TP Python 6 : Représentation graphique de fonctions

Correction

Préambule

Dans ce TP, nous allons nous intéresser à la **représentation graphique de fonctions**.
La bibliothèque `matplotlib.pyplot` contient une multitudes de fonctions déjà prêtes à l'emploi pour représenter des fonctions en Python.
Ainsi, si l'on souhaite représenter le graphe d'une fonction, il nous faudra donc l'importer, en plus de la librairie `numpy` . Tout comme on abrège `numpy` par `np` pour aller plus vite, il est d'usage d'abréger `matplotlib.pyplot` par `plt` .

Entrée[2]:
```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

I. Un peu de théorie

Considérons f une fonction définie sur $\mathbb{R}$

- La courbe représentative (ou graphe) de f est un ensemble de points du plan de la forme $M(x, f(x))$, où $x \in \mathbb{R}$.
- Ainsi, nous allons devoir nous munir d'outils en Python pour décrire le graphe de f. Pour ce faire, on va utiliser la notion de **liste** rapidement abordée au TP précédent.

L'objectif est le suivant :

On va se munir d'une **liste contenant les abscisses** x, et on va ***construire*** une **liste contenant les images** $f(x)$ **correspondantes**

I. 1) La fonction `linspace`

La fonction `linspace` est une fonction prête à l'emploi de la librairie `numpy` .
Pour l'appeler, on utilisera donc la commande `np.linspace`
Cette fonction permet de **générer une liste de valeurs**. Dans notre cas, on

l'utilisera pour *générer la liste des abscisses* que l'on souhaite considérer.

Exemple :
Exécuter le programme suivant, et intuiter le fonctionnement de la fonction `linspace` .

Entrée[3]:
```
a = np.linspace(0,10,3)
print(a)
b = np.linspace(0,10,5)
print(b)
c = np.linspace(0,10,11)
print(c,'\n')
d = np.linspace(3,7,4)
print(d)
```

```
[ 0.  5. 10.]
[ 0.   2.5  5.   7.5 10. ]
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]

[3.         4.33333333 5.66666667 7.        ]
```

La fonction `linspace` prends **trois variables en entrée**, via la commande `np.linspace(start,stop,num)` , où :

- La variable `start` correspond au **premier terme** de la liste. Il est **inclus** dans la liste.
- La variable `stop` correspond au **dernier terme** de la liste. Il est **inclus** dans la liste.
- La variable `num` correspond au **nombre de termes contenus dans la liste**.

Exercice 1 :

1. Définir une liste x contenant 13 valeurs entre 0 et 5, puis l'afficher.
2. On décrit l'intervalle $[0, 1]$ en partant de 0 et en avançant d'un **pas de discrétisation** de 0.25 jusqu'à arriver en 1. Définir la liste y d'abscisses correspondantes, puis l'afficher.
3. On décrit l'intervalle $[2, 12]$ en partant de 2 et en avançant d'un **pas de discrétisation** de 1 jusqu'à arriver en 12. Définir la liste z d'abscisses correspondantes, puis l'afficher.

Entrée[4]:

```
x = np.linspace(0,5,13)
print(x)
y = np.linspace(0,1,5)
print(y)
z = np.linspace(2,12,11)
print(z)
```

```
[0.         0.41666667 0.83333333 1.25       1.66666667 2.08333333
 2.5        2.91666667 3.33333333 3.75       4.16666667 4.58333333
 5.        ]
[0.   0.25 0.5  0.75 1.  ]
[ 2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]
```

Vous l''aurez compris, on utilise `linspace` pour définir un ensemble d'abscisses de la manière suivante : on décrit un intervalle $[a, b]$ ($a < b$ deux réels) avec un **pas de discrétisation** fixé δ_x.

Le risque est alors de **se tromper dans le choix de la variable `num`** .
Un bon garde-fou est la commande `len` , qui renvoie la **longueur d'une liste donnée**

Exemple :
Exécuter le programme suivant.

Entrée[5]:

```
a = np.linspace(0,1,4)
print('La liste a est de longueur ',len(a),'et a = ',a,'\n')
b = np.linspace(0,1,5)
print('La liste b est de longueur ',len(b),'et a = ',b,'\n')
```

```
La liste a est de longueur  4 et a =  [0.         0.33333333 0.6666
6667 1.        ]

La liste b est de longueur  5 et a =  [0.   0.25 0.5  0.75 1.  ]
```

<u>Remarque :</u>
Plus le pas de discrétisation (la troisième variable `num` de la fonction `linspace`) est **petit**, plus la liste obtenue est **longue**. On dit que le pas est **fin**.

I. 2) Définir une liste d'images

Soit f une fonction définie sur un intervalle $[a, b] \subset \mathbb{R}$. On sait maintenant définir une liste d'abscisses contenues dans cet intervalle $[a, b]$.

Comment dès lors définir la liste des images associées?

Un exemple est cette fois-ci plus parlant !

Exercice 2 :
On va se donner une fonction affine $f : x \longmapsto 2x + 1$ définie sur $\mathbb{R}$, et on va s'intéresser à sa restriction à l'intervalle $[0, 10]$.

1. Définir X la liste contenant les valeurs obtenues en parcourant l'intervalle $[0, 10]$ avec un pas de 1.
2. Définir la fonction `func_affine` qui étant donnée un réel x, renvoie $2x + 1$.
3. À l'aide d'une boucle `for` et de la commande `len` , définir une liste Y qui contient, pour chaque valeur de la liste X, l'image correspondante par f de cette valeur.

Rappel :

- On initialise une liste vide à l'aide de la commande `liste = []` .
- Si `k` est un entier, la commande `liste[k]` renvoie la $k + 1$ème valeur de la liste (*array starts at 0!*).
- La commande `liste.append(truc)` ajoute à la liste **déjà existante** `liste` la variable `truc` .

Entrée[6]:

```
X = np.linspace(0,10,11)
def func_affine(x):
    y = 2*x+1
    return y

Y = []
for k in range(len(X)):
    Y.append(X[k])
# Comparaison

print(X)
print(Y)
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
```

I. 3) Première utilisation de `matplotlib.pyplot`

On sait désormais se munir d'une liste d'antécédents, et construire la liste des images par une fonction f correspondantes. Il ne reste plus qu'à **représenter graphiquement les points de coordonnées** $(x, f(x))$. Dans le jargon, on dit aussi ***plotter*** la fonction f...

On va en effet utiliser la **fonction** `plot` de la bibliothèque `matplotlib.pyplot` , *via* la commande `plt.plot(x,y)` , où :

- la variable `x` est la liste des antécédents,
- la variable `y` est la liste des images correspondantes.

Remarques :

- **Produire l'image** (au sens informatique du terme) du graphe de f est fait par `plt.plot`, mais **afficher cette image dans une fenêtre graphique** relève d'une autre fonction : `plt.show()` qui ne prends pas d'argument!
- On utilise le raccourci `plt` pour désigner `matplotlib.pyplot`. Sans raccourci, l'appel à la fonction `plot` sera : `matplotlib.pyplot.plot` !

Exemple :
On cherche à tracer le graphe de la fonction $f : x \longmapsto 2x + 1$ sur l'intervalle $[0, 10]$. Vu que cette fonction est déjà codée par la fonction `func_aff`, pas besoin de la recoder, il suffit de l'appeler! On pourrait argumenter qu'il en va de même pour les listes d'images et d'antécédents, mais puisque les variables `x` et `y` ne cessent d'être redéfinies au cours de ce TP, ne prenons pas de risques.

Entrée[7]:

```
# Liste des antécédents
X = np.linspace(0,10,11)

# On construit les images
Y = []
for k in range(len(X)):
    Y.append(X[k])

# On représente graphiquement la fonction affine

plt.plot(X,Y)
plt.show()
```

Exercice 3 :
Tracer la fonction $f : x \longmapsto \frac{1}{2}e^x$ sur l'intervalle $[-1, 1]$, en choisissant le pas de discrétisation.

Remarque :
En Python, si l'on demande à plotter plusieurs fonctions, la **fenêtre graphique** reste la même! Cela peut nous empêcher d'obtenir un seul graphique à la fois. Pour éviter ce problème, on utilise la commande `plt.clf()` qui permet de **réinitialiser la fenêtre graphique**.

Entrée[8]:

```
def func_exp(x):
    y = 0.5*np.exp(x)
    return y

X = np.linspace(-1,1,100)
Y = []
for k in range(len(X)):
    Y.append(func_exp(X[k]))

plt.clf()
plt.plot(X,Y)
plt.show()
```

I. 4) Pour aller plus loin avec `matplotlib.pyplot` (HP)

Tracer le graphe d''une fonction demande d''annoter les axes, et de donner un titre à notre fonction. Tout cela est faisable avec `matplotlib.pyplot` , mais n'est pas exigible au programme :

- En **troisième argument** de `plot` , on peut modifier la couleur du graphique (`r` pour rouge, `b` pour bleu, `v` pour vert...).
- En accolant certains symboles à la couleur, on peut modifier l'aspect des points composant le graphique (`+` pour des croix, `*` pour des asterisques...).
- La fonction `title` permet de donner un titre à son graphique.
- La fonction `legend` permet de donner un titre à un graphique contenant plusieurs représentations graphiques.
- La fonction `xlabel` permet de donner un titre à l'axe des abscices.
- La fonction `ylabel` permet de donner un titre à l'axe des abscices.

Exemple :
On trace la courbe représentative de la fonction $f : x \longmapsto -3x + 4$ sur $[0.5, 2]$.
On change la couleur du graphique et donne des titres aux axes et à la figure.

Entrée[15]:

```
def f1(x):
    y = -3*x+4
    return y

X = np.linspace(0.5,2,100)
Y = []
for k in range(len(X)):
    Y.append(f1(X[k]))

plt.clf()
plt.plot(X,Y,'r')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Graphe de f(x)=-3x+4')
plt.show()
```

Exercice 4 :
Tracer la courbe représentative de la fonction $f : x \longmapsto cos(x)$ sur $[-10, 10]$.
Mettre la couleur du graphique en vert, changer l'aspect des points pour obtenir des croix et donne des titres aux axes et à la figure.

Entrée[20]:

```
def f2(x):
    y = np.cos(x)
    return y

X = np.linspace(-10,10,1000)
Y = []
for k in range(len(X)):
    Y.append(f2(X[k]))

plt.clf()
plt.plot(X,Y,'g+')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Graphe de f(x)=cos(x)')
plt.show()
```

Exemple :
Que fait le programme suivant?

Entrée[27]:

```
X = np.linspace(0,1,100)
Y2 = []
Y3 = []
Y4 = []

for k in range(len(X)):
    Y2.append(X[k]**2)
    Y3.append(X[k]**3)
    Y4.append(X[k]**4)

plt.clf()
plt.plot(X,X)
plt.plot(X,Y2)
plt.plot(X,Y3)
plt.plot(X,Y4)
plt.xlabel('antécédents')
plt.ylabel('images')
plt.legend(['y=x','y=x^2','y=x^3','y=x^4'])
plt.show()
```

II. À vous de jouer

Exercice 5 :
Tracer la courbe représentative de la fonction $f : x \longmapsto sin(x)$ sur $[-2\pi, 2\pi]$.
Mettre la couleur du graphique en rouge, changer l'aspect des points pour obtenir des étoiles et donne des titres aux axes et à la figure.
On poura utiliser la commande `np.pi` pour utiliser une valeur approchée du nombre π.

Entrée[31]:

```
X = np.linspace(-2*np.pi,2*np.pi,100)
Y=[]

for k in range(len(X)):
    Y.append(np.sin(X[k]))

plt.clf()
plt.plot(X,Y,'r*')
plt.xlabel('x')
plt.ylabel('y')
plt.title('f(x)=sin(x)')
plt.show()
```

Exercice 6 :

Tracer la courbe représentative de la fonction $g : x \longmapsto \dfrac{(x-3)^2}{(x-1)(x+5)}$ sur $[2, 10]$.

Entrée[33]:

```
1
2  def f3(x):
3      y = (x-3)**2/((x-1)*(x+5))
4      return y
5
6  X = np.linspace(2,10,100)
7  Y = []
8
9  for k in range(len(X)):
10     Y.append(f3(X[k]))
11
12 plt.clf()
13 plt.plot(X,Y)
14 plt.xlabel('x')
15 plt.ylabel('y')
16 plt.title('f(x)=(x-3)^2/(x-1)(x+5)')
17 plt.show()
```

Exercice 7 :
Représenter sur un même graphe au voisinage de 0 (disons, sur $[-2, 2]$) lesfonctions suivantes :

$$\begin{cases} g_0(x) = e^x \\ g_1(x) = 1 + x \\ g_2(x) = 1 + x + \dfrac{x^2}{2} \\ g_3(x) = 1 + x + \dfrac{x^2}{2} + \dfrac{x^3}{6} \end{cases}$$

Les fonctions g_1, g_2 et g_3 sont appellées *approximations de la fonction exponentielle au voisinage de* 0. Nous aborderons ce thème au second semestre.

Entrée[50]:

```
def g1(x):
    y = 1+x
    return y
def g2(x):
    y = 1+x +(x**2/2)
    return y
def g3(x):
    y = 1+x+(x**2/2)+(x**3/6)
    return y

X = np.linspace(-2,2,100)
Y0 = []
Y1 = []
Y2 = []
Y3 = []

for k in range(len(X)):
    Y0.append(np.exp(X[k]))
    Y1.append(g1(X[k]))
    Y2.append(g2(X[k]))
    Y3.append(g3(X[k]))

plt.clf()
plt.plot(X,Y0)
plt.plot(X,Y1)
plt.plot(X,Y1)
plt.plot(X,Y3)
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['y=exp(x)','y=1+x','y=1+x+x^2/2','y=1+x+x^2/2+x^3/6'])
plt.show()
```

III. Fin du TP

- **Effectuez** une sauvegarde personnelle du document obtenu :
  - utilisez le menu `Fichier / Enregistrer sous…` .
  - Le fichier sera alors téléchargé dans votre répertoire `Téléchargements` .
  - Charge à vous de le placer dans votre répertoire.
- **Rendez** votre travail en utilisant le bouton Rendre ce travail en haut à droite.
- **Fermez** ce document :
  - Cliquez sur le bouton Retour en haut à droite.
  - Ou fermez simplement le navigateur.