Math 361S Lecture Notes
Interpolation

Jeffrey Wong*

February 12, 2020

Contents

*Edited by Holden Lee

1 Polynomial interpolation

1.1 Background: Facts about polynomials

Given an integer $n \geq 1$, define $\mathcal{P}_n$ to be the space of polynomials with real coefficients of degree at most n. That is,

$$p(x) \in \mathcal{P}_n \iff p(x) = a_0 + a_1 x + \cdots + a_n x^n, \quad a_i \in \mathbb{R}^n.$$

Polynomials can be added or multiplied by scalars, so $\mathcal{P}_n$ is a vector space. There are $n+1$ independent coefficients, which means that

$$\dim \mathcal{P}_n = n + 1.$$

A **basis** for $\mathcal{P}_n$ consists of $n+1$ polynomials that span $\mathcal{P}_n$. There are many choices, the simplest of which are the 'monomials'

$$1, x, x^2, \ldots, x^n.$$

There are plenty of other bases, however, each with different properties; we will make use of this freedom to choose a basis for numerical methods.

It is useful to recall that a non-zero polynomial of degree n always has exactly n **complex** zeros (roots).

A polynomial $p \in \mathcal{P}_n$ should be thought of as containing $n+1$ pieces of information. To be precise, we have the following:

Theorem: A polynomial of degree n is defined by its values on a set of $n+1$ distinct points.

Proof. Suppose $x_0, x_1, \ldots, x_n$ are points and $p, q, \in \mathcal{P}_n$ such that

$$p(x_i) = q(x_i) \text{ for } i = 0, \ldots, n.$$

Let $r = p - q$. Then r is also a polynomial of degree n and

$$r(x_i) = 0 \text{ for } i = 0, \ldots, n.$$

Then r has degree n but has $n+1$ zeros, which is impossible unless $r(x) = 0$, i.e. $p = q$. □

1.2 The basic interpolation problem

Consider a set of $n+1$ points (x, y),

$$(x_0, y_0),\ (x_1, y_1),\ \ldots,\ (x_n, y_n).$$

The x-values are called the **abscissas** or **nodes**. The y-values are assumed to come from some underlying function f, i.e.

$$y_i = f(x_i),$$

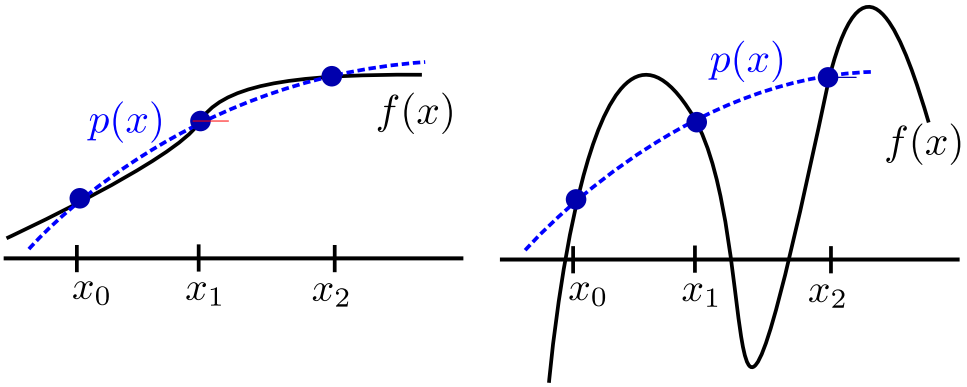


Figure 1: Interpolating polynomial for data at three abscissas (x_0, x_1, x_2) and two possible functions $f(x)$. Given three points, $p(x)$ may not be a good estimate of f (right) - the interpolant cannot know what f does between the data points.

but the nature of the function $f(x)$ may be unknown.

The goal of **interpolation** is to construction a simple function p that passes through ('interpolates') the data

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$$

and produces a useful approximation to the function $f(x)$. There are a number of different 'simple functions' to try; here we focus on the simplest choice (polynomials).

Definition: The **interpolating polynomial** for a set of $n+1$ points

$$(x_0, y_0), \ldots, (x_n, y_n)$$

(or for a function f through points $x_0, \ldots, x_n$) is the unique polynomial $p \in \mathcal{P}_n$ such that

$$p(x_i) = y_i, \qquad i = 0, 1, \ldots, n.$$

The uniqueness is a consequence of the previous theorem. Existence will be settled by constructing the interpolant in the next section. We will see that there are several methods for constructing the interpolating polynomial, which all give the same $p(x)$ in different forms.

1.3 General considerations

Given some basis $\phi_0, \cdots \phi_n$ for $\mathcal{P}_n$, we may represent p in the form

$$p = c_0\phi_0 + \cdots + c_n\phi_n.$$

Finding the interpolant then amounts to finding c_i's such that

$$y_i = c_0\phi_0(x_i) + \cdots + c_n\phi_n(x_i), \quad i = 0, \ldots, n$$

which is a set of $n+1$ linear equations for $n+1$ unknowns. To be of practical use, we should choose the basis so that the expression for p has nice properties, such as

- The ϕ_i's are easy to construct
- The c_i's are easy to compute
- The computations are numerically stable (safe against round off errors)
- $p(x)$ is easy to evaluate once the c_i's and ϕ_i's have been found
- The approximation is 'flexible' (it can be adjusted if the data changes).

The simplest choice for a basis is the set of monomials:

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

Writing out the equations (see textbook for details), we obtain a system that is solvable but not in any way nice. For small values of n, things are not too bad. However, the formula is a bit of a mess, and is not useful for developing theory and of limited use in practice.

1.4 Lagrange form of the interpolant

Suppose we want the c_i's to be as simple as possible. Then, at best, we would have

$$p(x) = y_0 L_0(x) + \cdots + y_n L_n(x)$$

for basis functions $L_0, \cdots, L_n$. That is, the coefficient of the i-th basis function is just the i-th function value. This is the **Lagrange form** of the interpolating polynomial.

To construct it, observe that it suffices to find polynomials L_i such that

(i) L_i has degree n

(ii) $L_i(x_i) = 1$

(iii) $L_i(x_j) = 0$ for $j \neq i$.

Property (iii) can be satisfied by constructing a polynomial with roots at x_j for $j \neq i$ (there are n of them, so (i) is not a concern):

$$L_i(x) = c \prod_{j \neq i} (x - x_j).$$

Then, the constant c is chosen to satisfy (ii), which gives

$$L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}.$$

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

$$L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}.$$

be the i-th '**Lagrange basis polynomial**'. Then the interpolating polynomial for the points $(x_0, y_0), \cdots (x_n, y_n)$ can be expressed as

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

Proof. It is clear from the construction that L_i is a polynomial of degree n such that

$$L_i(x_j) = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}.$$

Thus p is a polynomial of degree n and, for $0 \leq j \leq n$,

$$p(x_j) = \sum_{i=0}^{n} y_i L_i(x_j) = y_j L_j(x_j) = y_j.$$

□

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

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}, \qquad L_1(x) = \frac{x - x_0}{x_1 - x_0}$$

so

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

It is not apparent that the Lagrange form is valuable in practice since the L_i's are not nice to compute. It is mainly used in theory, as the expression is easy to manipulate and the function values appear in a natural way (we'll make use of it later for differentiation and integration).

Practical note: The Lagrange form can be constructed and evaluated efficiently, but it takes some effort to derive the methods. See Section 10.3 of the book for the **barycentric formula** for evaluation.

Example: Suppose we have the following data:

x_i	y_i
0	-1
1/2	2/3
1	8/9

and wish to construct the interpolating polynomial. The result (dashed) and the function the data came from ($f(x) = x - 9^{-x}$) are shown above. To derive $p(x)$, first find the Lagrange basis polynomials, which are

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

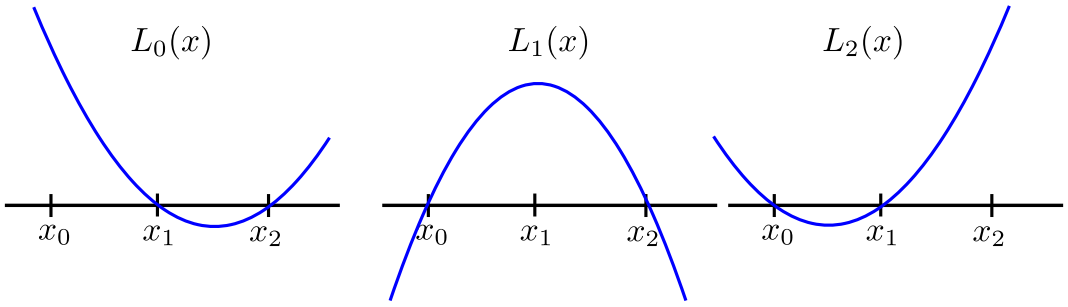
and

$$L_1(x) = -4x(x-1), \qquad L_2(x) = 2x(x-\frac{1}{2}).$$

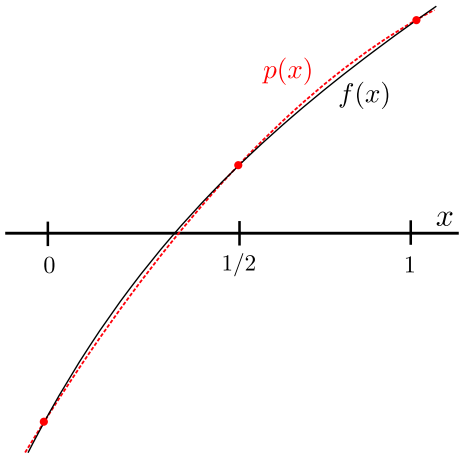
The function values at the three points are $-1, 2/3$ and $8/9$ so

$$p(x) = -2(x-\frac{1}{2})(x-1) - \frac{2}{3}x(x-1) + \frac{16}{9}x(x-\frac{1}{2}).$$

This is the unique quadratic function passing through the given data. For reference, the basis polynomials are sketched below.



The data was taken from the function $f(x) = x - 9^{-x}$ in this case. The function $f(x)$ and interpolant $p(x)$ (dashed) are shown below.



Note that even with three points, the approximation is quite close to the actual function (we study the error in detail in section 2).

1.5 Newton form

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

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

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

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

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

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

Notice that c_j depends only on the points up to x_j. This form is efficient to evaluate (using Horner's method; see homework), but 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.

1.6 Divided differences

For the detailed exposition, see Section 10.2 of the textbook. We define divided differences inductively as follows (using square brackets to distinguish from regular function evaluation):

$$f[x_i] = y_i$$
$$f[x_{i-1}, x_i] = \frac{f[x_i] - f[x_{i-1}]}{x_i - x_{i-1}}$$

and in general

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

for $0 \le i < j \le n$. Because the notation is unwieldy, let us define some shorthand:

$$\gamma_{j\ell} = f[x_{j-\ell}, x_{j-\ell+1}, \ldots x_j]$$

(the ℓ here is the distance between the first and last index). Then

$$\gamma_{j0} = y_j, \qquad \gamma_{j\ell} = \frac{\gamma_{j,\ell-1} - \gamma_{j-1,\ell-1}}{x_j - x_{j-\ell}}. \tag{2}$$

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

Theorem (Newton form of the interpolant): The polynomial interpolating f at the $n+1$ abscissas $x_0, \ldots, x_n$ can be expressed in 'Newton form',

$$p_n(x) = f(x_0) + \sum_{j=1}^{n} f[x_0, x_1, \ldots, x_j] \prod_{i=0}^{j-1} (x - x_i). \tag{3}$$

See Theorem 1 in Section 3 for the proof.

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:

i	x_i	γ_{j0}	γ_{j1}	γ_{j2}	γ_{j3}
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	$f[x_3]$	$f[x_2, x_3]$	$f[x_1, x_2, x_3]$	$f[x_0, x_1, x_2, x_3]$.

From the formula (1) or (2), it is easy to compute the divided differences in the table. Each entry is the difference between the two entries next to/above in the previous column, divided by the appropriate difference of x-values (x_j and $x_{j-\ell}$ for column ℓ). For the example,

i	x_i	γ_{j0}	γ_{j1}	γ_{j2}	γ_{j3}
0	0	1			
1	1	0	-1		
2	2	1	1	1	
3	3	10	9	4	1

This gives

$$\gamma_{11} = f[x_0, x_1] = -1, \quad \gamma_{22} = f[x_0, x_1, x_2] = 1, \quad \gamma_{33} = f[x_0, x_1, x_2, x_3] = 1$$

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 (2) with (careful) overwriting.

1.7 Evaluation

Usually, we construct the interpolant in advance and then evaluate it for a large collection of points. Thus the main measure of efficiency is the amount of work required to evaluate $p_n(x)$ **given the coefficients**.

The Newton form (3) can be efficiently evaluated using Horner's method:

$$p_n(x) = c_0 + (x - x_0)\Big(c_1 + (x - x_1)\big(c_2 + \cdots + (x - x_{n-1})c_n\big)\cdots\Big).$$

where $c_j = f[x_0, x_1, \cdots, x_j]$. This leads to a simple algorithm:
Input: x, abscissas/values $x_0, \ldots, x_n$ and $f_0, \ldots, f_n$ and coefficients $c_0, \ldots, c_n$
Output: $y = p_n(x)$ (Newton form)

$y \leftarrow c_n$
for $k = n-1, n-2, \ldots, 0$ **do**
  $y \leftarrow c_k + (x - x_k)y$
end for
return y

The algorithm takes roughly $3n$ flops, which is essentially optimal. The Lagrange form can be evaluated in a similar amount of operations, but the Newton version is simpler.

1.8 Generalizations

Divided differences can be generalized in several ways. The above method works whenever the x_i's are **distinct**.

- Typically, the x_i's are increasing, but they do not have to be! The above method also works when the x_i's are in any order. This is useful, for instance, if we need to add a point inside the interval, e.g. we have points $0, 0.5, 1$ and add a point at 0.75.

- The method can be extended to have repeated points, which is used to interpolate derivatives. See Section 10.7 of the textbook for details (e.g. **Hermite interpolation**).
- When the x_i's are equally spaced, an explicit formula can be derived.

2 Polynomial interpolation: Error analysis

Let us suppose that p_n is the interpolant of the function f, i.e.

$$p_n(x_j) = f(x_j), \qquad j = 0, \ldots, n.$$

Assume that the nodes are increasing and lie in an interval $[a, b]$

$$a < x_0 < x_1 < x_2 < \cdots < x_n < b,$$

If p_n is to be a good approximation to f, we care about how close $p_n(x)$ is to $f(x)$ for x in an interval $[c, d]$ contained in $[a, b]$ where we wish to use the interpolant. The relevant 'interpolation error' is

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

The key questions are:

- What is a bound on the interpolation error?
- How does it depend on n and the distribution of the points?
- How does the error depend on the interval?

The answers guide our strategy for approximating f in some interval by an interpolant.

Remark: Often, the interval of interest is just the interval between the endpoints x_0 and x_n. When x lies outside this interval, the estimate $p_n(x) \approx f(x)$ is called **extrapolation** rather than 'interpolation'. As we will see, extrapolation tends to do much worse, so it should be avoided unless it is absolutely needed.

2.1 Error formula

Before stating the result, it is helpful to contrast with Taylor's theorem. In this case, we are given a point x_0, a function $f \in C^{(n+1)}$ and the $n + 1$ values

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

and construct a polynomial $T_n(x)$ that approximates f:

$$f(x) = T_n(x) + R_n(x)$$

where

$$T_n(x) = f(x) + f'(x)(x - x_0) + \frac{f''(x)}{2}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n,$$

$$R_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)}(x - x_0)^{n+1}$$

and ξ_x is a number between x_0 and x.

For the interpolating polynomial, we instead have $n + 1$ values

$$f(x_0), \ldots, f(x_n)$$

at $n + 1$ **different** points. But the main result is similar:

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 every $x \in [a, b]$,

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

where the error is

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

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

See Theorem 2 in Section 3 for the proof. Note that the error looks similar to Taylor's theorem, except:

$$(x - x_0) \cdots (x - x_n) \text{ instead of } (x - x_0)^{n+1}.$$

The proof is essentially the same as for Taylor's theorem (omitted).

The theorem **does apply for extrapolation**, i.e. x can be outside of $[x_0, x_n]$.

2.2 Consequences

The error depends on three factors:

- The size of the $n + 1$-th derivative. Since ξ_x is only known to lie inside $[a, b]$, we can at best say that this factor is bounded by

$$M = \max_{x \in [a,b]} |f^{(n+1)}(x)|. \tag{4}$$

- The product
$$\omega_{n+1}(x) = \prod_{j=0}^{n}(x - x_j) \tag{5}$$
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!

- The factor $1/(n+1)!$ This factor helps as n increases; the other two factors tend to grow with n. Whether the error gets better as n grows typically depends on which of the competing factors win.

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

$$\max_{x\in I} |p_n(x) - f(x)| = \max_{x\in I} |E_n(x)| \leq \frac{M}{(n+1)!} \max_{x\in I} |\omega_{n+1}(x)|.$$

Note that even if we know x lies in some small interval I, we **cannot assume** the ξ where the $n+1$-th derivative is evaluated lies in I, so we are stuck with the bound (4) on $f^{(n+1)}$ over all of $[a, b]$.

On the other hand, ω only has to be bounded in I, so if we only care about interpolation in a small interval, say $[x_1, x_2]$ then ω only needs to be bounded inside $[x_1, x_2]$ but $f^{(n+1)}(x)$ must still be bounded in $[x_0, x_n]$ as in (4).

2.2.1 Example

Suppose we have the function
$$f(x) = e^x$$
and its values at points
$$x_1 = 0, \quad x_2 = 1$$
and wish to approximate it in the interval $[0, 1]$. The interpolating polynomial is
$$p_1 = 1 + (e - 1)x$$
and the error has the form
$$E(x) = \frac{f''(\xi_x)}{2!}x(x - 1)$$
for $\xi_x \in [0, 1]$. 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,
$$\max_{x\in[0,1]} |x(x - 1)| \leq 1/4$$
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 since p_1 just uses two function values, it should not be expected to do any better.

On the other hand, suppose we wanted to approximate e^x by p_1 (the same interpolant) in the interval $[0, 2]$. Then

$$M = \max_{x\in[0,1]} |f''(x)| = e^2, \quad \max_{x\in[0,2]} |x(x-1)| \leq 2$$

which gives the bound

$$|p_1(x) - f(x)| \leq 2e^2 \approx 15 \text{ for } x \text{ in } [0, 1].$$

Now, returning to $[0, 1]$, 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

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

Then

$$|f^{(10)}(\xi_x)| \leq e$$

as before. The product cannot be bounded precisely. We can obtain a crude estimate by noting that

$$|x - x_j| \leq 1$$

which gives

$$|E(x)| \leq \frac{e}{10!} \approx 7.5 \times 10^{-7}.$$

In fact, one can do a bit better since some of the nodes are closer to x; this is derived in general in the next section.

2.3 General error bound: equally spaced points

Let us suppose we now have points

$$a = x_0 < x_1 < \cdots < x_n = b$$

and a point $x \in [a, b]$, and the points are equally spaced with spacing h:

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

Let ω_n be defined as in (5). The error is worst when x is close to one of the endpoints (why is this plausible?). Suppose $x \in [x_0, x_1]$. Then

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

so it follows that (once you are convinced the error between x_1 and b is no worse)

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

Thus the error E_n for the degree n interpolant has the following bound:

$$|E_n(x)| \le \left(\max_{x \in [a,b]} |f^{(n+1)}(x)| \right) h^{n+1}.$$

What happens when points are added? The factor h^{n+1} goes to zero as $h \to 0$, but the derivative $f^{(n+1)}$ can grow. Whether increasing the number of points improves the error depends on how fast the n-th derivative grows. For instance, suppose

$$f(x) = 1/x, \qquad x_0 = 1, \quad x_n = 2.$$

Then for $x \in [0, 1]$.

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

The h^{n+1} decays fast enough to cancel this out:

$$|E_n(x)| \le (n+1)! \left(\frac{1}{n} \right)^{n+1} \sim \frac{\sqrt{2\pi n}}{e} e^{-n} \text{ as } n \to \infty$$

making use of Stirling's approximation

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

This cancellation is typical, leaving exponential decay. There are, however, catastrophic cases where $f^{(n+1)}$ grows too fast (see next section).

2.4 A very bad function: Runge's phenomenon

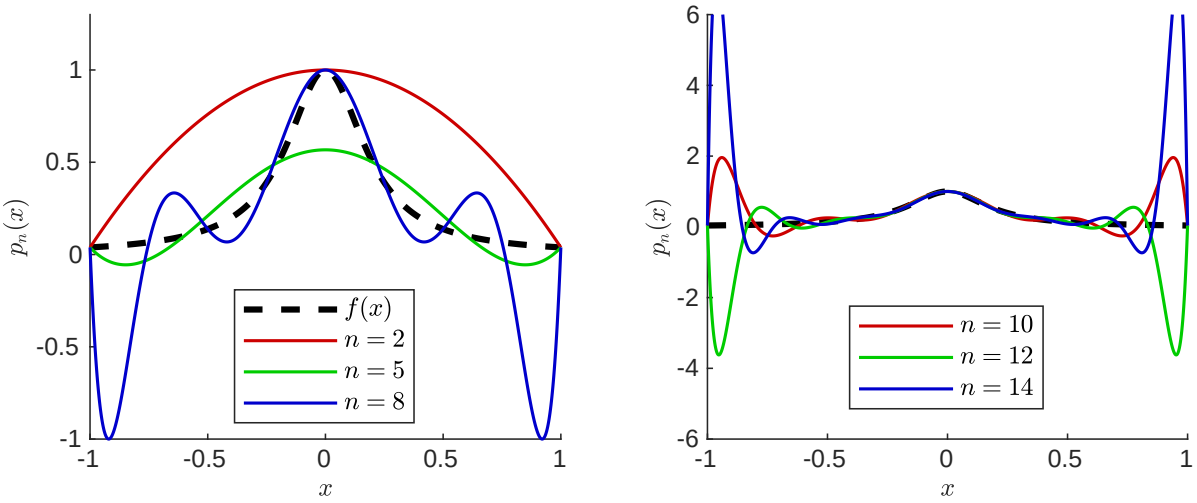
In general, it is dangerous to increase the number of nodes to try to improve the interpolant. The standard example is the seemingly nice function

$$f(x) = \frac{1}{x^2 + 25}.$$

Let $p_n(x)$ be the interpolant for nodes

$$-1 = x_0 < \cdots < x_n = 1$$

that are equally spaced. Plots of the interpolants 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 $n \to \infty$.

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.

How does this relate to the error formula? The competing factors are

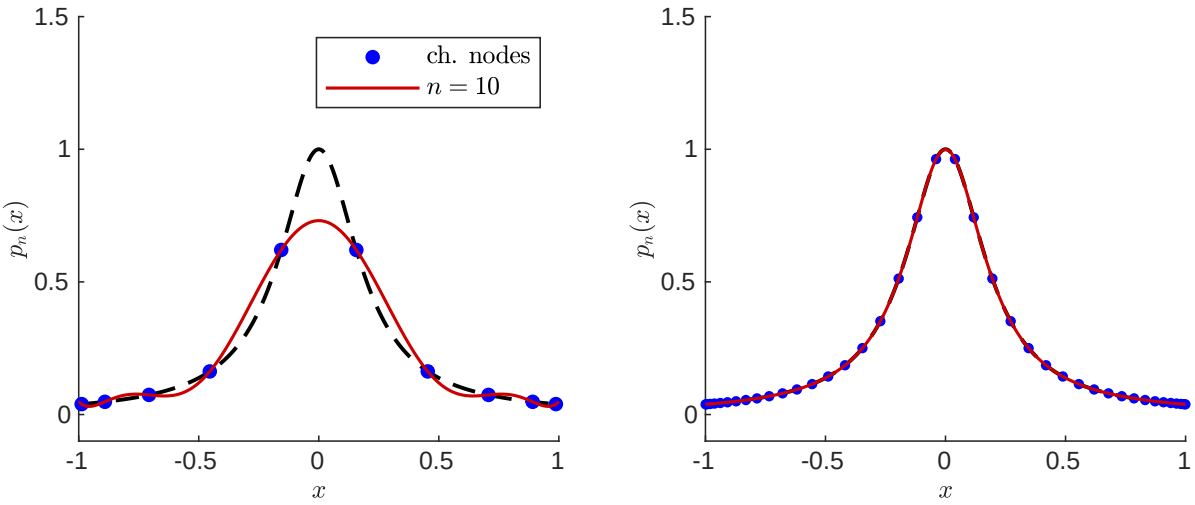
$$\frac{f^{(n+1)}(\xi)}{(n+1)!}, \qquad \omega_{n+1}(x) = \prod_{j=0}^{n}(x - x_j).$$

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.

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 **Chebyshev interpolation**. The nodes (for n points) are

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

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



Thus the problem is not the increase in the number of points itself, but the distribution of the points; a smarter choice is fine.

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.

More context: In general, the criterion for which sets of points and functions are problematic is subtle. It turns out that for any given scheme for choosing points, there is a function that causes problems, even for the Chebyshev nodes (however, such examples are so pathological for these nodes they are, for practical purposes, a good choice for all 'reasonable' functions).

For Runge's example, the problem is that f has a singularity in the complex plane that is too close to the interpolation interval, which is the right condition for when equally spaced points fail for a function f.

2.5 The other remedy: piecewise interpolation

The problem of high-degree interpolation can be avoided altogether by constructing a **piecewise approximation**. For nodes

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

we can use low-degree interpolants (degree up to 3, rarely higher) in sub-intervals. For instance, we could construct linear interpolants in $[x_i, x_{i+1}]$. Increasing the number of points never causes the problem of the previous section. The error decreases as the spacing $h = x_{i+1} - x_i$ goes to zero (by the Lagrange error formula). The piecewise linear approximation has the disadvantage that it is not differentiable at the nodes, however.

The most common choice, when the function is available, is to use a **cubic spline**. The idea here is to use cubic interpolants that use the data

$$f(x_i), f'(x_i), f(x_{i+1}), f'(x_{i+1})$$

(this is an example of Hermite interpolation, which matches function values and derivatives). These interpolants are then glued together such that the function value and derivative line up, creating a nice, smooth interpolant. See Section 11.3 of the textbook for details.

Cubic splines are a common choice when one needs to build a representation of a complex function out of data (and the function and its derivative are available to evaluate), for instance the outline of a shape in computer graphics.

3 Proofs

Theorem 1. The polynomial interpolating f at the $n+1$ abscissas $x_0, \ldots, x_n$ can be expressed in Newton form,

$$p_n(x) = \sum_{j=0}^{n} f[x_0, x_1, \ldots, x_j] \prod_{i=0}^{j-1} (x - x_i), \tag{6}$$

where the $f[x_i, \ldots, x_j]$ are recursively defined by

$$f[x_i] = f(x_i) \tag{7}$$

$$f[x_i, x_{i+1}, \ldots, x_{j-1}, x_j] = \frac{f[x_{i+1}, \ldots, x_{j-1}, x_j] - f[x_i, x_{i+1}, \ldots, x_{j-1}]}{x_j - x_i}. \tag{8}$$

Proof. We define $f[x_0, \ldots, x_j]$ as the coefficient of $\prod_{i=0}^{j-1}(x - x_i)$ in (6) and show the recursive formula holds (rather than define it by the recursive formula and show that they are the coefficients).

First, (7) holds because $f(x_i)$ is the constant polynomial interpolating $(x_i, f(x_i))$.

Let p_{n-1} be the polynomial interpolating f at $x_0, \ldots, x_{n-1}$ and q be the polynomial interpolating f at $x_1, \ldots, x_n$. We construct p_n from p_{n-1} and q, and then match coefficients to obtain the result. We claim

$$p_n(x) = p_{n-1}(x) + \frac{x - x_0}{x_n - x_0}(q(x) - p_{n-1}(x)). \tag{9}$$

The motivation is the following: $p_{n-1}(x)$ agrees with $p_n(x)$ at $x = x_0, \ldots, x_{n-1}$. We want to also make it agree at $x = x_n$. For this, we add $g(x)(q(x) - p_{n-1}(x))$ for some $g(x)$. We want $g(x) = 1$ for $x = x_n$, so that for $x = x_n$, this is just adding $q(x) - p_{n-1}(x)$, giving $p_n(x_n) = f(x_n)$. We want $g(x) = 0$ for $x = x_0$, because $p_{n-1}(x)$ already has the right value at $x = x_0$. Thus we can take $g(x) = \frac{x - x_0}{x_n - x_0}$.

Formally, let $g(x) = \frac{x - x_0}{x_n - x_0}$ and we check that:

- Both sides agree at $x = x_0$: We have $p_n(x_0) = f(x_0)$. Moreover, $g(x_0) = 0$, so

$$p_{n-1}(x_0) + g(x_0)(q(x_0) - p_{n-1}(x_0)) = p_{n-1}(x_0) + 0 = f(x_0)$$

  because p_{n-1} interpolates f at x_0.

- Both sides agree at $x = x_1, \ldots, x_{n-1}$: For $1 \le i \le n-1$, we have $p_n(x_i) = f(x_i)$. Moreover, $q(x_i) = p_{n-1}(x_i) = f(x_i)$ because both q and p_{n-1} interpolate f at x_i. Hence

$$p_{n-1}(x_i) + g(x_i)(q(x_i) - p_{n-1}(x_i)) = p_{n-1}(x_i) + 0 = f(x_i).$$

  Note it doesn't matter what $g(x_i)$ is.

- Both sides agree at $x = x_n$: We have $p_n(x_n) = f(x_n)$. Moreover, $g(x_n) = 1$, so

$$p_{n-1}(x_n) + g(x_n)(q(x_n) - p_{n-1}(x_n)) = p_{n-1}(x_n) + (q(x_n) - p_{n-1}(x_n)) = q(x_n) = f(x_n)$$

  because q interpolates f at x_n.

Now because both sides are polynomials of degree $\le n$, and they are the same at $n+1$ points, they must be the same polynomial. This shows (9).

Now consider the coefficient of x^n on both sides of (9). For the left-hand-side, note that in (6), the only summand with degree n is the last one, so the coefficient is $f[x_0, \ldots, x_n]$. For the right-hand side, the coefficient of x^n is the coefficient of x^{n-1} in $\frac{1}{x_n - x_0} q(x) - p_{n-1}(x)$, which by similar reasoning is $\frac{f[x_1, \ldots, x_j] - f[x_0, \ldots, x_{j-1}]}{x_n - x_0}$.

This shows (8) for $i = 0$, $j = n$. The equation holds for any $x_i, \ldots, x_j$ by relabeling. □

Theorem 2. 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 every $x \in [a, b]$,

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

where the error is

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

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

Proof. The idea is that we treat x as another point to interpolate f at. We create a function interpolating f at $x_0, \ldots, x_n, x$, and take its $(n+1)$th derivative.

The polynomial of degree $n+1$ interpolating f at $x_0, \ldots, x_n, x$ is

$$g(t) := p_n(t) + f[x_0, \ldots, x_n, x] \prod_{j=0}^{n} (t - x_j).$$

Now consider

$$h(t) := f(t) - g(t) = f(t) - p_n(t) - f[x_0, \ldots, x_n, x] \prod_{j=0}^{n} (t - x_j)$$

Because g interpolates f at $x_0, \ldots, x_n, x$, $h(t)$ has zeros at $n+2$ points, namely $x_0, \ldots, x_n, x$.

We use the following fact: if $h \in C[a, b]$ has k zeros in an interval $[a, b]$, then h' has at least $k - 1$ zeros in (a, b). This follows from Rolle's Theorem: if $h(c) = h(d) = 0$, then there is $\xi \in (c, d)$ such that $h'(\xi) = 0$.

Hence by induction $h^{(n+1)}$ as at least 1 zero in $[a, b]$, say $h^{(n+1)}(\xi_x) = 0$. Because the $(n + 1)$th derivative of a degree-n polynomial is 0,

$$h^{(n+1)}(t) = f^{(n+1)}(t) - 0 - (n + 1)! f[x_0, \ldots, x_n, x].$$

Then plugging in $t = \xi_x$ and rearranging gives

$$f[x_0, \ldots, x_n, x] = \frac{1}{(n + 1)!} f^{(n+1)}(\xi_x).$$

Because g interpolates f at x, we have

$$f(x) = p_n(x) + f[x_0, \ldots, x_n, x] \prod_{j=0}^{n} (x - x_j) = \frac{1}{(n + 1)!} f^{(n+1)}(\xi_x) \prod_{j=0}^{n} (x - x_j)$$

□

4 Hermite interpolation (optional)

Interpolants can also be constructed that match the derivatives of the function values at the nodes. To simplify matters, we will consider only one important case, **Hermite interpolation**, in which we construct a polynomial p such that

$$p(x_j) = f(x_j), \quad p'(x_j) = f'(x_j), \qquad j = 0, 1, 2, \ldots, n. \tag{10}$$

It is possible to generalize to allow any number of derivatives at each point. The main theorem is that

Theorem. Given a function f and its derivatives at x_j for $j = 0, 1, \cdots n$, there is a unique polynomial p of degreee (at most) $2n + 1$ such that (10) holds.

Note that the degree is $2n + 1$. In general, the degree will one less than the number of given conditions (here there are two per node, for a total of $2n + 2$). The error formula is

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

which is the same as the Lagrange formula for a set of $2n + 2$ points but with a duplicate set of $n + 1$ points (it is possible to make sense of this coincidence via some limits).

There is an Hermite basis analogous to the Lagrange basis but it is tedious to construct. Fortunately, Hermite interpolants are easy to compute via divided differences. We define a set of $2n + 2$ points

$$z_{2i} = z_{2i+1} = x_i, \qquad 0 \le i \le n$$

i.e. the new z-points are $x_0, x_0, x_1, x_1, x_2, x_2, \ldots$. Now we compute the divided differences $f[z_i, \ldots, z_{i+j}]$ the same way as before, except that if there is a division by zero we replace the quotient with a derivative:

$$f[z_{2i}, z_{2i+1}] = f'(x_i) \text{ instead of } \frac{f[z_{2i+1}] - f[z_{2i}]}{z_{2i+1} - z_{2i}}.$$

This only changes the first (non-trivial) column; the rest of the divided differences work as before. The result is that

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

Again, this is the same as before but with a set of $2n + 2$ points that have repeats. 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	$\boxed{-1}$		
2	1	0	-1	0	
3	1	0	$\boxed{3}$	2	1

so the Hermite interpolant is

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

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.

The high degree is still undesirable, however; it is more common to use the piecewise method described in subsection 2.5 (cubic interpolants in intervals between points).