Extra para ingenieros

Ya hemos visto que gran parte del interés del álgebra lineal es que a pequeña escala, cuando se estudian pequeñas variaciones, casi todo es aproximadamente una aplicación lineal. Sin embargo, hay también muchas situaciones en que combinaciones lineales, bases o aplicaciones lineales aparecen de forman natural sin mediar aproximaciones. Es de ejemplos de este tipo de los que nos ocuparemos aquí.

Curvas de Bézier. Los puntos del plano los podemos identificar con vectores de $\mathbb{R}^2$ y tiene sentido considerar combinaciones lineales de ellos. El ingeniero P. Bézier utilizó en los años 60 del siglo XX un tipo de combinación lineal variable mientras trabajaba en el diseño de carrocerías de automóviles. Con el desarrollo de la informática, su idea se ha convertido en un referente para que los usuarios tracen curvas libres de manera interactiva en una pantalla.

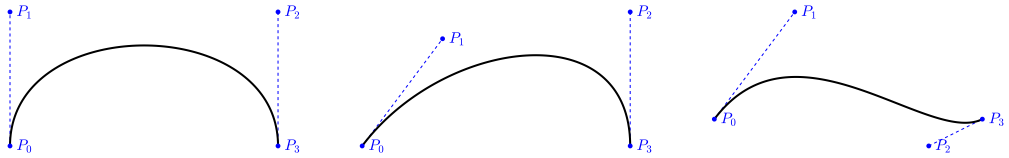
La *curva de Bézier* más empleada en el mundo de los gráficos por ordenador es la asociada a cuatro puntos P_0, P_1, P_2 y P_3, que suponemos distintos. Viene dada por

$$\sigma(t) = \lambda_0(t)P_0 + \lambda_1(t)P_1 + \lambda_2(t)P_2 + \lambda_3(t)P_3$$

con

$$\lambda_0(t) = (1-t)^3, \quad \lambda_1(t) = 3t(1-t), \quad \lambda_2(t) = 3t^2(1-t)^2, \quad \lambda_3(t) = t^3.$$

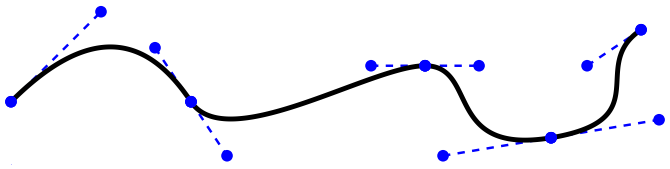
Cuando t varía en $[0,1]$, el resultado de $\sigma(t)$ produce puntos que describen una curva. Es evidente que $\sigma(0) = P_0$ y $\sigma(1) = P_3$ por tanto tal curva conecta P_0 y P_3. En general no pasa por P_1 y P_2, que se dice que son puntos de control. Dos propiedades interesantes son que la curva siempre está contenida en el cuadrilátero formado por los cuatro puntos y que siempre los segmentos P_0P_1 y P_2P_3 son tangentes a ella. Estas dos propiedades geométricas hacen que sea muy intuitivo modificar la curva de manera interactiva "tirando" de los puntos de control. Cuando los movemos, la curva se desplaza hacia ellos.



Ciertamente, no parece dar mucho juego trazar una curva que pase por dos puntos, por mucha flexibilidad que tengamos. Lo que se hace en muchos programas que permiten el trazado de curvas (como GIMP o Photoshop) es que el usuario pueda concatenar varias curvas de Bézier como las anteriores de manera que cada punto por el que pasa la curva sea el punto medio de dos puntos de control. La mecánica habitual es que uno señala con el ratón un nuevo punto de la curva y lo arrastra para definir un punto de control y otro simétrico. Normalmente, para que todo sea

más visual, mientras se está arrastrando se muestra el segmento formado por los dos puntos de control que será tangente a la curva resultante.

Sin duda, una imagen no puede reflejar la sensación de versatilidad que transmite utilizar una aplicación interactiva de este tipo. De todas formas, aquí va una ilustración del tipo de objetos que se consideran:



Hay también curvas de Bézier que añaden más de dos puntos de control, o equivalentemente combinaciones lineales más largas. Son más difíciles de interpretar geométricamente y por tanto menos usadas en la práctica.

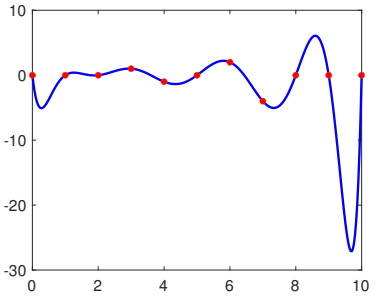
B-splines. Ya habíamos visto la interpolación cúbica con *splines* para conectar con una gráfica los puntos $(0, y_0), \ldots, (0, y_n)$ pero eso requería resolver un sistema de ecuaciones lineales. Las curvas de Bézier son más rápidas pero tienen cierta indeterminación, deseable en sus aplicaciones, que permite al usuario hacer ajustes mediante los puntos de control. Es posible conseguir de manera sencilla, utilizando combinaciones lineales, una curva polinómica automática que conecta los puntos anteriores. Para ello, se definen los *polinomios de Lagrange*:

$$L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x-k}{j-k} \qquad \text{con} \quad j = 0, 1, \ldots, n$$

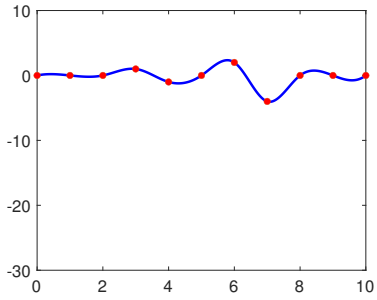
que cumplen $L_j(j) = 1$ y $L_j(l) = 0$ si $l \neq j$, $0 \leq l \leq n$. Por tanto el polinomio $P \in \mathbb{R}_n[x]$ definido por la combinación lineal

$$P(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x)$$

cumple $P(j) = y_j$, como deseábamos. El problema es que en la práctica muchas veces este polinomio muestra grandes oscilaciones. Por ejemplo, para $n = 10$ con $y_3 = 1$, $y_4 = -1$, $y_6 = 2$, $y_7 = -4$ y el resto de los y_j nulos, se obtiene la primera figura mientras que los *splines* dan el resultado de la segunda.



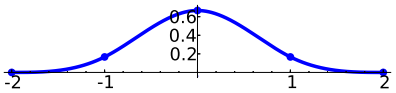
Polinomios de Lagrange



Splines cúbicos

Sea $b = b(x)$ la función polinómica a trozos que es par, esto es, $b(x) = b(-x)$ y que para $x \geq 0$ viene definida por

$$b(x) = \begin{cases} \frac{2}{3} - x^2 + \frac{1}{2}x^3 & \text{si } 0 \leq x < 1 \\ \frac{1}{6}(2-x)^3 & \text{si } 1 \leq x < 2 \\ 0 & \text{si } x \geq 2 \end{cases}$$



Gráfica de b

Es un sencillo ejercicio comprobar que b tiene dos derivadas continuas. Se dice que b es un *B-spline* y la "B" es de base. Más adelante veremos la razón para ello.

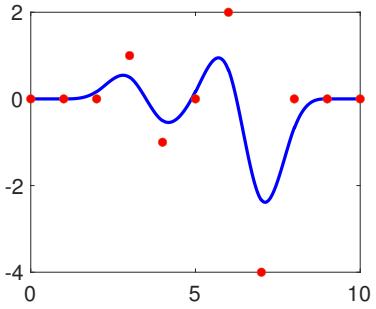
Consideremos la combinación lineal de los trasladados de b, es decir,

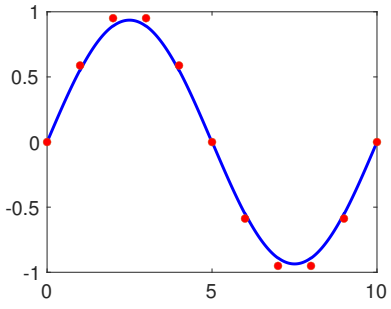
$$f(x) = y_0 b(x-0) + y_1 b(x-1) + \cdots + y_n b(x-n).$$

Esta f no pasará en general por los puntos (j, y_j) pero

$$f(j) = \frac{1}{6}(y_{j-1} + 4y_j + y_{j+1}) \qquad \text{para} \quad 0 < j < n$$

implica que si los y_j presentan pocas variaciones, se tendrá $f(j) \approx y_j$. Las siguientes figuras muestran el resultado para $n = 10$ cuando usamos los y_j empleados antes con los polinomios de Lagrange y con y_j más regulares que corresponden a valores de la función $\text{sen}(\pi x/5)$.





Está claro que en el segundo caso tenemos una aproximación bastante buena a unir los puntos y esa es la mayor fuente de interés de los *B-splines*. Con un número muy reducido de operaciones y sin resolver ningún sistema, logramos una solución aproximada del problema de interpolación. Una de las aplicaciones comunes es redimensionar una imagen. Si por ejemplo la agrandamos multiplicando por cuatro sus altura y anchura, entonces el píxel (x, y) lo podemos enviar a $(4x, 4y)$ pero todos los píxeles que no tengan coordenadas múltiplos de cuatro nos lo tenemos que inventar. Utilizar una interpolación exacta con *splines cúbicos* sería muy costoso porque hoy en día las fotos digitales típicas cuentan con millones de píxeles.

El *B-spline* anterior sirve para construir una base de los *splines cúbicos*, en el sentido de que todo spline cúbico es de la forma

$$s(x) = \lambda_0 b(x-0) + \lambda_1 b(x-1) + \cdots + \lambda_n b(x-n).$$

y la interpolación con ellos que habíamos visto en una sección anterior consiste en elegir los coeficientes λ_j de la combinación lineal de forma que se cumpla $s(j) = y_j$. Además se impone cierta condición más en los extremos ($s''(0) = s''(n) = 0$) porque si no el sistema de ecuaciones lineales resultante sería compatible indeterminado.

Detección y corrección de errores. Los espacios vectoriales y las aplicaciones lineales son muy relevantes en algunos algoritmos que dan mayor fiabilidad a la transmisión de información digital detectando y corrigiendo errores. El contexto es diferente al estudiado en el curso pues el cuerpo K subyacente no es $\mathbb{R}$ o $\mathbb{C}$ sino que solo cuenta con un número finito de elementos. Consideraremos únicamente el caso más sencillo $K = \{0, 1\}$ con la suma y el producto habituales excepto que decretamos $1 + 1 = 0$. Este cuerpo representa los posibles valores de un bit. El esquema habitual pasa por fijar una aplicación lineal $f : K^n \longrightarrow K^m$ del tipo

$$f(\vec{x}) = H\vec{x} \qquad \text{con} \quad H \in \mathcal{M}_{m\times n}(K) \quad \text{y} \quad \operatorname{rg}(H) = m < n.$$

A la matriz H se le llama *matriz de paridad*. Bajo estas condiciones, se tiene

$$\dim \operatorname{Ker}(f) = n - \operatorname{rg}(H) = n - m.$$

Si nuestra información está codificada como una tira de n bits que dan las coordenadas de un vector $\vec{x} \in K^n$ perteneciente a $\operatorname{Ker}(f)$, entonces $f(\vec{x}) = \vec{0}$. Supongamos que transmitimos $\vec{x}$ y se recibe $\vec{x}'$. Si $f(\vec{x}') \neq \vec{0}$ podemos asegurar que se ha producido un error, es decir, $\vec{x} \neq \vec{x}'$. Cuando la dimensión $n - m$ de $\operatorname{Ker}(f)$ es suficientemente pequeña en comparación con la del espacio ambiente K^n, lo común es que al cometer errores nos salgamos de $\operatorname{Ker}(f)$ y por eso un valor nulo de $f(\vec{x}')$ da fuertes indicios de que no hay errores. Por otro lado, $n - m$ pequeño en comparación con n hace que la codificación sea poco eficiente pues gastamos n coordenadas (bits) para algo que en realidad solo tiene $n - m$ grados de libertad. Hay que buscar un balance entre ambas situaciones.

El caso más simple, e históricamente uno de los primeros, consiste en añadir a la información que tengamos un bit extra artificial para que el número de unos sea par. Con la notación anterior, tenemos

$$H\vec{x} = 0 \qquad \text{donde} \quad H = (1, 1, \ldots, 1) \in \mathcal{M}_{1\times n}(K).$$

Si lo que se recibe es un $\vec{x}'$ con un número impar de unos, o equivalentemente $H\vec{x}' \neq 0$, seguro que se ha cometido un error. De este ejemplo proviene llamar matriz de *paridad* a H. Te sorprenderías al saber la cantidad de variantes, un poco más elaboradas, de esta idea que aparecen en la vida cotidiana, por ejemplo la lectura de los códigos de barras que ahora llevan casi todo los artículos de consumo, la composición de los números de cuentas bancarias y hasta la asignación de la letra del DNI.

Con matrices más complicadas que la del ejemplo anterior, podemos llegar no solo a detectar errores con mayor fiabilidad sino también a corregir los más básicos.

Veamos un ejemplo que en cierta manera es óptimo. Corresponde a la matriz

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \in \mathcal{M}_{3\times 7}(K).$$

Si leemos las columnas como si fueran números binarios de tres bits, obtendremos en cada una el número de columna. Por ejemplo, la sexta columna corresponde al número binario 110 que es $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 6$. La dimensión del núcleo de $f(\vec{x}) = H\vec{x}$ es $n - m = 4$ y $\mathcal{B} = \{\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4\}$ es una base donde

$$\vec{v}_1 = (1,1,1,0,0,0,0)^t, \qquad \vec{v}_3 = (0,1,0,1,0,1,0)^t,$$
$$\vec{v}_2 = (1,0,0,1,1,0,0)^t, \qquad \vec{v}_4 = (1,1,0,1,0,0,1)^t.$$

Para codificar información digital la dividimos en trozos de cuatro bits $b_1b_2b_3b_4$ y enviamos $\vec{x} = b_1\vec{v}_1 + b_2\vec{v}_2 + b_3\vec{v}_3 + b_4\vec{v}_4$, de esta forma $H\vec{x} = \vec{0}$. Si se produjera un solo error en la transmisión de $\vec{x}$, digamos en la coordenada j, se tendría $\vec{x}\,' = \vec{x} + \vec{e}_j$ con $\vec{e}_j$ el j-ésimo vector de la base canónica, el que tiene un 1 en la coordenada j y ceros en el resto (recuérdese que hemos decretado $1 + 1 = 0$). Por tanto

$$H\vec{x}\,' = H\vec{x} + H\vec{e}_j = H\vec{e}_j = \text{columna } j \text{ de } H.$$

En definitiva, al leer $H\vec{x}\,'$ como si fuera un número binario tendremos el lugar donde se ha producido al error y lo podremos corregir o, mejor todavía, que se corrija automáticamente.

Por ser más concretos, $b_1b_2b_3b_4 = 0011$ quedará codificado como $\vec{x} = \vec{v}_3 + \vec{v}_4 = (1,0,0,0,0,1,1)^t$. Si al transmitirlo se produce un error en la tercera coordenada, el receptor obtendrá $\vec{x}\,' = (1,0,1,0,0,1,1)^t$ y, sin información adicional, podrá corregirlo ya que $H\vec{x}\,' = (0,1,1)^t$ y 011 es 3 en binario.