

Figura 11.7: Interpolación con splines cúbicos naturales

Tenemos así dos problemas de interpolación desacoplados, esto es, cada uno se resuelve independiente del otro. Queda como cuestión pendiente la elección de T en (11.12) y de los puntos t_j en (11.13). Una buena elección es tomar

$$h_j := \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2},$$

que coincide con la longitud del segmento que une (x_j, y_j) y (x_{j+1}, y_{j+1}) , y hacer luego

$$t_0 = 0, \quad t_k := h_0 + h_1 + \dots + h_{k-1}, \quad T = t_n.$$

Una vez ya calculados, la curva se dibuja evaluando

$$\mathbf{x}(t) := (cx(t), cy(t)), \quad t \in [0, T].$$

Implementa un programa que siga este prototipo

```
01  SPLINE2D
02
03  SPLINE2D(X,Y) Traza una curva en el plano que pasa por
04      (X(i),Y(i)) utilizando splines cubicos
05
```

Ejercicio 11.15 El comando `ginput` permite introducir los puntos a través del ratón. Lee bien la ayuda y modifica el programa anterior para que haga posible la lectura de los datos con el ratón en caso de que se llame a la función sin argumentos.

Entre las posibilidades que ofrece está la de controlar qué tecla o botón del ratón se ha utilizado.

Ayuda. Deduce qué realiza el siguiente código...

```
x0=[]; y0=[]; n=length(x0);
salir=0;
cla
xlim([0,1]); ylim([0,1])
while salir==0
    [xi,yi,b]=ginput(1);
    if b==3                % boton derecho del raton
        cla
        if n>0            % borramos el ultimo punto
            x0(n)=[]; y0(n)=[];
            plot(x0,y0,'ro')
            xlim([0,1]); ylim([0,1])
            n=n-1;
        end
    elseif isempty(b) % Se ha pulsado return: finalizamos lectura
        salir=1;
    elseif b==2        % boton central: finalizamos la lectura
        salir=1
    else                % introducimos el punto
        x0=[x0 xi]; y0=[y0 yi];
        plot(x0,y0,'o')
        n=n+1;
        axis([0 1 0 1])
    end
end
end
```

Nota. El comando `gtext` es similar a `ginput`. Despliega un texto en el punto seleccionado con el ratón. □

11.3. Curvas Bézier

Las curvas Bézier son curvas en el plano que quedan determinadas por un conjunto de puntos que marcan su recorrido, aunque estas curvas no pasan necesariamente por todos los puntos. Éstos forman un *polígono de control* en el siguiente sentido: la curva está contenida en el polígono formado por esos puntos e imita con cierta libertad la poligonal que dibujan.

La curva se construye utilizando los polinomios de Bernstein de orden n :

$$B_j^n(t) = \binom{n}{j} t^j (1-t)^{n-j}, \quad j = 0, 1, \dots, n.$$

Es un simple ejercicio comprobar que los polinomios cumplen las siguientes propiedades:

- $\{B_j^n\}_{j=0}^n$ son una base de \mathbb{P}_n . Dicho de otra forma, cualquier polinomio se puede escribir como una combinación de estos polinomios.

$$\blacksquare \sum_{j=0}^n B_j^n(t) = 1, \quad \forall t \in \mathbb{R}.$$

A partir de unos puntos

$$\mathbf{x}_i = (x_i, y_i), \quad \text{con } i = 0, \dots, n,$$

se construye ahora la curva de Bézier utilizando estos polinomios:

$$S(t) := \sum_{j=0}^n B_j^n(t) \mathbf{x}_j = \left(\sum_{j=0}^n B_j^n(t) x_j, \sum_{j=0}^n B_j^n(t) y_j \right), \quad t \in [0, 1].$$

El parámetro t se mueve de forma que para $t = 0$ estamos en el punto inicial y para $t = 1$, en el final. Nótese que los puntos utilizados para dibujar la curva **están ordenados**: hay un punto inicial \mathbf{x}_0 , uno final \mathbf{x}_n y a \mathbf{x}_j le sigue \mathbf{x}_{j+1} ...

Podemos comprobar (véase la Figura 11.8), que la curva se *adapta* a la forma que marcan los puntos. Sin embargo, únicamente podemos garantizar que esta curva pasará por el primer y último punto ya que $S(0) = \mathbf{x}_0$ y $S(1) = \mathbf{x}_n$ pero en general $S(t) \neq x_j$ para $j \neq 0, n$.

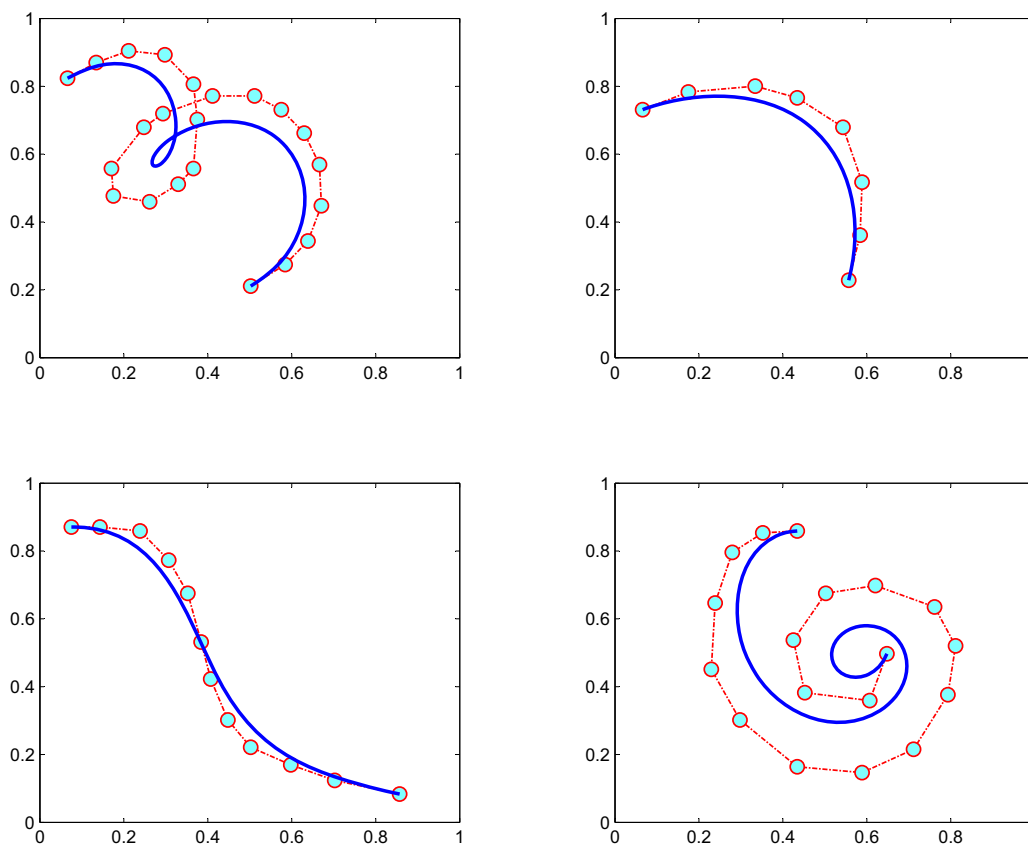


Figura 11.8: Curvas Bézier para algunas configuraciones de puntos.

Ejercicio 11.16 *Utilizando el código mostrado en el Ejercicio 11.15, programa las curvas Bézier. Compara con las curvas que definen la interpolación por splines y la polinómica.*

Ejercicio 11.17 *Implementa la siguiente extensión del programa anterior: permitir que el usuario seleccione puntos y pueda moverlos o borrarlos (por ejemplo, con el botón derecho del ratón) y así comprobar la sensibilidad de la curva al polígono de control.*

Una sugerencia: una vez leídos los puntos, y dibujada la correspondiente curva nos situamos

Escogeremos entonces j para que $|\cos(\theta_j)|$ sea mínimo, y construiremos la curva Bézier con puntos

$$\{\mathbf{x}_1, \dots, \mathbf{x}_j, \mathbf{y}, \mathbf{x}_{j+1}, \dots, \mathbf{x}_n\}$$

$$\{\mathbf{x}_j, \mathbf{x}_{j+1}\}.$$

Nota. Las curvas Bézier fueron introducidas por los ingenieros Pierre Bézier y Paul de Casteljaou, que trabajaban entonces en Renault y Citroën. Concretamente, buscaban herramientas para el diseño de parachoques de forma que la curva que diera su forma fuera controlada por una serie de puntos y que tuviera buenas propiedades geométricas. Los algoritmos iniciales (todavía se utilizan en la práctica hoy en día) seguían ideas muy geométricas¹⁶, y se tardó algún tiempo en darle la forma que hemos mostrado, más matemática. Como resultado de este estudio surgieron las curvas B-spline, que reemplazaban a los polinomios de Bernstein por splines de grados adecuados. Las curvas B-spline son más flexibles que las curvas Bézier y en su caso límite incluyen a éstas. Por lo tanto pueden interpretarse como una generalización de las mismas.

Por otro lado existen formas mucho más eficientes de evaluar y calcular este tipo de curvas que permiten en última media su implementación en una forma más interactiva, de forma que el usuario mueva los puntos (cambie el *polígono de control*) y que la curva se redibuje constantemente. \square

¹⁶La versión original ni siquiera hablaba de polinomios.

Bibliografía

- [1] Atkinson, K. (1993): *Elementary Numerical Analysis*. Wiley.
- [2] Burden, R.L. y Faires, J.D. (1998): *Análisis Numérico*. International THOMSON.
- [3] Chapman, S.J. (1997): *Fortran 90/95 for Scientists and Engineers*. McGraw-Hill.
- [4] Chapman, S.J. (2002): *Matlab Programming for Engineers*. Books/Cole.
- [5] Cooper, J. (2001): *A Matlab Companion for Multivariate Calculus*. Academic Press.
- [6] Eriksson, K.E., Step, D. y Johnson, C. (2004): *Applied Mathematics: body and soul*. Springer Verlag.
- [7] García de Jalón, J. (2004): *Aprenda Matlab 6.5 como si estuviera en primero*. Publicación electrónica, disponible en <http://mat21.etsii.upm.es/ayudainf/aprendainf/varios.htm>
- [8] Gander, W. y Hřebíček, J. (2004): *Solving problems in Scientific Computing using Maple and Matlab*. Springer-Verlag.
- [9] Golub, G.H. y van Loan, C.F. (1989): *Matrix Computations*. University Press.
- [10] Golub, G.H. y van der Vorst, H. (2000): “Eigenvalue computation in the 20th century” en J. Comput. Appl. Math. **123** (2000), 35–65.
- [11] Greenbaum, A. (1997): *Iterative Methods for Linear Systems*. SIAM.
- [12] Hänmerlin, G. y Hoffmann, K.H. (1991): *Numerical Mathematics*. Springer-Verlag.
- [13] Hanselman, D. y Littlefield, B.R. (2000): *Mastering Matlab 6*. Prentice Hall.
- [14] Higham, D.J. y Higham, N.J. (2005): *Matlab guide*. SIAM.
- [15] Hoffman, J., Johnson, C. y Logg, A. (2004): *Dreams of Calculus*. Springer.
- [16] Infante, J.A. y Rey, J.M. (2002): *Métodos numéricos. Teoría, problemas y prácticas con Matlab*. Anaya.
- [17] Kharab, A. y Guenther, R.B. (2001): *An Introduction to Numerical Methods: A MATLAB Approach*. Chapman & Hall.
- [18] Kincaid, D. y Cheney, W. (1994): *Análisis numérico. Las matemáticas del cálculo científico*. Addison-Wesley Iberoamericana.