Patch Kessler

February 23, 2006

Natural Cubic Spline Interpolation

We start with $n+1$ points (x_i, y_i) in the plane, with none of the x_i's the same[1]. We will construct a cubic function between each pair of adjacent points (as shown in fig 1 on the next page). Let $f_i(x)$ denote the function between point i and point $i+1$. In order for $f_i(x)$ to pass through (x_i, y_i) and (x_{i+1}, y_{i+1}) we must have

$$\begin{aligned} f_i(x_i) &= y_i \\ f_i(x_{i+1}) &= y_{i+1} \end{aligned} \tag{1}$$

for $i = 1, \ldots, n$. We require in addition that adjacent functions have the same slopes (first derivatives) and curvatures (second derivatives) at the point where they meet. This can be expressed as

$$\begin{aligned} f_i'(x_{i+1}) &= f_{i+1}'(x_{i+1}) \\ f_i''(x_{i+1}) &= f_{i+1}''(x_{i+1}) \end{aligned} \tag{2}$$

for $i = 1, \ldots, n$. It is convenient to express each of the n different functions $f_i(x)$ as

$$f_i(x) = \alpha_i(x - x_i)^3 + \beta_i(x - x_i)^2 + \gamma_i(x - x_i) + y_i \tag{3}$$

because this way, requirement $(1)_1$ is automatically satisfied. Our goal is to find the constants α_i, β_i, and γ_i. Assuming that $x_{i+1} - x_i = 1$, and expanding $f_i(x)$, equations $(1)_2$ and $(2)_{1,2}$ become

$$\begin{aligned} 3\alpha_i + 3\beta_i + 3\gamma_i &= 3(y_{i+1} - y_i) \\ 3\alpha_i + 2\beta_i + \gamma_i &= \gamma_{i+1} \\ 3\alpha_i + \beta_i &= \beta_{i+1} \end{aligned} \tag{4}$$

for $i = 1, \ldots, n-1$. We can eliminate α_i by substituting the last equation into the first two, obtaining

$$\begin{aligned} \beta_{i+1} + 2\beta_i + 3\gamma_i &= 3(y_{i+1} - y_i) \\ \beta_{i+1} + \beta_i + \gamma_i &= \gamma_{i+1} \end{aligned} \tag{5}$$

for $i = 1, \ldots, n-1$. At the next value of i, $(5)_1$ is given by $\beta_{i+2} + 2\beta_{i+1} + 3\gamma_{i+1} = 3(y_{i+2} - y_{i+1})$. Combining this with (5) eliminates the γ_i's, giving

$$\beta_i + 4\beta_{i+1} + \beta_{i+2} = 3(y_i - 2y_{i+1} + y_{i+2}) \tag{6}$$

for $i = 1, \ldots, n-2$. Now for the endpoints. Combining $f_n(x_{n+1}) = y_{n+1}$ and $f_n''(x_{n+1}) = 0$ gives

$$2\beta_n + 3\gamma_n = 3(y_{n+1} - y_n) \tag{7}$$

Combining this in turn with $(5)_{1,2}$ with $i = n-1$ gives

$$\beta_{n-1} + 4\beta_n = 3(y_{n-1} - 2y_n + y_{n+1}) \tag{8}$$

Requiring that $f_1''(x_1) = 0$ gives us $\beta_1 = 0$, and we have enough equations to solve for the remaining β_i's. In matrix form these are given by

$$\begin{bmatrix} 4 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 4 \end{bmatrix} \begin{bmatrix} \beta_2 \\ \beta_3 \\ \beta_4 \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix} = 3 \begin{bmatrix} \tilde{y}_2 \\ \tilde{y}_3 \\ \tilde{y}_4 \\ \vdots \\ \tilde{y}_{n-1} \\ \tilde{y}_n \end{bmatrix} \tag{9}$$

[1]The subscript i in an experession indicates that the statement being made is true for any i from a set of integers. This set is usually clear from the context, for instance we are given $n+1$ points in the plane and so the i in x_i can be any integer from 1 to $n+1$.

where we use $\tilde{y}_i$ to denote $y_{i-1} - 2y_i + y_{i+1}$. We now turn to the γ_i's. Note that γ_n follows immediately from (7)

$$\gamma_n = y_{n+1} - y_n - \frac{2}{3}\beta_n \tag{10}$$

and that $\gamma_{n-1}, \ldots, \gamma_1$ can then be computed one after another using $(5)_2$

$$\begin{aligned} \gamma_{n-1} &= \gamma_n - \beta_n - \beta_{n-1} \\ \gamma_{n-2} &= \gamma_{n-1} - \beta_{n-1} - \beta_{n-2} \\ &\vdots \\ \gamma_2 &= \gamma_3 - \beta_3 - \beta_2 \\ \gamma_1 &= \gamma_2 - \beta_2 - \beta_1 \end{aligned} \tag{11}$$

Our construction of the f_i's is completed by using $(4)_1$ to find the α_i's. Computing the value of y corresponding to some x on $[x_1, x_{n+1}]$ is a matter of determining which $f_i(x)$ to use, and then performing the computation in (3).

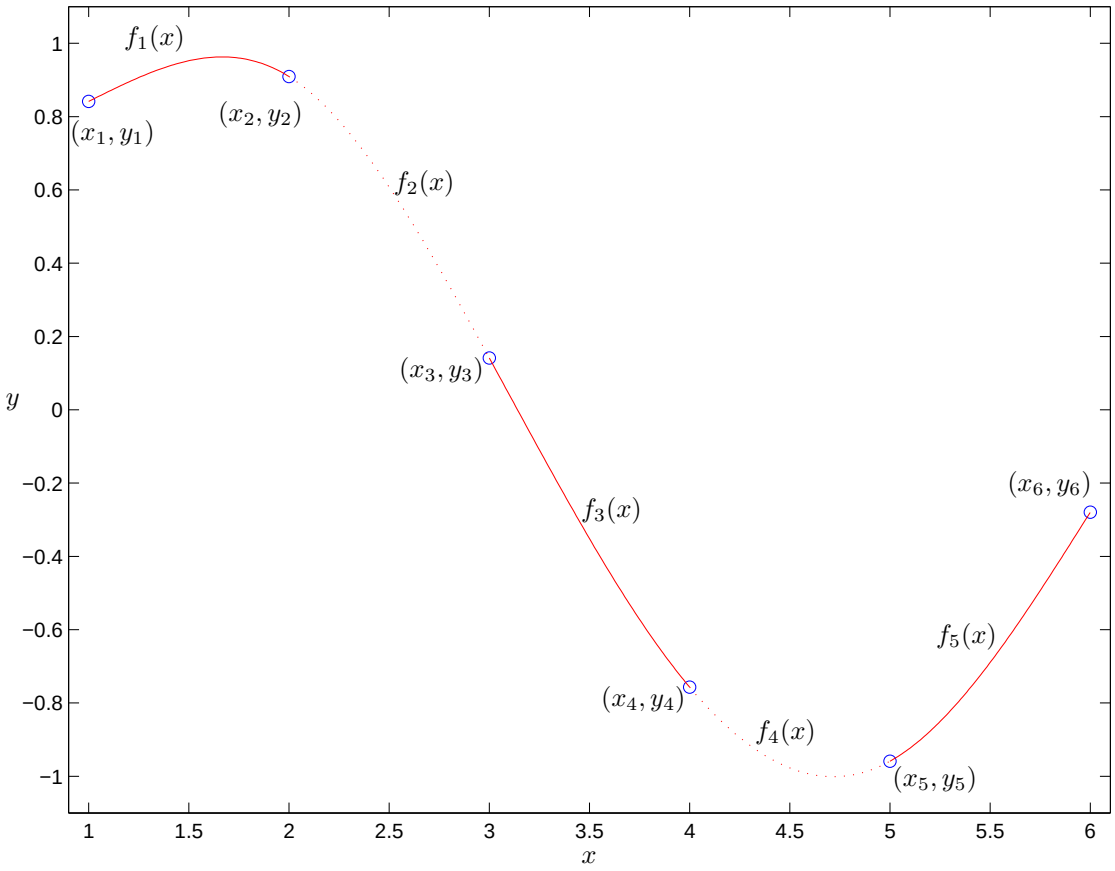

Figure 1: Example of Cubic Spline Interpolation using the equations from these notes. I used ten lines of code to generate the α_i's, β_i's and γ_i's, and another five lines of code to generate the output vector. I used a `for` loop to compute the γ_i's but didn't find loops necessary for anything else. I used the built-in functions `eye` and `diag` to construct the matrix from (9), however this involved the needless storage of lots of zeros. It would have been better to LU factorize this (tridiagonal) matrix and solve for the β_i's by forward and backward substitution.