



Rémy Malgouyres,
Algorithmes pour la synthèse d'images
et l'animation 3D,
DUNOD, 2002

TP n° 4 : Interpolation par des courbes

Objectifs :

Le but de ce TP est de commencer à construire une hiérarchie de classes pour représenter des courbes planes. Chaque classe dérivée correspondra soit à un type de courbe particulier, soit à un algorithme d'affichage particulier. On implémentera plusieurs algorithmes d'interpolation, d'abord pour des courbes *fonctionnelles* (Lagrange, Neville-Aitken, Spline cubique), puis pour des courbes *non fonctionnelles* en se ramenant au cas fonctionnel.

1 Objectifs à réaliser

On souhaite réaliser une application permettant à un utilisateur de définir et de visualiser dans une fenêtre graphique différentes sortes de courbes. On proposera un menu *Interpolation* dans lequel apparaîtront des commandes permettant de lancer le dessin d'une courbe définie par ses points de contrôle. Après activation d'une commande, l'utilisateur rentre des points de contrôle par des clics de souris. Une fois le dernier point de contrôle saisi, le programme affiche la courbe du type correspondant à la commande choisie, et pour les points de contrôle rentrés.

On commencera par créer un nouveau projet *Visual C++*, *MFC*, appelé MathSynth de type *single document*.

2 Les classes abstraites

2.1 La classe de base Courbe2D

On définira tout d'abord une classe de base *Courbe2D*, dont dériveront toutes les classes correspondant à des courbes tout au long des différents TP.

Cette classe est définie comme suit :

```

class Courbe2D{

protected:

public:
    Courbe2D(){

        void affiche(CDC& cdc, Pixel centrevue, double facteur_echelle);

        virtual void getpoint(double t, double & xpos, double & ypos)=0;
};

```

Le programme étant destiné à être par la suite intégré à un programme déjà existant, Il est fortement conseillé de respecter rigoureusement les identificateurs et les prototypes.

La fonction membre `getpoint` prend en paramètre un réel $t \in [0, 1]$ et retourne le point $Q(a(1 - t) + bt)$, où $Q : [a, b] \rightarrow \mathbb{R}^2$ est la courbe en question. La fonction `getpoint` est *virtuelle*, ce qui signifie que son implémentation se trouvera dans les classes dérivées de la classe `Courbe2D`.

La fonction `affiche` utilise la fonction `getpoint` en traçant calculant un certain nombre de points $Q(t_0), \dots, Q(t_n)$, et en traçant les segments entre tous les couples de points calculés successifs. Cette fonction `affiche` n'est pas virtuelle et doit fonctionner que que soit le type de courbe.

La classe `Pixel`, quand à elle contient deux données entières publiques x et y .

2.2 La classe Interpol

C'est de la classe abstraite `Interpol` que dériveront toutes les classes permettant de réaliser des interpolations par des courbes fonctionnelles. classe `Interpol` dérive de la classe `Courbe2D`. La définition de cette classe est la suivante :

```

class Interpol : public Courbe2D{
protected:
    Pixel *ctrlpoints; // tableau des points de contrôle
    int nbctrlpoints; // nombre de points de contrôle
public:
    Interpol();
    Interpol(Pixel *controlpoints, int nbcontrolpoints);
    virtual ~Interpol();

    virtual void getpoint(double t, double & xpos, double & ypos)=0;
};

```

Important : Dans le constructeur `Interpol(Pixel *controlpoints, int nbcontrolpoints)`, il faut trier les points de contrôle dans l'ordre des x croissant.

3 Interpolation de Lagrange

3.1 Principe de l'interpolation de Lagrange

Le principe d'interpolation de Lagrange est d'abord défini pour des courbes *fonctionnelles*, c'est à dire de la forme :

$$y = f(x).$$

Soit une suite de n points de contrôles P_0, \dots, P_{n-1} , avec, pour $i = 1, \dots, n$, $P_i = (x_i, y_i)$. On supposera que les x_i sont triés par ordre croissant (voir la remarque à propos du constructeur de la classe `Interpol`).

Considérons, pour $i = 0, \dots, n - 1$, le polynôme L_i défini par :

$$L_i(x) = \frac{\prod_{\substack{j=0 \\ j \neq i}}^{n-1} (x - x_j)}{\prod_{\substack{j=0 \\ j \neq i}}^{n-1} (x_i - x_j)}$$

Exercice 1.

Montrer que $L_i(x_k) = 0$ pour $i \neq k$ et $L_i(x_i) = 1$.

Soit maintenant P le polynôme (dit *polynôme de Lagrange associé aux points (x_i, y_i)*) défini par :

$$P(x) = \sum_{i=0}^{n-1} L_i(x) \cdot y_i$$

Exercice 2.

Montrer que le polynôme de Lagrange P interpole le points P_k , c'est à dire qu'on a $P(x_k) = y_k$ pour $k = 0, \dots, n - 1$.

3.2 Programmation de l'interpolation de Lagrange

- On définira une classe `InterpolLagrange`, dérivée de la classe `interpol`, et on implémentera la fonction `getpoint` correspondante d'après le principe d'interpolation de Lagrange.
- Rajouter une donnée membre `courbe`, de type `Courbe2D*`, à la classe vue du projet. Écrire une procédure de saisie des points de contrôle par des clics de souris. Allouer la variable `courbe` tout en appelant constructeur prenant en paramètre les points de contrôle. Dans la fonction `OnDraw` de la classe vue, faites afficher la courbe. Testez.
- Essayez d'interpoler un grand nombre de points. Que se passe-t-il ? Comment expliquez-vous ce phénomène ?

4 Interpolation de Neville-Aitken

4.1 Principe de l'interpolation de Neville-Aitken

À nouveau, soit une suite de n points de contrôles P_0, \dots, P_{n-1} , avec, pour $i = 1, \dots, n$, $P_i = (x_i, y_i)$. On supposera que les x_i sont triés par ordre croissant. Le principe consiste à

calculer les polyn ome $P_{i,j}(x)$ pour $i = 0, \dots, n - 1$ et $j = i, \dots, n - 1$ d efinis par r ecurrence comme suit :

Initialisation : pour $i = 0, \dots, n - 1$ et pour tout x

$$P_{i,i}(x) = y_i.$$

It eration : pour $i = 0, \dots, n - 1$ et pour $j > i$

$$P_{i,j}(x) = \frac{(x - x_i)P_{i+1,j}(x) - (x - x_j)P_{i,j-1}(x)}{x_j - x_i}$$

Le polyn ome d'interpolation cherch e est $y = P_{0,n-1}$.

Exercice 3.

Montrer que l'on peut calculer, pour i fix e, les nombres $P_{i,j}(x)$  a partir de $P_{i,i}(x)$ et des $P_{i+1,k}$. En d eduire comment calculer $P_{0,n-1}(x)$ en utilisant un tableau de n nombres r eels.

4.2 Programmation de l'interpolation de Neville-Aitken

- On d efinira une classe `InterpolNeville`, d eriv ee de la classe `interpol`, et on impl ementera la fonction `getpoint` correspondante d'apr es le principe d'interpolation de Neville-Aitken.
- Ajouter une commande  a l'application permettant de lancer la saisie des points de contr ole par des clics de souris. Allouer la variable `courbe` tout en appelant constructeur prenant en param etre les points de contr ole. Testez.

5 Interpolation spline cubique

Fixons $i \in \{0, \dots, n - 2\}$. Si l'on veut interpoler entre les points x_i et x_{i+1} par un segment de droite, on obtient les formules d'interpolation lin eaire suivantes :

$$y(x) = Ay_i + By_{i+1}.$$

avec

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i} \text{ et } B = \frac{x - x_i}{x_{i+1} - x_i}$$

En effet, cette  equation est l' equation param etrique d'une droite passant par les points y_i (pour $x = x_i$) et y_{i+1} (pour $x = x_{i+1}$).

Si l'on fait une telle interpolation lin eaire entre pour tous les $i \in \{0, \dots, n - 2\}$, on obtient une courbe qui est une ligne bris ee ayant les points (x_i, y_i) pour sommets. En cons equence, la d eriv ee et la d eriv ee seconde de cette courbe ne sont pas d efinies aux points (x_i, y_i) . Or, le but de l'interpolation spline cubique est d'obtenir une courbe de classe C^2 , c'est  a dire dont la d eriv ee et la d eriv ee seconde sont d efinies et continues en tout point.

Pour r ealiser cela, supposons que l'on puisse d ecider des valeurs y_i''  a donner aux d eriv ees secondes de y aux points y_i , pour $i = 0, \dots, n - 1$. Nous pouvons alors rendre la d eriv ee seconde de notre courbe continue en consid erant l' equation param etrique suivante :

$$y(x) = Ay_i + By_{i+1} + Cy_i'' + Dy_{i+1}''$$

avec $C = \frac{1}{6}(A^3 - A)(x_{i+1} - x_i)^2$ et $D = \frac{1}{6}(B^3 - B)(x_{i+1} - x_i)^2$.

Exercice 4.

Montrer que la d eriv ee seconde de l'expression ci-dessus est continue au point x_i . Qu'en est-il de la d eriv ee premi ere ? La fonction y ainsi d efinie est-elle de classe C^2 ?

Pour rendre la d eriv ee premi ere continue, nous pouvons essayer de fixer judicieusement les valeurs y_i'' pour $i = 0, \dots, n - 1$. Pour cela,  egalons la d eriv ee premi ere de $y(x)$ au points x_i , calcul ee dans l'intervalle $[x_i, x_{i+1}]$ et calcul ee dans l'intervalle $[x_{i-1}, x_i]$. On obtient pour $i = 1, \dots, n - 2$:

$$\left(\frac{1}{6}(x_i - x_{i-1})\right) y_{i-1}'' + \left(\frac{1}{3}(2x_i - x_{i-1} - x_{i+1})\right) y_i'' + \left(\frac{1}{6}(x_{i+1} - x_i)\right) y_{i+1}'' = \frac{y_{i-1} - y_i}{x_i - x_{i-1}} + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}.$$

C'est un syst eme de $n - 1$  equations pour les $n - 2$ inconnues y_i'' , avec $i = 1, \dots, n - 2$ (si l'on fixe par exemple les valeurs $y_0'' = y_{n-1}'' = 0$). Les propri etes de ce syst eme (syst eme tridiagonal), nous permettent de l'inverser pour le r esoudre en un nombre d'op erations $O(n)$.

5.1 Programmation de l'interpolation Spline cubique

- On d efinira une classe `InterpolSpline`, d eriv ee de la classe `interpol`, et on impl ementera la fonction `getpoint` correspondante d'apr es le principe d'interpolation spline cubique. La fonction permettant de calculer les d eriv ees secondes y_i'' par r esolution du syst eme tridiagonal ci-dessus est fournie dans le fichier `tridiagonal.cpp`.
- Ajouter une commande   l'application permettant de lancer la saisie des points de contr ole par des clicks de souris. Allouer la variable `courbe` tout en appelant constructeur prenant en param etre les points de contr ole. Testez.
- L'interpolation d'un grand nombre de points pose-t-elle les m emes probl emes qu'avec les autres types d'interpolation ?

6 Cas non fonctionnel

6.1 Principe de l'interpolation non fonctionnelle

Consid erons cette fois des points de contr ole (x_i, y_i) , non plus rang es dans l'ordre des x_i croissant, mais rang es dans l'ordre ou les a d efini l'utilisateur. Nous voulons interpoler ces points par une courbe param etr ee $M(t) = (x(t), y(t))$ quelconque, et non plus fonctionnelle.

Pour cela, nous allons consid erer deux courbes fonctionnelles $y_a(x)$ et $y_b(x)$, obtenues par le proc ede spline cubique ci-dessus, et obtenues par les points de contr ole suivants.

La courbe y_a est obtenue par les points de contr ole $(x_{a,i}, y_{a,i})$ pour $i = 0, \dots, n - 1$, avec $x_{a,i} = i$ et $y_{a,i} = x_i$.

La courbe y_b est obtenue par les points de contr ole $(x_{b,i}, y_{b,i})$ pour $i = 0, \dots, n - 1$, avec $x_{b,i} = i$ et $y_{b,i} = y_i$.

On pose :

$$x(t) = y_a(t) \text{ et } y(t) = y_b(t)$$

Exercice 5.

Montrer que la courbe M ainsi d efinie interpole les points de contr ole (x_i, y_i) .

6.2 Programmation de l'interpolation non fonctionnelle

- On définira une classe `InterpolNonFonct`, dérivée de la classe `Courbe2D` comprenant deux données membres de type pointeur sur `InterpolSpline`. Définir un constructeur ayant pour paramètre le tableau des points de contrôle. On implémentera la fonction `InterpolNonFonct : :getpoint` utilisant la fonction `getpoint` de la classe `InterpolSpline`.
- Ajouter une commande à l'application permettant de lancer la saisie des points de contrôle par des clicks de souris. Allouer la variable `courbe` tout en appelant constructeur prenant en paramètre les points de contrôle. Testez.