Chapter 3

Interpolation

Interpolation is the problem of fitting a smooth curve through a given set of points, generally as the graph of a function. It is useful at least in data analysis (interpolation is a form of regression), industrial design, signal processing (digital-to-analog conversion) and in numerical analysis. It is one of those important recurring concepts in applied mathematics. In this chapter, we will immediately put interpolation to use to formulate high-order quadrature and differentiation rules.

3.1 Polynomial interpolation

Given $N+1$ points $x_j \in \mathbb{R}$, $0 \leq j \leq N$, and sample values $y_j = f(x_j)$ of a function at these points, the polynomial interpolation problem consists in finding a polynomial $p_N(x)$ of degree N which reproduces those values:

$$y_j = p_N(x_j), \qquad j = 0, \ldots, N.$$

In other words the graph of the polynomial should pass through the points (x_j, y_j). A degree-N polynomial can be written as $p_N(x) = \sum_{n=0}^{N} a_n x^n$ for some coefficients $a_0, \ldots, a_N$. For interpolation, the number of degrees of freedom ($N+1$ coefficients) in the polynomial matches the number of points where the function should be fit. If the degree of the polynomial is strictly less than N, we cannot in general pass it through the points (x_j, y_j). We can still try to pass a polynomial (e.g., a line) in the "best approximate manner", but this is a problem in approximation rather than interpolation; we will return to it later in the chapter on least-squares.

Let us first see how the interpolation problem can be solved numerically in a direct way. Use the expression of p_N into the interpolating equations $y_j = p_N(x_j)$:

$$\sum_{n=0}^{N} a_n x_j^n = y_j, \qquad j = 0, \ldots, N.$$

In these $N+1$ equations indexed by j, the unknowns are the coefficients $a_0, \ldots, a_N$. We are in presence of a linear system

$$V\mathbf{a} = \mathbf{y}, \qquad \Leftrightarrow \qquad \sum_{n=0}^{N} V_{jn} a_n = y_j,$$

with V the so-called Vandermonde matrix, $V_{jn} = x_j^n$, i.e.,

$$V = \begin{pmatrix} 1 & x_0 & \cdots & x_0^N \\ 1 & x_1 & \cdots & x_1^N \\ \vdots & \vdots & & \vdots \\ 1 & x_N & \cdots & x_N^N \end{pmatrix}.$$

We can then use a numerical software like Matlab to construct the vector of abscissas x_j, the right-hand-side of values y_j, the V matrix, and numerically solve the system with an instruction like $\mathbf{a} = V \setminus \mathbf{y}$ (in Matlab). This gives us the coefficients of the desired polynomial. The polynomial can now be plotted in between the grid points x_j (on a finer grid), in order to display the interpolant.

Historically, mathematicians such as Lagrange and Newton did not have access to computers to display interpolants, so they found explicit (and elegant) formulas for the coefficients of the interpolation polynomial. It not only simplified computations for them, but also allowed them to understand the error of polynomial interpolation, i.e., the difference $f(x) - p_N(x)$. Let us spend a bit of time retracing their steps. (They were concerned with applications such as fitting curves to celestial trajectories.)

We'll define the interpolation error from the uniform (L^∞) norm of the difference $f - p_N$:

$$\|f - p_N\|_\infty := \max_x |f(x) - p_N(x)|,$$

where the maximum is taken over the interval $[x_0, x_N]$.

Call P_N the space of real-valued degree-N polynomials:

$$P_N = \{\sum_{n=0}^{N} a_n x^n : a_n \in \mathbb{R}\}.$$

Lagrange's solution to the problem of polynomial interpolation is based on the following construction.

Lemma 1. *(Lagrange elementary polynomials) Let $\{x_j, j = 0, \ldots, N\}$ be a collection of disjoint numbers. For each $k = 0, \ldots, N$, there exists a unique degree-N polynomial $L_k(x)$ such that*

$$L_k(x_j) = \delta_{jk} = \begin{cases} 1 & \text{if } j = k; \\ 0 & \text{if } j \neq k. \end{cases}$$

Proof. Fix k. L_k has roots at x_j for $j \neq k$, so L_k must be of the form[1]

$$L_k(x) = C \prod_{j \neq k} (x - x_j).$$

Evaluating this expression at $x = x_k$, we get

$$1 = C \prod_{j \neq k} (x_k - x_j) \qquad \Rightarrow \qquad C = \frac{1}{\prod_{j \neq k}(x_k - x_j)}.$$

Hence the only possible expression for L_k is

$$L_k(x) = \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}.$$

These elementary polynomials form a basis (in the sense of linear algebra) for expanding any polynomial interpolant p_N.

[1] That's because, if we fix j, we can divide $L_k(x)$ by $(x - x_j)$, $j \neq k$. We obtain $L_k(x) = (x - x_j)q(x) + r(x)$, where $r(x)$ is a remainder of lower order than $x - x_j$, i.e., a constant. Since $L_k(x_j) = 0$ we must have $r(x) = 0$. Hence $(x - x_j)$ must be a factor of $L_k(x)$. The same is true of any $(x - x_j)$ for $j \neq k$. There are N such factors, which exhausts the degree N. The only remaining degree of freedom in L_k is the multiplicative constant.

Theorem 4. *(Lagrange interpolation theorem)*

Let $\{x_j, j = 0, \ldots, N\}$ be a collection of disjoint real numbers. Let $\{y_j, j = 0, \ldots, N\}$ be a collection of real numbers. Then there exists a unique $p_N \in P_N$ such that

$$p_N(x_j) = y_j, \qquad j = 0, \ldots, N.$$

Its expression is

$$p_N(x) = \sum_{k=0}^{N} y_k L_k(x), \tag{3.1}$$

where $L_k(x)$ are the Lagrange elementary polynomials.

Proof. The justification that (3.1) interpolates is obvious:

$$p_N(x_j) = \sum_{k=0}^{N} y_k L_k(x_j) = \sum_{k=0}^{N} y_k L_k(x_j) = y_j.$$

It remains to see that p_N is the unique interpolating polynomial. For this purpose, assume that both p_N and q_N take on the value y_j at x_j. Then $r_N = p_N - q_N$ is a polynomial of degree N that has a root at each of the $N+1$ points $x_0, \ldots, x_N$. The fundamental theorem of algebra, however, says that a nonzero polynomial of degree N can only have N (complex) roots. Therefore, the only way for r_N to have $N + 1$ roots is that it is the zero polynomial. So $p_N = q_N$.

By definition,

$$p_N(x) = \sum_{k=0}^{N} f(x_k) L_k(x)$$

is called the *Lagrange interpolation polynomial* of f at x_j.

Example 7. *Linear interpolation through (x_1, y_1) and (x_2, y_2):*

$$L_1(x) = \frac{x - x_2}{x_1 - x_2}, \qquad L_2(x) = \frac{x - x_1}{x_2 - x_1},$$

$$\begin{aligned} p_1(x) &= y_1 L_1(x) + y_2 L_2(x) \\ &= \frac{y_2 - y_1}{x_2 - x_1} x + \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1} \\ &= y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1). \end{aligned}$$

Example 8. *(Example (6.1) in Suli-Mayers) Consider $f(x) = e^x$, and interpolate it by a parabola ($N = 2$) from three samples at $x_0 = -1, x_1 = 0, x_2 = 1$. We build*

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{1}{2}x(x - 1)$$

Similarly,

$$L_1(x) = 1 - x^2, \qquad L_2(x) = \frac{1}{2}x(x + 1).$$

So the quadratic interpolant is

$$\begin{aligned} p_2(x) &= e^{-1}L_0(x) + e^0 L_1(x) + e^1 L_2(x), \\ &= 1 + \sinh(1)\, x + (\cosh(1) - 1)\, x^2, \\ &\simeq 1 + 1.1752\, x + 0.5431\, x^2. \end{aligned}$$

Another polynomial that approximates e^x reasonably well on $[-1, 1]$ is the Taylor expansion about $x = 0$:

$$t_2(x) = 1 + x + \frac{x^2}{2}.$$

Manifestly, p_2 is not very different from t_2.

(Insert picture here)

Let us now move on to the main result concerning the interpolation error of smooth functions.

Theorem 5. *Let $f \in C^{N+1}[a, b]$ for some $N > 0$, and let $\{x_j : j = 0, \ldots, N\}$ be a collection of disjoint reals in $[a, b]$. Consider p_N the Lagrange interpolation polynomial of f at x_j. Then for every $x \in [a, b]$ there exists $\xi(x) \in [a, b]$ such that*

$$f(x) - p_N(x) = \frac{f^{(N+1)}(\xi(x))}{(N + 1)!}\pi_{N+1}(x),$$

where

$$\pi_{N+1}(x) = \prod_{j=1}^{N+1}(x - x_j).$$

An estimate on the interpolation error follows directly from this theorem. Set

$$M_{N+1} = \max_{x \in [a,b]} |f^{(N+1)}(x)|$$

(which is well defined since $f^{(N+1)}$ is continuous by assumption, hence reaches its lower and upper bounds.) Then

$$|f(x) - p_N(x)| \leq \frac{M_{N+1}}{(N+1)!}|\pi_{N+1}(x)|$$

In particular, we see that the interpolation error is zero when $x = x_j$ for some j, as it should be.

Let us now prove the theorem.

Proof. (Can be found in Suli-Mayers, Chapter 6)

In conclusion, the interpolation error:

- depends on the smoothness of f via the high-order derivative $f^{(N+1)}$;
- has a factor $1/(N+1)!$ that decays fast as the order $N \to \infty$;
- and is directly proportional to the value of $\pi_{N+1}(x)$, indicating that the interpolant may be better in some places than others.

The natural follow-up question is that of convergence: can we always expect convergence of the polynomial interpolant as $N \to \infty$? In other words, does the factor $1/(N+1)!$ always win over the other two factors?

Unfortunately, the answer is no in general. There are examples of very smooth (analytic) functions for which polynomial interpolation diverges, particularly so near the boundaries of the interplation interval. This behavior is called the *Runge phenomenon*, and is usually illustrated by means of the following example.

Example 9. *(Runge phenomenon) Let $f(x)$ for $x \in [-5, 5]$. Interpolate it at equispaced points $x_j = 10j/N$, where $j = -N/2, \ldots, N/2$ and N is even. It is easy to check numerically that the interpolant diverges near the edges of $[-5, 5]$, as $N \to \infty$. See the Trefethen textbook on page 44 for an illustration of the Runge phenomenon.*

(Figure here)

If we had done the same numerical experiment for $x \in [-1, 1]$, the interpolant would have converged. This shows that the size of the interval matters. Intuitively, there is divergence when the size of the interval is larger than the "features", or characteristic length scale, of the function (here the width of the bump near the origin.)

The analytical reason for the divergence in the example above is due in no small part to the very large values taken on by $\pi_{N+1}(x)$ far away from the origin — in contrast to the relatively small values it takes on near the origin. This is a problem intrinsic to equispaced grids. We will be more quantitative about this issue in the section on Chebyshev interpolants, where a remedy involving non-equispaced grid points will be explained.

As a conclusion, polynomial interpolants can be good for small N, and on small intervals, but may fail to converge (quite dramatically) when the interpolation interval is large.

3.2 Polynomial rules for integration

In this section, we return to the problem of approximating $\int_a^b u(x)dx$ by a weighted sum of samples $u(x_j)$, also called a quadrature. The plan is to form interpolants of the data, integrate those interpolants, and deduce corresponding quadrature formulas. We can formulate rules of arbitrarily high order this way, although in practice we almost never go beyond order 4 with polynomial rules.

3.2.1 Polynomial rules

Without loss of generality, consider the local interpolants of $u(x)$ formed near the origin, with $x_0 = 0, x_1 = h$ and $x_{-1} = -h$. The rectangle rule does not belong in this section: it is not formed from an interpolant.

- The trapezoidal rule, where we approximate $u(x)$ by a line joining $(0, u(x_0))$ and $(h, u(x_1))$ in $[0, h]$. We need 2 derivatives to control the error:

$$u(x) = p_1(x) + \frac{u''(\xi(x))}{2}x(x-h),$$
$$p_1(x) = u(0)L_0(x) + u(h)L_1(x),$$
$$L_0(x) = \frac{h-x}{h}, \qquad L_1(x) = \frac{x}{h},$$
$$\int_0^h L_0(x)dx = \int_0^h L_1(x)dx = h/2, \qquad \text{(areas of triangles)}$$
$$\int_0^h \left|\frac{u''(\xi(x))}{2}x(x-h)\right| dx \le C \max_\xi |u''(\xi)|h^3.$$

The result is

$$\int_0^h u(x)\,dx = h\left(\frac{u(0)+u(h)}{2}\right) + O(h^3).$$

As we have seen, the terms combine as

$$\int_0^1 u(x)\,dx = \frac{h}{2}u(x_0) + h\sum_{j=1}^{N-1} u(x_{j-1}) + \frac{h}{2}u(x_N) + O(h^2).$$

- Simpson's rule, where we approximate $u(x)$ by a parabola through $(-h, u(x_{-1}))$, $(0, u(x_0))$, and $(h, u(x_1))$ in $[-h, h]$. We need three derivatives to control the error:

$$u(x) = p_2(x) + \frac{u'''(\xi(x))}{6}(x+h)x(x-h),$$

$$p_2(x) = u(-h)L_{-1}(x) + u(0)L_0(x) + u(h)L_1(x),$$

$$L_{-1}(x) = \frac{x(x-h)}{2h^2}, \qquad L_0(x) = \frac{(x+h)(x-h)}{-h^2}, \qquad L_1(x) = \frac{(x+h)x}{2h^2},$$

$$\int_{-h}^h L_{-1}(x)dx = \int_{-h}^h L_1(x)dx = h/3, \qquad \int_{-h}^h L_0(x)dx = 4h/3,$$

$$\int_0^h \left|\frac{u'''(\xi(x))}{6}(x+h)x(x-h)\right| dx \le C\max_\xi |u'''(\xi)|h^4.$$

The result is

$$\int_{-h}^h u(x)\,dx = h\left(\frac{u(-h)+4u(0)+u(h)}{3}\right) + O(h^4).$$

The composite Simpson's rule is obtained by using this approximation on $[0, 2h] \cup [2h, 4h] \cup \ldots [1-2h, 1]$, adding the terms, and recognizing that the samples at $2nh$ (except 0 and 1) are represented twice.

$$\int_0^1 u(x)dx = h\left(\frac{u(0)+4u(h)+2u(2h)+4u(3h)+\ldots+2u(1-2h)+4u(1-h)+u(1)}{3}\right.$$

It turns out that the error is in fact $O(h^5)$ on $[-h, h]$, and $O(h^4)$ on $[0, 1]$, a result that can be derived by using symmetry considerations (or canceling the terms from Taylor expansions in a tedious way.) For this, we need u to be four times differentiable (the constant in front of h^5 involves u''''.)

The higher-order variants of polynomial rules, called Newton-Cotes rules, are not very interesting because the Runge phenomenon kicks in again. Also, the weights (like 1,4,2,4,2, etc.) become negative, which leads to unacceptable error magnification if the samples of u are not known exactly.

Piecewise spline interpolation is not a good choice for numerical integration either, because of the two leftover degrees of freedom, whose arbitrary choice affects the accuracy of quadrature in an unacceptable manner. (We'll study splines in a later section.)

We'll return to (useful!) integration rules of arbitrarily high order in the scope of spectral methods.

3.3 Polynomial rules for differentiation

A systematic way of deriving finite difference formulas of higher order is to view them as derivatives of a polynomial interpolant passing through a small number of points neighboring x_j. For instance (again we use $-h, 0, h$ as reference points without loss of generality):

- The forward difference at 0 is obtained from the line joining $(0, u(0))$ and $(h, u(h))$:

$$p_1(x) = u(0)L_0(x) + u(h)L_1(x),$$
$$L_0(x) = \frac{h-x}{h}, \qquad L_1(x) = \frac{x}{h},$$
$$p_1'(0) = \frac{u(h)-u(0)}{h}.$$

  We already know that $u'(0) - p_1'(0) = O(h)$.

- The centered difference at 0 is obtained from the line joining $(-h, u(-h))$ and $(h, u(h))$ (a simple exercise), but it is also obtained from differentiating the parabola passing through the points $(-h, u(-h))$, $(0, u(0))$, and $(h, u(h))$. Indeed,

$$p_2(x) = u(-h)L_{-1}(x) + u(0)L_0(x) + u(h)L_1(x),$$
$$L_{-1}(x) = \frac{x(x-h)}{2h^2}, \qquad L_0(x) = \frac{(x+h)(x-h)}{-h^2}, \qquad L_1(x) = \frac{(x+h)x}{2h^2},$$
$$p_2'(x) = u(-h)\frac{2x-h}{2h^2} + u(0)\frac{2x}{-h^2} + u(h)\frac{2x+h}{2h^2},$$

$$p_2'(0) = \frac{u(h) - u(-h)}{2h}.$$

We already know that $u'(0) - p_2'(0) = O(h^2)$.

- Other examples can be considered, such as the centered second difference (-1 2 -1), the one-sided first difference (-3 4 -1), etc.

Differentiating one-sided interpolation polynomials is a good way to obtain one-sided difference formulas, which comes in handy at the boundaries of the computational domain. The following result establishes that the order of the finite difference formula matches the order of the polynomial being differentiated.

Theorem 6. *Let $f \in C^{N+1}[a,b]$, $\{x_j, j = 0, \ldots, N\}$ some disjoint points, and p_N the corresponding interpolation polynomial. Then*

$$f'(x) - p_N'(x) = \frac{f^{(N+1)}(\xi)}{N!}\, \pi_N^*(x),$$

for some $\xi \in [a,b]$, and where $\pi_N^*(x) = (x - \eta_1)\ldots(x - \eta_N)$ for some $\eta_j \in [x_{j-1}, x_j]$.

The proof of this result is an application of Rolle's theorem that we leave out. (It is not simply a matter of differentiating the error formula for polynomial interpolation, because we have no guarantee on $d\xi/dx$.)

A consequence of this theorem is that the error in computing the derivative is a $O(h^N)$ (which comes from a bound on the product $\pi_N^*(x)$.)

It is interesting to notice, at least empirically, that the Runge's phenomenon is absent when the derivative is evaluated at the center of the interval over which the interpolant is built.

3.4 Piecewise polynomial interpolation

The idea of piecewise polynomial interpolation, also called spline interpolation, is to subdivide the interval $[a,b]$ into a large number of subintervals $[x_{j-1}, x_j]$, and to use low-degree polynomials over each subintervals. This helps avoiding the Runge phenomenon. The price to pay is that the interpolant is no longer a C^∞ function – instead, we lose differentiability at the junctions between subintervals, where the polynomials are made to match.

If we use polynomials of order n, then the interpolant is piecewise C^∞, and overall C^k, with $k \leq n-1$. We could not expect to have $k = n$, because it would imply that the polynomials are identical over each subinterval. We'll see two examples: linear splines when $n = 1$, and cubic splines when $n = 3$. (We have seen in the homework why it is a bad idea to choose $n = 2$.)

3.4.1 Linear splines

We wish to interpolate a continuous function $f(x)$ of $x \in [a, b]$, from the knowledge of $f(x_j)$ at some points x_j, $j = 0, \ldots, N$, not necessarily equispaced. Assume that $x_0 = a$ and $x_N = b$. The piecewise linear interpolant is build by tracing a straight line between the points $(x_{j-1}, f(x_{j-1}))$ and $(x_j, f(x_j)))$; for $j = 1, \ldots, N$ the formula is simply

$$s_L(x) = \frac{x_{j-1} - x}{x_j - x_{j-1}} f(x_{j-1}) + \frac{x - x_{j-1}}{x_j - x_{j-1}} f(x_j), \qquad x \in [x_{j-1}, x_j].$$

In this case we see that $s_L(x)$ is a continuous function of x, but that its derivative is not continuous at the junction points, or nodes x_j.

If the function is at least twice differentiable, then piecewise linear interpolation has second-order accuracy, i.e., an error $O(h^2)$.

Theorem 7. *Let $f \in C^2[a, b]$, and let $h = \max_{j=1,\ldots,N}(x_j - x_{j-1})$ be the grid diameter. Then*

$$\|f - s_L\|_{L^\infty[a,b]} \leq \frac{h^2}{8} \|f''\|_{L^\infty[a,b]}.$$

Proof. Let $x \in [x_{j-1}, x_j]$ for some $j = 1, \ldots, N$. We can apply the basic result of accuracy of polynomial interpolation with $n = 1$: there exists $\xi \in [x_{j-1}, x_j]$ such that

$$f(x) - s_L(x) = \frac{1}{2} f''(\xi)\, (x - x_{j-1})(x - x_j), \qquad x \in [x_{j-1}, x_j].$$

Let $h_j = x_j - x_{j-1}$. It is easy to check that the product $(x - x_{j-1})(x - x_j)$ takes its maximum value at the midpoint $\frac{x_{j-1}+x_j}{2}$, and that the value is $h_j^2/4$. We then have

$$|f(x) - s_L(x)| \leq \frac{h_j^2}{8} \max_{\xi \in [x_{j-1}, x_j]} |f''(\xi)|, \qquad x \in [x_{j-1}, x_j].$$

The conclusion follows by taking a maximum over j.

We can express any piecewise linear interpolant as a superposition of "tent" basis functions $\phi_k(x)$:

$$s_L(x) = \sum_k c_k \phi_k(x),$$

where $\phi_k(x)$ is the piecewise linear function equal to 1 at $x = x_k$, and equal to zero at all the other grid points x_j, $j \neq k$. Or in short, $\phi_k(x_j) = \delta_{jk}$. On an equispaced grid $x_j = jh$, an explicit formula is

$$\phi_k(x) = \frac{1}{h} S_{(1)}(x - x_k),$$

where

$$S_{(1)}(x) = x_+ - 2(x-h)_+ + (x-2h)_+,$$

and where x_+ denotes the positive part of x (i.e., x if $x \geq 0$, and zero if $x < 0$.)

Observe that we can simply take $c_k = f(x_k)$ above. The situation will be more complicated when we pass to higher-order polynomials.

3.4.2 Cubic splines

Let us now consider the case $n = 3$ of an interpolant which is a third-order polynomial, i.e., a cubic, on each subinterval. The most we can ask is that the value of the interpolant, its derivative, and its second derivative be continuous at the junction points x_j.

Definition 5. *(Interpolating cubic spline) Let $f \in C[a,b]$, and $\{x_j, j = 0, \ldots, N\} \subset [a,b]$. An interpolating cubic spline is a function $s(x)$ such that*

1. *$s(x_j) = f(x_j)$;*
2. *$s(x)$ is a polynomial of degree 3 over each segment $[x_{j-1}, x_j]$;*
3. *$s(x)$ is globally C^2, i.e., at each junction point x_j, we have the relations*

$$s(x_j^-) = s(x_j^+), \qquad s'(x_j^-) = s'(x_j^+), \qquad s''(x_j^-) = s''(x_j^+),$$

where the notations $s(x_j^-)$ and $s(x_j^+)$ refer to the adequate limits on the left and on the right.

Let us count the degrees of freedom. A cubic polynomial has 4 coefficients. There are $N+1$ points, hence N subintervals, for a total of $4N$ numbers to be specified.

The interpolating conditions $s(x_j) = f(x_j)$ specify two degrees of freedom per polynomial: one value at the left endpoint x_{j-1}, and one value at the right endpoint x_j. That's $2N$ conditions. Continuity of $s(x)$ follows automatically. The continuity conditions on s', s'' are imposed only at the interior grid points x_j for $j = 1, \ldots, N-1$, so this gives rise to $2(N-1)$ additional conditions, for a total of $4N-2$ equations.

There is a mismatch between the number of unknowns ($4N$) and the number of conditions ($4N-2$). Two more degrees of freedom are required to completely specify a cubic spline interpolant, hence the precaution to write "an" interpolant in the definition above, and not "the" interpolant.

The most widespread choices for fixing these two degrees of freedom are:

- Natural splines:
$$s''(x_0) = s''(x_N) = 0.$$
If $s(x)$ measures the displacement of a beam, a condition of vanishing second derivative corresponds to a free end.

- Clamped spline:
$$s'(x_0) = p_0, \qquad s'(x_N) = p_N,$$
where p_0 and p_N are specified values that depend on the particular application. If $s(x)$ measures the displacement of a beam, a condition of vanishing derivative corresponds to a horizontal clamped end.

- Periodic spline: assuming that $s(x_0) = s(x_N)$, then we also impose
$$s'(x_0) = s'(x_N), \qquad s''(x_0) = s''(x_N).$$

Let us now explain the algorithm most often used for determining a cubic spline interpolant, i.e., the $4N$ coefficients of the N cubic polynomials, from the knowledge of $f(x_j)$. Let us consider the natural spline.

It is advantageous to write a system of equations for the second derivatives at the grid points, that we denote $\sigma_j = s''(x_j)$. Because $s(x)$ is piecewise cubic, we know that $s''(x)$ is piecewise linear. Let $h_j = x_j - x_{j-1}$. We can write
$$s''(x) = \frac{x_j - x}{h_j}\sigma_{j-1} + \frac{x - x_{j-1}}{h_j}\sigma_j, \qquad x \in [x_{j-1}, x_j].$$

Hence

$$s(x) = \frac{(x_j - x)^3}{6h_j}\sigma_{j-1} + \frac{(x - x_{j-1})^3}{6h_j}\sigma_j + \alpha_j(x - x_{j-1}) + \beta_j(x_j - x).$$

We could have written $ax + b$ for the effect of the integration constants in the equation above, but writing it in terms of α_j and β_j makes the algebra that follows simpler. The two interpolation conditions for $[x_{j-1}, x_j]$ can be written

$$s(x_{j-1}) = f(x_{j-1}) \qquad \Rightarrow \qquad f(x_{j-1}) = \frac{\sigma_{j-1}h_j^2}{6} + h_j\beta_j,$$

$$s(x_j) = f(x_j) \qquad \Rightarrow \qquad f(x_j) = \frac{\sigma_j h_j^2}{6} + h_j\alpha_j.$$

One then isolates α_j, β_j in this equation; substitutes those values in the equation for $s(x)$; and evaluates the relation $s'(x_j^-) = s'(x_j^+)$. Given that $\sigma_0 = \sigma_N = 0$, we end up with a system of $N - 1$ equations in the $N - 1$ unknowns $\sigma_1, \ldots, \sigma_N$. Skipping the algebra, the end result is

$$h_j\sigma_{j-1} + 2(h_{j+1}+h_j)\sigma_j + h_{j+1}\sigma_{j+1} = 6\left(\frac{f(x_{j+1}) - f(x_j)}{h_{j+1}} - \frac{f(x_j) - f(x_{j-1})}{h_j}\right).$$

We are in presence of a tridiagonal system for σ_j. It can be solved efficiently with Gaussian elimination, yielding a LU decomposition with bidiagonal factors. Unlike in the case of linear splines, there is no way around the fact that a linear system needs to be solved.

Notice that the tridiagonal matrix of the system above is diagonally dominant (each diagonal element is strictly greater than the sum of the other elements on the same row, in absolute value), hence it is always invertible.

One can check that the interpolation for cubic splines is $O(h^4)$ well away from the endpoints. This requires an analysis that is too involved for the present set of notes.

Finally, like in the linear case, let us consider the question of expanding a cubic spline interpolant as a superposition of basis functions

$$s(x) = \sum_k c_k\phi_k(x).$$

There are many ways of choosing the functions $\phi_k(x)$, so let us specify that they should have as small a support as possible, that they should have the

same smoothness C^2 as $s(x)$ itself, and that they should be translates of each other when the grid x_j is equispaced with spacing h.

The only solution to this problem is the cubic B-spline. Around any interior grid point x_k, is supported in the interval $[x_{k-2}, x_{k+2}]$, and is given by the formula

$$\phi_k(x) = \frac{1}{4h^3} S_{(3)}(x - x_{k-2}),$$

where

$$S_{(3)}(x) = x_+^3 - 4(x-h)_+^3 + 6(x-2h)_+^3 - 4(x-3h)_+^3 + (x-4h)_+^3,$$

and where x_+ is the positive part of x. One can check that $\phi_k(x)$ takes the value 1 at x_k, 1/4 at $x_{k\pm1}$, zero outside of $[x_{k-2}, x_{k+2}]$, and is C^2 at each junction. It is a bell-shaped curve. it is an interesting exercise to check that it can be obtained as the convolution of two "tent" basis functions of linear interpolation:

$$S_{(3)}(x) = cS_{(1)}(x) * S_{(1)}(x),$$

where c is some constant, and the symbol $*$ denotes convolution:

$$f * g(x) = \int_{\mathbb{R}} f(y)g(x-y)\,dy.$$

Now with cubic B-splines, we cannot put $c_k = f(x_k)$ anymore, since $\phi_k(x_j) \neq \delta_{jk}$. The particular values of c_k are the result of solving a linear system, as mentioned above. Again, there is no way around solving a linear system. In particular, if $f(x_k)$ changes at one point, or if we add a datum point (x_k, f_k), then the update requires re-computing the whole interpolant. Changing one point has ripple effects throughout the whole interval $[a, b]$. This behavior is ultimately due to the more significant overlap between neighboring ϕ_k in the cubic case than in the linear case.

18.330 Introduction to Numerical Analysis
Spring 2012