3 TP3

3.1 Courbes de Bézier

3.1.1 Rappels

Une courbe de Bézier avec $n+1$ points de contrôle s'écrit :

$$C(t) = \sum_{i=0}^{n} P_i \mathcal{B}_{(i,n)}(t) \text{ avec } t \in [0,1] \tag{3.1}$$

Les polynômes de Bernstein associés sont :

$$\mathcal{B}_{(i,n)}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} \text{ , pour } t \in [0,1] \tag{3.2}$$

La dérivée $\mathcal{B}'_{i,n}(t)$ du polynôme de Bernstein $\mathcal{B}'_{i,n}(t)$ s'écrit :

$$\mathcal{B}'_{(i,n)} = n(\mathcal{B}_{i-1,n-1}(t) - \mathcal{B}_{i,n-1}(t)) \text{ , pour } t \in [0,1] \tag{3.3}$$

3.1.2 Calcul par la méthode directe

Soient 4 points $P_0(0,0)$, $P_1(0,1)$, $P_2(1,1)$, $P_3(2,0)$. $Q(t)$ est la courbe de Bézier (cubique) dont ces 4 points[1] sont les points de contrôles.

1. tracer les 4 courbes liées aux polynômes de Bernstein d'une courbe de degré 3 (fonctions de base),
2. tracer la courbe de Bézier correspondant à ces 4 points
3. tracer la courbe dérivée de cette courbe : $C'(t) = \sum_{i=0}^{n} P_i \mathcal{B}'_{(i,n)}(t)$ avec $t \in [0,1]$

Comme décrire l'ensemble des points de contrôle et les points de la courbe ? On peut utiliser un vecteur de vecteur (matrice) car nous aurons besoin d'opérations vectorielles (cf éq. 3.1). Par exemple, les points P sont définis par des vecteurs ligne :

```
P0=[0, 0, 0];
P1=[1, 1, 0];
P2=[2, 1, 0];
P3=[3, 0, 0];
```

Les points de contrôles sont un ensemble de points (une colonne de vecteurs 3) :

```
ptsControle=[P0; P1; P2; P3];
```

Le calcul d'un point sur la courbe peut alors se faire par des opérations vectorielles. `BernsteinValue(t, i, degre+1)` renvoie un scalaire (cf. éq.3.2). `ptsControle(i, :)` est le vecteur 3 de la i^e ligne de la matrice des points de contrôle .

```
function p = computeBezierPoint(nbPtsControle, ptsControle, t)
...
        p = p + ptsControle(i, :) * BernsteinValue(t, i, degre+1);
...
endfunction
```

Dans cette écriture `p` est un tableau de vecteur 3. On peut l'utiliser dans l'affichage à condition de séparer la 1ère colonne (les x) de la 2e (les y) :

```
ptsContX= ptsControle(:, 1);
ptsContY= ptsControle(:, 2);
plot(ptsContX, ptsContY, "-*r");
```

1. On a choisi des points de dimension 2 mais cela ne change rien pour la formule 3.1

3.1.3 Paramétrisation/discrétisation

Lorsque l'on veut afficher une courbe de Bézier, on effectue une discrétisation de l'espace paramétrique $[0, 1]$. Dans l'exemple précédent vous avez peut-être utilisé des valeurs discrètes de t (par ex. $t = \{0\ 0,1\ 0,2\ 0,3\ ...\ 0,9\ 1\}$) pour calculer des points de $C(t)$ et l'afficher par une polyligne. Cette discrétisation d'un espace au départ continu (on avait dit $t \in [0, 1]$) s'est donc fait de manière uniforme.

On peut utiliser d'autres ensembles discrets, par exemple $t = \{0\ 0,3\ 0,4\ 0,5\ 1\}$ qui représente encore la courbe grâce à ces points extrêmes $C(0)$ et $C(1)$ mais cela pourrait être aussi $t = \{0,1\ 0,11\ 0,12\ 0,13\ 0,14\ ...\ 0,2\}$.

Cela dépend donc de ce que l'on veut extraire de la courbe. On peut même relier la discrétisation à la paramétrisation de la courbe. Si la discrétisation est uniforme comme vous l'avez fait, on peut penser que t serait un point se déplaçant dans $[0, 1]$. Rien n'assure que $t_{i+1} - t_i =$ constante entraîne $C(t_{i+1}) - C(t_i) =$ constante, cela dépend de la courbure de $C(t)$ (ou de la vitesse d'un mobile qu'aurait cette courbe pour reprendre l'analogie).

1. exprimer sous forme de fonction uniforme la discrétisation de l'ensemble ($f(u) \to t_{\text{discret}} \in [0...1]$ (on obtiendra le même résultat qu'à la question précédente $u = 0\ 0,1\ 0,2\ 0,3\ ...\ 0,9\ 1 \to t = 0\ 0,1\ 0,2\ 0,3\ ...\ 0,9\ 1$). Calculer pour chaque point sur la courbe $C(t_i)$ la distance au point précédent $|C(t_i) - C(t_{i-1})|$. Est-ce uniforme ? Tracer la polyligne représentant ces distances ($dist(0) = 0$, $dist(1) = |C(t_1) - C(t_0)|$, ...).
2. utiliser la fonction $f(x) = x^2$ pour $x \in [0, 1]$. On aura donc $u = 0\ 0,1\ 0,2\ 0,3\ ...\ 0,9\ 1 \to t = 0\ 0,01\ 0,04\ 0,09\ ...\ 0,81\ 1$. À quoi cela correspondrait sur la polyligne des distances successives ?
3. et si $f(x) = -(2x - 1)^2 + 1$?

Comment peut-on faire cela avec Scilab ? On peut passer comme paramètre d'une fonction une autre fonction. Dans notre cas, on passe à une fonction qui calcule le vecteur de paramètres une fonction qui décrit comment sont réparties ces paramètres.

```
function y = homogene(x)
    y = x;
endfunction

function tabRetours = computeParamValues(maFonction, numValeurs)
    tabRetours = [];
    for u = 0:1/numValeurs:1
        v = maFonction(u);
        tabRetours = [tabRetours, v]; // attention à faire un vecteur ligne
    end
endfunction
```

On appelera ensuite cette fonction pour remplir un tableau :

```
    param = computeParamValues(homogene, 10);
```

Et on utilisera directement ce tableau dans une boucle `for` pour calculer les points sur la courbe de Bézier correspondant :

```

function curvePoints = computeBezierCurve(ptsControle)
  ...
    //for p = 0:0.1:1 // inutile dorénavant
    for p = param
        curvePoints = [curvePoints; computeBezierPoint(3, ptsControle, p)]; // attention à faire une
    matrice de 3 valeurs par ligne
        ...
    end
...
endfunction
```

3.1.4 C^0 et C^1-continuités

La C^0-continuité entre 2 courbes de Bézier est assurée lorsqu'un point extrême d'un polygone de contrôle est confondu avec un point extrême d'un autre polygone de contrôle. Par exemple $C(1) = P_0 = D(0) = Q_3$ assure que les 2 courbes $C(t)$ et $D(u)$ sont jointives en P_0, Q_3 respectivement pour chacune d'entre elle. C'est dû à la propriété des courbes de Bézier d'interpoler les 2 points extrêmes ($C(t = 0) = P_0$, $C(t = 1) = P_3$ pour une courbe de 4 points de contrôle).

De même la C^1-continuité assure que les tangentes, en $t = 1$ par exemple, sont les mêmes en sens, module et direction, à gauche et à droite du point de contact (il faut déjà avoir effectué la C^0-continuité). C'est induit par le calcul des tangentes aux points extrêmes d'une courbe de Bézier : $C'(t = 0) = k\overrightarrow{P_0P_1}$, k dépendant du degré de la courbe. Idem pour $C'(t = 1) = k\overrightarrow{P_2P_3}$ (avec le même k que précédemment). Pour avoir la C^1-continuité des courbes $C(t)$ définie par les points de contrôle P_0, P_1, P_2, P_3 et $D(u)$ de points Q_0, Q_1, Q_2, Q_3, il suffit d'assurer $P_2, P_3 = Q_0, Q_1$ alignés et $|P_2P_3| = |Q_0Q_1|$.

1. construire des courbes C^0 et C^1-continues.

3.1.5 Calcul par De Casteljau

Utiliser l'algorithme de De Casteljau pour visualiser une courbe de Bézier. Tracer les polygones de contrôles intermédiaires de quelques valeurs ($t = 0,25$, $t = 0,5$, $t = 0,75$).

3.2 Annexe

3.2.1 Factorielle

La factorielle n'existe pas avec Scilab, on peut très bien la recoder avec `prod(1:n)` qui calcule $1 * 2 * 3 * ... * n$. Attention, par convention $0! = 1$:

```
function f = fact(n)
    if (n == 0) then
        f = 1;
    else
        f = prod(1:n);
    end;
endfunction
```

3.2.2 Tracé

On peut tracer une polyligne en marquant ces sommets avec `plot()` plutôt que `plot2d` (cf le wikilivre à la section « graphisme et son »[2]). Par exemple, pour une courbe décrite par des lignes continues (`-`), des marques `*` et en rouge (`r`) :

```
plot(pX, pY, "-*r");
```

Alors que pour une courbe décrite par des traits tiretés (`--`), des marques circulaires (`o`) et en noir(`k`) :

```
plot(curveX, curveY, "--ok");
```

2. `https://fr.wikibooks.org/wiki/Découvrir_Scilab/Graphiques_et_sons`