Math 563 Lecture Notes
Polynomial Interpolation: Piecewise (splines)

Spring 2020

Overview

The point: An introduction to splines and a sample of the various approaches. The point is that cubic splines strike a good balance between efficiency and accuracy, and are reasonably straightforward to construct - they are therefore a good 'default choice' for interpolating data.

Related reading: Section 8.7.1 of Quarteroni for the splines discussed here, plus 8.7.2, 8.8 for the more technical B-splines (not covered here).

1 Splines

A good way to avoid the problem of Runge's example is to use only low degree polynomials to interpolate. Instead, we break up the interval $[a, b]$ into sub-intervals

$$I_i = [x_i, x_{i+1}], \quad i = 0, \cdots, n-1 \tag{1}$$

and define a 'piecewise' interpolant $p(x)$ by

$$p(x) = p_i(x) \text{ for } x \in I_i, \quad i = 0, \cdots, n-1$$

where each $p_i(x)$ is some function to be constructed on the sub-interval I_i. For later use, also define the 'grid spacing' and max. grid spacing

$$h_i = x_{i+1} - x_i, \quad h_{\max} = \max_{0 \le i \le n-1} h_i.$$

Now, each 'piece' p_i is defined only on an interval of width h_i which shrinks to zero as the number of points $\to \infty$, so any reasonable approximation will have good convergence.

The main questions are:

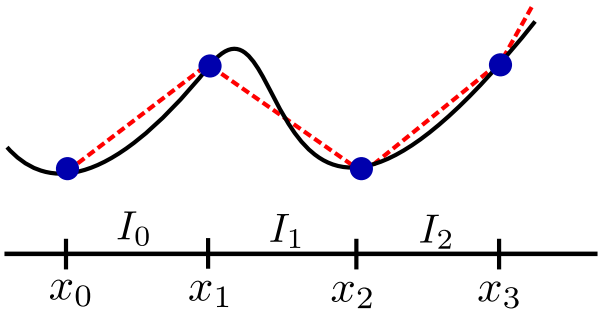
- What are good choices of functions to use (that balance efficiency/accuracy well)?
- How does the error behave as the max. grid spacing $h_{\max} \to 0$?
- How does one efficiently **compute** and **evaluate** the interpolant?

1.1 Piecewise linear interpolant

The simplest piecewise interpolant is the **piecewise linear** one. We choose

$$p_i(x) = \text{the linear interpolant for } x_i, x_{i+1} \text{ in } I_i.$$

The construction is more or less trivial; just connect the data by lines.



Error analysis Let $h_j = x_{j+1} - x_j$ be the interval width. By the Lagrange error formula,

$$|p(x) - f(x)| \leq \frac{f''(\eta_i(x))}{4} h_i^2 \quad \text{for } x \in I_i,$$

where $\eta_i \in I_i$ after bounding

$$|(x - x_i)(x - x_{i+1})| \leq h_i^2/2.$$

Assuming $|f''(x)| \leq M$,

$$|p(x) - f(x)| \leq \frac{M}{4} h_{\max}^2 \text{ for } x \in [a, b].$$

Thus as the max spacing goes to zero, the error goes to zero like $O(h_{\max}^2)$. Since the number of points scales with $1/h$, the error goes to zero like $O(1/n^2)$ as $n \to \infty$. We are using the shrinking of the interval size rather than an increase of degree to improve the error.

However, note that $p'(x)$ is not defined at the x_j's. The approximation is **not differentiable**, which may not be ideal. To improve it, we need a smoother (higher degree) interpolating polynomial that can be further adjusted.

1.2 hats

There is some hidden structure behind the linear interpolant. Suppose, generally, we want to construct an interpolant in the form

$$p(x) = \sum_{i=0}^{n} c_i \phi_i(x)$$

where each $\phi_i(x)$ is a **local basis function** that has 'compact support': it is non-zero only in a small interval around x_i (using only a few nearby sub-intervals).

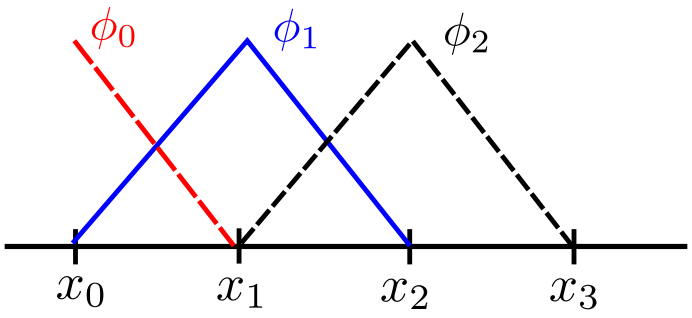
We can cast the piecewse linear interpolant in terms of the 'local basis' by defining basis functions ϕ_i such that

$$\begin{aligned}&\phi_i \text{ is a piecewise linear function,}\\ &\phi_i \text{ is non-zero only for } [x_{i-1}, x_{i+1}],\\ &\phi_i(x_i) = 1,\ \phi_i(x_{i\pm1}) = 0.\end{aligned}$$

This function is easy to construct: it is a piecewise linear 'hat'

$$\phi_j = \begin{cases} \dfrac{x_{j+1} - x}{x_{j+1} - x_j} & x > x_j \\ \dfrac{x - x_{j-1}}{x_j - x_{j-1}} & x < x_j \end{cases}, \qquad \phi_j = 0 \text{ otherwise}$$

suitably modified at the endpoints (half hats).



Then the piecewise linear interpolant - as with the Lagrange basis - is given by

$$p(x) = \sum_{i=0}^{n} f_i \phi_i(x).$$

This is true by construction; the ϕ_i's are piecewise linear so $p(x)$ is also piecewise linear and

$$p(x_j) = \sum_{i=0}^{n} f_i \phi_i(x_j) = \sum_{i=0}^{n} f_i \delta_{ij} = f_j$$

by the properties of ϕ_i (and δ_{ij} is the Kronecker delta).

Aside: elements The hats here are simple examples of piecewise linear 'elements' used in the finite element method, where a basis of this kind (local, made of simple functions) is used to represent solutions to differential equations.

1.3 PCHIPs

We can improve the approximation by upgrading the degree to 3. Recall that given data on f and f' at two points, there is a unique 'cubic Hermite interpolating polynomial' (CHIP). Gluing these pieces together creates the piecewise CHIP (PCHIP). Because the derivatives also match, the PCHIP is a continuous derivative, so it is much smoother than the linear approximation.[1]

What if f' is not known? If $f'(x)$ is not available, a derivative approximation can be used. One has to be careful, however, to use the right method to ensure the spline is reasonable; the discussion here is involved and for simplicity, we will assume f' is known (see: monotonicity preserving splines). One choice, for instance, is the **centered difference** $f' \approx (f_{i+1} - f_{i-1})/(2h)$ for equally spaced points.

Here, we define

$$p_i(x) = \text{CHIP for } [x_i, x_{i+1}] \text{ with data } f_i, f_i' \text{ and } f_{i+1}, f_{i+1}'.$$

Each CHIP can be written in Newton form as

$$p_i(x) = f_i + b_i(x - x_i) + c_i(x - x_i) + d_i(x - x_i)^2(x - x_{i+1})$$

or in terms of the Hermite basis as

$$p_i(x) = f_i q_{1,i} + f_i' q_{2,i} + f_{i+1} q_{3,i} + f_{i+1}' q_{4,i}$$

where $\{q_{k,i},\ k = 0, 1, 2, 3\}$ are the basis polynomials (see HW) that can be computed in advance (the formulas are not complicated).

In the Newton case, divided differences can be used; the table is

	f	Df	$D^2 f$	$D^3 f$
x_i	f_i			
x_i	f_i	$\boxed{f_i'}$		
x_{i+1}	f_{i+1}	$f[x_i, x_{i+1}]$	$\cdots$	
x_{i+1}	f_{i+1}	$\boxed{f_{i+1}'}$	$\cdots$	$\cdots$

with the 'derivative replacement' entries boxed and the $\cdots$ are computed in the usual way (e.g. the top entry is $(f[x_i, x_{i+1}] - f_i')/h_i$) with $h_i = x_{i+1} - x_i$).

The other basis has the advantage that most of the computation is independent of f.

[1] This method is a good general purpose interpolation scheme. Matlab's `interp1`, for instance, uses a version of PCHIP that estimates the derivatives.

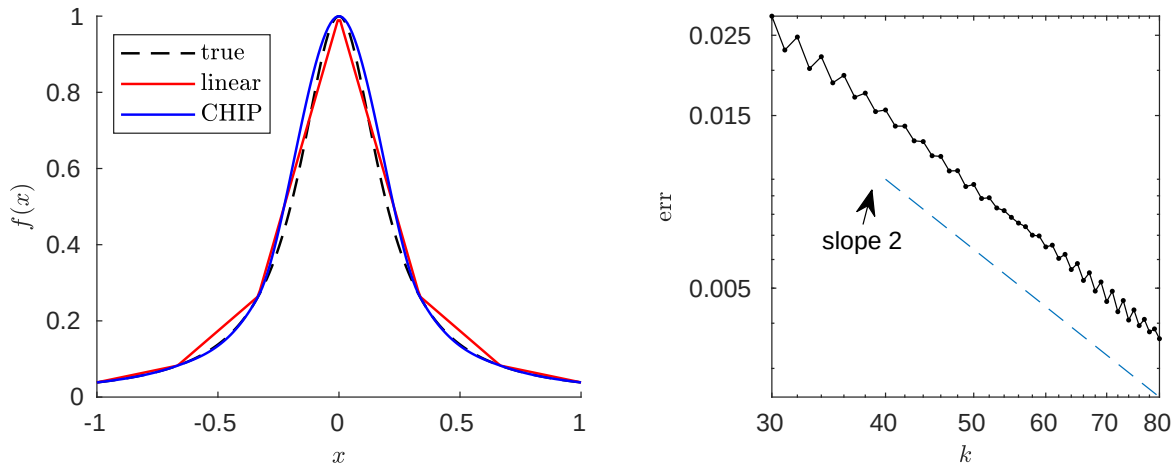


Figure 1: Left: linear and CHIP approximations with $n = 7$ to Runge's example ($f(x) = 1/(1 + 25x^2)$). Right: error plot (loglog) for the piecewise linear case with a reference line.

The error in the CHIP can be shown[2] to have the form

$$E = \frac{f^{(4)}}{4!}(x - x_i)^2(x - x_{i+1})^2$$

(similar to the Lagrange formula, so it follows that

$$|\text{error}| \leq \frac{M}{96}h_{\max}^4, \qquad \text{assuming } |f^{(4)}(x)| \leq M$$

where $h_{\max}$ denotes the maximum spacing. In particular, the error scales like the fourth-power of h, which is quite good. For this reason, cubic splines are a preferred method of interpolation - they have a good balance of simplicity (fast to compute) and accuracy.

One can also use a spline to estimate the derivative. By the construction, p' is continuous. It is also easily computed, and the error turns out to behave nicely (see HW).

Aside: Local basis for cubics? A similar local basis can be constructed for cubic splines, analogous to the hat functions for piecewise linear interpolants. Splines based on this structure are called **B-splines**. The details are somewhat involved (compared to the simpler hats) and will not be pursued here. Note that PCHIPs and natural splines (below) can be constructed directly, without appeal to any underlying basis.

[2]The proof is similar to the Lagrange formula; note that it is the same, but with 'repeated' nodes x_i, x_{i+1} since there are two elements of data at each node.

1.4 natural splines

A variant on the PCHIP strategy relaxes the required data by asking only for the function values. The pieces are cubic polynomials

$$p_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

and instead of the values of f_i' matched, we instead impose 'continuity' conditions to ensure the pieces are smooth (up to second derivatives) across the nodes.

The conditions imposed are (where $p(x)$ is the full interpolant):

i) p interpolates the data: $p(x_j) = f_j$ for the nodes x_j

ii) p, p' and p'' are continuous in (a, b) (in particular, at $x_1, \cdots, x_{n-1}$)

iii) some 'boundary conditions' at the endpoints x_0 and x_n

Boundary conditions: No continuity conditions are imposed at the endpoints. One can check (exercise) that the problem is undertermined if only (i) and (ii) are imposed - we are short by two equations.

There are several common choices. One is the **natural spline** boundary condition

$$p''(x_0) = p''(x_n) = 0. \tag{2}$$

The analogy is to a string pinned at the nodes

The system: Consider the natural spline. The unknowns to solve for are b_i, c_i and d_i for each piece. Imposing condition (i) gives $a_i = f_i$ and one more equation $p_{n-1}(x_n) = f_n$. Imposing condition (ii) at the right endpoint of each sub-interval I_i for $i = 0, \cdots, n-2$ gives

$$p_i''(x_{i+1}) = p_{i+1}''(x_{i+1}) \implies c_i + 3d_i h_i = c_{i+1}, \quad 0 \le i < n-1 \tag{3}$$
$$p_i'(x_{i+1}) = p_{i+1}'(x_{i+1}) \implies b_i + 2c_i h_i + 3d_i h_i^2 = b_{i+1}, \quad 0 \le i < n-1 \tag{4}$$
$$p_i(x_{i+1}) = p_{i+1}(x_{i+1}) \implies f_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = f_{i+1}, \quad 0 \le i < n \tag{5}$$

where $h_i = x_{i+1} - x_i$ (not that the $i = n-1$ case of (5) is the interpolation property (i), not (ii)). The boundary conditions (2) require that

$$c_0 = 0, \quad c_{n-1} + 3d_{n-1}h_{n-1} = 0.$$

While messy, the critical point is that the above is a **linear system** for the coefficients and there are exactly enough equations (see homework). Moreover, each coefficient is related only to adjacent ones, so the system is **sparse** - if written in matrix form, most of the entries of the matrix are zero.

In a better form: We can simplify further to better see the structure. Eliminating d_i

is trivial using (3). One can also eliminate b_i (see the nasty details in subsection 1.5 to obtain a system only for the c's of the form

$$u_i c_{i-1} + v_i c_i + w_i c_{i+1} = r_i, \quad i = 1, \cdots, n-1$$

with the understanding that $c_0 = c_n = 0$ (so the first term drops out when $i = 0$ and the last term drops when $i = n$). The coefficients are given in the referenced section.

In matrix form, this system is

$$A\mathbf{c} = \mathbf{r}, \quad A = \begin{bmatrix} v_1 & w_1 & 0 & 0 & \cdots & 0 \\ u_2 & v_2 & w_2 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & u_{n-2} & v_{n-2} & w_{n-2} \\ 0 & \cdots & \cdots & 0 & u_{n-1} & v_{n-1} \end{bmatrix}$$

with $\mathbf{c} = (c_1, \cdots, c_{n-1})$. The matrix A is **tri-diagonal**: it has non-zero entries only up to one diagonal above/below the main one. By standard methods (e.g. Gaussian elimination[3]), this system can be solved in $O(n)$ operations (and one can show that the system always has a unique solution). Since there are also that many coefficients to solve for, this amount of work is ideal. Thus, the fact that the equations are coupled together does not pose a problem, since there is a good way to compute the solution to the system of n equations.

Comparison to CHIPs: Unlike the CHIP construction, all the pieces are linked together, so we must solve one large linear system at once. This requires more from the writer of the code, but is not any less efficient because the tri-diagonal system is easy to solve numerically. The natural spline also has a continuous second derivative.

[3]You'll occasionally see a version of this referred to as the Thomas algorithm. See LU factorization for tri-diagonal/banded matrices for the details.

1.5 Natural splines: the details

Plug in for d_i to get

$$h_i c_i + h_i c_{i+1} = b_{i+1} - b_i$$
$$h_i b_i + h_i^2 c_i + \frac{h_i^2}{3}(c_{i+1} - c_i) = f_{i+1} - f_i$$

with $c_0 = c_n = 0$. Now write

$$h_i b_{i-1} + h_i h_{i-1} c_{i-1} + \frac{h_i h_{i-1}}{3}(c_i - c_{i-1}) = \frac{h_i}{h_{i-1}}(f_i - f_{i-1})$$

and

$$h_{i-1}(c_i + c_{i-1}) = b_i - b_{i-1}.$$

Then

$$h_i h_{i-1}(c_i + c_{i-1}) + h_i^2 c_i - h_i h_{i-1} c_{i-1} + \frac{h_i^2}{3}(c_{i+1} - c_i) - \frac{h_i h_{i-1}}{3}(c_i - c_{i-1}) = r_i$$

where

$$r_i = f_{i+1} - f_i - \frac{h_i}{h_{i-1}}(f_i - f_{i-1}).$$

Group by c's:

$$\left(h_i h_{i-1} - h_i h_{i-1} + \frac{1}{3} h_i h_{i-1}\right) c_{i-1} + \left(h_i h_{i-1} + h_i^2 - \frac{h_i^2}{3} - \frac{h_i h_{i-1}}{3}\right) c_i + \left(\frac{h_i^2}{3}\right) c_{i+1} = r_i$$

$$\frac{1}{3} h_i h_{i-1} \cdot c_{i-1} + \left(\frac{2}{3} h_i h_{i-1} + \frac{2}{3} h_i^2\right) c_i + \frac{1}{3} h_i^2 \cdot c_{i+1} = r_i.$$

Multiply all this by $3/h_i$ to get the standard form:

$$h_{i-1} c_{i-1} + 2(h_{i-1} + h_i) c_i + h_i c_{i+1} = \frac{3}{h_i}(f_{i+1} - f_i) - \frac{3}{h_{i-1}}(f_i - f_{i-1})$$

with $c_0 = c_n = 0$. This is a linear system $(n-1)$ equations for $\mathbf{c} = (c_1, \cdots, c_{n-1})$ that can be solved efficiently. From here, the rest of the coefficients are computed by the formulas

$$b_i = \frac{1}{h_i}(f_{i+1} - f_i) - h_i c_i - \frac{h_i}{3}(c_{i+1} - c_i)$$

$$d_i = \frac{c_{i+1} - c_i}{3 h_i}.$$

Furthermore, note that the associated matrix is diagonally dominant since

$$2(h_{i-1} + h_i) > h_{i-1} + h_i$$

so it follows that the system always has a unique solution and that the numerical procedure is well-behaved (it is an 'easy' system to solve numerically).