Python for dummies : problème 4
Approximation B-splines

Une courbe plane paramétrique basée sur de B-splines de degré p pour $n-p$ points de contrôle $[X_i, Y_i]$ et n noeuds T_i s'écrit sous la forme :

$$\begin{cases} x^h(t) = \sum_{i=0}^{n-p-1} B_i^p(t)\ X_i \\ y^h(t) = \sum_{i=0}^{n-p-1} B_i^p(t)\ Y_i \end{cases} \qquad T_p \leq t < T_{n-p}$$

Considérons uniquement le cas de degré $p = 3$. En outre, nous allons nous limiter à de noeuds qui sont des entiers successifs, pour éviter de bêtement rendre le problème trop compliqué :-)

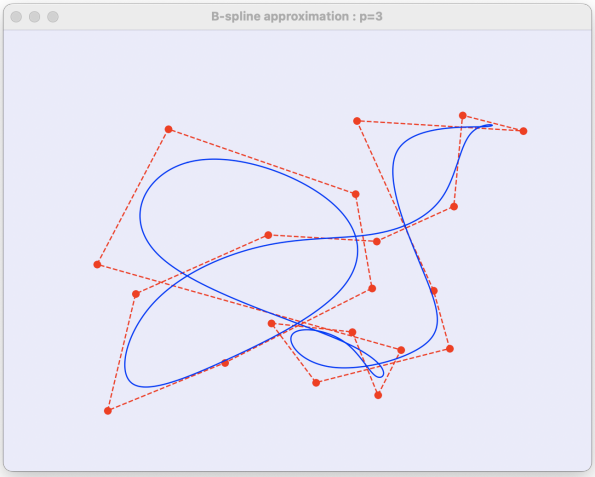
A titre d'exemple, prenons. $p = 3$ avec $n = 8$ et $n + 1 = 9$ noeuds répartis de manière uniforme.
Plus précisément, définissons $\mathbf{T} = (-3, -2, -1, 0, 1, 2, 3, 4, 5)$.
Choisissons une liste de $m = n - p = 5$ points de contrôle $[X_i, Y_i] = \{(0, 0); (1, 3); (2, 0); (3, 3); (4, 0)\}$.
Notre courbe paramétrique est alors définie pour $t \in [0, 2]$

La mission consistera à obtenir une courbe fermée périodique $(x(t), y(t))$ qui sera définie à partir de m points (X_k, Y_k) correspondant aux noeuds suivants :

$$T_k = k - 3 \qquad\qquad k = 0, \ldots, m-1$$

Pour obtenir une telle courbe périodique, il faudra juste répéter les 3 premiers points de contrôle à la fin de la liste et également ajouter trois noeuds : so easy !

La courbe paramétrique sera définie pour $t \in [0, m]$.



En reprenant notre exemple, il faudra donc modifier le vecteur de noeuds et la liste des points de contrôle de la manière suivante pour fermer la courbe paramétrique de manière harmonieuse et obtenir notre approximation périodique pour les cinq points originaux.

$$\begin{aligned} \mathbf{T} &= (-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8) \\ [X_i, Y_i] &= \{(0, 0); (1, 3); (2, 0); (3, 3); (4, 0); (0, 0); (1, 3); (2, 0)\} \end{aligned}$$

Plus précisément, votre mission consiste à :

1. Ecrire une fonction `x,y = bspline(X,Y,t)` qui calcule les valeurs `x` et `y` d'une courbe fermée bspline de degré 3 pour les abscisses `t` . Les coordonnées de points de contrôle seront donnés dans deux listes ou tableaux unidimensionnels `X` ou `Y` de même taille m. Les noeuds seront définis comme les entiers successifs $(-3, -2, \ \ldots \ , m+3)$. Le vecteur `t` sera un tableau `numpy` unidimensionnel contenant des temps $t_i \in [0, m]$. Les tailles des deux tableaux `x` et `y` seront identiques à celle du tableau `t`.
   Attention : c'est au sein de votre fonction, qu'il convient d'ajouter les coordonnées supplémentaires pour obtenir une courbe fermée :-) C'est aussi, votre job de définir le vecteur des noeuds :-)

2. Pour cette mission, il est important d'obtenir une programme rapide même si c'est au prix d'une utilisation un peu plus importante de la mémoire.

3. Pour obtenir facilement la solution, c'est sans doute une très bonne idée de vous inspirer des programmes fournis par l'enseignant à la fin du cours sur les NURBS qui sont disponibles sur le site web du cours.

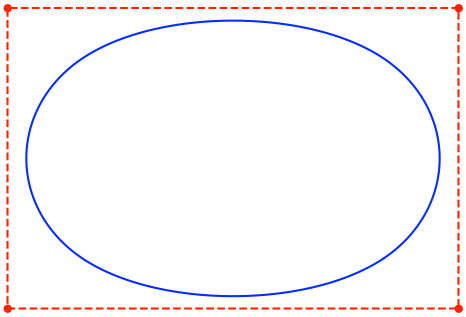
4. Si vous souhaitez réaliser un programme vraiment efficace, la meilleure idée est d'utiliser l'algorithme de l'ingénieur un peu inconnu de Citroën : oui, c'est cela la meilleure manière de faire le calcul ! Mais, une solution plus simple est aussi parfaitement acceptable.
   Le temps consacré au devoir ne doit pas être excessif !

5. Ensuite pour tester, votre programme, on vous a fourni un tout petit programme tout simple `bsplineTest.py` pour approximer un rectangle.

```
X = [0,3,3,0]
Y = [0,0,2,2]
t = linspace(0,len(X),len(X)*100 + 1)

x,y = bspline(X,Y,t)

fig = plt.figure("Approximation avec des B-spline
plt.plot(X,Y,'.r',markersize=10)
plt.plot([*X,X[0]],[*Y,Y[0]],'--r')
plt.plot(x,y,'-b')
plt.axis("equal"); plt.axis("off")
plt.show()
```



6. Finalement, nous avons aussi fourni un programme un peu plus rigolo `bsplineTestFun.py` qui permet de définir plein de points en cliquant sur la figure. Un double click permet d'obtenir la courbe fermée obtenue. Pour les petits futés et curieux, le listing de ce programme est donné à la fin de cet énoncé.

7. Votre fonction (avec les éventuelles sous-fonctions que vous auriez créées) sera soumise sur `zouLab` sans y adjoindre le programme de test fourni ! Cette fonction devra être soumise via le web : ce travail est individuel et sera évalué.

Un petit programme interactif avec `matplotlib`...

Pour les petits curieux, voici l'implémentation de `bsplineTestFun.py` pour définir de manière magique une courbe à approximer...

```
import matplotlib
from matplotlib import pyplot as plt
from numpy import *
from bsplineTest import bspline


# ====================== callback pour les événements avec la souris ======
#
#  Observer la gestion distincte du clic simple et double :-)
#  Apres un événement, on redessine la figure avec draw()
#

def mouse(event):
  global X,Y,n
  if (event.dblclick):
    if (n > 2) :
      plt.plot([*X,X[0]],[*Y,Y[0]],'--r')
      t = linspace(0,n,n*1000+1)
      x,y = bspline(X,Y,t)
      plt.plot(x,y,'-b')
      X,Y = [],[]; n = 0
  else :
    x = event.xdata
    y = event.ydata
    if (x != None and y != None) :
      n = n + 1
      X = append(X,[x])
      Y = append(Y,[y])
      print("New data : " + str(x) + "," + str(y))
      plt.plot([x],[y],'.r',markersize=10)
  fig.canvas.draw()


# ============================ mainProgram ================================

matplotlib.rcParams['toolbar'] = 'None'
matplotlib.rcParams['lines.linewidth'] = 1
plt.rcParams['figure.facecolor'] = 'lavender'

X,Y = [],[]; n = 0
fig = plt.figure("B-spline approximation : p=3")
fig.canvas.mpl_connect('button_press_event',mouse)
plt.ylim((0,1)); plt.xlim((0,1.3)); plt.axis("off")

plt.show()
```