Math 563 Lecture Notes
Polynomial interpolation: the fundamentals

Spring 2020

Overview

The point: Here we introduce polynomial interpolation - a critical tool used throughout computational math for building approximations to functions. Some properties of the important error formula are considered.

Related reading: Quarteroni, Section 8.1.1 and 8.2. For a bit more on stability and the Runge example, see 8.1.2 (the section with the 'Lebesgue constant').

1 Introduction (approximation)

1.1 Motivation

Let $f(x)$ be a function on an interval $[a, b]$. The goal is to construct a function $g(x)$ that approximates f to within a given error. To do so, we must specify:

- The structure of the approximating functions
- The way of measuring error

These choices lead to different schemes, which are the basis of **approximation theory**. Note that there are really two problems in practice, depending on what is known:

- **Problem I:** The function $f(x)$ is given (like $f(x) = e^x$), can be evaluated at any point x and we seek a simple function $g(x)$ (like $g(x) = ax + b$) that best approximates f. The 'simple' part constrains the possible accuracy.
- **Problem II:** Values are given at a set of points: $(x_0, f_0), \cdots (x_n, f_n)$ with $f_j = f(x_j)$ but $f(x)$ is not known. The amount and type of data constrains the possible accuracy and what approximations may be constructed.

One reasonable measure of size is the 'max norm' (or 'L^∞ [ell-infinity]' norm)

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

where $f(x)$ is a continuous function.[1]. This quantity measures the maximum magnitude of f. Thus, $\|f-g\|_\infty$ measures the maximum error between f and g. The discrete analogue, of course, is just the same maximum (the 'ℓ^∞ norm'):

$$\mathbf{f} = (f_0, f_1, \cdots, f_n) \implies \|\mathbf{f}\|_\infty = \max_j |f_j|.$$

(**Preview:** Later, we will consider the L^2 norm $\int_a^b |f(x)-g(x)|^2\,dx$, which is the 'mean-square' error (that you may know from analysis or statistics).)

Approximation by **polynomials** is most straightforward. Denote

$$\mathbb{P}_n = \{\text{polynomials of degree} \le n\}.$$

Larger degree means a more complicated approximation (with $n+1$ coefficients). A polynomial has the obvious advantage that it is easy to compute and manipulate.

One may ask, first, whether it is possible to approximate any function by a polynomial to any accuracy. A classical theorem from analysis assures us this is true:

Theorem (Weierstrass' theorem): Let $f(x)$ be a continuous function on the (closed) interval $[a,b]$. Then there is a sequence of polynomials $P_n(x)$ (of degree n) such that

$$\lim_{n\to\infty} \|P_n - f\|_\infty = 0.$$

Informally: Every continuous function on a closed interval can be approximated arbitrarily well by a polynomial.

However, Wierstrass' theorem does not say how to construct such a polynomial! To do so, constructive methods are needed.

[1] Technically, this is the 'sup' norm $\sup_{x\in[a,b]} |f(x)|$, but if f is continuous then it achieves its maximum and the sup can be replaced by the simpler max. The term is used here since in the discrete case, it represents the maximum error.

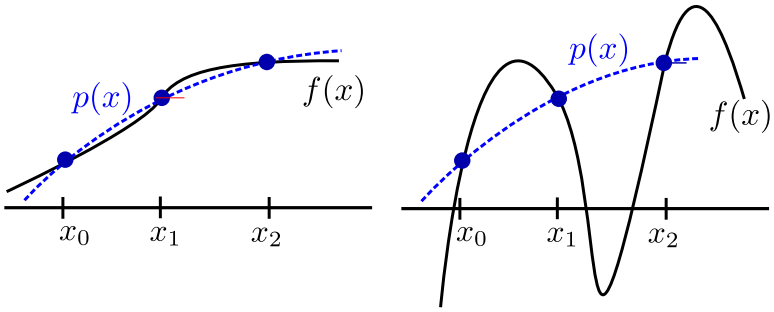


Figure 1: Interpolating polynomial for data at three nodes (x_0, x_1, x_2) and two possible functions $f(x)$. Given three points, $p(x)$ may not be a good estimate of f (right) - the interpolant cannot know what f does between the data points.

2 Polynomial interpolation (Lagrange)

One approach to approximation is called **interpolation**. Suppose we have the data

$$\text{'nodes' } x_0, \cdots, x_n, \quad \text{values } f_j = f(x_j), \ j = 0, 1, \cdots, n. \tag{1}$$

An **interpolant** for $f(x)$ is a function $p(x)$ such that

$$p(x_j) = f_j \text{ for } j = 0, 1, \cdots, n. \tag{2}$$

That is, an interpolant agrees with f at the given nodes. Note that if only the values at $n+1$ points are given, we have no information about $f(x)$ in between, so it could do anything - this is just a limitation of the data (see Figure 1).

In the context where $f(x)$ is known, we are 'sampling' $f(x)$ to construct a simpler approximation; there is more freedom to choose the nodes optimally.

The **interpolating polynomial** $p_n(x)$ (or $p(x)$ if n is implied) for the nodes/data (1) is defined to be the polynomial of degree $\leq n$ that interpolates the data (i.e. satisfies (2)).

Lemma (uniqueness): For a given function $f(x)$, there is a unique polynomial $p_n(x) \in \mathbb{P}_n$ (i.e. degree $\leq n$) interpolating $f(x)$ at the $n+1$ nodes $x_0, \cdots, x_n$.

The proof is useful to know. The uniqueness follows from the fact that a polynomial of degree n has exactly n (complex) zeros (so $\leq n$ real zeros):

Proof. Suppose there are two such polynomials $p(x)$ and $q(x)$. Let $r(x) = p(x) - q(x)$. Then

$$r(x) \text{ has degree} \leq n, \quad r(x_j) = p(x_j) - q(x_j) = 0 \text{ for } j = 0, \cdots n.$$

Thus r is a degree $\leq n$ polynomial with $n+1$ zeros. But a non-zero polynomial of degree n has at most n real zeros (by the fundamental theorem of algebra). Thus, $r(x)$ must be equal

to the zero function (so it is trivial), and so $p(x) = q(x)$ for all x.

Equivalently, we can factor $r(x)$:

$$r(x) = \cdots (x - x_0) \cdots (x - x_n).$$

But the $n+1$ factors of $(x - x_j)$ give a degree $n+1$ polynomial, which makes no sense unless the $\cdots$ is zero, in which case $r(x) = 0$ trivially. □

Construction (general idea): From here, we would like to write

$$f(x) \approx p_n(x) = \sum_{i=0}^{n} c_i \phi_i(x)$$

where the ϕ_k's are some basis that spans $\mathbb{P}_n(x)$. One choice is $1, x, x^2, \cdots x^n$, in which case

$$p_n(x) = c_0 + c_1 x + \cdots + c_n x^n$$

but then the equations $p_n(x_j) = f_j$ just yield a messy system of equations (see below). Two more elegant approaches (often more useful in practice) are considered in more detail.

Uniqueness: To emphasize, because the interpolating polynomial is unique, any method will end up constructing the same polynomial, just in a different form. The methods should be evaluated, then, by their advantages/disadvantages in practice.

Caution (interpolation vs. approximation): Note that 'interpolation' is not exactly the same as 'approximation' - it is a strategy that one hopes will approximate the function. In the case of Problem II where data is given, interpolation is natural since it uses precisely the data we are given.

For Problem I (where f is given), it is not obvious that interpolation is the right way to obtain a small max-norm (or any other error). Compare, for instance, to a best-fit line that does not have to pass through any points.

2.1 Simple basis

For contrast, we start by constructing the interpolant in the basis $\{1, x, x^2, \cdots, x^n\}$. The interpolating polynomial then has the form

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

Imposing the conditions $p(x_j) = f_j$, we obtain the system of equations

$$\sum_{i=0}^{n} c_i x_j^i = f_j, \qquad j = 0, \cdots n. \tag{3}$$

This is a linear system for the unknowns. Let $\mathbf{c} = (c_0, \cdots, c_n)^T$ (a vector of dim. $n+1$) and $\mathbf{f} = (f_0, \cdots, f_n)^T$ and define the **Vandermonde matrix**

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}.$$

Then the system (3) can be written in matrix form as

$$V\mathbf{c} = \mathbf{f}.$$

If n is small (say, $n = 1$ or $n = 2$) then this approach works find (when $n = 1$, the process is just finding the line $ax + b$ between two points). For larger n, the problem is not so nice. It would be convenient if V were simply a diagonal matrix...

2.2 Lagrange basis

A more clever choice of basis makes solving for the coefficients trivial. The **Lagrange form** uses basis polynomials $\ell_i(x)$ such that

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

That is, the coefficients are just the function values. This formula works if and only if each ℓ_i vanishes at all the nodes except the k-th:

$$\ell_i(x_j) = \delta_{ij} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}, \qquad \text{for } i = 0, \cdots, n. \tag{4}$$

where δ_{ij} is the 'Kronecker delta' (defined in-line). But this just says that ℓ_i has zeros at all the x_j's except x_i. There are then n zeros and $\ell_i \in \mathbb{P}_n$, so the polynomial can be factored. The condition $\ell_i(x_i) = 1$ yields the leading factor:

$$\ell_i(x) = C \prod_{j=0, j\neq i}^{n} (x - x_j) \implies \ell_i(x) = \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}. \tag{5}$$

Theorem (Lagrange form of the interpolant): Let $x_0, \cdots, x_n$ be a set of $n+1$ distinct nodes and let

$$\ell_i(x) = \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

be the i-th '**Lagrange basis polynomial**'. Then the interpolating polynomial for the data $(x_0, f_0), \cdots (x_n, f_n)$ can be expressed as

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

(The proof is immediate from the construction).

2.3 Computing: Lagrange form

The construction is straightforward in Lagrange form, if a bit tedious. Note that the basis functions are independent of $f(x)$; they depend only on the choice of nodes $\{x_j\}$. Thus, we can compute the ℓ_i's separately from f, which can save work if several different f's are to be evaluated at the same nodes.

Two alternate forms are more typical for computation (see section 5 for the other). To improve the Lagrange formula (LI), let

$$\ell(x) = \prod_{j=0}^{n} (x - x_j).$$

Factor this out of the Lagrange formula to get

$$p(x) = \ell(x) \sum_{j=0}^{n} w_j \frac{f_j}{x - x_j} \tag{M-LI}$$

where the 'weights' w_j are the leftover constants from the product:

$$w_j = \prod_{k=0,\, k\neq j}^{n} \frac{1}{x_j - x_k}.$$

This is called the **modified Lagrange formula**. The weights depend only on the nodes, so they can be computed in advance. Then the interpolant (M-LI) can be evaluated at any value of x painlessly. For further simplification into 'barycentric form', see homework.

2.4 Note on adding points

One disadvantage of the Lagrange form is that it is not easy to add more points to the interpolant. Suppose $p_n(x)$ has been constructed for nodes $x_0, \cdots x_n$. Then, we find that a new points x_{n+1} must be added (say, to improve accuracy).

Now the basis functions and/or weights must be re-computed - the entire polynomial has changed. Ideally, we would like to keep the old computation and just add a new term. Fortunately, another approach makes this easier (see Newton form, section 5).

2.5 Examples

Example (linear case): We can use this form to construct a line through two points (x_0, y_0) and (x_1, y_1). The Lagrange basis functions are

$$\ell_0(x) = \frac{x - x_1}{x_0 - x_1}, \qquad \ell_1(x) = \frac{x - x_0}{x_1 - x_0}$$

so

$$p_1(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0}.$$

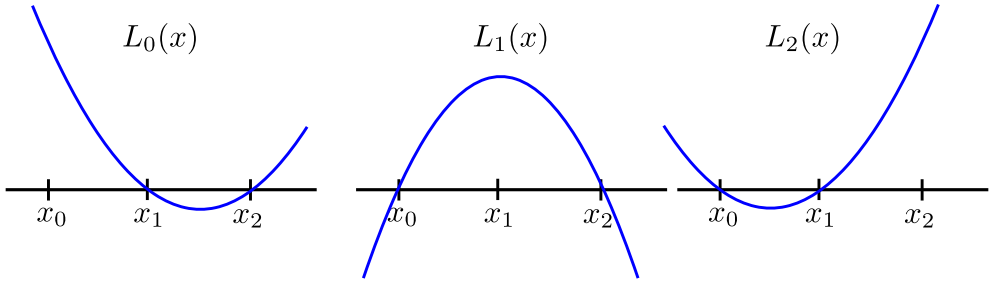
Example (basis functions for $n = 2$) Consider the nodes $x = 0, 1/2$ and 1. The Lagrange basis polynomials are

$$\ell_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 1/2)(x - 1)}{1/2} = 2(x - \frac{1}{2})(x - 1)$$

and

$$\ell_1(x) = -4x(x - 1), \qquad \ell_2(x) = 2x(x - \frac{1}{2}).$$

the basis polynomials are sketched below.



2.6 Comparison to Taylor's theorem

Recall **Taylor's theorem** (simplified slightly):

Theorem (Taylor series) Let $f \in C^{(n+1)}([a,b]$ (continuous $n+1$-st derivative in $[a,b]$). Then for each point x_0 in the interval, f can be written as

$$f(x) = P_n(x) + R(x),$$
$$T_n(x) := \sum_{j=0}^{n} \frac{f^{(j)}(x_0)}{j!}(x-x_0)^j, \quad R(x) := \frac{f^{(n+1)}(\eta_x)}{(n+1)!}(x-x_0)^{n+1} \tag{6}$$

for some value η_x inside $[a,b]$ depending on x.

The function can be approximated by an n-th degree polynomial plus error using values

$$f(x_0), f'(x_0), \cdots f^{(n)}(x_0).$$

This theorem, of course, is a valid way to approximate $f(x)$. Like the Lagrange interpolant, $n+1$ pieces of information of used, but all are at one point.

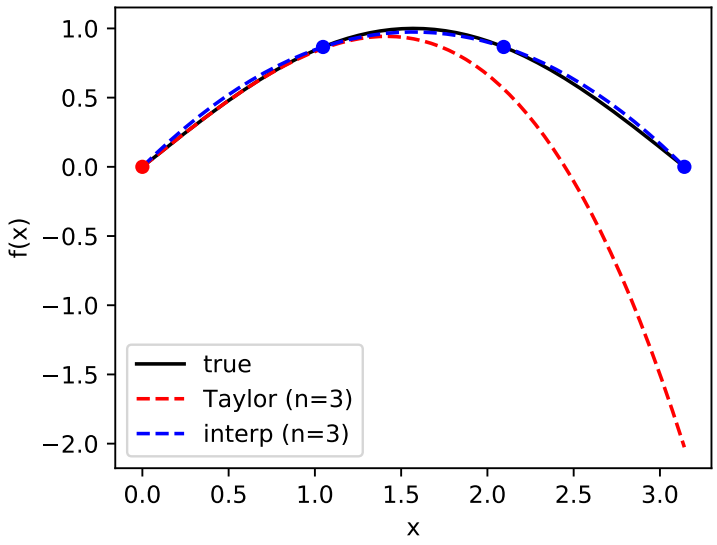
Because the Taylor series uses only information at x_0, it tends to do poorly away from x_0. Given data at several points, interpolants tend to do better - since it takes into account information on the whole interval. An example is shown below for

$$f(x) = \sin x, \quad x \in [0, \pi].$$

The Taylor series for $n = 3$ is

$$T_3(x) = x - x^3/6.$$

The interpolating polynomial $p_3(x)$ is also shown with equally spaced points. Note that the interpolant has the appropriate shape, while the Taylor polynomial drops off when it leaves the neighborhood of $x = 0$.



3 Error analysis

The error bound can be written in a similar form to that of Taylor's theorem. The fundamental result is the following (extremely important!) formula for interpolation error:

Theorem (Lagrange error formula): Suppose $f \in C^{n+1}([a,b])$ with the $n+1$ nodes

$$x_0 < x_1 < \cdots < x_n$$

contained in $[a,b]$. Let $p_n(x)$ be the interpolating polynomial. Then for $x \in [a,b]$,

$$f(x) = p_n(x) + E(x)$$

where the error can be written (in the 'Lagrange form')

$$E(x) = \frac{f^{(n+1)}(\eta_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j)$$

for some η_x (depending on x) in $[a,b]$.

Comparison to Taylor: The only difference is that

$$\text{(Taylor)} \prod_{j=0}^{n} (x - x_0) \quad \longrightarrow \quad \text{(Lagrange)} \prod_{j=0}^{n} (x - x_j)$$

and the rest (assumptions, η_x, etc.) are the same. The difference in the error (as in the previous page) comes from the behavior of the product factor.

The proof is similar to the proof of Taylor's theorem and is not really important; see subsection 5.2 for the details.

3.1 Consequences

There are two key questions to answer: When is the error small? When can we find a good bound on the error? To restate, the error in the interpolant is

$$E(x) = \frac{f^{(n+1)}(\eta_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j),$$

Note that η_x is only known to be inside $[a,b]$; otherwise we have no control over it. Thus any bound derived for the error cannot depend on η_x.

The error depends on two factors (and the $1/(n+1)!$):

- The size of the $n+1$-th derivative. Due to the η_x, we can only bound this term by

$$M = \max_{x \in [a,b]} |f^{(n+1)}(x)|. \tag{7}$$

- The product

$$\omega_n(x) = \prod_{j=0}^{n}(x - x_j) \tag{8}$$

  which is **small** if the nodes x_j's and x are close together, and **large** if they are far apart or x is far from the nodes.

  In particular, it is clear that $\omega(x)$ will cause trouble for extrapolation (when x is outside the interval $[x_0, x_n]$).

- The factor $1/(n+1)!$,, which goes to zero quickly as $n \to \infty$.

Which term wins? The factorial helps make the error small, while the other two may grow with n (depending on f and the nodes). Thus, the size of the error depends on a competition between these factors.

A bound on the error in an interval I can be obtained by bounding each part, so

$$\max_{x\in I} |p_n(x) - f(x)| \leq \frac{M}{(n+1)!} \max_{x\in I} |\omega_n(x)|. \tag{9}$$

This bound works for any interval I, so for instance the error between the first two points ($I = [x_0, x_1]$) would require bounding ω only in that range.

3.2 General error bound: equally spaced points

To get a better sense of $\omega_n(x)$, suppose we now have **equally spaced** nodes in $[a, b]$:

$$x_i = a + jh, \quad h = \frac{b-a}{n}$$

so $x_0 = a$ and $x_n = b$. Let ω_n be defined as in (8). We want to know what happens to the error as the 'grid spacing' h goes to zero.

To do so, the product ω_n mus be bounded. First note that the error is worst near the endpoints (why is this claim plausible?).
Now suppose that x is between the first two nodes, i.e. $x \in [x_0, x_1]$. Then each node after that is h further away, so

$$|x - x_j| \leq (j+1)h \text{ for } j = 0, \cdots n.$$

It follows (omitting the argument for the other sub-intervals) that

$$|\omega_n(x)| \leq \prod_{j=0}^{n} |x - x_j| \leq (n+1)!h^{n+1} \text{ for } x \in [a, b].$$

From Stirling's approximation,

$$n! \sim e^{-n}n^n\sqrt{2\pi n} \text{ as } n \to \infty,$$

the h^{n+1} factor is (barely) enough to win over the $(n+1)!$:

$$(n+1)!(\frac{b-a}{n})^{n+1} \sim Cn^{1/2}(\frac{b-a}{e})^n \text{ as } n \to \infty.$$

Thus ω_n decays like e^{-n} (times $\sqrt{n}$). From the bound

$$|E(x)| \leq \frac{\left(\max_{x\in[a,b]} |f^{(n+1)}(x)|\right)}{(n+1)!} \max|\omega_n(x)|$$

we see that if the derivative over $(n+1)!$ decays faster than $((b-a)/e)^n$ grows, the error will go to zero (exponentially!).[2]

Example: Consider $f(x) = 1/x$ with $x_0 = 1$ and $x_n = 2$.
Then for $x \in [0,1]$ we have $f^n = (-1)^n n!/x^{n+1}$ so

$$\frac{|f^{n+1}(x)|}{(n+1)!} \leq \left(\max_{x\in[1,2]} |1/x^{n+1}|\right) = 1.$$

since $1/x$ is decreasing on $[1,2]$. The ω_n term decays fast enough to cancel this out:

$$|E_n(x)| \leq 1 \cdot \max|\omega_n| \sim Cn^{1/2}e^{-n} \text{ as } n \to \infty,$$

so the error decreases like e^{-n}. This cancellation is typical, but far from guaranteed. [3]

3.3 Example: error bound

Here is a practical example of an error estimate. Suppose we have the function/nodes

$$f(x) = e^x, \quad x_0 = 0,\ x_1 = 1$$

and wish to approximate in $[0,1]$. The interpolating polynomial (Lagrange form) is

$$p_1 = 1 \cdot (1-x) + e \cdot x$$

and the error has the form

$$E(x) = \frac{f''(\eta_x)}{2!}x(x-1)$$

for $\eta_x \in [0,1]$. The best we can do for the f'' term is the bound

$$M = \max_{x\in[0,1]} |f''(x)| = \max_{x\in[0,1]} |e^x| \leq e.$$

For the other term,

$$\max_{x\in[0,1]} |x(x-1)| \leq 1/4$$

[2]The condition here is not and if an only if; proving the precise conditions takes more work.

[3]For those familiar with complex analysis - if f is analytic in a disk of radius R, then by Cauchy's integral formula, $|f^{(n)}(z)| \leq C/R^n$ by integrating $f/(z-x_0)^{n+1}$ around the disk. Then the error bound looks like $((b-a)/R)^n$, suggesting that analytic functions, at least, behave nicely with interpolation.

by finding the maximum (which is at $x = 1/2$). Thus

$$|p_1(x) - f(x)| \leq \frac{e}{8} \approx 0.34 \text{ for } x \text{ in } [0, 1]$$

which is not great, but with only two functions values, we should not expect better.

Adding more points: Now suppose that we now use 10 equally spaced points

$$x_j = jh, \text{ for } j = 0, \cdots 9$$

where $h = 1/9$ is the spacing between points. Then the error is

$$E(x) = \frac{f^{(10)}(\xi_x)}{10!} \prod_{j=0}^{9} (x - x_j).$$

Bounding $|f^{(10)}(\xi_x)| \leq e$ is easy. The product is bounded as previously discussed, yielding

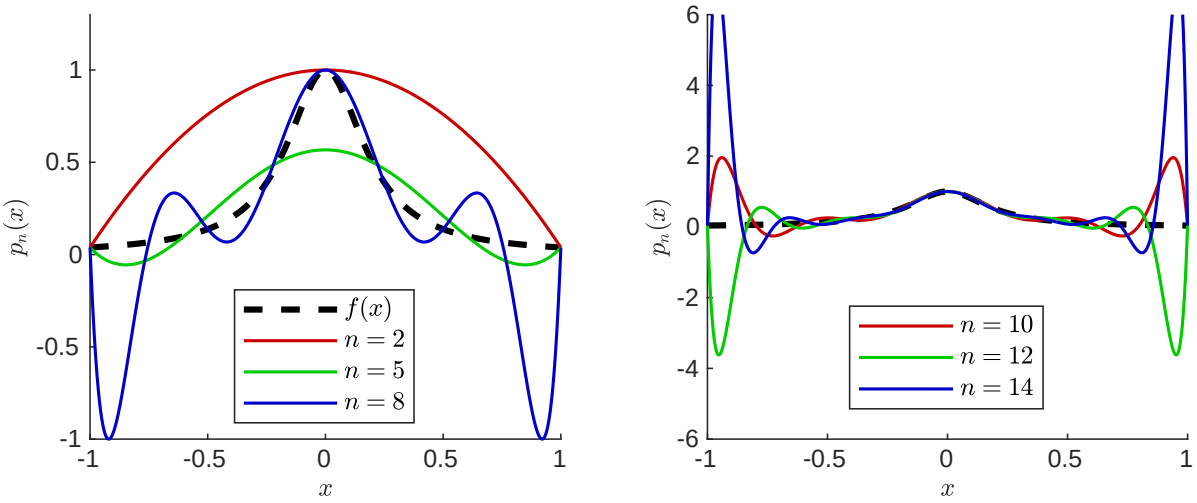
$$|E(x)| \leq eh^{10} \approx 7.8 \times 10^{-10}.$$

4 What can go wrong: Runge's example

In general, it is dangerous to increase the number of nodes to try to improve the interpolant. To see how badly things can go, consider the function

$$f(x) = \frac{1}{1+25x^2}, \quad x \in [-1,1]$$

and equally spaced nodes $-1 = x_0 < \cdots < x_n = 1$.
Plots of the interpolants $p_n(x)$ for small and large n are shown below.



The polynomial oscillates wildly near the endpoints, getting worse and worse as n increases. In fact, the maximum error

$$\max_{x\in[-1,1]} |p_n(x) - f(x)|$$

grows **exponentially** as $n \to \infty$ (black line in Figure 2 below).

The poor behavior is not really unexpected, given some thought. If the polynomial is increasing through one node, it needs to turn sharply to get back to the next node, causing rapid variation. Thus, once the polynomial ends up with a large derivative, it gets out of control and there is not much hope.

Relation to the error formula: The competing factors are

$$\frac{f^{(n+1)}(\xi)}{(n+1)!}, \qquad \omega_n(x) = \prod_{j=0}^{n}(x - x_j).$$

It can be checked that if n is large, $f^{(n)}/(n+1)!$ grows quite rapidly with n, fast enough that the good behavior of $\omega_{n+1}/(n+1)!$ (exponential decay) is not enough to counteract it.

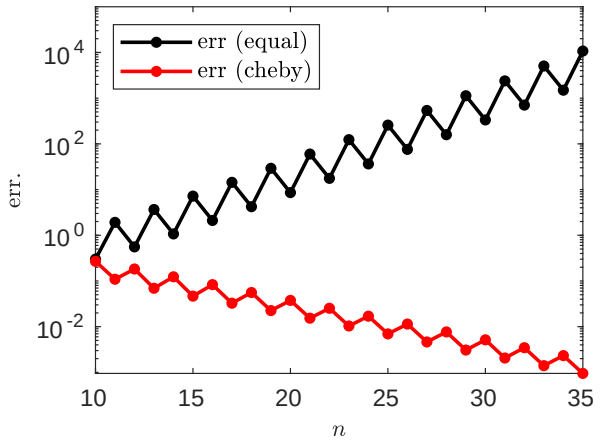
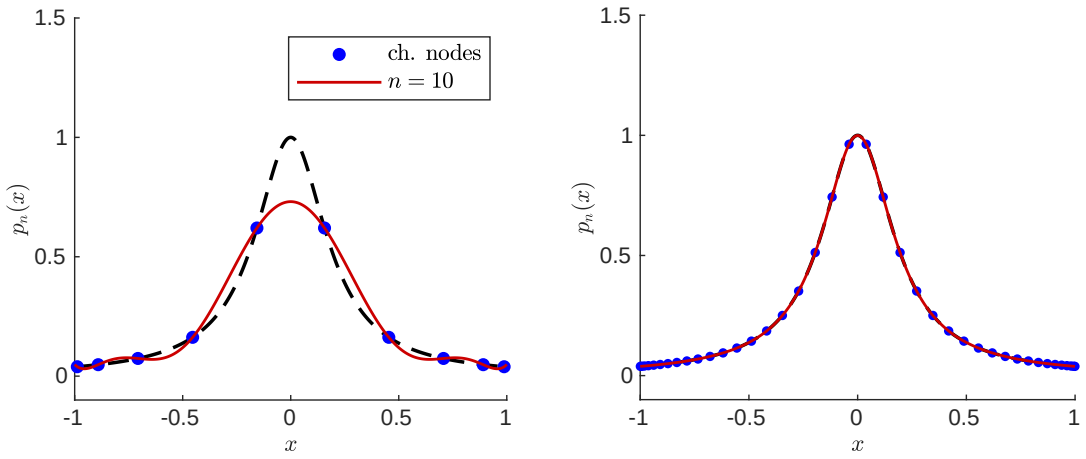


Figure 2: Max. error of the interpolating polynomial for Runge's example, with equally spaced nodes (black) and Chebyshev nodes (red). One converges; the other does not.

A better choice of nodes: However, by choosing different points so that ω_n is smaller, we can optimize the size of ω_n enough that the interpolant works; this is the basis of **Chebyschev interpolation**. The nodes (for n points) are

$$x_j = \cos\left(\frac{2j+1}{2n}\pi\right), \quad j = 0, \cdots, n-1.$$

As shown in the plot below ($n = 10$ and $n = 50$), the error is now well-behaved!



Thus the problem is not inherent to high-degree interpolants; a smarter choice of points can work. The interesting theory of this will be explored later.

Key point: The lesson here is that **equally spaced interpolants** of high degree can be disastrous (and should be avoided if possible). However, with the right choice of points, high degree interpolants can work.

5 Newton form

The idea here is to build up the polynomial inductively, adding one point at a time. Define

$$p_k = \text{interpolating polynomial through } (x_0, y_0), \cdots (x_k, y_k).$$

The $k = 0$ case is trivial, since p_0 is just a constant function:

$$p_0(x) = y_0.$$

Now suppose p_{k-1} is known. We wish to add the point (x_k, y_k) and adjust p_{k-1} so that it also passes through the new point, adding one to the degree. Write p_k in the form

$$p_k = p_{k-1} + c_k(x - x_0) \cdots (x - x_{k-1}).$$

for a constant c_k. The added term does not 'spoil' the previous work, since

$$p_k(x_i) = p_{k-1}(x_i) \text{ for } 0 \leq i < k.$$

Thus we need only choose c_k so that $p_k(x_k) = y_k$:

$$c_k = \frac{y_k - p_{k-1}(x_k)}{(x_k - x_0) \cdots (x_k - x_{k-1})}.$$

This inductively constructs p_k in 'Newton' form

$$p_n = c_0 + c_1(x - x_0) + c_2(x - x_1)(x - x_0) + \cdots + c_k(x - x_0) \cdots (x - x_{k-1})$$

or more succinctly (with $\prod_{i=0}^{-1} \cdots$ understood to be 1)

$$p_n = \sum_{j=0}^{n} c_j \prod_{i=0}^{j-1} (x - x_i).$$

Notice that c_j depends only on the points up to x_j. Some nice properties:

- This form is efficient to evaluate by using 'Horner's method'. We factor out each $(x - x_j)$ to put it in nested form, e.g. for $n = 3$:

  $$p_n = c_0 + (x - x_0)\Big(c_1 + (x - x_1)\big(c_2 + (x - x_2)(c_3)\big)\Big)$$

  and then evaluate from inside out, saving many multiplications of the product terms.

- By the construction, adding a new point to the interpolation is easy because the coefficients already computed do not change. Given p_n, adding a point x_{n+1} simply requires updating

  $$\underbrace{p_{n+1}(x)}_{\text{new}} = \underbrace{p_n(x)}_{\text{old}} + c_{n+1} \prod_{j=0}^{n} (x - x_j)$$

  for one new coefficient c_{n+1}.

- However, we still need to find a good way to compute the coefficients c_j. It turns out there is a rather nice method (fast and simple!) for doing so.

5.1 Divided differences

Here is the technique for computing the Newton coefficients. We define **divided differences** inductively as follows (using square brackets to distinguish from regular function evaluation):

$$f[x_i] = f(x_i)$$
$$f[x_{i-1}, x_i] = \frac{f[x_i] - f[x_{i-1}]}{x_i - x_{i-1}}$$
$$f[x_{i-2}, x_{i-1}, x_i] = \frac{f[x_{i-1}, x_i] - f[x_{i-2}, x_{i-1}]}{x_i - x_{i-2}}$$

and in general

$$f[x_{i-j}, x_{i-j+1}, \cdots, x_{i-1}, x_i] = \frac{f[x_{i-j+1}, \cdots, x_{i-1}, x_i] - f[x_{i-j}, x_{i+1}, \cdots, x_{i-1}]}{x_i - x_{i-j}} \tag{10}$$

for $0 \le i < j \le n$. The square brackets indicate that the divided difference is a function of the nodes between x_{i-j} and x_i and the f-data, so it keeps track of the dependence.

Some shorthand is useful. Define the 'first differences'[4]

$$(Df)_i = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}, \qquad i \ge 1,$$

and then the second differences

$$(D^2 f)_i = \frac{(Df)_i - (Df)_{i-1}}{x_i - x_{i-2}}, \qquad i \ge 2,$$

and in general the j-th differences

$$(D^j f)_i = \frac{(D^{j-1} f)_i - (D^{j-1} f)_{i-1}}{x_i - x_{i-j}}, \qquad i \ge j, \tag{11}$$

That is, each new set is obtained by taking successive differences (adjacent indices) and dividing by differences in the x's with indices separated by j.

This definition is just short for the divided difference from x_{i-j} to x_i:

$$(D^j f)_i = f[x_{i-j}, \cdots, x_i].$$

It is not hard (but a little tedious) to show the following:

Theorem (Newton form of the interpolant): The polynomial interpolating f at the $n+1$ nodes $x_0, \cdots, x_n$ in Newton form is given by

$$p_n(x) = f(x_0) + \sum_{j=1}^{n} f[x_0, x_1, \cdots x_j] \prod_{i=0}^{j-1} (x - x_i). \tag{12}$$

Note that $f[x_0, x_1, \cdots, x_j] = (D^j f)_j$ (the first element in the list of values of $D^j f$)

[4]The operator D here is an example of a 'backward difference', usually denoted ∇. It is a discrete version of the derivative, and the divided differences are analogous to repeated applications of the derivative.

Fortunately, a table makes the pattern clear. **Example:** Let

$$f(x) = x^3 - 2x^2 + 1, \qquad x_i = 0, 1, 2, 3.$$

The divided differences are best arranged in a table, where each one depends on the values to the left/upper-left in the previous column. Showing both types of notation, it looks like:

i	x_i	f	Df	$D^2 f$	$D^3 f$
0	0	$f[x_0]$			
1	1	$f[x_1]$	$f[x_0, x_1]$		
2	2	$f[x_2]$	$f[x_1, x_2]$	$f[x_0, x_1, x_2]$	
3	3	$f[x_3]$	$f[x_2, x_3]$	$f[x_1, x_2, x_3]$	$f[x_0, x_1, x_2, x_3]$.

The Newton coefficients are then the diagonal entries.

It's easiest to think of $D^j f$ as a column vector (with index $i = j, j+1, \cdots, n$), and the next column $D^{j+1} f$ is obtained by taking successive differences of the previous column. (Note that the correct x-values also have to be used; this is clear once you calculate a few).

For the example,

i	x_i	f	Df	$D^2 f$	$D^3 f$
0	0	1			
1	1	0	-1		
2	2	1	1	1	
3	3	10	9	4	1

This gives

$$(Df)_1 = f[x_0, x_1] = -1, \quad (D^2 f)_2 = f[x_0, x_1, x_2] = 1, \quad (D^3 f)_3 = f[x_0, x_1, x_2, x_3] = 1$$

so the Newton form of the interpolating polynomial is

$$p_3(x) = 1 - x + x(x-1) + x(x-1)(x-2).$$

Note that p_3 and f are the same function (why?).

Practical note: If we only need to compute the entries for the interpolant, the algorithm can be improved to use only one column of space for the divided differences by overwriting entries into the same column at each step. For the example it would look like

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ 10 \end{bmatrix} \to \begin{bmatrix} 1 \\ -1 \\ 1 \\ 9 \end{bmatrix} \to \begin{bmatrix} 1 \\ -1 \\ 1 \\ 4 \end{bmatrix} \to \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}.$$

This amounts to running the formula (11) with (careful) overwriting.

5.2 Proof of Lagrange error formula

The strategy is to define an auxiliary function q that has zeros at the $n+1$ interpolation points and x, then use the mean value theorem repeatedly to conclude that $q^{(n+1)}$ has one zero - this will be the η_x.

Theorem (Lagrange error formula): Suppose $f \in C^{n+1}([a,b])$ with the $n+1$ nodes

$$x_0 < x_1 < \cdots < x_n$$

contained in $[a,b]$. Let $p_n(x)$ be the interpolating polynomial. Then for $x \in [a,b]$,

$$f(x) = p_n(x) + E(x)$$

where the error can be written (in the 'Lagrange form')

$$E(x) = \frac{f^{(n+1)}(\eta_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j)$$

for some η_x (depending on x) in $[a,b]$.

Proof. Fix a value x in the interval $[a,b]$ (not one of the x_j's) and define

$$q(y) = f(y) - p_n(y) + (p_n(x) - f(x))r(y) \qquad r(y) := \prod_{j=0}^{n} \frac{y - x_j}{x - x_j}.$$

Note that r has zeros at the x_j's and $r(x) = 1$, so it is easy to check that

$$q(x_j) = 0 \text{ for } j = 0, \cdots, n \text{ and } q(x) = 0.$$

Then by repeated application of the mean value theorem, we conclude that q' has $n+1$ zeros (one each between zeros of q) and so on to find a point η_x such that

$$q^{(n+1)}(\eta_x) = 0.$$

Now note that since p_n has degree n, its $n+1$-th derivative is zero, so

$$q^{(n+1)}(y) = f^{(n+1)}(y) + (p_n(x) - f(x))r^{(n+1)}(y).$$

But $r(y)$ is a polynomial of degree $n+1$, so only its leading term contributes to the formula above. This is easy to calculate:

$$r(y) = \left(\prod_{j=0}^{n} \frac{1}{x - x_j} \right) y^{n+1} + (\text{deg. n}) \implies r^{(n+1)}(y) = (n+1)! \prod_{j=0}^{n} \frac{1}{x - x_j}.$$

Plugging in η_x and rearranging then gives the desired formula. □

6 Hermite interpolation (briefly)

Suppose we are instead given nodes $x_0, x_1, \cdots, x_n$ and data

$$f_j = f(x_j), \quad f'_j = f'(x_j), \qquad j = 0, \cdots n.$$

With the function value and derivative specified, there are now $2n+2$ pieces of data to use. The **Hermite interpolating polynomial** is the unique polynomial of degree $2n+1$ that agrees with f and f' at the given nodes. That is,

$$H(x_j) = f_j, \; H'(x_j) = f'_j, \quad j = 0, \cdots n.$$

Lemma (uniqueness): The Hermite interpolating polynomial H_{2n+1} (exists and) is unique. (Proof: similar to the Lagrange proof; see HW)

Warning: The 'Hermite *interpolating* polynomial' is different from the 'Hermite polynomial', which is one of a set of orthogonal polynomials that we will study later (unrelated!).

The construction can be done in two ways. One way is to use an analogue of the Lagrange basis, writing the polynomial as

$$H_{2n+1}(x) = \sum_{i=0}^{n} f_i h_i(x) + \sum_{i=0}^{n} f'_i \hat{h}_i(x)$$

where h_j and $\hat{h}_j$ are suitably defined (h_i and h'_i vanish at the x_j's except for $h_i(x_i) = 1$ etc.).

A much more elegant approach uses divided differences. The rule is:

- Let $z_0, z_1, z_2, \cdots, z_{2n+1}$ be the sequence $x_0, x_0, x_1, x_1, \cdots x_n, x_n$
- Compute 'divided differences' using the z_j's (and $f_j = f(z_j)$ as before)
- Replace 'divide by zero' results with derivatives, e.g. since $z_0 = z_1 = x_0$,

$$f[z_0, z_1] = \frac{f(z_1) - f(z_0)}{z_1 - z_0} \to f'(x_0)$$

This means the divided difference table looks the same, but every other entry in the Df column of first differences is f' instead of f. The rest of it is calculated the usual way.

One then obtains (rather miraculously), the Newton form

$$H_{2n+1}(x) = \sum_{i=0}^{2n+1} f[z_0, \cdots, z_i] \prod_{j=0}^{i-1} (x - z_j)$$

which will look like

$$f(x_0) + f'(x_0)(x - x_0) + c(x - x_0)^2 + d(x - x_0)^2(x - x_1) + \cdots$$

For example, let us compute the cubic Hermite interpolant for the data

$$f(-1) = 2, \quad f'(-1) = \boxed{\text{-1}}, \quad f(1) = 0, \quad f'(1) = \boxed{3}.$$

The divided difference table is (with the derivatives in boxes and diagonal entries used for $p(x)$ in red as before)

i	z_i	$f[z_i]$	$f[z_i, z_{i-1}]$	$\cdots$	$\cdots$
0	-1	2			
1	-1	2	$\boxed{\text{-1}}$		
2	1	0	-1	0	
3	1	0	$\boxed{3}$	2	1

so the Hermite interpolant is

$$p(x) = 2 - (x + 1) + 0 \cdot (x + 1)^2 + 1 \cdot (x + 1)^2(x - 1).$$

Remark (accuracy): Typically, derivative information greatly improves the quality of the interpolant (the disadvantage: we need to know the derivative). It can be shown that if p_{2n+1} is the Hermite interpolant for the Runge example with $n + 1$ equally spaced points (from earlier), then p_{2n+1} does converge to f.