Accuracy of Interpolation and Splines

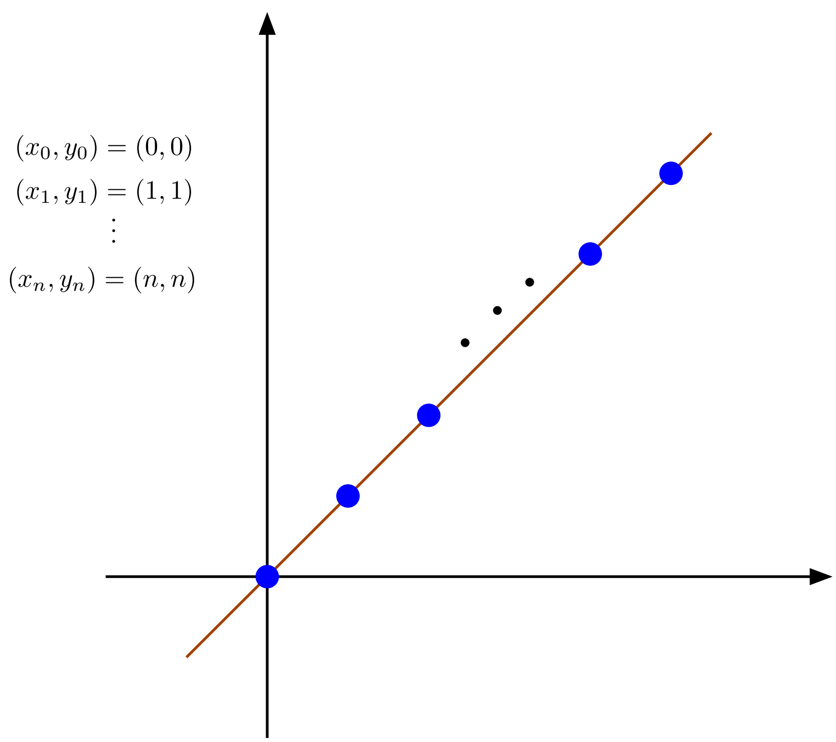
Mridul Aanjaneya

November 28, 2017

We saw three methods for polynomial interpolation (Vandermonde, Lagrange, Newton). It is important to understand that all three methods compute (in theory) the *same exact interpolant* $\mathcal{P}_n(x)$, just following different paths which may be better or worse from a computational perspective. The question however remains:

- How accurate is this interpolation, or in other words,
- How close is $\mathcal{P}_n(x)$ to the "real" function $f(x)$?

Example:



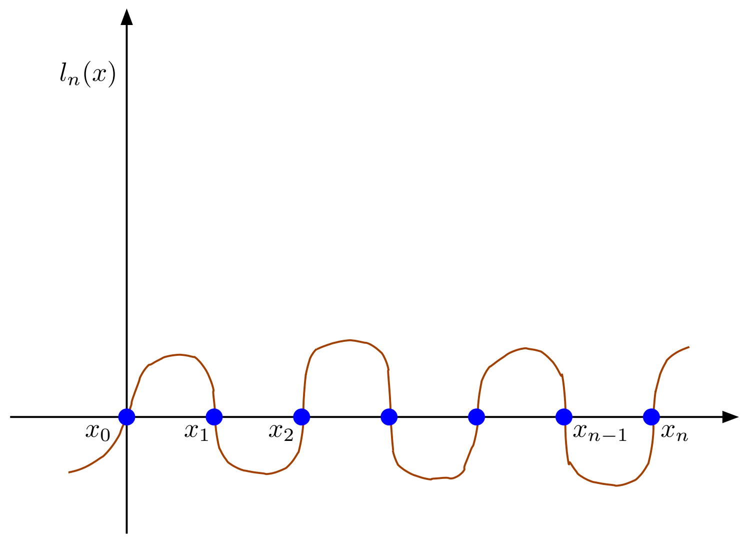
Using Lagrange polynomials $\mathcal{P}_n(x)$ $(= x)$ is written as

$$f(x) = \sum_{i=0}^{n} y_i l_i(x)$$

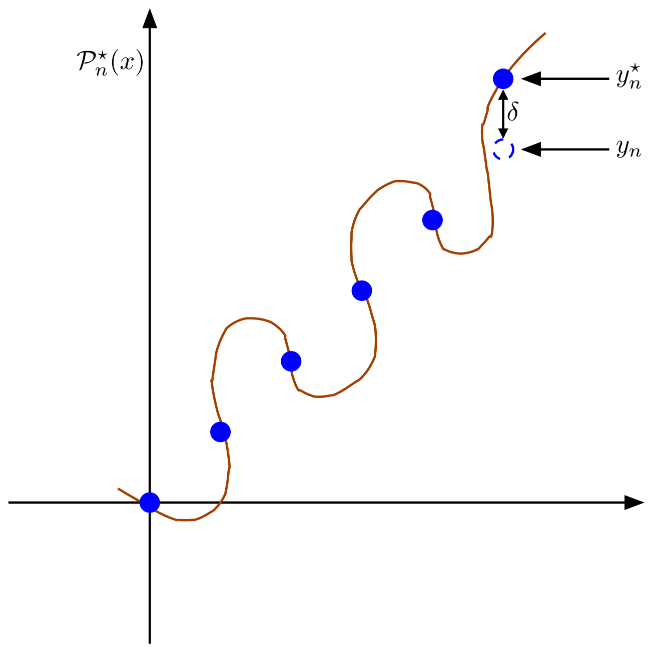
Let us "shift" y_n by a small amount δ. The new value is $y_n^\star = y_n + \delta$. The updated interpolant $\mathcal{P}_n^\star(x)$ then becomes:

$$\mathcal{P}_n^\star(x) = \sum_{i=0}^{n-1} y_i l_i(x) + y_n^\star l_n(x)$$

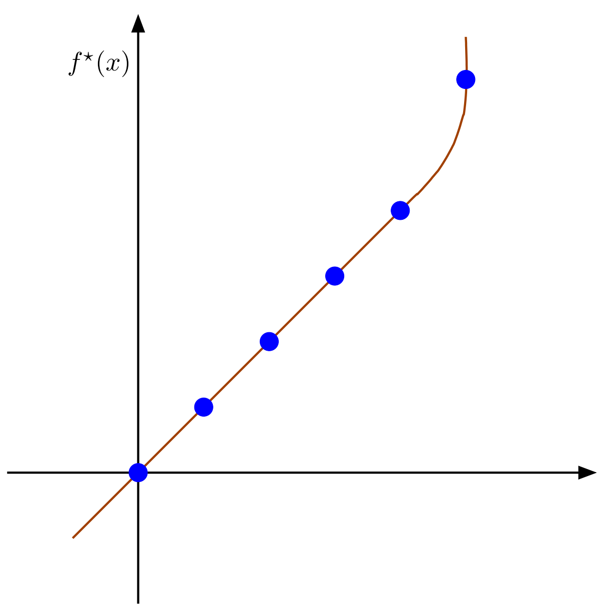
Thus, $\mathcal{P}_n^\star(x) - \mathcal{P}_n(x) = \delta \cdot l_n(x)$. Note that l_n is a function that "oscillates" through zero several times:



Thus, $\mathcal{P}_n^\star(x)$ looks like



What we observe is that a *local* change in y-values caused a *global* (and drastic) change in $\mathcal{P}_n(x)$. Perhaps the "real" function f would have exhibited a more graceful and localized change, e.g.:



We will use the following theorem to compare the "real" function f being sampled, and the reconstructed interpolant $\mathcal{P}_n(x)$.

Theorem 1. *Let*

- $x_0 < x_1 < \ldots < x_{n-1} < x_n$
- $y_n = f(x_n)$, $k = 0, 1, \ldots, n$, *where f is a function which is n-times differentiable with continuous derivatives*
- $\mathcal{P}_n(x)$ *is a polynomial that interpolates* $(x_0, y_0), (x_1, y_1) \ldots, (x_n, y_n)$

then for any $x \in (x_0, x_n)$, there exists a $\theta = \theta(x) \in (x_0, x_n)$ such that

$$f(x) - \mathcal{P}_n(x) = \frac{f^{(n+1)}(\theta)}{(n+1)!}(x - x_0)(x - x_1)\ldots(x - x_n)$$

This theorem may be difficult to apply directly since:

- θ is not known
- θ changes with x
- The $(n+1)$-th derivative $f^{(n+1)}(x)$ may not be fully known.

However, we can use it to derive a conservative bound:

Theorem 2. *If $M = \max_{x\in[x_0,x_n]} |f^{(n+1)}(x)|$ and $h = \max_{0\le i\le n} |x_{i+1} - x_i|$, then*

$$|f(x) - \mathcal{P}_n(x)| \le \frac{Mh^{n+1}}{4(n+1)}$$

for all $x \in [x_0, x_n]$.

How good is this, especially when we keep adding more and more data points (e.g., $n \to \infty$ and $h \to 0$), really depends on the higher order derivatives of $f(x)$. For example, $f(x) = \sin(x)$, $x \in [0, 2\pi]$, alll derivatives of f are $\pm\sin(x)$ or $\pm\cos(x)$. Thus, $|f^{(k)}(x)| \le 1$ for *any* k. In this case, $M = 1$, and as we add more (and denser) data points, we have

$$|f(x) - \mathcal{P}_n(x)| \le \frac{Mh^{n+1}}{4(n+1)} \xrightarrow[h\to 0]{n\to\infty} 0$$

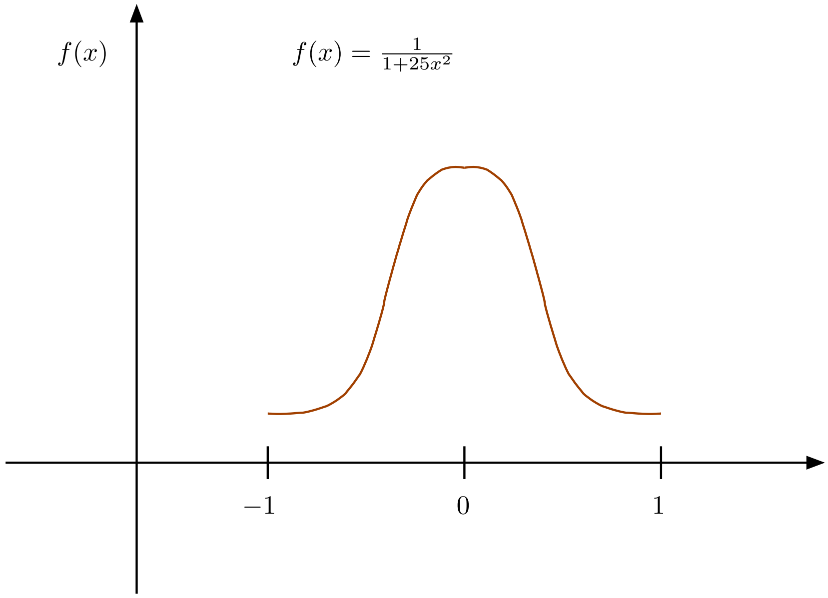
For some functions, however, the values of $|f^{(k)}(x)|$ grow vastly as $k \to \infty$ (i.e., when we introduce additional points). For example,

$$f(x) = \frac{1}{x}, \quad x \in (0.5, 1) \Rightarrow |f^{(n)}(x)| = n!\frac{1}{x^{n+1}}, M = \max_{x\in(0.5,1)} |f^{(n)}(x)| = n!2^{n+1}$$

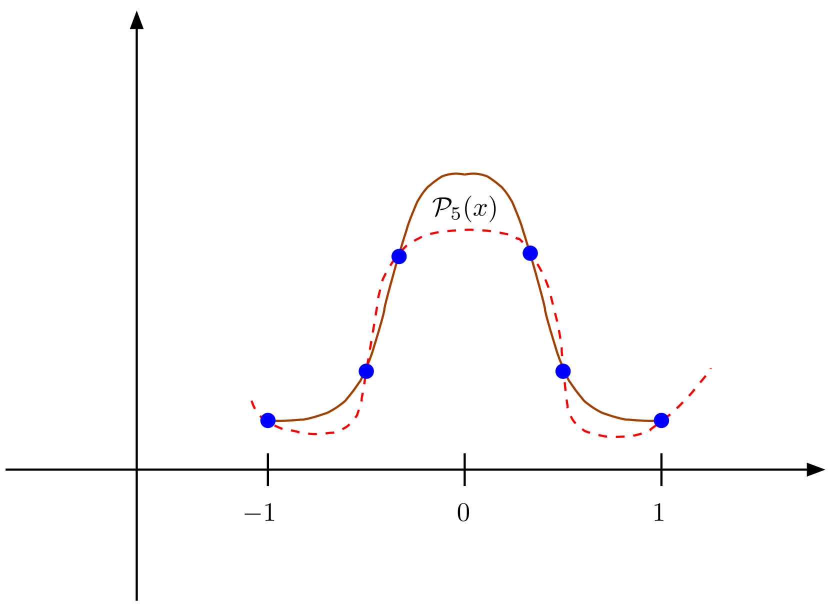
In this case, as $n \to \infty$:

$$\frac{Mh^n}{4n} = \frac{n!2^{n+1}h^n}{4n} \xrightarrow{n\to\infty} \infty$$

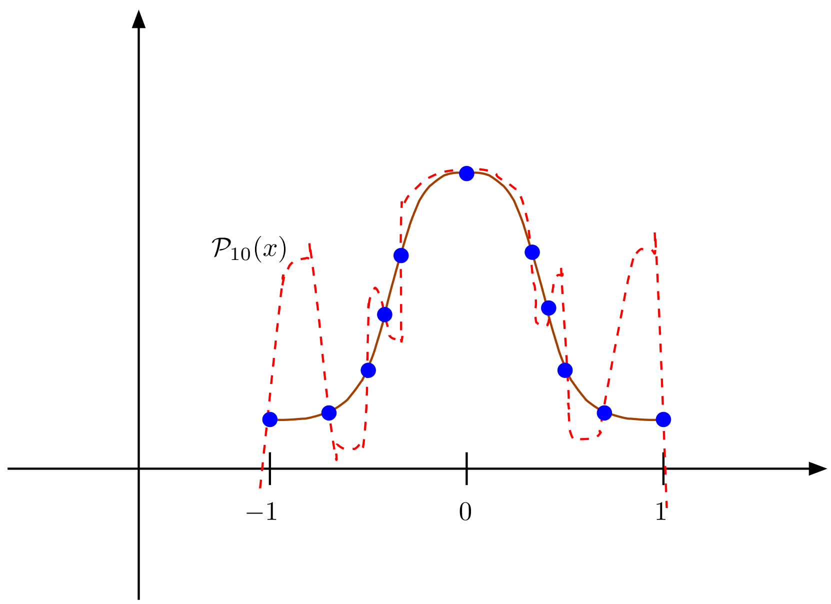
Another commonly cited example is *Runge's function*:



Approximation with a degree-5 polynomial:



Approximation with a degree-10 polynomial:



Thus, in this case, the polynomial $\mathcal{P}_k(x)$ do *not* uniformly converge to $f(x)$ as we add more points.

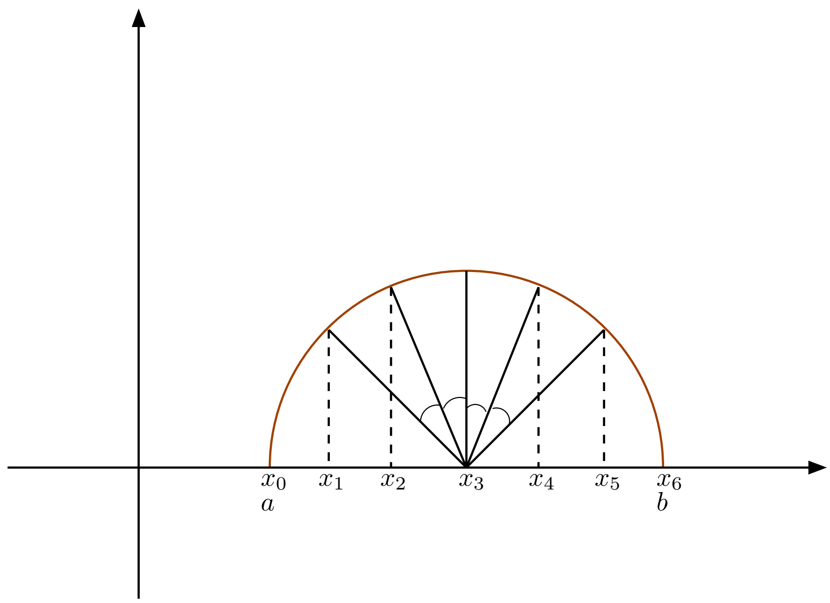
A possible improvement stems from the following idea:

$$f(x) - \mathcal{P}_n(x) = \underbrace{\frac{f^{(n+1)}(\theta)}{(n+1)!}}_{\text{this can be arbitrary}} \underbrace{(x - x_0)\dots(x - x_n)}_{\text{select points to minimize this product}}$$

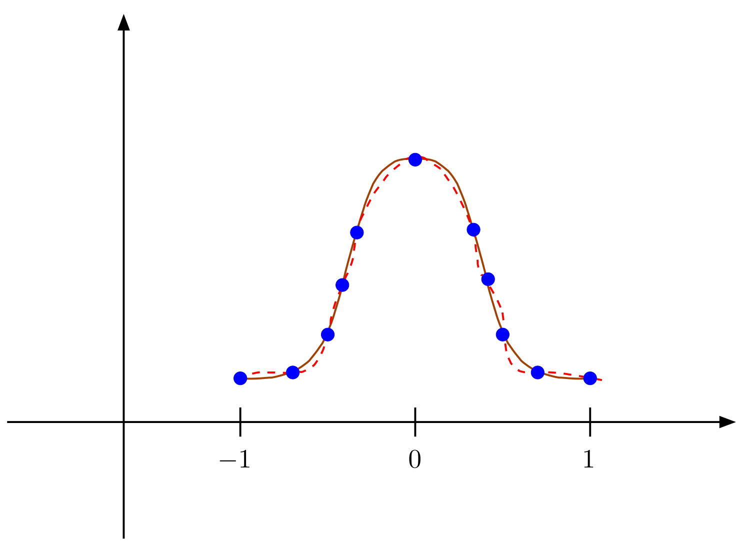
The value of the product $(x - x_0)\dots(x - x_n)$ is minimized by selecting the x_i's as the *Chebyshev points*. If the interpolation interval is $[a, b]$, the Chebyshev points are given by:

$$x_i = \frac{a+b}{2} + \frac{a-b}{2}\cos\left(\frac{i\pi}{n}\right), \quad i = 0, 1, 2, \dots, n$$

Graphically, these points are the projections on the x-axis of the $n+1$ points located along the half circle with diameter the interval $[a, b]$ at equal arc-lengths:



Now, we can re-try Runge's function using Chebyshev points:



In fact, it is possible to show that using Chebyshev points, we can guarantee that

$$|f(x) - \mathcal{P}_n(x)| \xrightarrow{n\to\infty} 0$$

provided that over $[a, b]$ both $f(x)$ and its derivative $f'(x)$ remain bounded (the benefit is that this condition does not place restrictions on higher-order derivatives of $f(x)$).

Although using Chebyshev points mitigates some of the drawbacks of high-order polynomial interpolants, this is still a non-ideal solution, since:

- We do not always have the flexibility to pick the x_i's.
- Polynomial interpolants of high degree typically require more than $O(n)$ computational cost to construct.
- Local changes in the data points affect the entire extent of the interpolant.

Piecewise Polynomials

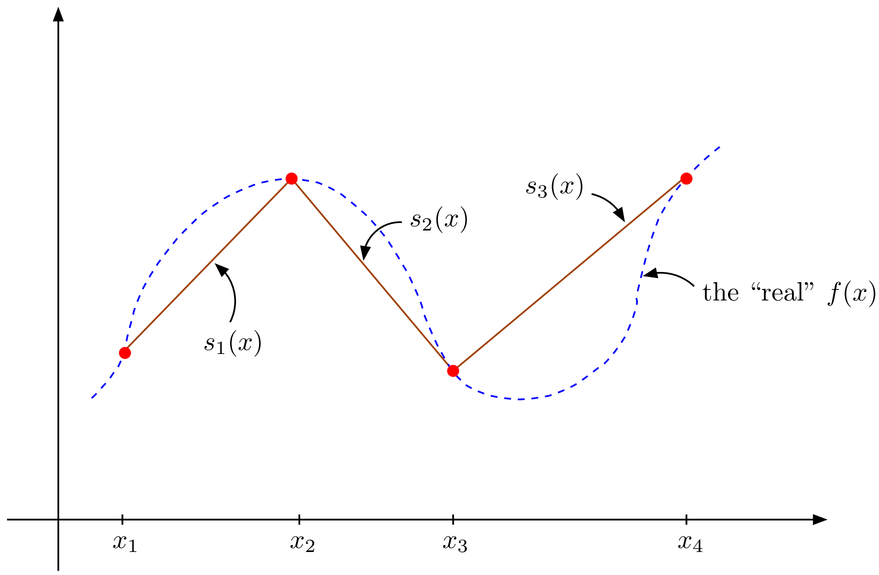
A better remedy is to use piecewise polynomials. Assume that the x-values $\{x_i\}_{i=1}^n$ are sorted in ascending order:

$$a = x_1 < x_2 < \ldots < x_n = b$$

Define $I_k = [x_k, x_{k+1}]$ and $h_k = |x_{k+1} - x_k|$. We also define the polynomials $s_1(x), s_2(x), \ldots, s_{n-1}(x)$ and use each of them to define the interpolant $s(x)$ at the respective interval I_k:

$$s(x) = \begin{cases} s_1(x), x \in I_1 \\ s_2(x), x \in I_2 \\ \vdots \\ s_{n-1}(x), x \in I_{n-1} \end{cases}$$

The benefit of using piecewise polynomial interpolants is that each $s_k(x)$ can be relatively low order and thus, non-oscillatory and easier to compute. The simplest piecewise polynomial interpolant is a *piecewise linear* curve:



In this case, every s_k can be written out explicitly as:

$$s_k(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k)$$

The next step is to examine the error $e(x) = f(x) - s_k(x)$ in the interval I_k. From the theorem we presented in the last lecture, we have that, for any $x \in I_k$ there is a $\theta_k = \theta(x_k) \in I_k$ such that:

$$e(x) = f(x) - s_k(x) = \frac{f''(\theta)}{2} \underbrace{(x - x_k)(x - x_{k+1})}_{q(x)} \tag{1}$$

We are interested in the *maximum* value of $|q(x)|$ in order to determine a bound for the error. $q(x)$ is a quadratic function which crosses zero at x_k and x_{k+1}, thus the extreme value is obtained at the midpoint $x_m = (x_k + x_{k+1})/2$. Thus,

$$|q(x)| \le |q(x_m)| = \left(\frac{x_{k+1} - x_k}{2}\right)^2 = \frac{h_k^2}{4}$$

for all $x \in I_k$. Using equation (1) gives:

$$\begin{aligned}
|f(x) - s_k(x)| &\le \max_{x\in I_k}\left|\frac{f''(x)}{2}\right| \cdot \max_{x\in I_k}|(x - x_k)(x - x_{k+1})| \\
&= \max_{x\in I_k}\left|\frac{f''(x)}{2}\right| \cdot \frac{h_k^2}{4} \\
\Rightarrow |f(x) - s_k(x)| &\le \frac{1}{8}\max_{x\in I_k}|f''(x)| \cdot h_k^2
\end{aligned}$$

for all $x \in I_k$.

Additionally, if we assume all data points are equally spaced, i.e.,

$$h_1 = h_2 = \ldots = h_{n-1} = h = \left(\frac{b-a}{n-1}\right)$$

we can additionally write:

$$|f(x) - s(x)| \leq \frac{1}{8} \max_{x \in [a,b]} |f''(x)| \cdot h^2$$

We often express the quantity on the right hand side using the "infinity norm" of a given function, defined as

$$||f||_\infty = \max_{x \in [a,b]} |f(x)|$$

Thus, using this notation:

$$\boxed{|f(x) - s(x)| \leq \frac{1}{8} ||f''||_\infty \cdot h^2}$$

Note that

- As $h \to 0$, the maximum discrepancy between f and s vanishes (proportionally to h^2)
- As we introduce more points, the quality of the approximation increases consistently, since the criterion above only considers the second derivative $f''(x)$ and not any higher order.

Piecewise cubic interpolation

In this approach, each $s_k(x)$ is a *cubic* polynomial, designed such that it interpolates the four data points:

$$(x_{k-1}, y_{k-1}), (x_k, y_k), (x_{k+1}, y_{k+1}), (x_{k+2}, y_{k+2})$$

As we will see, the benefit is that the error can be made even smaller than with piecewise linear curves; the drawback is that $s(x)$ can develop "kinks" (or corners) where two pieces s_k and s_{k+1} are joined.

Error of piecewise cubics:

$$f(x) - s_k(x) = \frac{f''''(\theta_k)}{4!} \underbrace{(x - x_{k-1})(x - x_k)(x - x_{k+1})(x - x_{k+2})}_{q(x)}$$

An analysis similar to the linear case can show that

$$|q(x)| \leq \frac{9}{16} \max\{h_{k-1}, h_k, h_{k+1}\}^4$$

If we again assume that $h_1 = h_2 = \ldots = h_{n-1} = h$, the error bound becomes:

$$\begin{aligned} |f(x) - s(x)| &\leq \frac{1}{24}||f''''||_\infty \frac{9}{16}h^4 \\ \Rightarrow |f(x) - s(x)| &\leq \frac{9}{384}||f''''||_\infty h^4 \end{aligned}$$

The next possibility we shall consider, is a piecewise cubic curve

$$s(x) = \begin{cases} s_1(x), x \in I_1 \\ s_2(x), x \in I_2 \\ \vdots \\ s_{n-1}(x), x \in I_{n-1} \end{cases}$$

where each $s_k(x) = a_3^{(k)}x^3 + a_2^{(k)}x^2 + a_1^{(k)}x + a_0^{(k)}$ and the coefficients $a_i^{(j)}$ are chosen as to *force* that the curve has continuous values, first and second derivatives:

$$\begin{aligned} s_k(x_{k+1}) &= s_{k+1}(x_{k+1}) \\ s'_k(x_{k+1}) &= s'_{k+1}(x_{k+1}) \\ s''_k(x_{k+1}) &= s''_{k+1}(x_{k+1}) \end{aligned}$$

The curve constructed this way is called a *cubic spline* interpolant.

The Cubic Spline

As always, our goal in this interpolation task is to define a curve $s(x)$ which interpolates the n data points

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \quad \text{(where } x_1 < x_2 < \ldots < x_n\text{)}$$

In the fashion of piecewise polynomials, we will define $s(x)$ as a different cubic polynomial $s_k(x)$ at each sub-interval $I_k = [x_k, x_{k+1}]$, i.e.,

$$s(x) = \begin{cases} s_1(x), x \in I_1 \\ s_2(x), x \in I_2 \\ \vdots \\ s_{n-1}(x), x \in I_{n-1} \end{cases}$$

Each of the s_k's is a cubic polynomial:

$$s_k(x) = a_3^{(k)}x^3 + a_2^{(k)}x^2 + a_1^{(k)}x + a_0^{(k)}$$

where $a_3^{(k)}$, $a_2^{(k)}$, $a_1^{(k)}$, $a_0^{(k)}$ are unknown coefficients. Since we have $n-1$ piecewise polynomials, in total we shall have to determine $4(n-1) = 4n - 4$ unknown coefficients. The points $(x_2, x_3, \ldots, x_{n-1})$ where the formula for $s(x)$ changes from one cubic polynomial (s_k) to another (s_{k+1}) are called *knots*.

Note: In some textbooks, the extreme points x_1 and x_n are also included in the definition of what a knot is. We will stick with the definition we stated above.

The piecewise polynomial interpolation method described as *cubic spline* also requires the neighboring polynomials s_k and s_{k+1} to be joined at x_{k+1} with a certain degree of smoothness. In detail:

- The curve should be continuous: $s_k(x_{k+1}) = s_{k+1}(x_{k+1})$
- The derivative (slope) should be continuous: $s'_k(x_{k+1}) = s'_{k+1}(x_{k+1})$
- The 2nd derivative should be continuous as well: $s''_k(x_{k+1}) = s''_{k+1}(x_{k+1})$

(*Note:* If we force the next (3rd) derivative to match, this will force s_k and s_{k+1} to be exactly identical.)
When determining the unknown coefficients $\{a_i^{(j)}\}$, each of these 3 smoothness constraints (for knots $k = 2, 3, \ldots, n-1$) needs to be satisfied, for a total of $3(n-2) = 3n-6$ constraint equations. We should not forget that we additionally want to *interpolate* all n data points, i.e.,

$$s(x_i) = y_i \quad \text{for } i = 1, 2, \ldots, n$$

In total, we have $3n - 6 + n = 4n - 6$ total equations to satisfy, and $4n - 4$ unknowns! Consequently, we will need 2 more equations to ensure that the unknown coefficients will be uniquely determined. Several plausible options exist on how to do that:

1. The "not-a-knot" approach: We stipulate that at the locations of the first knot (x_2) and last knot (x_{n-1}) the *third* derivative of $s(x)$ should also be continuous, e.g.:

$$s'''_1(x_2) = s'''_2(x_2) \quad \text{and} \quad s'''_{n-2}(x_{n-1}) = s'''_{n-1}(x_{n-1})$$

   As we discussed before, these two additional constraints will effectively cause $s_1(x)$ to be identical with $s_2(x)$, and $s_{n-2}(x)$ to coincide with $s_{n-1}(x)$. In this sense, x_2 and x_{n-1} are no longer "knots" in the sense that the formula for $s(x)$ "changes" at these points (hence the name).

2. Complete spline: If we have access to the derivative f' of the function being sampled by the y_i's (i.e., $y_i = f(x_i)$), we can formulate the two additional constraints as:

$$s'_1(x_1) = f'(x_1) \quad \text{and} \quad s'_{n-1}(x_n) = f'(x_n)$$

   Note that qualitatively, using the complete spline approach is a better utilization of the flexibility of the spline curve in matching yet one more property of f. In contrast, the not-a-knot approach makes the spline "less flexible" by removing two degrees of freedom, in order to obtain a unique solution. However, we cannot always assume knowledge of f'.

3. The natural cubic spline: We use the following two constraints:

$$s''(x_1) = 0 \quad \text{and} \quad s''(x_n) = 0$$

   Thus, $s(x)$ reaches the endpoints looking like a straight line (instead of a curved one).

4. Periodic spline: The following two constraints are used:

$$s'(x_1) = s'(x_n) \quad \text{and} \quad s''(x_1) = s''(x_n)$$

   This is useful when the underlying function f is also known to be periodical over $[a, b]$.

Since $s(x)$ is piecewise cubic, its second derivative $s''(x)$ is piecewise linear on $[x_1, x_n]$. The linear Lagrange interpolation formula gives the following representation for $s''(x) = s''_k(x)$ on $[x_k, x_{k+1}]$:

$$s''_k(x) = s''(x_k)\frac{x - x_{k+1}}{x_k - x_{k+1}} + s''(x_{k+1})\frac{x - x_k}{x_{k+1} - x_k}$$

Defining $m_k = s''(x_k)$ and $h_k = x_{k+1} - x_k$ gives

$$s''_k(x) = \frac{m_k}{h_k}(x_{k+1} - x) + \frac{m_{k+1}}{h_k}(x - x_k)$$

for $x_k \leq x \leq x_{k+1}$ and $k = 1, 2, \ldots, n-1$. Integrating the above equation twice will introduce two constants of integration, and the result can be manipulated so that it has the form:

$$s_k(x) = \frac{m_k}{6h_k}(x_{k+1} - x)^3 + \frac{m_{k+1}}{6h_k}(x - x_k)^3 + p_k(x_{k+1} - x) + q_k(x - x_k) \tag{2}$$

Substituting x_k and x_{k+1} into equation (2) and using the values $y_k = s_k(x_k)$ and $y_{k+1} = s_k(x_{k+1})$ yields the following equations that involve p_k and q_k respectively:

$$y_k = \frac{m_k}{6}h_k^2 + p_k h_k \quad \text{and} \quad y_{k+1} = \frac{m_{k+1}}{6}h_k^2 + q_k h_k$$

These two equations are easily solved for p_k and q_k, and when these values are substituted into equation (2), the result is the following expression for the cubic function $s_k(x)$:

$$\begin{aligned} s_k(x) &= \frac{m_k}{6h_k}(x_{k+1} - x)^3 + \frac{m_{k+1}}{6h_k}(x - x_k)^3 + \left(\frac{y_k}{h_k} - \frac{m_k h_k}{6}\right)(x_{k+1} - x) \\ &+ \left(\frac{y_{k+1}}{h_k} - \frac{m_{k+1} h_k}{6}\right)(x - x_k) \end{aligned} \tag{3}$$

Notice that equation (3) has been reduced to a form that involves only the unknown coefficients $\{m_k\}$. To find these values, we must use the derivative of equation (3), which is

$$
\begin{aligned}
s'_k(x) &= -\frac{m_k}{2h_k}(x_{k+1}-x)^2 + \frac{m_{k+1}}{2h_k}(x-x_k)^2 - \left(\frac{y_k}{h_k} - \frac{m_k h_k}{6}\right) \\
&+ \frac{y_{k+1}}{h_k} - \frac{m_{k+1}h_k}{6} \qquad (4)
\end{aligned}
$$

Evaluating equation (4) at x_k and simplifying the result yields:

$$
s'_k(x_k) = -\frac{m_k}{3}h_k - \frac{m_{k+1}}{6}h_k + d_k, \quad \text{where } d_k = \frac{y_{k+1}-y_k}{h_k} \qquad (5)
$$

Similarly, we can replace k by $k-1$ in equation (4) to get the expression for $s'_{k-1}(x)$ and evaluate it at x_k to obtain

$$
s'_{k-1}(x_k) = \frac{m_k}{3}h_{k-1} + \frac{m_{k-1}}{6}h_{k-1} + d_{k-1} \qquad (6)
$$

Now using the continuity of derivatives and equations (5) and (6) gives an important relation involving m_{k-1}, m_k and m_{k+1}:

$$
h_{k-1}m_{k-1} + 2(h_{k-1}+h_k)m_k + h_k m_{k+1} = u_k \qquad (7)
$$

where $u_k = 6(d_k - d_{k-1})$ for $k = 2,\ldots,n-1$. Observe that the unknowns in equation (7) are the desired values $\{m_k\}$, and the other terms are constants obtained by performing simple arithmetic with the data points $\{x_k, y_k\}$. Therefore, in reality, system (7) is an underdetermined system of $n-2$ linear equations involving n unknowns. Hence, two additional equations must be supplied. They are used to eliminate m_1 and m_n. Consider the natural cubic spline strategy where m_1 and m_n are given ($=0$). The first equation (for $k=2$) of system (7) is:

$$
2(h_1+h_2)m_2 + h_2 m_3 = u_2 - h_1 m_1 \qquad (8)
$$

and similarly, the last equation is:

$$
h_{n-2}m_{n-2} + 2(h_{n-2}+h_{n-1})m_{n-1} = u_{n-1} - h_{n-1}m_n \qquad (9)
$$

Equations (8) and (9) with (7) used for $k = 3,4,\ldots,n-2$ form a tridiagonal $(n-2)\times(n-2)$ linear system $HM = V$ involving the coefficients $m_2, m_3, \ldots, m_{n-1}$:

$$
\begin{bmatrix}
b_2 & c_2 & & & & \\
a_3 & b_3 & c_3 & & & \\
 & & \ddots & & & \\
 & & & a_{n-3} & b_{n-2} & c_{n-2} \\
 & & & & a_{n-2} & b_{n-1}
\end{bmatrix}
\begin{bmatrix} m_2 \\ m_3 \\ \vdots \\ m_{n-2} \\ m_{n-1} \end{bmatrix}
=
\begin{bmatrix} v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{bmatrix}
$$

After the coefficients $\{m_k\}$ are determined, the spline coefficients $a_k^{(j)}$ for $s_k(x)$ are computed using the formulas

$$
a_k^{(0)} = y_k, \quad a_k^{(1)} = d_k - \frac{h_k}{6}(2m_k + m_{k+1}), \quad a_k^{(2)} = \frac{m_k}{2}, \quad a_k^{(3)} = \frac{m_{k+1}-m_k}{6h_k}
$$

Error analysis

For simplicity, we will again assume that

$$h_2 = h_3 = \ldots = h_{n-1} = h \qquad (h_k = x_{k+1} - x_k)$$

For the not-a-knot method, we have

$$|f(x) - s(x)| \lessapprox \frac{5}{384} ||f^{(4)}||_\infty \cdot h^4$$

The "approximate" inequality is used because the interpolation error can be slightly larger near the endpoints of the interval $[a, b]$. This is a very comparable result with the (non-smooth) piecewise cubic polynomial method:

$$|f(x) - s(x)| \leq \frac{9}{384} ||f^{(4)}||_\infty \cdot h^4$$

Note though that the computation of the piecewise cubic method was *very local* and simple (every interval could be independently evaluated) while the computation of the coefficients of the cubic spline is more elaborate.