Lycée La Prat's
Classe de PT

Pour le 25 décembre 2014

Devoir d'informatique numéro 2

Rendez le DL sous forme d'un (ou deux) fichier `.py`, qui commence par votre nom (sans accents) suivi de « _ » (tiret du 8, appelé underscore en anglais).

Exercice 1 (Tracé de courbes)

Tracer les courbes du TD de mathématiques sur les courbes. Voici un exemple de code :

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

Les bibliothèques nécessaires, on ne les présente plus.

```
1  """ Variables globales: valeurs max en x et y, intervalle I pour t
       """
2
3  xmin = -10
4  xmax = 10
5  ymin = -10
6  ymax = 10
7
8  tmin = 0
9  tmax = 9*np.pi
10 nb_point = 6000
11
12
13 """ Fonction tracée """
14
15
16 def f(t):
17     q = 9/4.
18     x = (q+1) * np.cos(t) - np.cos((q+1)*t)
19     y = (q+1) * np.sin(t) - np.sin((q+1)*t)
20     return (x, y)
```

On stocke en début de fichier les variables qu'on modifiera, selon les fonctions à tracer.

Ici, on trace une épicycloïde $\begin{cases} x(t) &= (q+1)\cos t - \cos((q+1)t) \\ y(t) &= (q+1)\sin t - \sin((q+1)t) \end{cases}$ pour $q = 9/4$.

Le reste du programme n'a aucune raison de changer selon la fonction à tracer.

```
""" Construction du cadre du tracé : objet "figure", style des axes"""

ma_fig = plt.figure()  #figure vide, sans axes a priori.

ax = plt.gca()  #Axes : construction des axes (cf TP an dernier)
ax.spines['right'].set_color('none')  #l'axe de droite est effacé
ax.spines['top'].set_color('none')    #idem pour l'axe du haut
ax.xaxis.set_ticks_position('bottom')  #graduations : axe du bas uniquement
ax.yaxis.set_ticks_position('left')    #idem vertical
ax.spines['bottom'].set_position(('data', 0))  #l'axe du bas est placé en 0
ax.spines['left'].set_position(('data', 0))    #idem vertical

plt.axis('equal')       #repère orthonormé
plt.xlim(xmin, xmax)    #limites de la boite x
plt.ylim(ymin, ymax)    #limites de la boite y
```

Quelques explications et rappels :

- ligne 3 : `ma_fig` est un objet auquel seront rattaché toutes les propriétés de la figure : les axes en particulier.
- ligne 5 à 11 : pour avoir la présentation usuelle des axes en maths (au lieu d'une boite, vous pouvez tester la différence en commentant ces lignes).
- ligne 13 : pour que le repère soit orthonormé, les deux axes doivent avoir des graduations « égales ».

```
""" Tracé : valeurs de t, calculs de x(t) et y(t) """

t_tab = np.linspace(tmin, tmax, nb_point)  #tableau des t_i possibles
x_tab, y_tab = f(t_tab)  #tableau des x(t_i) et y(t_i)

plt.plot(x_tab, y_tab)  #tracé

plt.show()
```

- ligne 3 : création d'une liste de points dans un tableau Numpy.
- ligne 4 : les fonctions Numpy sont vectorisée : écrire `f(t_tab)`, où `t_tab= array([t1,..., tnb_points])` est un tableau Numpy, signifie `array([f(t1),...,f(tnb_points)])`. La fonction « rentre » dans le tableau et s'applique élément par élément. Il en est de même pour les tableaux à plus d'une dimension (matrices, etc).

Récupérez le fichier sur ma page web, ne retapez pas tout ! Lisez-le, et comprenez-le.

Exercice 2 (des sapins – sujet facultatif non noté)

Vous pouvez faires des arbres (de différentes tailles), placés aléatoirement, plus ou moins compliqués, avec des boules et des guirlandes ou pas, de la couleur ou pas, des flocons de neige (récursifs), etc... Libre à vous d'imaginer ce que vous voulez, suivant ce que vous vous sentez capable de faire.
Plan de travail :

1) Faire une liste des types de dessins que vous voulez faire : une fonction pour chacun. Spécifier les paramètres des fonctions (laissez un maximum de variables).

2) Faites des dessins tout simple pour commencer : vous ferez plus complexe ensuite. Au besoin, pour faire plus complexe, vous ferez de nouvelles fonctions. Toujours décomposer un problème complexe en sous-problèmes simples. Vous pouvez le faire à plusieurs (pas plus de 2), et vous partager les tâches.

3) Gardez à l'esprit les principes généraux de la programmation : noms de variables et de fonctions explicites (mais pas trop longs), commentaires en début de fonction pour expliquer ce qu'elle fait.

4) Testez toujours vos fonction une par une.

5) Petites astuces pour aller vite : Utiliser <tab> pour compléter un nom de variable lorsque vous tapez dans spyder, utilisez les raccourcis clavier (Ctrl-C/Ctrl-X, Ctrl-V, Ctrl-1).

6) N'hésitez pas à utiliser la documentation en ligne. Par exemple via google, avec des mots clés en anglais (quitte à passer par google traduction avant) et en rajoutant au besoin `docs.python.org` (documentation officielle) ou `stackoverflow` (forum de discussion) pour tomber sur des documents fiables. Partez de l'exemple donné, et adaptez le.

Pour rechercher dans une page de documentation : Ctrl-F (ou menu Edit -> rechercher).

Pour information, on utilise la version 3.4 de python (pas la branche 2.*).

7) N'hésitez pas à me demander conseil si vous êtes bloqués.

Outils : on utilisera la bibliothèque `matplotlib.pyplot`, abrégée en `plt`. On peut s'inspirer de l'exercice précédent, avec `plt.axis('off')`.
Si vous voulez utiliser `Scikit.image` ou `Turtle`, libre à vous !

1) Pour tracer un segment entre $A(x_A, y_A)$ et $B(x_B, y_B)$, utiliser `plt.plot(x,y, color='red')` avec $x = [x_A, x_B]$ et $y = [y_A, y_B]$. C'est la même commande que pour tracer une courbe.

2) Pour un motif plein : `plt.fill(x, y, color='chartreuse')`

3)

4) Pour des animations (neige, feux d'artifice) : soit la commande `matplotlib.animation.FuncAnimation(...)` (avancé), soit placer des `plt.pause(temps)` entre chaque tracé. Exemple :

```
n = 10
r = 5  #rayon
x = [r*np.cos(2*i*np.pi/n) for i in range(n)]
y = [r*np.sin(2*i*np.pi/n) for i in range(n)]
for i in range(n):
    plt.plot([x[i-1],x[i]], [y[i-1],y[i]])
    plt.pause(0.1)
```

N'hésitez pas à mettre de l'informatique dans vos TIPE. Et à me consulter pour toute question liée à l'informatique.

Exemple :

```
def triangle_iso(base, hauteur, milieu_base):
    ... #retourne les points d'un triangle isocèle à plat.
    return (x, y) #liste des abscisses, puis liste des ordonnées

def rectangle(horizontal, vertical, milieu_base):
    x_b, y_b = milieu_base
    A = [x_b - horizontal/2., y_b]
    B = [x_b + horizontal/2., y_b]
    C = [x_b + horizontal/2., y_b + vertical]
    D = [x_b - horizontal/2., y_b + vertical]
    x, y = zip(*[A, B, C, D])  #Explication en bas.
    return (x, y)

def tronc(racine, taille):
    haut_tronc = taille/4
    larg_tronc = taille/6
    x_tab, y_tab = rectangle(larg_tronc, haut_tronc, racine)
    plt.fill(x_tab, y_tab, color='burlywood')
    return haut_tronc

def feuilles(racine, taille):
    hauteur_feuilles = taille
    x_tab, y_tab = triangle_iso(hauteur_feuilles/2,
        hauteur_feuilles, racine)
    plt.fill(x_tab, y_tab, color='darkgreen')
    return hauteur_feuilles

def sapin(racine, taille):
    racine = np.array(racine)
    haut_tronc = tronc(racine, taille)
    feuilles(racine + [0, haut_tronc], taille)
    return None

... #Comme pour l'exercice 1, initialisation de la figure

sapin((0,0), 10)
plt.show()
```

`zip(*[(x1,y1), (x2,y2), (x3,y3)])` retourne `((x1,x2,x3),(y1,y2,y3))`, ce qui est pratique pour passer de la notation habituelle (liste de points) à la notation matplotlib.