

## Lecture 12: Hermite Interpolation; Piecewise Polynomial Interpolation

An example in the previous lecture demonstrated that polynomial interpolants to  $\sin(x)$  attain arbitrary accuracy for  $x \in [-5, 5]$  as the polynomial degree increases, even if the interpolation points are taken exclusively from  $[-1, 1]$ . In fact, as  $n \rightarrow \infty$  interpolants based on data from  $[-1, 1]$  will converge to  $\sin(x)$  for all  $x \in \mathbb{R}$ . More precisely, for any  $x \in \mathbb{R}$  and any  $\varepsilon > 0$ , there exists some positive integer  $N$  such that  $|\sin(x) - p_n(x)| < \varepsilon$  for all  $n \geq N$ , where  $p_n$  interpolates  $\sin(x)$  at  $n + 1$  uniformly-spaced interpolation points in  $[-1, 1]$ .

In fact, this is not as surprising as it might first appear. The Taylor series expansion uses derivative information at a single point to produce a polynomial approximation of  $f$  that is accurate at nearby points. In fact, the interpolation error bound derived in the previous lecture bears close resemblance to the remainder term in the Taylor series. If  $f \in C^{(n+1)}[a, b]$ , then expanding  $f$  at  $x_0 \in (a, b)$ , we have

$$f(x) = \sum_{k=0}^n \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k + \frac{f^{(n+1)}(\xi)}{(n+1)!} (x - x_0)^{n+1}$$

for some  $\xi \in [x, x_0]$  that depends on  $x$ . The first sum is simply a degree  $n$  polynomial in  $x$ ; from the final term – the Taylor remainder – we obtain the bound

$$\max_{x \in [a, b]} \left| f(x) - \left( \sum_{k=0}^n \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k \right) \right| \leq \left( \max_{\xi \in [a, b]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \right) \left( \max_{x \in [a, b]} |x - x_0|^{n+1} \right),$$

which should certainly remind you of the interpolation error formula derived in the last lecture.

One can view polynomial interpolation and Taylor series as two extreme approaches to approximating  $f$ : one uses global information, but only about  $f$ ; the other uses only local information, but about both  $f$  and its derivatives. In this lecture we will discuss an alternative based on the best features of each of these ideas: use global information about both  $f$  and its derivatives. As a general rule, high degree polynomials are numerically fragile objects, though the degree of fragility varies with the polynomial basis used. (Barycentric interpolation, discussed on the third problem set, is ideal.) We will thus also discuss an alternative that sacrifices smoothness for accuracy, approximating  $f$  with many low degree polynomials that are accurate on small subintervals of  $[a, b]$ .

### 2.6. Hermite interpolation.

In cases where the polynomial interpolants of the previous sections incurred large errors for some  $x \in [a, b]$ , one typically observes that the slope of the interpolant differs markedly from that of  $f$  at some of the interpolation points  $\{x_j\}$ . A natural alternative is to force the interpolant to match both  $f$  and  $f'$  at the interpolation points. Often the underlying application provides a motivation for such derivative matching. For example, if the interpolant approximates the position of a particle moving in space, we might wish the interpolant to match not only position, but also velocity.<sup>†</sup> *Hermite interpolation* is the general procedure for constructing such interpolants.

<sup>†</sup>Typically the position of a particle is given in terms of a second-order differential equation (in classical mechanics, arising from Newton's second law,  $F = ma$ ). To solve this second-order ODE, one usually writes it as a system of first-order equations whose numerical solution we will study later in the semester. One component of the system is position, the other is velocity, and so one automatically obtains values for both  $f$  (position) and  $f'$  (velocity) simultaneously.

Given data points  $\{(x_j, f(x_j), f'(x_j))\}_{j=0}^n$ , we wish to construct a polynomial  $h_n \in \mathcal{P}_{2n+1}$  such that

$$h_n(x_j) = f(x_j), \quad h'_n(x_j) = f'(x_j). \quad (12.1)$$

Note that  $h$  must generally be a polynomial of degree  $2n + 1$  to have sufficiently many degrees of freedom to satisfy the  $2n + 2$  constraints. We begin by addressing the existence and uniqueness of this interpolant.

Existence is best addressed by explicitly constructing the desired polynomial. We adopt a variation of the Lagrange approach used in Section 2.4. We seek degree- $(2n + 1)$  polynomials  $\{A_k\}_{k=0}^n$  and  $\{B_k\}_{k=0}^n$  such that

$$A_k(x_j) = \begin{cases} 0, & j \neq k, \\ 1, & j = k, \end{cases} \quad A'_k(x_j) = 0 \text{ for } j = 0, \dots, n;$$

$$B_k(x_j) = 0 \text{ for } j = 0, \dots, n, \quad B'_k(x_j) = \begin{cases} 0, & j \neq k \\ 1, & j = k \end{cases}.$$

These polynomials would yield a basis for  $\mathcal{P}_{2n+1}$  in which  $h_n$  has a simple expansion:

$$h_n(x) = \sum_{k=0}^n f(x_k)A_k(x) + \sum_{k=0}^n f'(x_k)B_k(x). \quad (12.2)$$

To show how we can construct the polynomials  $A_k$  and  $B_k$ , we recall the Lagrange basis polynomials used for the standard interpolation problem,

$$\ell_k(x) = \prod_{j=0, j \neq k}^n \frac{(x - x_j)}{(x_k - x_j)}.$$

Consider the definitions

$$A_k(x) := (1 - 2(x - x_k)\ell'_k(x_k))\ell_k^2(x),$$

$$B_k(x) := (x - x_k)\ell_k^2(x).$$

Note that since  $\ell_k \in \mathcal{P}_n$ , we have  $A_k, B_k \in \mathcal{P}_{2n+1}$ . Verification that these  $A_k$  and  $B_k$ , and their first derivatives, obtain the specified values at  $\{x_j\}$  is straightforward, and left as an exercise on the third problem set.

These interpolation conditions at the points  $\{x_j\}$  ensure that the  $2n + 2$  polynomials  $\{A_k, B_k\}_{k=0}^n$ , each of degree  $2n + 1$ , form a basis for  $\mathcal{P}_{2n+1}$ , and thus we can always write  $h_n$  via the formula (12.2).

Here are a couple of basic results whose proofs follow the same techniques as the analogous proofs for the standard interpolation problem.<sup>‡</sup>

**Theorem.** The Hermite interpolant  $h_n \in \mathcal{P}_{2n+1}$  is unique.

**Theorem.** Suppose  $f \in C^{2n+2}[x_0, x_n]$  and let  $h_n \in \mathcal{P}_{2n+1}$  such that  $h_n(x_j) = f(x_j)$  and  $h'_n(x_j) = f'(x_j)$  for  $j = 0, \dots, n$ . Then for any  $x \in [x_0, x_n]$ , there exists some  $\xi \in [x_0, x_n]$  such that

$$f(x) - h_n(x) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \prod_{j=0}^n (x - x_j)^2.$$

---

<sup>‡</sup>Note that the uniqueness result hinges on the fact that we interpolate  $f$  and  $f'$  both at all interpolation points. If we vary the number of derivatives interpolated at each data point, we open the possibility of non-unique interpolants.

The proof of this latter result is directly analogous to the standard polynomial interpolation error given in the previous lecture. The reader is encouraged to understand how to derive this result.

**2.6.1. Hermite–Birkhoff interpolation.** Of course, one need not stop at interpolating  $f$  and  $f'$ . Perhaps your application has more general requirements, where you want to interpolate higher derivatives, too, or have the number of derivatives interpolated differ at each interpolation point. Such polynomials are called *Hermite–Birkhoff interpolants*, and you already have the tools at your disposal to compute them. Simply formulate the problem as a linear system and find the desired coefficients, but beware that in some situations, *there may be infinitely many polynomials that satisfy the interpolation conditions*.

**2.6.2. Hermite–Fejér interpolation.** Another Hermite-like scheme initially sounds like a bad idea: Construct  $h_n \in \mathcal{P}_{2n+1}$  such that

$$h_n(x_j) = f(x_j), \quad h_n'(x_j) = 0.$$

That is, explicitly construct  $h_n$  such that its derivatives in general *do not match those of  $f$* . This method, called Hermite–Fejér interpolation, turns out to be remarkably effective, even better than standard Hermite interpolation in certain circumstances. In fact, Fejér proved that if we choose the interpolation points  $\{x_j\}$  in the right way,  $h_n$  is guaranteed to converge to  $f$  uniformly as  $n \rightarrow \infty$ .

**Theorem.** For each  $n \geq 1$ , let  $h_n$  be the Hermite–Fejér interpolant of  $f \in C[a, b]$  at the Chebyshev interpolation points

$$x_j = \frac{a+b}{2} + \left(\frac{b-a}{2}\right) \cos\left(\frac{(2j+1)\pi}{2n+2}\right).$$

Then  $h_n(x)$  converges uniformly to  $f$  on  $[a, b]$ .

For a proof of this result, see page 57 of Natanson, *Constructive Function Theory*, volume 3.

## 2.7. Piecewise polynomial interpolation.

We have seen through examples that high degree polynomial interpolation can lead to large errors when the  $(n+1)$ st derivative of  $f$  is large in magnitude. In other cases, the interpolant converges to  $f$ , but the polynomial degree must be fairly high to deliver an approximation of acceptable accuracy throughout  $[a, b]$ . A robust alternative is to replace the single high-degree interpolating polynomial over the entire interval with many low-degree polynomials valid between consecutive interpolation points.

**2.7.1. Piecewise linear interpolation.** The simplest piecewise polynomial interpolation uses linear polynomials to interpolate between adjacent data points. Informally, the idea is to ‘connect the dots.’ Given  $n+1$  data points  $\{(x_j, f_j)\}_{j=0}^n$ , we need to construct  $n$  linear polynomials  $\{s_j\}_{j=1}^n$  such that

$$s_j(x_{j-1}) = f_{j-1}, \quad \text{and} \quad s_j(x_j) = f_j$$

for each  $j = 1, \dots, n$ .<sup>§</sup> It is simple to write down a formula for these polynomials,

$$s_j(x) = f_j - \frac{(x_j - x)}{(x_j - x_{j-1})}(f_j - f_{j-1}).$$

Each  $s_j$  is valid on  $x \in [x_{j-1}, x_j]$ , and the interpolant  $S(x)$  is defined as  $S(x) = s_j(x)$  for  $x \in [x_{j-1}, x_j]$ .

---

<sup>§</sup>Note that all the  $s_j$ 's are *linear* polynomials—the subscript  $j$  *does not* denote the polynomial degree.

To analyze the error, we can apply the interpolation bound developed in the last lecture. If we let  $\Delta$  denote the largest space between interpolation points,

$$\Delta := \max_{j=1,\dots,n} |x_j - x_{j-1}|,$$

then the standard interpolation error bound gives

$$\max_{x \in [x_0, x_n]} |f(x) - S(x)| \leq \max_{x \in [x_0, x_n]} \frac{|f''(x)|}{2} \Delta^2.$$

In particular, this proves convergence as  $\Delta \rightarrow 0$  provided  $f \in C^2[x_0, x_n]$ .

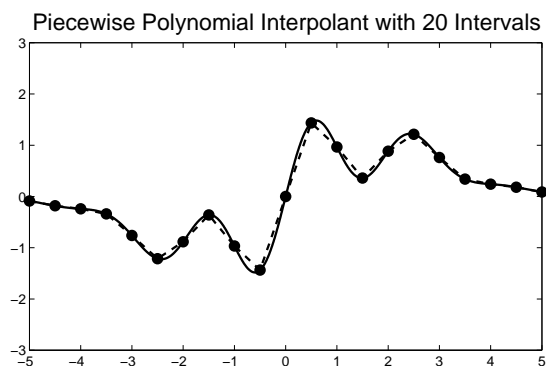
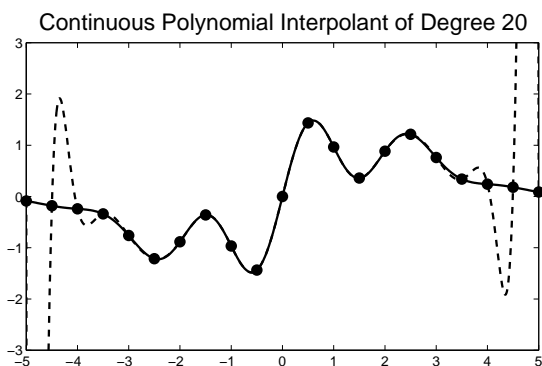
What could go wrong with this simple approach? The primary difficulty is that the interpolant is *continuous*, but generally not *continuously differentiable*. Still, these functions are easy to construct and cheap to evaluate, and can be very useful despite their simplicity.

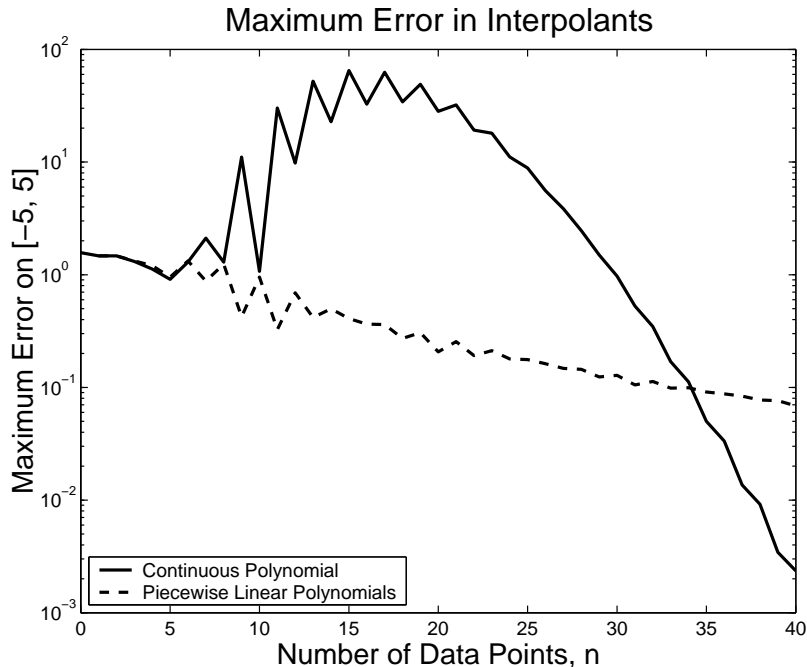
In MATLAB, linear interpolation is implemented in the commands `yy = interp1q(x,y,xx)` and `yy = interp1(x,y,xx,'linear')`.

In the figures below, we compare piecewise linear interpolation with the continuous polynomial interpolation of §§2.2–2.4 over the interval  $[-5, 5]$  for the function

$$f(x) = (x + \sin(3x))e^{-x^2/6}.$$

(The interpolants are shown as dashed lines; the interpolation points are the solid dots.) Eventually, the continuous polynomial interpolants converge, but there is some initial period (when modest degree polynomials do not have enough ‘wiggle’ to capture the oscillations due to the  $\sin(3x)$  term) where the continuous polynomial interpolant is poor, but piecewise linear interpolation does better.





The code that generated these plots is as follows:

```
% compare continuous polynomial interpolation with piecewise linear interpolation

maxdeg = 40;
xx = linspace(-5,5,1000);           % fine grid
fxx = (xx+sin(3*xx)).*exp(-(xx.^2)/6); % true function value on a fine grid
err_poly = zeros(maxdeg+1,1); err_pwpoly = zeros(maxdeg,1);

% continuous polynomial approximation
for n=0:maxdeg
    x = linspace(-5,5,n+1);
    fx = (x+sin(3*x)).*exp(-(x.^2)/6);
    p = polyfit(x,fx,n); % MATLAB's polynomial interpolation routine
    err_poly(n+1) = max(abs(polyval(p,xx)-fxx));
    figure(1), clf
    plot(xx,fxx, 'b-', 'linewidth',2), hold on
    plot(xx,polyval(p,xx), 'r--', 'linewidth',2)
    plot(x,fx, 'r.', 'markersize',30)
    axis equal, axis([-5 5 -3 3])
    title(sprintf('Continuous Polynomial Interpolant of Degree %d',n), 'fontsize',20)
    input('');
end

% piecewise linear interpolation
for n=1:maxdeg
    x = linspace(-5,5,n+1)';
    fx = (x+sin(3*x)).*exp(-(x.^2)/6);
    pxx = interp1(x,fx,xx,'linear'); % MATLAB's piecewise linear interpolation routine
    err_pwpoly(n) = max(abs(pxx-fxx));
end
```

```
figure(2), clf
plot(xx, fxx, 'b-', 'linewidth', 2), hold on
plot(xx, pxx, 'r--', 'linewidth', 2)
plot(x, fx, 'r.', 'markersize', 30)
axis equal, axis([-5 5 -3 3])
title(sprintf('Piecewise Polynomial Interpolant with %d Intervals', n), 'fontsize', 20)
input('');
end

% compare the errors
figure(3), clf
semilogy([0:maxdeg], err_poly, 'b-', 'linewidth', 2), hold on
semilogy([1:maxdeg], err_pwpoly, 'r--', 'linewidth', 2)
legend('Continuous Polynomial', 'Piecewise Linear Polynomials', 3)
title('Maximum Error in Interpolants', 'fontsize', 20)
xlabel('Number of Data Points, n', 'fontsize', 18)
ylabel('Maximum Error on [-5, 5]', 'fontsize', 18)
```