Vade-mecum Python
Tracé de graphes

Que l'on acquière des données via un fichier `.csv` ou qu'on les entre soi-même dans un programme, il est souvent utile de représenter ces données sur un graphe.

On peut vouloir aussi tracer une courbe théorique à partir d'une équation.

Les principes vus en 1. et 2. concernent ces deux aspects.

En 3. on rajoute l'utilisation de `numpy` qui permet très rapidement de tracer des courbes représentatives de fonctions. La différence est que `numpy` utilise un autre objet de Python : les tableaux, et non des listes. Une liste peut contenir tout type d'objets alors qu'un tableau ne contient que des nombres.

Numpy permet de réaliser des opérations qui étaient impossibles avec les listes et cela peut s'avérer très utile.

1 La base

Pour tracer des graphes, on a besoin du module `pyplot` de la bibliothèque `matplotlib`.

Pour repérer les points sur le graphe, on va construire 2 listes, une liste X contenant les abscisses, et une liste Y contenant les ordonnées.

On souhaite tracer la courbe représentative de la fonction $f : x \longmapsto f(x) = x^2 + 1$ dans l'intervalle $[-10, 10]$

code Python

```
import matplotlib.pyplot as plt
X, Y = [ ], [ ] # on intialise les 2 listes
for x in range(-10,11) : # x prend les valeurs -10, -9,.. 0,1,..., 10
    X.append(x) # on ajoute chaque valeur de x à la liste X
    y = x**2 + 1 # on construit l'image de x
    Y.append(y) # on ajoute chaque valeur de y à la liste Y
# à présent X et Y sont prêtes ; on peut le vérifier avec print(X) et print(Y) par exemple
plt.plot(X,Y) # on construit la courbe
plt.show() # on l'affiche
```

Tapez ce code, exécutez-le. La courbe apparaît mais il manque des choses : un titre, une légende, l'intitulé des axes ...

De plus, le pas de 1 des abscisses pour le calcul des images est beaucoup trop grand. Tâchez de modifier le programme pour obtenir une représentation plus fidèle à l'allure théorique.

Si vous l'avez trouvé, tant mieux. Nous utiliserons plus loin une méthode bien plus simple pour arriver à ce résultat.

2 Améliorations

On donne ici quelques commandes pour compléter notre graphe ; on se place en dessous de la ligne `# à présent X et Y sont prêtes...`. Observer ce qu'entraînent ces modifications pour comprendre leur utilité.

code Python

```
plt.plot(X, Y, 'r+- -', linewidth = 0.5, label = "courbe")
# linewidth pour épaisseur du trait ; voir plus bas pour la partie 'r+- -'
# le label est la référence de la construction de la légende (voir plus bas)
plt.xlabel("abscisse") ; plt.ylabel("ordonnée")
plt.title("une belle courbe") ; plt.grid() # affiche une grille aux graduations principales
plt.xlim(min(X), max(X)) ; plt.ylim(min(Y), max(Y)) # valeurs limites des axes
plt.legend() # on construit la légende de la courbe à partir de son label
plt.savefig("courbe.png") # enregistre la courbe en format png (ou pdf ou eps ...)
plt.show()
```

Avec `r+- -`, on a défini ensemble la couleur, le type de marqueurs et le type de ligne (dans cet ordre).

couleur	marqueur	type de ligne
`r` pour red	`+`	`-` pour ligne pleine
`b` pour blue	`.`	`- -` pour pointillés
`g` pour green	`o`	`-.`
etc	`x`	etc

Comme marqueurs, il y a aussi `s` pour square (carré), `^` pour afficher des triangles ...

On aurait pu écrire à la place : `color = "red", marker = "+", line = "-"`

REMARQUES :

— Il est possible de personnaliser beaucoup plus le graphe ; toutes les parenthèses pour les commandes (légendes, label des axes, grille...) peuvent intégrer des options ; voir la documentation de matplotlib pour aller plus loin : `www.matplotlib.org`

— Python interprète les commandes les unes après les autres ; ainsi, si vous placez `plt.legend()` après `plt.show()`, la légende n'apparaîtra pas sur la visualisation.

— Il est bien sûr possible de construire plusieurs courbes sur le même graphique ; pour cela il faudra intégrer plusieurs `plt.plot(...)` en détaillant les listes utilisées et les options voulues. Il faudra aussi tenir compte de la remarque précédente ; si vous voulez voir afficher la légende d'une deuxième courbe, la commande `plt.legend()` doit être placée après la commande de constructions des courbes (les `plt.plot()`).

3 Tracé de courbes en utilisant `numpy`

Pour la suite, on a importé numpy sous l'alias `np`.
Numpy propose entre autres les fonctions :

— Pour la création d'un nouveau tableau vide : `T = np.array([ ])`

— `np.arange(début_inclus, fin_exclue, pas)` : crée un tableau ; similaire à range pour les listes, sauf qu'on peut mettre un pas décimal ; gros avantage par rapport aux listes

— `np.linspace(début_inclus, fin_incluse, nombre_de_points)` : pratique aussi

— On peut rajouter une valeur à la fin d'un tableau T avec la commande : `np.append(T, nbe_à_ajouter)`

— on peut transformer une liste L en tableau T : `T = np.array(L)` ; cela permet d'effectuer des opérations sur T qui n'était pas permise sur L. (Les signes `+, -, *, / , **` n'ont pas le même sens entre listes et tableaux).

On peut faire toutes les opérations sur les tableaux ; `2.5*T` multiplie tous les nombres du tableau T par 2,5. Cela est impossible avec les listes (`2*L` concatène L à elle-même, et donc `2.5*L` ne veut rien dire).

`Numpy` est capable de calculer l'image d'un tableau par une fonction. Ainsi, `np.sin(T)` construit un tableau de valeurs images des valeurs du tableau T par la fonction sinus.

On comprend dès lors l'intérêt de `numpy` pour tracer des graphes de courbes théoriques.

Le code ci-dessous trace la courbe représentative de la fonction $f : x \longmapsto f(x) = 5 \times \dfrac{\sin^2(\pi \times x)}{(\pi \cdot x)^2}$

code Python

```
import matplotlib.pyplot as plt
import numpy as np
from math import pi
X = np.linspace(-5,5,200)
Y = 5*np.sin(pi*X)**2/(pi*X)**2
plt.plot(X, Y)
plt.show()
```

Il serait encore ici tout à fait possible de personnaliser le graphe, de l'enregistrer...

4 Plusieurs sous-graphes

Je le signale par acquis de conscience, on peut créer des sous-graphes :

Il faut définir la dimension, par exemple 2 graphes sur 2, puis les graphes sont numérotés de la droite vers la gauche et du haut vers le bas :

221 : il y a 4 graphes (2 par 2) et celui-là se situe en haut à gauche.

code Python

```
import matplotlib.pyplot as plt
import numpy as np
from math import pi
X = np.linspace(-5,5,200)
Y = 5*np.sin(pi*X)**2/(pi*X)**2; Z = 5*X; T = np.exp(-X)
A = plt.subplot(221)
A.plot(X,Y)
B = plt.subplot(222)
B.plot(X,Z)
C = plt.subplot(223)
C.plot(X,T)
D = plt.subplot(224, facecolor='yellow')
D.plot(X,Y)
plt.show()
```