Interpolation polynômiale par morceaux

Nicolas Pourcelot

5 mars 2010

Le but de ce document est d'offrir des interpolations polynômiales par morceaux utiles à la réalisation d'exercices de lecture graphique, en particulier en classe de seconde.

Il s'agit de générer des courbes passant par des points donnés, et semblant le plus élégantes possible, c'est-à-dire sans artefacts.

1 Interpolation par une courbe de fonction cubique par morceaux

1.1 Énoncé du problème

On cherche à tracer la courbe $\mathcal{C}$ d'une *fonction* f passant simplement par un certain nombre de points (en particulier, l'utilisateur final ne fournit pas de conditions sur les dérivées).

On supposera toujours ces points d'abscisses deux à deux distinctes, et rangés par ordre (strictement) croissant d'abscisse.

Soient trois points A, B, C par laquelle notre courbe doit passer [1].

La courbe doit en outre respecter les contraintes suivantes :
- si $y_B > y_A$ et $y_B > y_C$, alors y_B doit être le maximum de f sur l'intervalle $[x_A; x_C]$;
- si $y_B < y_A$ et $y_B < y_C$, alors y_B doit être le minimum de f sur l'intervalle $[x_A; x_C]$;
- sinon (c-à-d. si $y_A < y_B < y_C$, ou $y_A > y_B > y_C$), la pente de la courbe $\mathcal{C}$ au point B est celle de la droite (AC).

1. Si il n'y a que deux points A et B, l'interpolation la plus naturelle est le segment $[AB]$.

Les deux premières contraintes permettent de réaliser facilement des courbes d'extrema locaux choisis.

La dernière contrainte est, elle, purement esthétique.

1.2 Interpolation entre $(0;0)$ et $(1;1)$

Soient $a, b, c, d \in \mathbb{R}$.

Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = ax^3 + bx^2 + cx + d$.

On cherche à déterminer a, b, c, d en fonction de $f(0) = 0$, $f'(0)$, $f(1) = 1$ et $f'(1)$.

$\forall x \in \mathbb{R}, f'(x) = 3ax^2 + 2bx + c$.

On obtient donc :

$$\begin{cases} 0 = d \\ f'(0) = c \\ 1 = a + b + c + d \qquad (L_3) \\ f'(1) = 3a + 2b + c \qquad (L_4) \end{cases} \Leftrightarrow \begin{cases} 0 = d \\ f'(0) = c \\ 3 - f'(1) = b + 2c \qquad (3L_3 - L_4) \\ f'(1) = 3a + 2b + c \end{cases}$$

$$\Leftrightarrow \begin{cases} 0 = d \\ f'(0) = c \\ 3 - f'(1) - 2f'(0) = b \\ \dfrac{f'(1) - 2b - c}{3} = a \end{cases}$$

$$\Leftrightarrow \begin{cases} d = 0 \\ c = f'(0) \\ b = 3 - f'(1) - 2f'(0) \\ a = \dfrac{f'(1) - 2(3 - f'(1) - 2f'(0)) - f'(0)}{3} \end{cases}$$

$$\Leftrightarrow \begin{cases} d = 0 \\ c = f'(0) \\ b = 3 - f'(1) - 2f'(0) \\ a = \dfrac{3f'(1) - 6 + 3f'(0)}{3} \end{cases}$$

$$\Leftrightarrow \begin{cases} a = f'(1) - 2 + f'(0) \\ b = 3 - f'(1) - 2f'(0) \\ c = f'(0) \\ d = 0 \end{cases}$$

1.3 Cas général : interpolation entre A et B

On se ramène au cas précédent par transformation affine (attention, les dérivées sont modifiées !).

Puis, on obtient une interpolation entre A et B en appliquant à la fonction solution trouvée la transformation affine inverse.

$$\forall x \in \mathbb{R}, f(x) = k_y \left(a \left(\frac{x}{k_x} - x_A \right)^3 + b \left(\frac{x}{k_x} - x_A \right)^2 + c \left(\frac{x}{k_x} - x_A \right) \right) + y_A$$

avec :

$$\begin{cases} k_x = x_B - x_A \qquad (k_x \neq 0 \text{ car } x_A \neq x_B) \\ k_y = y_B - y_A \\ a = \dfrac{k_x}{k_y} f'(x_B) - 2 + \dfrac{k_x}{k_y} f'(x_A) \\ b = 3 - \dfrac{k_x}{k_y} f'(x_B) - 2\dfrac{k_x}{k_y} f'(x_A) \\ c = f'(x_A) \end{cases}$$

1.4 Pente en un point d'interpolation :

Si le point est encadré par deux valeurs, on reprend les cas précédents (cf. 1.1) :

- si $y_A < y_B < y_C$ ou $y_C < y_B < y_A$, alors $f'(x_B) = \dfrac{y_C - y_A}{x_C - x_A}$;
- sinon, $f'(x_B) = 0$.

Pour le point de départ et le point d'arrivée de la courbe, on peut choisir comme valeurs respectives :

- $f'(x_A) = \dfrac{y_B - y_A}{x_B - x_A}$
- $f'(x_C) = \dfrac{y_C - y_B}{x_C - x_B}$

1.5 Implémentation

La courbe est définie si et seulement si deux points distincts du nuage ont des abscisses distinctes également.

```
assert len([point.x for point in points]) == len(set(point.x for point in
points))
```

On commence par ordonner les points par abscisse croissante.

```
points.sort(key = operator.attrgetter("x"))
```

On construit ensuite la courbe de proche en proche.

On intialise :

```
pas = 0.01      (par exemple)
A, B = points[:2]
penteA = (B.y - A.y)/(B.x - A.x)
```

On itère :

```
for i, point in enumerate(points[1:-1]):
  A, B, C = points[i:i+3]
  kx = B.x - A.x
  ky = B.y - A.y
  r = kx/ky
  a=r*penteB-2+r*penteA
  b=3-r*penteB-2*r*penteA
  c=penteA
  if min(A.y, C.y) <= B.y <= max(A.y, C.y):
    penteB = (C.y - A.y)/(C.x - A.x)
  else:
    penteB = 0
  x = arange(A.x, B.x, pas)
  y = ky*(a*(x/kx - A.x)**3 + b*(x/kx - A.x)**2 + c*(x/kx - A.x)) + A.y
  penteA = penteB
```

Et on termine la courbe :

```
A, B = points[-2:] penteB = (B.y - A.y)/(B.x - A.x)
```