Curve fitting: piecewise polynomial interpolation (splines)

We have seen that, unless n is quite small, trying to make a polynomial go through all the data points

$$(x_1, y_1),\ (x_2, y_2),\ (x_3, y_3),\ \ldots, (x_n, y_n)$$

can result in a very poor fit (i.e. the curve does a bad job of describing the trend in the data), even though the fitting error is nominally zero. This is due to *polynomial wiggle*, also known as *Runge's phenomenon*.

To get around this problem, while still requiring that the fitted curve go through all data points, we can instead fit a low-degree polynomial (called a *spline*) between each pair of consecutive data points. In other words, the fitted curve will be defined *piecewise* on the subintervals

$$x_1 \le x \le x_2, \quad x_2 \le x \le x_3 \quad \ldots \quad x_{n-2} \le x \le x_{n-1}, \quad x_{n-1} \le x \le x_n,$$

which means that it may have a different formula on each of the $n-1$ subintervals. The data points (x_i, y_i), where adjacent splines join up, are called "knots".

The simplest piecewise fit is obtained by connecting each pair of consecutive data points with a straight line, i.e. using a degree-1 polynomial on each subinterval. Indeed, this is what Matlab's `plot` command does by default with the arrays of x and y values that you give it. This *piecewise linear* fit is easy to implement but has the drawback that there will usually be "corners" at the data points, i.e. the gradients of adjacent line segments do not match, so the fitted "curve" is not smooth.

To get a smoother fit, we can use a quadratic or cubic polynomial on each subinterval (but generally not a polynomial of degree higher than three, because then the wiggle problem will arise again, and the extra coefficients would also make the calculations more complex).

Let $S(x)$ be the piecewise polynomial fit composed of $S_1(x), S_2(x), \ldots, S_{n-1}(x)$, where $S_k(x)$ denotes the spline on the kth subinterval $[x_k, x_{k+1}]$, for $k = 1, 2, \ldots, n-1$.
If each $S_k(x)$ is a cubic polynomial, with four coefficients, then we have a total of $4(n-1)$ coefficients to determine. We impose the following conditions:

1. $S_k(x_k) = y_k$ for $k = 1, \ldots, n-1$ and $S_{n-1}(x_n) = y_n$
   (the fitted curve $S(x)$ must go through all the data points: n conditions)

2. $S_k(x_{k+1}) = S_{k+1}(x_{k+1})$ for $k = 1, \ldots, n-2$
   (adjacent splines join up at the interior knots: $n-2$ conditions)

3. $S'_k(x_{k+1}) = S'_{k+1}(x_{k+1})$ for $k = 1, \ldots, n-2$
   (the gradients of adjacent splines match at the interior knots: $n-2$ conditions)

4. $S''_k(x_{k+1}) = S''_{k+1}(x_{k+1})$ for $k = 1, \ldots, n-2$
   (the second derivatives of adjacent splines match at the interior knots: $n-2$ conditions)

Conditions 3 and 4 are for ensuring smoothness of the fitted curve.
This makes a total of $4n-6$ conditions, whereas we have $4n-4$ unknown coefficients, so another two conditions are needed. There are a variety of choices, with the most common being:

- zero second derivative at the endpoints: $S''(x_1) = 0$ and $S''(x_n) = 0$ ("natural" boundary)
- gradient specified at the endpoints: $S'(x_1) = \gamma_{\text{left}}$ and $S'(x_n) = \gamma_{\text{right}}$ ("clamped" boundary)

- the third derivatives of adjacent splines match at the knots x_2 and x_{n-1} (this is equivalent to using a single cubic polynomial on the first two subintervals, $x_1 \le x \le x_3$, and a single cubic polynomial on the last two subintervals, $x_{n-2} \le x \le x_n$; so, in effect, x_2 and x_{n-1} are not knots, and this is called the "not-a-knot" condition)

For $k = 1, \ldots, n-1$, we can write each cubic polynomial $S_k(x)$ in the form

$$S_k(x) = a_k + b_k(x - x_k) + c_k(x - x_k)^2 + d_k(x - x_k)^3 \qquad \text{for } x_k \le x \le x_{k+1},$$

and hence $S_k'(x) =$

$S_k''(x) =$

Let $\boxed{h_k = x_{k+1} - x_k}$ denote the successive "step sizes" in the x-direction from one knot to the next.
Condition $\boxed{1}$ gives

Condition $\boxed{2}$ gives

Condition $\boxed{3}$ gives

Condition $\boxed{4}$ gives

Note that all the "a"s have already been determined from $\boxed{1}$.
From the equation of $\boxed{4}$, we express the "d"s in terms of the "c"s:

$$d_k =$$

Then $\boxed{3}$ becomes

and $\boxed{2}$ becomes

Solve for the "b"s in terms of the "a"s and "c"s:

$$b_k =$$

Note that $\dfrac{a_{k+1} - a_k}{h_k} = \dfrac{y_{k+1} - y_k}{x_{k+1} - x_k}$ is the gradient of the line segment joining the kth and the $(k+1)$st data points. Denote this gradient by g_k, i.e. define $\boxed{g_k = \dfrac{a_{k+1} - a_k}{h_k} = \dfrac{y_{k+1} - y_k}{x_{k+1} - x_k}}$.

Then we have

$$b_k =$$

$$b_{k+1} =$$

Substitute these expressions into the equation from $\boxed{3}$:

Multiply through by 3 and bring all "c"s to the left-hand side and all "g"s to the right-hand side of the equation:

Re-organise the left-hand side:

The index k can range from 1 to $n-3$, so we have a system of $n-3$ equations for the $n-1$ unknown "c"s. In matrix form $M\mathbf{c} = \boldsymbol{\beta}$:

$$\begin{pmatrix} & & & & & & & \\ h_1 & 2(h_1+h_2) & h_2 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 & 0 & \cdots & 0 \\ \vdots & \vdots & & & \ddots & & & \vdots \\ \vdots & \vdots & & & & \ddots & & \vdots \\ & & & & & & & 0 \\ 0 & 0 & 0 & \cdots & 0 & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ & & & & & & & \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \vdots \\ \vdots \\ c_{n-3} \\ c_{n-2} \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} \\ 3(g_2-g_1) \\ 3(g_3-g_2) \\ \vdots \\ \vdots \\ 3(g_{n-2}-g_{n-3}) \\ \\ \end{pmatrix}$$

M and $\boldsymbol{\beta}$ currently have only $n-3$ rows. The extra two conditions that we need in order to determine all $n-1$ of the "c" coefficients go in the top and bottom rows.

Once the c_k have been determined, we can then find the b_k and d_k using the formulas derived from conditions $\boxed{4}$ and $\boxed{2}$. Remember, however, that these formulas are valid only for $k = 1, \ldots, n-2$, so how do we find b_{n-1} and d_{n-1}?
We haven't yet used the rightmost equation in condition $\boxed{1}$, $S_{n-1}(x_n) = y_n$. This gives

Now apply the extra boundary conditions. For "natural" boundary conditions,

$$S''_{n-1}(x_n) = 0 \implies$$

and so the previous equation becomes