POLYTECH LILLE

Projet de Calcul Numérique

Rémy Gaudré et Matthias Raverdy

GIS4 – 2019/2020

TABLE DES MATIÈRES

1 Premier jeu de données 3

2 En R ou en Python 4

3 Calculer une spline lissante à partir du paramètre p 5

4 Traiter un exemple plus volumineux 7

5 Partie théorique 9

6 Calculer p en fonction de S 10

7 Degré de liberté 10

8 Paramètre optimal 10

INTRODUCTION

Le but du projet est de construire et afficher une spline lissante à l'aide d'un jeu de données. Nous utilisons pour ce faire, différentes méthodes vues pendant les cours de calcul numérique.

Les splines lissantes résultent de méthodes d'interpolation et sont très souvent utilisées dans les problèmes de lissage de données expérimentales ou de statistiques.

1 PREMIER JEU DE DONNÉES

Dans un premier temps, nous utilisons le jeu de données ci-dessous :

```
x = [0., 1.7, 3.3, 5.0, 6.7, 8.4, 10.1]
y = [1.7, 3.4, 0.8, 2.5, 4.3, 4.3, 0.8]
```

Listing 1 – Données d'entrainement

Pour construire n splines, nous considérons $n+1$ points. Soit, sous python :

```
n = len(x) - 1
```

Listing 2 – Calcul de n

2 En R ou en Python

Python met à disposition des modules et des fonctions haut niveau qui permettent de calculer les splines directement avec le jeu de données grâce.
La fonction CubicSpline du package "scipy.interpolate" nous permet d'obtenir les splines cubiques. Le paramètre de lissage p est optimal, on ne le précise pas quand on construit la spline, nous avons juste à fournir les données (x et y). La fonction nous retourne une fonction polynomiale. Nous n'avons plus qu'à afficher les données.

Voici notre code haut niveau :

```
import scipy.linalg as nla
from scipy.interpolate import CubicSpline #contient des splines
import numpy as np #def matrice
import matplotlib.pyplot as plt #graphique

x = [0., 1.7, 3.3, 5.0, 6.7, 8.4, 10.1]
y = [1.7, 3.4, 0.8, 2.5, 4.3, 4.3, 0.8]
#n+1 points donc n = 6

def spline(x,y):
    cs = CubicSpline(x, y)
    xs = np.arange(min(x),max(x),(max(x)-min(x))/1000)
    fig, ax = plt.subplots(figsize=(6.5, 4))
    ax.plot(x, y, 'o', label='data')
    ax.plot(xs, cs(xs), label="Spline")
    ax.set_xlim(min(x)-1,max(x)+1)
    ax.legend(loc='lower left', ncol=2)
    plt.show()

spline(x,y)
```

Listing 3 – Programme python haut niveau

Ce qui nous donne le résultat suivant :

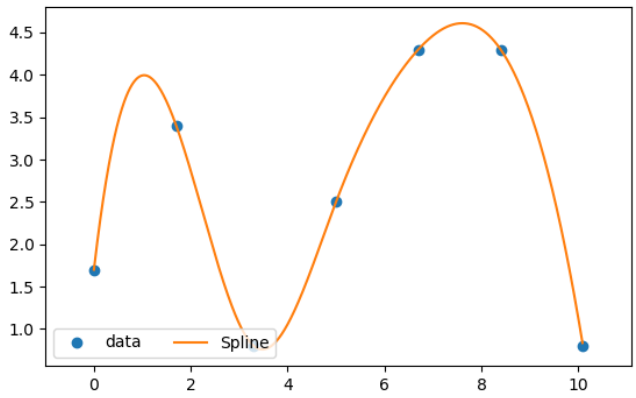

FIGURE 2.1 – La spline produite par le programme haut niveau

3 Calculer une spline lissante à partir du paramètre p

Nous avons prédéfini un paramètre p assez grand pour avoir un résultat assez proche des données ($p = 10^5$). Ainsi nous avons établi notre programme bas niveau étape par étape en référence à la partie 3.4 du cours de Calcul Numérique :

1. Nous importons les données.
2. Nous calculons n à partir des données.
3. Nous calculons $H = x_{i+1} - x_i$, $(0 \leq x_i \leq x_{i+1})$.
4. Nous calculons T de dimension $(n-1) \times (n-1)$.
5. Nous calculons $G = \frac{1}{H}$.
6. Nous calculons Q de dimension $(n+1) \times (n-1)$.
7. Nous calculons $Q^T \sum Q + pT$.
8. Nous calculons $pQ^T y$.
9. Nous calculons L tel que $LL^T = A$.
10. Nous calculons w tel que $L^T w = pQ^T y$.
11. Nous calculons c tel que $Lc = w$
12. Nous calculons a tel que $a = y - \frac{1}{p} \sum Qc$
13. Nous ajoutons les conditions de spline naturelle : $c_0 = c_n = 0$
14. Nous calculons d tel que $d_i = \frac{c_{i+1} - c_i}{3h_i}, (0 \leq i \leq n-1)$
15. Nous calculons b tel que $b_i = \frac{a_{i+1} - a_i}{h_i} - c_i h_i - d_i h_i^2, (0 \leq i \leq n-1)$
16. Nous calculons pour un intervalle donné et un pas donné, les coordonnées correspondantes
17. Nous affichons la spline lissante.

Voici le code correspondant :

```
import numpy as np #def matrice
import matplotlib.pyplot as plt #graphique

x = [0., 1.7, 3.3, 5.0, 6.7, 8.4, 10.1]
y = [1.7, 3.4, 0.8, 2.5, 4.3, 4.3, 0.8]
n = len(x) - 1

h = np.array([x[1]-x[0]])
for i in range(1,n):
    h = np.append(h, x[i+1]-x[i])

T1 = np.diag(2*(h[:n-1]+h[1:n]))
T2 = np.diag(h[1:n-1], -1)
T3 = np.diag(h[1:n-1], 1)

T = (T1+T2+T3)/3
g = 1/h

Qpremiereligne = np.append([g[0]], np.zeros(n-2))
Q = np.array([Qpremiereligne])

Q1 = np.diag(g[1:n-1],1)
Q2 = np.diag(-g[:n-1]-g[1:n])
Q3 = np.diag(g[1:n-1],-1)
Q = np.append(Q, Q1 + Q2 + Q3, axis=0)

Qderniereligne = np.append(np.zeros(n-2), [g[n-1]])
Q = np.append(Q, [Qderniereligne], axis=0)

p = pow(10,5)
QT = np.transpose(Q)

A = np.matmul(QT,Q) + p*T
b = p*np.matmul(QT, np.transpose(y))

L = np.linalg.cholesky(A)
w = np.linalg.solve(L,b)
c = np.linalg.solve(np.transpose(L),w)
a = y - (1/p)*np.matmul(Q,c)
c0 = np.append([[0]], c)
c0 = np.append(c0,[[0]])

d = np.array([(c0[1]-c0[0])/(3*h[0])])
for i in range(1,n):
    d = np.append(d, (c0[i+1]-c0[i])/(3*h[i]))


b = np.array([((a[1]-a[0])/(h[0]))-c0[0]*h[0]-d[0]*h[i]**2])
for i in range(1,n):
    b = np.append(b,((a[i+1]-a[i])/(h[i]))-c0[i]*h[i]-d[i]*h[i]**2)

```

```
x1=np.linspace(x[0],x[n],1000*n)
y1 = np.array([])
for i in range(0,n):
    for j in range(0,1000):
        y1 = np.append(y1,a[i]+b[i]*((x1[i*1000+j]-x[i]))+c0[i]*pow((x1[i*1000+j]-x[i]),2)+d[i]*pow(((x1[i*1000+j]-x[i])),3))

fig, ax = plt.subplots(figsize=(6.5, 4))
ax.plot(x, y, 'o', label='data')
ax.plot(x1,y1, label="Spline")
ax.set_xlim(min(x)-1,max(x)+1)
ax.legend(loc='lower left', ncol=2)
plt.show()
```

Listing 4 – Programme python haut niveau

Ce qui nous donne le résultat suivant :

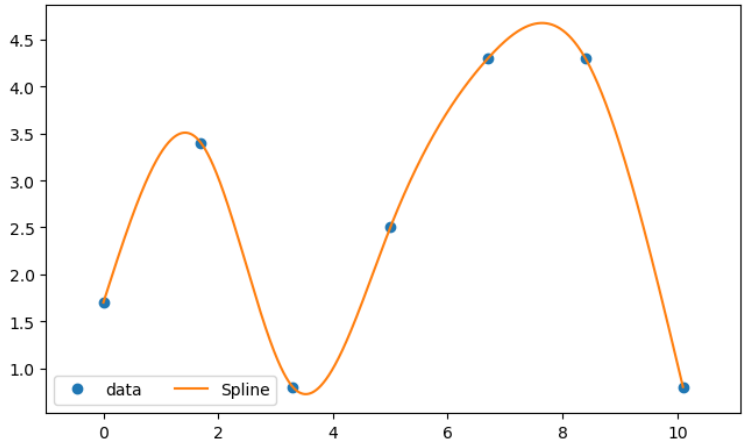

FIGURE 3.1 – La spline produite par le programme bas niveau

4 TRAITER UN EXEMPLE PLUS VOLUMINEUX

Nous souhaitons maintenant tester notre code sur un jeu de données plus volumineux.

Nous avons choisi d'utiliser l'exemple proposé dans le sujet, à savoir la densité minérale osseuse. Les données sont disposées dans un fichier csv qui regroupe des études réalisées sur un grand nombre d'individus. Pour chaque personne, nous avons l'évolution de la densité minérale osseuse prise à différents âges. Cela pose un problème, car nous obtenons plusieurs observations pour chaque coordonnées. Nous devons donc faire quelque chose aux données avant de faire la spline.
Nous devons grouper les observations sur les x. Pour cela, il existe plusieurs statistique :

moyenne, médiane, écart-type... Nous avons choisit d'utiliser la médiane, car elle n'est pas biaisée par les valeurs aberrantes. Ensuite, comme le nombre d'observation est variable pour chaque valeurs de l'âge présente dans les données, nous avons choisit d'utiliser la médiane mobile avec pour intervalle $[x_i - 1; x_i + 1]$. Nous créons comme cela des classes d'âge qui limite le nombre de points de la spline. Cela va réduire le bruit présent sur la spline et ainsi faciliter la lecture des données. Avec ce traitement, nous avons nos x et y auxquels nous appliquons la résolution bas niveau.
Voici le code correspondant au calcul de x et y :

```
data = np.genfromtxt('spnbmd.csv',delimiter=',',usecols=(2,4),skip_header
    =1)
xall = data[:,0]
yall = data[:,1]
x= range(int(min(xall)),int(max(xall)))
n = len(x) - 1
y = np.array([elem for elem in data if elem[0] >= int(min(x))-1 and elem
    [0] <= int(min(x))+1])


y=np.array([])
for i in x:
    val = np.array([elem for elem in data if elem[0] >= i-1 and elem[0]
    <= i+1])
    y = np.append(y,np.median(val[:,1]))
```

Listing 5 – Programme python haut niveau

Ce qui nous donne le résultat suivant :

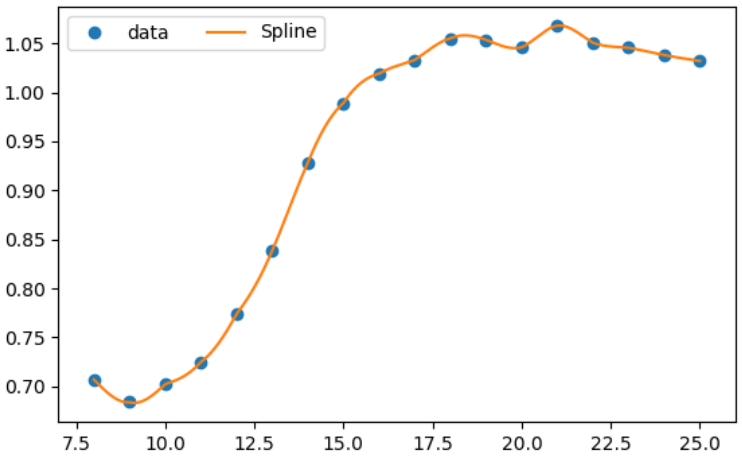

FIGURE 4.1 – La spline produite par le programme bas niveau sur les données de densité osseuse

Pour vérifier que cela correspond bien aux données brutes nous avons affiché les données de base avec la spline :

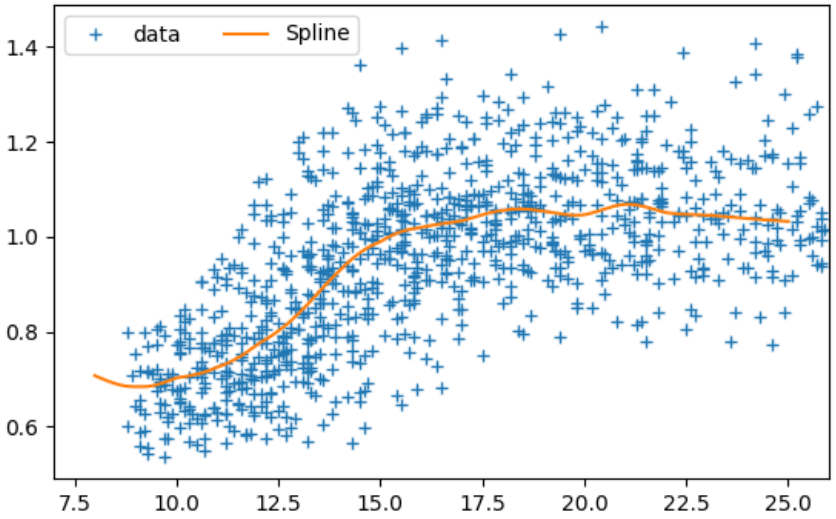

FIGURE 4.2 – Correspondance entre données réelles et la spline lissante

D'après ce dernier graphique, la spline semble bien suivre le centre du nuage de point.

5 PARTIE THÉORIQUE

Un multiplicateur de Lagrange est une variable qui permet de transformer un problème d'optimisation sous contrainte en un problème équivalent mais sans contrainte.

Dans notre cas, on cherche à minimiser l'intégrale suivante :

$$\int_a^b (g'')^2 \, \mathrm{d}x$$

à l'aide de cette contrainte d'inégalité :

$$\sum_{i=0}^{n} \left(\frac{g(x_i) - y_i}{\sigma_i} \right)^2 \leq S$$

Nous introduisons une variable d'écart z pour transformer cette contrainte d'inégalité en une contrainte d'égalité :

$$\sum_{i=0}^{n} \left(\frac{g(x_i) - y_i}{\sigma_i} \right)^2 = S - z^2$$

6 Calculer p en fonction de S

7 Degré de liberté

8 Paramètre optimal

Conclusion