécifier dans quelle bibliothèque chercher une fonction déjà créée, il est d'usage de renommer les bibliothèques avec un nom plus court.

Ainsi, il y a de fortes chances que les programmes que vous écrirez commencent par :

```
1 import numpy as np
2 import matplotlib.pyplot as pypl
```

II Calcul numérique

L'importation de *numpy* permet alors d'utiliser différentes fonctions mathématiques dont voici un extrait :

Numpy	np.exp(x)	np.sin(x)	np.cos(x)	np.log(x)	np.log10(x)	np.pi	np.sqrt(x)	x**y
Maths	e^x	$\sin x$	$\cos x$	$\ln(x)$	$\log(x)$	π	$\sqrt{x}$	x^y

Pour obtenir le résultat d'un calcul, on peut alors faire appel à la fonction `print` dans l'écriture du script, comme :

```
1 import numpy as np
2 print(np.cos(np.pi/3))
```

puis exécuter le programme.

On peut aussi directement taper dans la console (après avoir chargé les bibliothèques) la commande `np.cos(np.pi/3)` avant de taper entrée.

Question 1.

Utiliser la console pour déterminer le résultat de cos(0,45) et de $\sqrt{\ln(3)}$.

Question 2.

Ecrire un programme affichant le résultat de $4^{\tan(3)}$.

III Affichage de courbes

III.1 La fonction plot

La fonction principale du module *pyplot* de *matplotlib* s'appelle tout simplement `plot` (et s'appelle donc en fonction de l'alias du module, ici comme `pypl.plot`).

Cette fonction doit prendre au moins 2 arguments obligatoires, qui sont des listes (ou des *arrays* numpy) **de même taille** N, ainsi que des arguments optionnels. La première liste correspond aux valeurs des abscisses et la seconde à celles des ordonnées. La fonction `plot` place alors chacun des N points et les relie les uns aux autres successivement (leur représentation et le style de courbe dépendent des options choisies). Ceci crée un objet manipulable par python (on peut le sauvegarder, le modifier en ajoutant des titres , légendes, axes, etc), que l'on peut visualiser avec la fonction `show` qui ne prend pas d'argument.

Un exemple minimal de graphe serait le suivant :

```
1 import matplotlib.pyplot as pypl
2 x = [1,2,5,4]
3 y = [0,4,1,3]
4 pypl.plot(x,y)
5 pypl.show()
```

Les options permettant de spécifier prennent la place du troisième argument, par exemple on peut utiliser `pypl.plot(x,y,"o")` qui tracera alors un nuage de points ronds, non reliés entre eux. On peut ainsi :

— spécifier la forme des points : `"o"` (pour des ronds), `"+"` (pour des croix), `"*"` (pour des étoiles) ;
— spécifier la manière de relier les points entre eux : `"-"` pour une ligne continue, `":"` pour une ligne pointillée (*dotted*), `"--"` pour une ligne tiretée (*dashed*), `"-."` pour une ligne alternant tirets et points ;
— spécifier la couleur : `"r"` rouge, `"g"` vert, `"b"` bleu ;

et combiner ces spécifications, par exemple `pypl.plot(x,y,"r:*")` reliera en rouge les points placés sous forme d'étoiles avec des pointillés.

On peut aussi associer une étiquette à cette courbe avec l'option label selon `pypl.plot(x,y,label = "...")`

III.2 Les options

Une fois l'objet graphe créé, il est possible de l'habiller avec les fonctions suivantes, qui seront à placer entre la fonction `plot` et la fonction `show` :

— `pypl.title(str)` : place la chaîne de caractère `str` en titre (les chaînes de caractères doivent être entre guillemets, comme par exemple "Titre") ;
— `pypl.xlabel(str)` : nom de la grandeur en abscisse ;
— `pypl.ylabel(str)` : nom de la grandeur en ordonnée ;
— `pypl.grid()` : pour placer une grille de quadrillage dans le graphe ;
— `pypl.xlim(x1,x2)` : pour fixer les valeurs limites x1 et x2 des abscisses du graphe ;
— `pypl.ylim(y1,y2)` : pour fixer les valeurs limites y1 et y2 des ordonnées du graphe ;
— `pypl.legend()` : pour légender le graphe

On peut enfin superposer plusieurs courbes sur le même graphe en appelant plusieurs fois la fonction `plot` avant les autres. Python s'occupe alors tout seul de changer les couleurs des courbes afin qu'elle ne soit pas confondues (à part si la couleur de chaque courbe a été forcée dans les options). Pour faciliter la lecture, l'option label de chaque courbe est primordiale puisqu'il suffit alors d'utiliser la fonction `legend()` pour faire apparaitre dans un coin du graphe l'étiquette de chaque courbe. ‘

IV Au travail !

IV.1 Tracés de graphes

On souhaite tracer la fonction sinus entre $-\pi/2$ et $3\pi/2$. Plutôt que de créer des listes à la main de valeurs pour les points à placer on utilise la fonction `linspace(x,y,n)` de numpy qui crée un *array* de n valeurs équiréparties entre x et y. Il est alors possible d'utiliser directement les fonctions numpy sur chacune des valeurs de l'*array* en une seule ligne.

On peut donc écrire le programme suivant :

```
1 import numpy as np
2 import matplotlib.pyplot as pypl
```

```
3 xmin = - np.pi/2
4 xmax = 3*np.pi/2
5 N = 100
6 x = np.linspace(xmin,xmax,N)
7 y = np.sin(x)
8 pypl.plot(x,y)
9 pypl.show()
```

Question 3.

Recopier le programme précédent et modifier les différents paramètres afin de voir leur influence. En particulier, commenter l'effet du choix de la valeur de N.

Question 4.

Adaptez le programme précédent afin de visualiser les fonction $f(x) = e^{-x}$ et $g(x) = 2e^{-x/2}$ pour x allant de -1 à 5. Le graphe devra faire aussi apparaitre une grille de quadrillage, chaque courbe devra être étiquetée $f(x)$ et $g(x)$ respectivement, et l'axe des abscisses devra être légendé. Le titre du graphe sera "Un joli graphe". Une fois la courbe obtenue, recopiez le script utilisé sur le compte-rendu.

Exploitation de données

On souhaite maintenant utiliser Python pour visualiser un jeu de données et déterminer quel modèle expérimental est le plus adapté à l'étude de la réfraction d'un rayon lumineux traversant un dioptre. On mesure donc les angles d'incidence i et de réfraction r d'un rayon lumineux et on trouve les couples de valeurs suivantes :

i (°)	0	10	20	30	40	50	60	70	80
r (°)	0	7,5	15	22	29	35	41	45	48

On suppose deux modèles différents, et effectue alors une régression linéaire sur ces données pour chaque modèle.

Le premier modèle est un modèle pour lequel r est proportionnel à i. Dans ce modèle, la droite passant le plus près de tous les points a pour équation $r = a_1 i + b_1$ avec $a_1 \simeq 0,613$ et $b_1 \simeq 2,347$.

Le second modèle est un modèle dans lequel $\sin r$ est proportionnel à $\sin i$. Dans ce modèle, la droite passant le plus près de tous les points a pour équation $\sin r = a_2 \sin i + b_2$ avec $a_1 \simeq 0,754$ et $b_1 \simeq -3,89 * 10^{-4}$.

Pour ne pas perdre de temps sur le traitement des données, vous pouvez copier les lignes suivantes afin de bien avoir les deux jeux de données sous forme d'*arrays numpy* :

```
1 import numpy as np
2 import matplotlib.pyplot as pypl
3 i = np.linspace(0,80,9)
4 r = np.array([0,7.5,15,22,29,35,41,45,48])
```

Question 5.

Afficher sur un graphe les points de données (sous forme de points non liés entre eux) ainsi que la modélisation donnée par la régression linéaire (sous forme de ligne continue) sur le premier jeu de données. La régression ainsi obtenue vous parait-elle satisfaisante ?

Question 6.

Faire de même avec le deuxième modèle.