Devoir Surveillé 1

Tracer des courbes de Bézier –

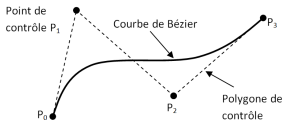
Informatique

PT – PT⋆

1 Présentation

Les courbes de Bézier ont été inventées par l'ingénieur Pierre Bézier (ingénieur Arts et Métiers (Pa. 1927) ingénieur chez Renault). Il s'agit de courbes paramétrées utilisées dans les logiciels de dessin, en conception assistée par ordinateur ou encore pour définir certaines polices de caractères. Même si ces courbes sont remplacées par des courbes de types « NURBS » elles restent néanmoins encore très utilisées.

Fonte définie par des courbes de Bézier



Courbe de Bézier et polygone de contrôle

Définition Soient P_0, P_1, ..., P_n, $n+1$ points de contrôle (ou pôles) de coordonnées (x_{Pi}, y_{Pi}). Pour une courbe plane, la position d'un point M de coordonnées $(x(t), y(t))$ dans la base $(\vec{x}, \vec{y})$ est définie par :

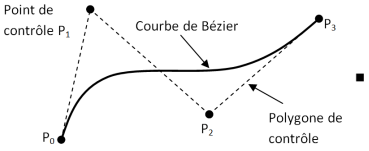
$$\forall t \in [0,1] \begin{cases} x(t) = \sum_{i=0}^{n} B_i^n(t) x_{Pi} \\ y(t) = \sum_{i=0}^{n} B_i^n(t) y_{Pi} \end{cases} \quad \text{avec} \quad B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad \text{et} \quad \binom{n}{i} = \frac{n!}{(n-i)!i!}$$

La fonction $B_i^n(t)$ est appelée polynôme de base de Bernstein.

▪ **Exemple**

Pour 4 pôles P_0, P_1, P_2 et P_3, (courbe de Bézier de degré 3), on a :

$$\forall t \in [0,1] \begin{cases} x(t) = (1-t)^3 t^0 x_{P_0} + 3(1-t)^2 t^1 x_{P_1} + 3(1-t)^1 t^2 x_{P_2} + (1-t)^0 t^3 x_{P_3} \\ y(t) = (1-t)^3 t^0 y_{P_0} + 3(1-t)^2 t^1 y_{P_1} + 3(1-t)^1 t^2 y_{P_2} + (1-t)^0 t^3 y_{P_3} \end{cases}$$



▪

Objectif L'objectif est de tracer les courbes de Bézier en utilisant des méthodes différentes.

2 Tracé naïf d'une courbe de Bézier

Le tracé d'une courbe de Bézier de degré n fait appel à la fonction `fact(n)` permettant de calculer $n!$.

Question 1 *Écrire cette fonction en utilisant un algorithme récursif* `factRec(n)`. *Vous prendrez soin de documenter votre fonction.*

Question 2 *Écrire cette fonction en utilisant un algorithme itératif* `factIt(n)`. *Vous prendrez soin de documenter votre fonction.*

Question 3 *En utilisant la fonction* `calculPointCourbe(poles,t)`, *réaliser le programme permettant de tracer une courbe sur 100 points. On rappelle que pour utiliser la fonction* `plot` *il est nécessaire de réaliser la liste des abscisses, qu'on pourra nommer* `les_x`, *et la liste des ordonnées, qu'on pourra nommer* `les_y`. *On fera l'hypothèse que la liste de pôles a déjà été renseignée dans la variable* `poles`.

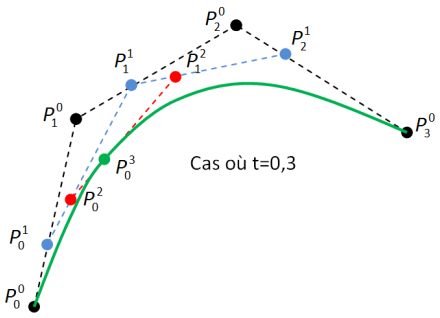
Question 4 *On fait l'hypothèse (forte) que la complexité algorithmique de la fonction* `pow`, *appelée dans la fonction* `fonctionBernstein`, *est linéaire. Donner la complexité algorithmique temporelle de la fonction* `fonctionBernstein`.

3 Utilisation de l'algorithme de De Casteljau

L'algorithme de De Casteljau repose sur le fait qu'une restriction d'une courbe de Bézier est aussi une courbe de Bézier. En notant P_0, P_1, P_2 et P_3 les 4 points de contrôle (ou pôles) d'une courbe de Bézier et t un réel donné appartenant à $[0;1]$:

- on construit les 3 barycentres P_j^1, $j \in [0;2]$ des pôles P_i^0, $i \in [0;3]$: $P_j^1 = (1-t)P_j^0 + tP_{j+1}^0$;
- on construit les 2 barycentres P_j^2, $j \in [0;1]$: $P_j^2 = (1-t)P_j^1 + tP_{j+1}^1$;
- on construit le dernier barycentre $P_0^3 = (1-t)P_0^2 + tP_1^2$.

Le dernier barycentre est un point de le courbe de Bézier.



Question 5 *On donne la fonction* `deCasteljau` *permettant de calculer l'abscisse (ou l'ordonnée) d'un point d'une courbe. Déterminer ce que retourne l'appel suivant (en justifiant et détaillant votre démarche) :* `deCasteljau([0,0,40,40],0,3,0.5)`.

Question 6 *Évaluer la complexité algorithmique de l'algorithme de De Casteljau en fonction du nombre de pôles.*

Question 7 *En identifiant un variant de boucle, montrer que l'algorithme se termine.*

4 Utilisation de l'algorithme de Horner

En mettant le polynôme sous la forme de Horner, on peut écrire que : $f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n = a_0 + x(a_1 + x(a_2 + ...x(a_{n-1} + a_n x)))$. Ainsi, sous la forme de Horner, $x(t) = \sum_{i=0}^{n} B_i^n(t) x_{Pi} = \sum_{i=0}^{n} a_i t^i$.

▪ Exemple

Pour une courbe de Bézier cubique, on a : $x(t) = (1-t)^3 t^0 x_{P_0} + 3(1-t)^2 t^1 x_{P_1} + 3(1-t)^1 t^2 x_{P_2} + (1-t)^0 t^3 x_{P_3}$.
Sous la forme de Horner, on a : $x(t) = \left(\left(\left(x_{P_3} - 3x_{P_2} + 3x_{P_1} - x_{P_0}\right)t + 3\left(x_{P_2} - 2x_{P_1} + x_{P_0}\right)\right)t + 3\left(x_{P_1} - x_{P_0}\right)\right)t + x_{P_0}$. ▪

Question 8 *Monter que* $a_i = \binom{n}{i} \sum_{j=0}^{i} \left((-1)^{i-j} \binom{i}{j} x_{Pj} \right)$.

Question 9 *Écrire un algorithme récursif, permettant de calculer un point de la courbe par la méthode de Horner. La fonction* `horner` *prendra comme argument* `L` *la liste des* a_i *(*`[an,a(n-1),...,a1,a0]`*) et le paramètre* t.

Question 10 *Quel peut-être l'avantage d'évaluer un polynôme en un point par la méthode de Horner plutôt que par une méthode naïve ?*

Algorithmes

Python

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

def coef_binom(i,n):
    """
    Retourne le coefficient binomial :
    C_n^i = n! / (i! (n-i)!)
    """
    res = fact(n)/(fact(i)*fact(n-i))
    return res

def calculPointCourbe(poles,u):
    """
    Retourne le point de la courbe de Bézier pour un paramètre donné.
    Entrées :
        * poles (list): liste des coordonnées des poles [[x1,y1],[x2,y2],...]
        * u (float) : paramètre appartenant à[0,1]
    Sortie :
        * pointM (list): point appartenant àla courbe de Bézier au paramètre u
    """
    px,py = [],[]
    for i in range(len(poles)):
        px.append(poles[i][0])
        py.append(poles[i][1])

    pointM = [fonctionBernstein(px,u),fonctionBernstein(py,u)]
    return pointM

def fonctionBernstein(p,u):
    """
    Calcul d'une des coordonnées d'un point appartenant àune courbe de Bézier.
        Entrées :
            * p (list): tableau contenant l'abscisse des poles
            * u (float): paramètre
        Sortie :
            * x (float) : une des coordonnées (suivant x ou y) d'un point de la courbe
    """
    n = len(p)
    x=0
    for i in range(n):
        x=x+coef_binom(i,n-1)*pow(u,i)*pow((1-u),n-i-1)*p[i]
    return x

def deCasteljau(P, i, j, t):
    """
    Retourne l'abscisse(ou l'ordonnées) d'un point de la courbe de Bézier pour un paramètre donné.
    Entrées :
     * P (list) : listes des abscisses (ou des ordonnéees) des poles
     * i,j (int) : poles considérés
     * t (float) : paramètre compris entre 0 et 1
    Sortie :
     * float : abscisse ou ordonnée d'un point de la courbe
    """
    if j == 0:
        return P[i]
    else :
        return deCasteljau(P,i,j-1,t)*(1-t)+deCasteljau(P,i+1,j-1,t)*t
```

DM 01