Lecture 4: Constructing Finite Difference Formulas

1.7 Application: Interpolants for Finite Difference Formulas

The most obvious use of interpolants is to construct polynomial models of more complicated functions. However, numerical analysts rely on interpolants for many other numerical chores. For example, in a few weeks we shall see that common techniques for approximating definite integrals amount to exactly integrating a polynomial interpolant. Here we turn to a different application: the use of interpolating polynomials to derive finite difference formulas that approximate derivatives, the use those formulas to construct approximations of differential equation boundary value problems.

1.7.1 Derivatives of Interpolants

Theorem 1.3 from the last lecture showed how well the interpolant \( p_n \in \mathcal{P}_n \) approximates \( f \). Here we seek deeper connections between \( p_n \) and \( f \).

How well do derivatives of \( p_n \) approximate derivatives of \( f \)?

Let \( p \in \mathcal{P}_n \) denote the degree-\( n \) polynomial that interpolates \( f \) at the distinct points \( x_0, \ldots, x_n \). We want to derive a bound on the error \( f'(x) - p'(x) \). Let us take the proof of Theorem 1.3 as a template, and adapt it to analyze the error in the derivative.

For simplicity, assume that \( \tilde{x} \in \{ x_0, \ldots, x_n \} \), i.e., assume that \( \tilde{x} \) is one of the interpolation points. Suppose we extend \( p(x) \) by one degree so that the derivative of the resulting polynomial at \( \tilde{x} \) matches \( f''(\tilde{x}) \). To do so, use the Newton form of the interpolant, writing the new polynomial as

\[
p(x) + \lambda w(x),
\]

where

\[
w(x) := \prod_{j=0}^{n} (x - x_j).
\]

The derivative interpolation condition at \( \tilde{x} \) is

\[
f'(\tilde{x}) = p'(\tilde{x}) + \lambda w'(\tilde{x}),
\]

and since \( w(x_j) = 0 \) for \( j = 0, \ldots, n \), the new polynomial maintains the standard interpolation at the \( n + 1 \) interpolation points:

\[
f(x_j) = p(x_j) + \lambda w(x_j), \quad j = 0, \ldots, n.
\]
Here we must tweak the proof of Theorem 1.3 slightly. As in that proof, define the error function
\[ \phi(x) := f(x) - (p(x) + \lambda w(x)). \]

Because of the standard interpolation conditions (1.9) at \( x_0, \ldots, x_n \), \( \phi \) must have \( n + 1 \) zeros. Now Rolle’s theorem implies that \( \phi' \) has (at least) \( n \) zeros, each of which occurs strictly between every two consecutive interpolation points. But in addition to these points, \( \phi' \) must have another root at \( \hat{x} \) (which we have required to be one of the interpolation points, and thus distinct from the other \( n \) roots). Thus, \( \phi' \) has \( n + 1 \) distinct zeros on \([a, b]\).

Now, repeatedly apply Rolle’s theorem to see that \( \phi'' \) has \( n \) distinct zeros, \( \phi''' \) has \( n - 1 \) distinct zeros, etc., to conclude that \( \phi^{(n+1)} \) has a zero: call it \( \xi \). That is,

\[ 0 = \phi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - (p^{(n+1)}(\xi) + \lambda w^{(n+1)}(\xi)). \]

We must analyze
\[ \phi^{(n+1)}(x) = f^{(n+1)}(x) - (p^{(n+1)}(x) + \lambda w^{(n+1)}(x)). \]

Just as in the proof of Theorem 1.3, note that \( p^{(n+1)} = 0 \) since \( p \in \mathcal{P}_n \) and \( w^{(n+1)}(x) = (n + 1)! \). Thus from (1.10) conclude
\[ \lambda = \frac{f^{(n+1)}(\xi)}{(n + 1)!}. \]

From (1.8) we arrive at
\[ f'(\hat{x}) - p'(\hat{x}) = \lambda w'(\hat{x}) = \frac{f^{(n+1)}(\xi)}{(n + 1)!} w'(\hat{x}). \]

To arrive at a concrete estimate, perhaps we should say something more specific about \( w'(\hat{x}) \). Expanding \( w \) and computing \( w' \) explicitly will take us far into the weeds; it suffices to invoke an interesting result from 1889.

**Lemma 1.1 (Markov brothers’ inequality for first derivatives).**

For any polynomial \( q \in \mathcal{P}_n \),
\[ \max_{x \in [a,b]} |q'(x)| \leq \frac{2n^2}{b - a} \max_{x \in [a,b]} |q(x)|. \]

We can thus summarize our discussion as the following theorem, an analogue of Theorem 1.3.

---

Lemma 1.1 was proved by Andrey Markov in 1889, generalizing a result for \( n = 2 \) that was obtained by the famous chemist Mendeleev in his research on specific gravity. Markov’s younger brother Vladimir extended it to higher derivatives (with a more complicated right-hand side) in 1892. The interesting history of this inequality (and extensions into the complex plane) is recounted in a paper by Ralph Boas, Jr. on ‘Inequalities for the derivatives of polynomials,’ *Math. Magazine* 42 (4) 1969, 165–174. The result is called the ‘Markov brothers’ inequality’ to distinguish it from the more famous ‘Markov’s inequality’ in probability theory (named, like ‘Markov chains,’ for Andrey; Vladimir died of tuberculosis at the age of 25 in 1897).
Theorem 1.6 (Bound on the derivative of an interpolant).

Suppose \( f \in C^{(n+1)}[a,b] \) and let \( p_n \in \mathcal{P}_n \) denote the polynomial that interpolates \( \{(x_j,f(x_j))\}_{j=0}^n \) at distinct points \( x_j \in [a,b], \ j = 0, \ldots, n. \) Then for every \( x_k \in \{x_0, \ldots, x_n\} \), there exists some \( \xi \in [a,b] \) such that

\[
f'(x_k) - p'_n(x_k) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w'(x_k),
\]

where \( w(x) = \prod_{j=0}^n (x-x_j) \). From this formula follows the bound

\[
|f'(x_k) - p'_n(x_k)| \leq \frac{2n^2}{b-a} \left( \max_{\xi \in [a,b]} \left| f^{(n+1)}(\xi) \right| \right) \left( \max_{x \in [a,b]} \prod_{j=0}^n |x-x_j| \right).
\]

Contrast the bound (1.11) with (1.7) from Theorem 1.3: the bounds are the same, aside from the leading constant \( 2n^2/(b-a) \) inherited from Lemma 1.1.

For our later discussion it will help to get a rough bound for the case where the interpolation points are uniformly distributed, i.e.,

\[
x_j = a + j/h, \quad j = 0, \ldots, n
\]

with spacing equal to \( h := (b-a)/n \). We seek to bound

\[
\max_{x \in [a,b]} \prod_{j=0}^n |x-x_j|,
\]

i.e., maximize the product of the distances of \( x \) from each of the interpolation points. Consider the sketch in the margin. Think about how you would place \( x \in [x_0,x_n] \) so as to make \( \prod_{j=0}^n |x-x_j| \) as large as possible. Putting \( x \) somewhere toward the ends, but not too near one of the interpolation points, will maximize product. Convince yourself that, regardless of where \( x \) is placed within \([x_0,x_n]\):

- at least one interpolation point is no more than \( h/2 \) away from \( x \);
- a different interpolation point is no more than \( h \) away from \( x \);
- a different interpolation point is no more than \( 2h \) away from \( x \);
- the last remaining (farthest) interpolation point is no more than \( nh = b-a \) away from \( x \).

This reasoning gives the bound

\[
\max_{x \in [a,b]} \prod_{j=0}^n |x-x_j| \leq \frac{h}{2} \cdot h \cdot 2h \cdots nh = \frac{h^{n+1}n!}{2}.
\]

Substituting this into (1.11) and using \( b-a = nh \) gives the following result.
Corollary 1.1 (The derivative of an interpolant at equispaced points).
Suppose \( f \in C^{(n+1)}[a,b] \) and let \( p_n \in \mathcal{P}_n \) denote the polynomial that interpolates \( \{ (x_i, f(x_i)) \}_{i=0}^n \) at equispaced points \( x_j = a + jh \) for \( h = (b-a)/n \). Then for every \( x_k \in \{ x_0, \ldots, x_n \} \),
\[
|f'(x_k) - p'_n(x_k)| \leq \frac{nh^n}{n+1} \left( \max_{\xi \in [a,b]} |f^{(n+1)}(\xi)| \right).
\]

1.7.2 Finite difference formulas

The preceding analysis was toward a very specific purpose: to use interpolating polynomials to develop formulas that approximate derivatives of \( f \) from the value of \( f \) at a few points.

Example 1.3 (First derivative). We begin with the simplest case: formulas for the first derivative \( f'(x) \). Pick some value for \( x_0 \) and some spacing parameter \( h > 0 \).

First construct the linear interpolant to \( f \) at \( x_0 \) and \( x_1 = x_0 + h \). Using the Newton form, we have
\[
p_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x - x_0) = f(x_0) + \frac{f(x_1) - f(x_0)}{h} (x - x_0).
\]
Take a derivative of the interpolant:
\[
p'_1(x) = \frac{f(x_1) - f(x_0)}{h},
\]
which is precisely the conventional definition of the derivative, if we take the limit \( h \to 0 \). But how accurate an approximation is it? Appealing to Corollary 1.1 with \( n = 1 \) and \( [a,b] = [x_0, x_1] = x_0 + [0, h] \), we have
\[
|f'(x_k) - p'_1(x_k)| \leq \left( \frac{1}{2} \max_{\xi \in [x_0, x_1]} |f''(\xi)| \right) h
\]

Does the bound (1.15) improve if we use a quadratic interpolant to \( f \) through \( x_0, x_1 = x_0 + h \) and \( x_2 = x_0 + 2h \)? Again using the Newton form, write
\[
p_2(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x - x_0) + \frac{f(x_2) - f(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x_2 - x_0)}{x_2 - x_0} (x - x_0) (x - x_1)
\]
\[
= f(x_0) + \frac{f(x_1) - f(x_0)}{h} (x - x_0) + \frac{f(x_0) - 2f(x_1) + f(x_2)}{2h^2} (x - x_0) (x - x_1).
\]
Taking a derivative of this interpolant with respect to \( x \) gives
\[
p'_2(x) = \frac{f(x_1) - f(x_0)}{h} + \frac{f(x_0) - 2f(x_1) + f(x_2)}{2h^2} (2x - x_0 - x_1).
\]
Evaluate this at \( x = x_0 \), \( x = x_1 \), and \( x = x_2 \) and simplify as much as possible to get:
\[
(1.17) \quad p'_2(x_0) = -\frac{3f(x_0) + 4f(x_1) - f(x_2)}{2h}
\]
\[
(1.18) \quad p'_2(x_1) = \frac{f(x_2) - f(x_0)}{2h}
\]
\[
(1.19) \quad p'_2(x_2) = \frac{f(x_0) - 4f(x_1) + 3f(x_2)}{2h}.
\]
These beautiful formulas are right-looking, central, and left-looking approximations to \( f' \). Though we used an interpolating polynomial to derive these formulas, those polynomials are now nowhere in sight: they are merely the scaffolding that lead to these formulas. How accurate are these formulas? Corollary 1.1 with \( n = 2 \) and \([a, b] = [x_0, x_2] = x_0 + [0, 2h]\) gives
\[
(1.20) \quad |f'(x_k) - p'_2(x_k)| \leq \left( \frac{2}{3} \max_{\xi \in [x_0, x_2]} |f'''(\xi)| \right) h^2.
\]
Notice that these approximations indeed scale with \( h^2 \), rather than \( h \), and so the quadratic interpolant leads to a much better approximation to \( f' \), at the cost of evaluating \( f \) at three points (for \( f'(x_0) \) and \( f'(x_2) \)), rather than two.

**Example 1.4** (Second derivative). While we have only proved a bound for the error in the first derivative, \( f'(x) - p'(x) \), you can see that similar bounds should hold when higher derivatives of \( p \) are used to approximate corresponding derivatives of \( f \). Here we illustrate with the second derivative.

Since \( p_1 \) is linear, \( p''_1(x) = 0 \) for all \( x \), and the linear interpolant will not lead to any meaningful bound on \( f''(x) \). Thus, we focus on the quadratic interpolant to \( f \) at the three uniformly spaced points \( x_0, x_1, \) and \( x_2 \). Take two derivatives of the formula (1.16) for \( p_2(x) \) to obtain
\[
(1.21) \quad p''_2(x) = \frac{f(x_0) - 2f(x_1) + f(x_2)}{h^2},
\]
which is a famous approximation to the second derivative that is often used in the finite difference discretization of differential equations. One can show that, like the approximations \( p'_2(x_k) \), this formula is accurate to order \( h^2 \).

**Example 1.5** (*Mathematica* code for computing difference formulas).

*Code to follow...*
1.7.3 Application: Boundary Value Problems

Example 1.6 (Dirichlet boundary conditions). Suppose we want to solve the differential equation

\[-u''(x) = g(x), \quad x \in [0, 1]\]

for the unknown function \(u\), subject to the Dirichlet boundary conditions

\[u(0) = u(1) = 0.\]

One common approach to such problems is to approximate the solution \(u\) on a uniform grid of points

\[0 = x_0 < x_1 < \cdots < x_n = 1\]

with \(x_j = j/N\).

We seek to approximate the solution \(u(x)\) at each of the grid points \(x_0, \ldots, x_n\). The Dirichlet boundary conditions give the end values immediately:

\[u(x_0) = 0, \quad u(x_n) = 0.\]

At each of the interior grid points, we require a local approximation of the equation

\[-u''(x_j) = g(x_j), \quad j = 1, \ldots, n - 1.\]

For each, we will (implicitly) construct the quadratic interpolant \(p_{2,j}\) to \(u(x)\) at the points \(x_{j-1}, x_j,\) and \(x_{j+1}\), and then approximate

\[-p''_{2,j}(x_j) \approx -u''(x_j) = g(x_j).\]

Aside from some index shifting, we have already constructed \(p''_{2,j}\) in equation (1.21):

\[p''_{2,j}(x) = \frac{u(x_{j-1}) - 2u(x_j) + u(x_{j+1})}{h^2}.\]

Just one small caveat remains: we cannot construct \(p''_{2,j}(x)\), because we do not know the values of \(u(x_{j-1}), u(x_j),\) and \(u(x_{j+1})\): finding those values is the point of our entire endeavor. Thus we define approximate values

\[u_j \approx u(x_j), \quad j = 1, \ldots, n - 1.\]

and will instead use the polynomial \(p_{2,j}\) that interpolates \(u_{j-1}, u_j,\) and \(u_{j+1}\), giving

\[p''_{2,j}(x) = \frac{u_{j-1} - 2u_j + u_{j+1}}{h^2}.\]
Let us accumulate all our equations for \( j = 0, \ldots, n \):

\[
\begin{align*}
  u_0 &= 0 \\
  u_0 - 2u_1 + u_2 &= -h^2g(x_1) \\
  u_1 - 2u_2 + u_3 &= -h^2g(x_2) \\
  \vdots \hspace{1cm} & \hspace{1cm} \vdots \\
  u_{n-3} - 2u_{n-2} + u_{n-1} &= -h^2g(x_{n-2}) \\
  u_{n-2} - 2u_{n-1} + u_n &= -h^2g(x_{n-1}) \\
  u_n &= 0.
\end{align*}
\]

Notice that this is a system of \( n + 1 \) linear equations in \( n + 1 \) variables \( u_0, \ldots, u_{n+1} \). Thus we can arrange this in matrix form as

\[
\begin{bmatrix}
  1 \\
  1 & -2 & 1 \\
  1 & -2 & 1 \\
  \vdots & \ddots & \ddots \\
  1 & -2 & 1 \\
  1 & -2 & 1
\end{bmatrix}
\begin{bmatrix}
  u_0 \\
  u_1 \\
  u_2 \\
  \vdots \\
  u_{n-2} \\
  u_{n-1} \\
  u_n
\end{bmatrix} =
\begin{bmatrix}
  0 \\
  -h^2g(x_1) \\
  -h^2g(x_2) \\
  \vdots \\
  -h^2g(x_{n-2}) \\
  -h^2g(x_{n-1}) \\
  0
\end{bmatrix},
\]

where the blank entries are zero. Notice that the first and last entries are trivial: \( u_0 = u_n = 0 \), and so we can trim them off to yield the slightly simpler matrix

\[
\begin{bmatrix}
  -2 & 1 \\
  1 & -2 & 1 \\
  \vdots & \ddots & \ddots \\
  1 & -2 & 1 \\
  1 & -2 & 1
\end{bmatrix}
\begin{bmatrix}
  u_1 \\
  u_2 \\
  \vdots \\
  u_{n-2} \\
  u_{n-1}
\end{bmatrix} =
\begin{bmatrix}
  -h^2g(x_1) \\
  -h^2g(x_2) \\
  \vdots \\
  -h^2g(x_{n-2}) \\
  -h^2g(x_{n-1})
\end{bmatrix}.
\]

Solve this \((n - 1) \times (n - 1)\) linear system of equations using Gaussian elimination. One can show that the solution to the differential equation inherits the accuracy of the interpolant: the error \(|u(x_j) - u_j|\) behaves like \(O(h^2)\) as \( h \to 0 \).

**Example 1.7 (Mixed boundary conditions).** Modify the last example to keep the same differential equation

\[-u''(x) = g(x), \quad x \in [0, 1]\]
but now use mixed boundary conditions,

\[ u'(0) = u(1) = 0. \]

The derivative condition on the left is the only modification; we must change the first row of equation (1.23) to encode this condition. One might be tempted to use a simple linear interpolant to approximate the boundary condition on the left side, adapting formula (1.14) to give:

\[ \frac{u_1 - u_0}{h} = 0. \]  

(1.25)

This equation makes intuitive sense: it forces \( u_1 = u_0 \), so the approximate solution will have zero slope on its left end. This gives the equation

\[
\begin{bmatrix}
-1 & 1 & & & \\
1 & -2 & 1 & & \\
& 1 & -2 & 1 & \\
& & \ddots & \ddots & \ddots \\
& & & 1 & -2 \\
& & & & 1
\end{bmatrix}
\begin{bmatrix}
u_0 \\
u_1 \\
u_2 \\
\vdots \\
u_n-2 \\
u_{n-1}
\end{bmatrix}
= \begin{bmatrix} 0 \\
-h^2g(x_1) \\
-h^2g(x_2) \\
\vdots \\
-h^2g(x_{n-2}) \\
-h^2g(x_{n-1}) \end{bmatrix},
\]

(1.26)

where we have trimmed off the elements associated with the \( u_n = 0 \).

The approximation of the second derivative (1.22) is accurate up to \( O(h^2) \), whereas the estimate (1.25) of \( u'(0) = 0 \) is only \( O(h) \) accurate. Will it matter if we compromise accuracy that little bit, if only in one of the \( n \) equations in (1.26)? What if instead we approximate \( u'(0) = 0 \) to second-order accuracy?

Equations (1.17)–(1.19) provide three formulas that approximate the first derivative to second order. Which one is appropriate in this setting? The right-looking formula (1.17) gives the approximation

\[ u'(0) \approx \frac{-3u_0 + 4u_1 - u_2}{2h}, \]

(1.27)

which involves the variables \( u_0, u_1, \) and \( u_2 \) that we are already considering. In contrast, the centered formula (1.18) needs an estimate of \( u(-h) \), and the left-looking formula (1.19) needs \( u(-h) \) and \( u(-2h) \). Since these values of \( u \) fall outside the domain \([0, 1]\) of \( u \), the centered and left-looking formulas would not work.

Combining the right-looking formula (1.27) with the boundary condition \( u'(0) = 0 \) gives

\[ \frac{-3u_0 + 4u_1 - u_2}{2h} = 0, \]
with which we replace the first row of (1.26) to obtain

\[
\begin{pmatrix}
  -3 & 4 & -1 \\
  1 & -2 & 1 \\
  1 & -2 & 1 \\
  \vdots & \vdots & \vdots \\
  1 & -2 & 1 \\
  1 & -2 & 1 \\
\end{pmatrix}
\begin{pmatrix}
  u_0 \\
  u_1 \\
  u_2 \\
  \vdots \\
  u_{n-2} \\
  u_{n-1} \\
\end{pmatrix}
= 
\begin{pmatrix}
  0 \\
  -h^2 g(x_1) \\
  -h^2 g(x_2) \\
  \vdots \\
  -h^2 g(x_{n-2}) \\
  -h^2 g(x_{n-1}) \\
\end{pmatrix}.
\]

(1.28)

Is this $O(h^2)$ accurate approach at the boundary worth the (rather minimal) extra effort? Let us investigate with an example. Set the right-hand side of the differential equation to

\[ g(x) = \cos(\pi x/2), \]

which corresponds to the exact solution

\[ u(x) = \frac{4}{\pi^2} \cos(\pi x/2). \]

Verify that $u$ satisfies the boundary conditions $u'(0) = 0$ and $u(1) = 0$. Indeed, we used this small value of $n$ because it is difficult to see the difference between the exact solution and the approximation from (1.28) for larger $n$. Instead, compute maximum error at the interpolation points,

\[ \max_{0 \leq j \leq n} |u(x_j) - u_j| \]

Figure 1.10 compares the solutions obtained by solving (1.26) and (1.28) with $n = 4$. Clearly, the simple adjustment that gave the $O(h^2)$ approximation to $u'(0) = 0$ makes quite a difference! This figure shows that the solutions from (1.26) and (1.28) differ, but plots like this are not the best way to understand how the approximations compare as $n \to \infty$. Instead, compute maximum error at the interpolation points,

\[ \max_{0 \leq j \leq n} |u(x_j) - u_j| \]

Figure 1.10: Approximate solutions to $-u''(x) = \cos(\pi x/2)$ with $u'(0) = u(1) = 0$. The black curve shows $u(x)$. The red approximation is obtained by solving (1.26), which uses the $O(h)$ approximation $u'(0) = 0$; the blue approximation is from (1.28) with the $O(h^2)$ approximation of $u'(0) = 0$. Both approximations use $n = 4$. 
for various values of \( n \). Figure 1.11 shows the results of such experiments for \( n = 2^2, 2^3, \ldots, 2^{12} \). Notice that this figure is a ‘log-log’ plot; on such a scale, the errors fall on straight lines, and from the slope of these lines one can determine the order of convergence. The slope of the red curve is \(-1\), so the accuracy of the approximations from (1.26) is \( \mathcal{O}(n^{-1}) = \mathcal{O}(h) \) accurate. The slope of the blue curve is \(-2\), so (1.28) gives an \( \mathcal{O}(n^{-2}) = \mathcal{O}(h^2) \) accurate approximation.

This example illustrates a general lesson: when constructing finite difference approximations to differential equations, one must ensure that the approximations to the boundary conditions have the same order of accuracy as the approximation of the differential equation itself. These formulas can be nicely constructed by from derivatives of polynomial interpolants of appropriate degree.