Sobre los B-Splines

Geometría diferencial aplicada

PER 1582

1. Introducción

El objetivo de estos apuntes es ampliar el material de soporte para la sesión novena de este curso, esto es, la sesión dedicada a los B-splines. Este tema se desarrolla dentro del marco de la aproximación numérica por polinomios de curvas en el plano (o en el espacio). Una de las formas más sencillas de aproximar una curva $c : I \mapsto \mathbb{R}^n$ es mediante su polinomio de Taylor[1]:

$$c(t) \approx \sum_{i=0}^{n} \frac{c^{i)}(0)}{i!} t^i,$$

y $c^{i)}(0) \in \mathbb{R}^n$ denota la derivada de orden i de c evaluada en el origen. Entonces, fijado un número natural positivo n, se puede escribir una aproximación de orden n, como una combinación lineal de los vectores $\{1, t, t^2, \dots t^n\}$, la base canónica del espacio de polinomios de grado n. El error de la fórmula de Taylor viene dada por una expresión del tipo $R_n = C_n t^{k+1}$. Esto significa que la aproximación es muy buena para t cercanos a cero y empeora significativamente cuando $t > 1$.
Una forma habitual de conseguir una aproximación buena entorno a una malla $\{t_0, t_1, \dots, t_{n-1}\}$ es utilizar la base de los polinomios de Lagrange:

$$\ell_j = \prod_{i=0, i\neq j} \frac{t - t_i}{t_j - t_i}.$$

En la sesión # 8, vimos cómo podíamos calcular los coeficientes de una aproximación con una base de Lagrange mediante los métodos de Lagrange y de Newton. También comentamos ciertos comportamientos patológicos que muestran estas aproximaciones en algunos casos.

Para solventar estos problemas, vimos que podíamos utilizar splines. La idea de los splines es sencilla, en vez de calcular un polinomio que interpole todos los nodos, se calcula un polinomio para cada par de nodos. Esto es, cada polinomio pasa por dos de los nodos. El grado de los polinomios interpoladores se puede asignar independientemente del número de nodos (en los casos de Lagrange y Newton, el grado de los polinomios queda determinado por el número de puntos) aunque, es muy común escojer polinomios cúbicos. Esta elección no es casual, ya que, usando polinomios cúbicos, uno puede imponer derivabilidad hasta segundo orden en los empalmes de los polinomios en los nodos.

Los métodos de interpolación son, en esencia, maneras de transformar un conjunto de puntos en el plano (o el espacio) en un conjunto de coeficientes que permiten recuperar esos puntos y que,

[1] Para simplificar la exposición, hemos tomado en desarrollo de Taylor respecto al origen. Podríamos haber elegido cualquier punto del intervalo I.

además, nos dan información de su entorno. En el caso de los métodos de Lagrange y Newton, la transformación resulta en el mismo número de coeficientes que de puntos de interpolación: Esto es, guardando la misma cantidad de información, podemos describir mejor la relación entre los puntos.

El problema de los coeficientes de los polinomios de interpolación es que dan muy poca información geométrica acerca de la curva que pasa por los puntos que interpolan. Los puntos de control represantan una alternativa a los coeficientes para codificar información de una curva. Además, estos definen el llamado polinomio de control (esto es, el polinomio cuyos vértices son los puntos de control) que localiza la curva en el plano (espacio).

2. Curvas de Bézier

Otra base de polinomios usada frecuentemente son los poliomios de Bernstein de grado n, definidos por:

$$B_i^n = \binom{n}{i} t^i (1-t)^{n-i}, \quad \binom{n}{i} = \frac{n!}{i!(n-i)!}. \tag{1}$$

Los polinomios de Bernstein verifican las siguientes propiedades:

1. $B_i^n(t) > 0$ para $t \in [0,1]$,
2. $B_0^n(0) = 1$ y $B_i^n(0) = 0$ para $i \neq 0$,
3. $B_n^n(1) = 1$ y $B_i^n(1) = 0$ para $i \neq 1$,

Si $c : [a,b] \mapsto \mathbb{R}^n$, entonces se puede reducir a una curva definida en el intervalo $[0,1]$ utilizando la reparametrización

$$t' = \frac{t-a}{b-a}.$$

Una curva polinomial descrita como combinación lineal de la base de polinomios de Bernstein de grado n se llama curva de Bézier. Si escribimos

$$c(t) = \sum_{i=0}^{n} c_i B_i^n(t), \quad t \in [0,1].$$

Los coeficientes c_i se llaman **puntos de control** de la curva o vértices del polígono de control. Dadas las propiedades 1. y 3., los coeficientes, se cumple $c(0) = c_0$ y $c(1) = c_n$. Esto es, una curva de Bézier se ancla en el primer y último puntos de control. Sin embargo, esto no sucede con el **resto de puntos de control que, en general, no están dentro de la curva**.

A continuación mostramos un trozo de código escrito en lenguaje C que evalúa una curva de Bézier en una malla de valores de t:

```
1     for(i=0; i<=GRID_SIZE; i++){
2                 t=step*i;
3                 s=0.;
4
5                 for(j=0; j<N; j++){
6                         s+=cn[j]*eval_Bernstein(j, DEG, t);
7                 }
8         }
```

Aquí, la variable `GRID_SIZE` indica el tamaño de la malla, la variable `N` indica el número de puntos de control, el vector `cn` contiene los puntos de control (la entrada del programa) y la variable `DEG` indica el grado de los polinomios de Benstein. La función `eval_Bernstein` evalúa el polinomio de Bernstein `j` de grado `DEG` en el instante `t` . Esta función se implementa a parte:

```
1     double combinatorial(int n, int i)
2     {
3             int j;
4             double s;
5
6             s=0.;
7             for(j=1; j<=n; j++) s+=log((double)j);
8             for(j=1; j<=i; j++) s-=log((double)j);
9             for(j=i; j<n; j++) s-=log((double)(n-j));
10
11    return exp(s);
12    }
13
14    double eval_Bernstein(int i, int n, double t)
15    {
16    return combinatorial(n,i)*pow(t, i)*pow(1.-t, n-i);
17    }
```

Vemos que, de hecho, la función `eval_Bernstein` es muy sencilla, solamente implementa la expresión explícita que define los polinomios de Bernstein, esto es, la ecuación (1). La función `combinatorial` es una implementación de los números combinatorios, necesarios para calcular los coeficientes de los polinomios de Berstein. Hacemos notar que se evita calcular los factoriales de esa expresión, aplicando un logaritmo y transformando los productos en sumas.

Ejercicio 1 *Implementa el algoritmo de evaluación de curvas de Bézier en Matlab. Puedes adaptar el código* `C` *que se muestra en este documento. Para ello, ten en cuenta que los vectores en* `C` *empiezan a contar sus componentes desde 0 y en Matlab desde 1.*

3. Nodos de interpolación, Coeficientes y puntos de control

En esta sección vamos a entender un poco mejor la diferencia que existe entre nodos de interpolación, esto es, los puntos de $\mathbb{R}^r$ por los que pasa un polinomio, los coeficientes del polinomio y los puntos de control para las curvas de Bézier (o para los B-splines). Como ya hemos mencionado, el conjunto de polinomios de grado menor o igual que n forma un espacio vectorial de dimensión $n+1$. Este espacio tiene, como base canónica, $\{1, t, t^2, \ldots, t^n\}$. Un polinomio de grado n, se expresa como una combinación lineal de esa base canónica y los coeficientes de la combinación lineal son los coeficientes del polinomio. El espacio vectorial de los polinomios de grado menor o igual que n también puede ser generado con los polinomios de Bernstein. Como en cualquier otro espacio vectorial, podemos preguntarnos por la matriz de cambio de la base canónica a la base de Bernstein. Si expresamos un polinomio como un vector cuyos componentes son sus coeficientes, podemos obtener una curva de Bézier simplemente multiplicando los coeficientes por la matriz de cambio. Antes de continuar, pondremos un ejemplo sencillo para fijar ideas.

Ejemplo 1 *Supongamos que tenemos dos puntos* $(x_0, 0)$ *y* $(x_1, 1)$ *del plano. Notemos que su polinomio interpolador será un polinomio de grado 1. Ahora nos preguntamos que pasará si calculamos los*

coeficientes del polinomio interpolador y utilizamos una matriz de cambio de base para convertirlos en puntos de control.

La curva de Bézier con esos puntos de control también interpolará esos dos puntos originales (ya que las curvas de Bézier se anclan en el primer y el último punto). Es decir, por la unicidad del polinomio interpolador, la curva de Bézier y el polinomio coincidirán. Podemos obtener el polinomio interpolador mediante el método de Lagrange o el de Newton y viene dado por la siguiente expresión:

$$p(t) = x_0 + (x_1 - x_0)t.$$

El polinomio p se puede expresar como vector del espacio de polinomios de grado menor o igual que uno como $v_p = (x_0, x_1 - x_0)$. *Los polinomios de Bernstein de grado 1 son* $B_0^1(t) = 1 - t$ *y* $B_1^1(t) = t$. *Es fácil ver que se verifica:*

$$\begin{pmatrix} B_0^1(t) \\ B_1^1(t) \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \end{pmatrix}.$$

Llamemos A a la matriz de cambio de la ecuación anterior. Según razonamientos previos, si queremos generar los puntos de control correspondintes a la curva de Bézier que interpola los puntos $(x_0, 0)$ *y* $(x_1, 1)$, *tenemos que multiplicar el vector* v_p *por la matriz A, esto es* $c^t = v_p^t M^{-1}$. *El vector c, será el vector de puntos de control. En este caso, el cálculo es muy sencillo, tenemos* $c = (x_0, x_1)$. *Esto es, tal y como esperábamos, recuperamos los puntos de interpolación.*

Ejercicio 2 *Demuestra que la matriz de cambio de base entre la base canónica del espacio de polinomios de grado k o inferior y los polinomios de Bernstein de grado k es la siguiente:*

$$M_{i,j} = \begin{cases} \binom{k}{i}\binom{k-i}{j-i}(-1)^{j-1} \text{ si } j \leq i \\ 0 \text{ en otro caso.} \end{cases}$$

4. B-splines

Los *B*-splines combinan ideas de la construcción de las curvas de Bézier y de los splines para obtener interpolaciones que solventan los problemas mencionados en otros métodos. En este sentido, Los B-splines se construyen de una manera similar a las curvas de Bézier, esto es, se escriben como una combinación lineal de una base de polinomios. Un B-spline es una curva de la forma

$$p(t) = \sum_{i=0}^{n} c_i N_i^k(t). \tag{2}$$

Donde N_i^k es una base de polinomios de grado k con soporte finito i.e. se anulan en toda la recta real salvo en un intervalo finito. Las funciones base se construyen a partir de una malla de nodos. Cada una de las funciones base, será distinta de cero solamente en el intervalo definido por dos nodos.

Dada una malla $\{a_0, \ldots, a_m\}$, cont $m = n + k + 1$, definimos la base $\{N_i^k\}_i$ mediante la siguiente recursión:

$$N_i^0 = \begin{cases} 1 \text{ si } t \in [a_i, a_{i+1}), \\ 0 \text{ en otro caso.} \end{cases}$$

$$N_i^k(t) = \alpha_i^{k-1}(t) N_i^{k-1}(t) + (1 - \alpha_{i+1}^{k-1}(t)) N_{i+1}^{k-1}(t),$$

donde,

$$\alpha_i^{k-1}(t) = \frac{t - a_i}{a_{i+k} - a_i}.$$

La base $\{N_i^k\}_i$ verifica las siguientes propiedades:

- N_i^k es una función polinomica a trozos,
- $N_i^k = 0$ si $t \in [a_i, a_{i+n+1})$.

A diferencia de la base de polinomios de Bernstein, la base de los B-splines depende de la malla y no tiene, en general, una expresión analítica. Si la tiene en casos particulares de mallas sencillas. Por ejemplo, si tomamos la malla definida por $\{0, 1\}$, entonces

$$N_i^k(t) = B_i^k(t),$$

esto es, las bases de los B-splines son una generalización de los polinomios de Bernstein.

4.1. El algoritmo de de Boor

Consideremos un B-spline generico dado por una combinación lineal de una base de B-splines:

$$p(t) = \sum_{i=0}^{k} c_i^0 N_i^k(t).$$

Con la base $\{N_i\}$ definida sobre una cierta malla $\{a_i\}$. Utilizando la recurrencia que usamos en la definición de los N_i's, podemos obtener:

$$\begin{aligned} p(t) &= \sum_{i=1}^{k} c_i^1 N_i^{k-1}(t), \\ &\vdots \\ p(t) &= \sum_{i=k}^{k} c_i^k N_i^{k-1}(t) = c_k^k, \end{aligned}$$

donde,

$$c_i^r = (1-\alpha)c_{i-1}^{r-1} + \alpha c_i^{r-1}, \qquad \alpha = \frac{t - a_i}{a_{i+k-1-r} - a_i}.$$

Podemos montar la siguiente tabla triangular:

$$\begin{array}{ccccc} c_0^0 & & & & \\ c_1^0 & c_1^1 & & & \\ c_2^0 & c_2^1 & c_2^2 & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_k^0 & c_1^k & c_2^k & \dots & c_k^k. \end{array}$$

Para generar elementos no nulos de esta tabla, hay que multiplicar por α para obtener el elemento de la derecha y por $1 - \alpha$ para obtener el elemento de la siguiente columna en la fila de abajo (la diagonal). De esta manera, podemos evaluar un B-spline en t sin necesidad de calcular, numéricamente, los elementos de la base.
Si aplicamos el algoritmo de de Boor a los puntos de control $c_0^0, \dots, c_k^0$, obtenemos una nueva combinación lineal

$$p_k(t) \sum_{i=0}^{k} c_i^k N_i^k(t),$$

que cumple $p_k(t) = p(t)$ si $t \in [a_k, a_{k+1}]$.

4.2. Formas polares

Dado un polinomio p de grado k, existe un único polinomio multivariante en las variables $t_1, t_2, \ldots, t_k$ y que denotaremos por $p[t_1, \ldots t_k]$ tal que:

1. $p[t_1, \ldots t_k]$ es un polinomio simétrico, esto es, para cualquier permutación σ de los índices, se tiene
$$p[t_{\sigma(1)}, \ldots t_{\sigma(k)}] = p[t_1, \ldots t_k],$$
2. $p[t, \ldots, t] = p(t)$,
3. $p[t_1, \ldots t_k]$ es afín:
$$p[t_1, \ldots, \alpha t + (1-\alpha)\tilde{t}, \ldots, t_k] =$$
$$\alpha p[t_1, \ldots, t, \ldots, t_k] + (1-\alpha)p[t_1, \ldots, \tilde{t}, \ldots, t_k].$$

El polinomio simétrico $p[t_1, \ldots t_k]$ se denomina la forma polar de p.

Recordemos que el algoritmo de de Boor calcula un nuevo B-spline p_k a partir de los puntos de control que coincide con el B-spline original p en el intervalo $[a_k, a_{k+1}]$. Este nuevo B-spline, además sirve para rescatar los puntos de control originales si se expresa en su forma polar, esto es

$$c_i^0 = p_j[t_{i+1}, \ldots, t_{i+k}] \quad i = j-k, \ldots, j.$$

La implementación del algoritmo de de Boor usando la forma polar es más eficiente que con coordenadas cartesianas. En particular si m de las variables $t_1, \ldots, t_k$ coinciden con los nodos, se requieren sólo $k-m$ pasos recursivos para evaluar $s[t_1, \ldots, t_k]$.