Lecture 24: Richardson Extrapolation and Romberg Integration

Throughout numerical analysis, one encounters procedures that apply some simple approximation (e.g., linear interpolation) to construct some equally simple algorithm (e.g., the trapezoid rule). An unfortunate consequence is that such approximations often converge slowly, with errors decaying only like h or h^2, where h is some discretization parameter (e.g., the spacing between interpolation points).

In this lecture we describe a remarkable, fundamental tool of classical numerical analysis. Like alchemists who sought to convert lead into gold, so we will take a sequence of slowly convergent data and extract from it a highly accurate estimate of our solution. This procedure is *Richardson extrapolation*, an essential but easily overlooked technique that should be part of every numerical analyst's toolbox. When applied to quadrature rules, the procedure is called *Romberg integration*.

4.3. Richardson extrapolation.

We begin in a general setting: Suppose we wish to compute some abstract quantity, x_*, which could be an integral, a derivative, the solution to a differential equation at a certain point, or something else entirely. Further suppose we cannot compute x_* exactly; we can only access numerical approximations to it, generated by some function ϕ that depends upon a mesh parameter h. We compute $\phi(h)$ for several values of h, expecting that $\phi(h) \to \phi(0) = x_*$ as $h \to 0$. To obtain good accuracy, one naturally seeks to evaluate ϕ with increasingly smaller values of h. There are two reasons not to do this: (1) often ϕ becomes increasingly expensive to evaluate as h shrinks;[†]; (2) the numerical accuracy with which we can evaluate ϕ may deteriorate as h gets small, due to rouding errors in floating point arithmetic. (For an example of the latter, try computing estimates of $f'(\alpha)$ using the formula $f'(\alpha) \approx (f(\alpha + h) - f(\alpha))/h$ as $h \to 0$.)

Assume that ϕ is infinitely continuously differentiable *as a function of* h, thus allowing us to expand $\phi(h)$ in the Taylor series

$$\phi(h) = \phi(0) + h\phi'(0) + \tfrac{1}{2}h^2\phi''(0) + \tfrac{1}{6}h^3\phi'''(0) + \cdots.$$

The derivatives here may seem to complicate matters (e.g., what are the derivatives of a quadrature rule with respect to h?), but we shall not need to compute them: they key is that the function ϕ behaves *smoothly* in h. Recalling that $\phi(0) = x_*$, we can rewrite the Taylor series for $\phi(h)$ as

$$\phi(h) = x_* + c_1 h + c_2 h^2 + c_3 h^3 + \cdots$$

for some constants $\{c_j\}_{j=1}^{\infty}$.

This expansion implies that taking $\phi(h)$ as an approximation for x_* incurs an $O(h)$ error. Halving the parameter h should roughly halve the error, according to the expansion

$$\phi(h/2) = x_* + c_1 \tfrac{1}{2}h + c_2 \tfrac{1}{4}h^2 + c_3 \tfrac{1}{8}h^3 + \cdots.$$

Here comes the trick that is key to the whole lecture: Combine the expansions for $\phi(h)$ and $\phi(h/2)$ in such a way that eliminates the $O(h)$ term. In particular, define

$$\psi(h) := 2\phi(h/2) - \phi(h)$$

[†]For example, computing $\phi(h/2)$ often requires at least twice as much work as $\phi(h)$. In some cases, $\phi(h/2)$ could require 4, or even 8, times as much work at $\phi(h)$, i.e., the expense of ϕ could grow like h^{-1} or h^{-2} or h^{-3}, etc.

$$= 2\Big(x_* + c_1\tfrac{1}{2}h + c_2\tfrac{1}{4}h^2 + c_3\tfrac{1}{8}h^3 + \cdots\Big) - \Big(x_* + c_1h + c_2h^2 + c_3h^3 + \cdots\Big)$$
$$= x_* - c_2\tfrac{1}{2}h^2 - c_3\tfrac{3}{4}h^3 + \cdots.$$

Thus, $\psi(h)$ also approximates $x_* = \psi(0) = \phi(0)$, but with an $O(h^2)$ error, rather than the $O(h)$ error that pollutes $\phi(h)$. For small h, this $O(h^2)$ approximation will be considerably more accurate.

Why stop with $\psi(h)$? Repeat the procedure, combining $\psi(h)$ and $\psi(h/2)$ to eliminate the $O(h^2)$ term. Since

$$\psi(h/2) = x_* - c_2\tfrac{1}{8}h^2 - c_3\tfrac{3}{32}h^3 + \cdots,$$

we have

$$\theta(h) := \frac{4\psi(h/2) - \psi(h)}{3} = x_* + c_3\tfrac{1}{8}h^3 + \cdots.$$

To compute $\theta(h)$, we need to compute both $\psi(h)$ and $\psi(h/2)$. These, in turn, require $\phi(h)$, $\phi(h/2)$, and $\phi(h/4)$. Usually, ϕ becomes increasingly expensive to compute as the mesh size is reduced. Thus there is some practical limit to how small we can take h when evaluating $\phi(h)$.

One could continue this procedure repeatedly, each time improving the accuracy by one order, at the cost of one additional ϕ computation with a smaller h. To facilitate generalization and to avoid a further tangle of Greek characters, we adopt a new notation: Define

$$R(j,0) := \phi(h/2^j), \qquad j \geq 0;$$
$$R(j,k) := \frac{2^k R(j,k-1) - R(j-1,k-1)}{2^k - 1}, \quad j \geq k > 0.$$

Thus, for example, $R(0,0) = \phi(h)$, $R(1,0) = \phi(h/2)$, and $R(1,1) = \psi(h)$. This procedure is called *Richardson extrapolation* after the British applied mathematician Lewis Fry Richardson, a pioneer of the numerical solution of partial differential equations, weather modeling, and mathematical models in political science. The numbers $R(j,k)$ are arranged in a triangular *extrapolation table*:

$$\begin{array}{llllll}
R(0,0) & & & & & \\
R(1,0) & R(1,1) & & & & \\
R(2,0) & R(2,1) & R(2,2) & & & \\
R(3,0) & R(3,1) & R(3,2) & R(3,3) & & \\
\cdots & \cdots & \cdots & \cdots & \ddots & \\
\uparrow & \uparrow & \uparrow & \uparrow & & \\
O(h) & O(h^2) & O(h^3) & O(h^4) & &
\end{array}$$

To compute any given element in the table, one must first determine entries above and to the left. The only expensive computations are in the first column; the extrapolation procedure itself is simple arithmetic. We expect the bottom-right element in the table to be the most accurate approximation to x_*. (This will usually be the case, but we can run into trouble if our assumption that ϕ is infinitely continuously differentiable does not hold, e.g., where floating point roundoff errors spoil what otherwise would have been an infinitely continuously differentiable procedure. In this case, rounding errors that are barely noticeable in the first column destroy the accuracy of the bottom right entries in the extrapolation table.)

4.3.1. Example: Finite difference approximation of the first derivative.

To see the power of Richardson extrapolation, consider the finite difference approximation of the first derivative. Given some $f \in C^\infty[a,b]$, expand in Taylor series about the point $\alpha \in [a,b]$:

$$f(\alpha+h) = f(\alpha) + hf'(\alpha) + \tfrac{1}{2}h^2 f''(\alpha) + \tfrac{1}{6}h^3 f'''(\alpha) + \cdots.$$

This expansion can be rearranged to give a finite difference approximation to $f'(\alpha)$:

$$f'(\alpha) = \frac{f(\alpha+h) - f(\alpha)}{h} + O(h),$$

so we define

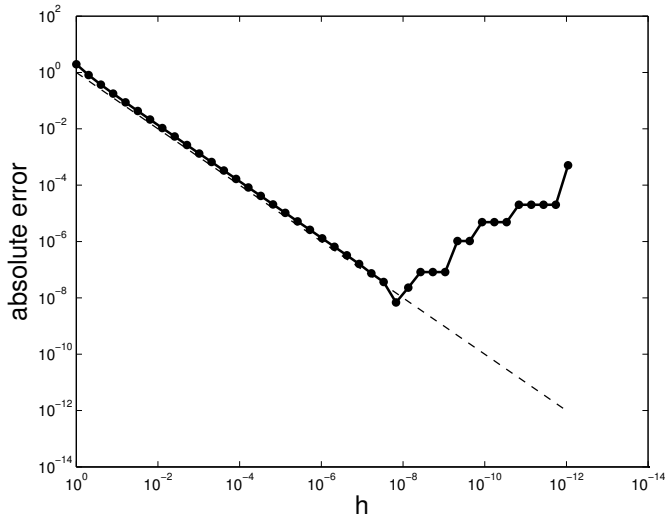
$$\phi(h) = \frac{f(\alpha+h) - f(\alpha)}{h}.$$

As a simple test problem, take $f(x) = \mathrm{e}^x$. We will use ϕ and Richardson extrapolation to approximate $f'(1) = \mathrm{e} = 2.7182818284\ldots$.

The simple finite difference method produces crude answers:

h	$\phi(h)$	error
1	4.670774270	1.95249×10^{0}
1/2	3.526814484	8.08533×10^{-1}
1/4	3.088244516	3.69963×10^{-1}
1/8	2.895480164	1.77198×10^{-1}
1/16	2.805025851	8.67440×10^{-2}
1/32	2.761200889	4.29191×10^{-2}
1/64	2.739629446	2.13476×10^{-2}
1/128	2.728927823	1.06460×10^{-2}
1/256	2.723597892	5.31606×10^{-3}
1/512	2.720938130	2.65630×10^{-3}

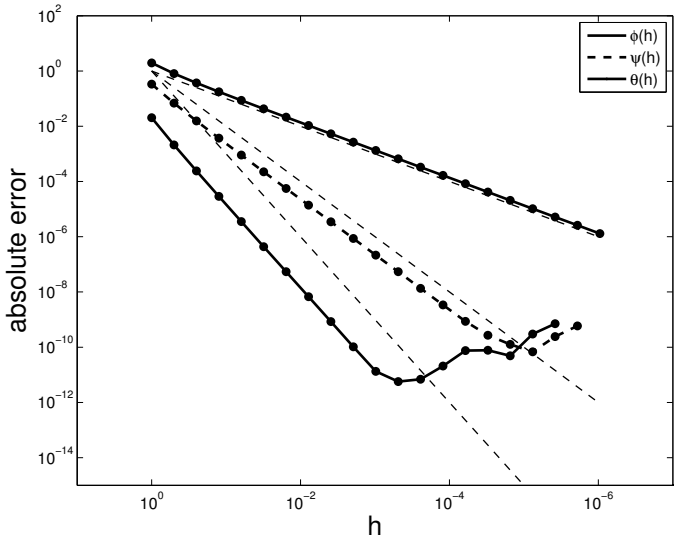
Even with $h = 1/512$ we fail to obtain three correct digits. As we take h smaller and smaller, finite precision arithmetic eventually causes unacceptable errors, as seen in the figure below showing the error in $\phi(h)$ as $h \to 0$. (The dashed line shows what perfect $O(h)$ convergence would look like.)



Yet a few steps of Richardson extrapolation on the above data reveals greatly improved solutions, five correct digits in $R(4,4)$:

$R(j,0)$	$R(j,1)$	$R(j,2)$	$R(j,3)$	$R(j,4)$
4.67077427047160				
3.52681448375804	2.38285469704447			
3.08824451601118	2.64967454826433	2.73861449867095		
2.89548016367188	2.70271581133258	2.72039623235534	2.71779362288168	
2.80502585140344	2.71457153913500	2.71852344840247	2.71825590783778	2.71828672683485

The plot below illustrates how the rounding errors made in the initial data pollute the Richardson iterates before long. (For comparison, the dashed lines indicate what an exact error of $O(h)$, $O(h^2)$, and $O(h^3)$ would like.)



4.3.2. Extrapolation for higher order approximations.

In many cases, the initial algorithm $\phi(h)$ is better than $O(h)$ accurate, and in this case the formula for $R(j,k)$ should be adjusted to take advantage. Suppose that

$$\phi(h) = x_* + c_1 h^r + c_2 h^{2r} + c_3 h^{3r} + \cdots$$

for some integer $r \geq 1$. Then define

$$R(j,0) := \phi(h/2^j) \qquad \text{for } j \geq 0$$

$$R(j,k) := \frac{2^{rk} R(j,k-1) - R(j-1,k-1)}{2^{rk} - 1} \qquad \text{for } j \geq k > 0.$$

In this case, the $R(:,k)$ column will be $O(h^{(k+1)r})$ accurate.

4.3.3. Extrapolating the composite trapezoid rule: Romberg integration.

Suppose $f \in C^\infty[a,b]$, and we wish to approximate $\int_a^b f(x)\,\mathrm{d}x$ with the composite trapezoid rule,

$$T(h) = \frac{h}{2}\Big[f(a) + 2\sum_{j=1}^{n-1} f(a+jh) + f(b)\Big].$$

One can show that if $f \in C^\infty[a,b]$, then

$$T(h) = \int_a^b f(x)\,\mathrm{d}x + c_1 h^2 + c_2 h^4 + c_3 h^6 + \cdots.$$

Now perform Richardson extrapolation on $T(h)$ with $r = 2$:

$$\begin{aligned} R(j,0) &= T(h/2^j) \qquad \text{for } j \geq 0 \\ R(j,k) &= \frac{4^k R(j,k-1) - R(j-1,k-1)}{4^k - 1} \qquad \text{for } j \geq k > 0. \end{aligned}$$

This procedure is called *Romberg integration*. In cases where $f \in C^\infty[a,b]$ (or if f has many continuous derivatives), the Romberg table will converge to high accuracy, though it may be necessary to take h to be relatively small before this is observed. When f does not have many continuous derivatives, each column of the Romberg table will still converge to the true integral, but not at the ever-improving clip we expect for smoother functions.

The significance of this procedure is best appreciated through an example. For purposes of demonstration, we should use an integral we know exactly, say

$$\int_0^\pi \sin(x)\,\mathrm{d}x = 2.$$

Start the table with $h = \pi$ to generate $R(0,0)$, requiring 2 evaluations of $f(x)$. To build out the table, compute the composite trapezoid approximation based on an increasing number of function evaluations at each step.[‡] The final entry in the first column requires 129 function evaluations, and has four digits correct. This may not seem particularly impressive, but after refining these computations through a few steps of Romberg integration, we have an approximation that is accurate to full precision.

0.000000000000						
1.570796326795	2.094395102393					
1.896118897937	2.004559754984	1.998570731824				
1.974231601946	2.000269169948	1.999983130946	2.000005549980			
1.993570343772	2.000016591048	1.999999752455	2.000000016288	1.999999994587		
1.998393360970	2.000001033369	1.999999996191	2.000000000060	1.999999999996	2.000000000001	
1.999598388640	2.000000064530	1.999999999941	2.000000000000	2.000000000000	2.000000000000	2.000000000000

Be warned that Romberg results are not always as clean as the example shown here, but this procedure is important tool to have at hand when high precision integrals are required. The general strategy of Richardson extrapolation can be applied to great effect in a wide variety of numerical settings.

MATLAB code implementing Romberg integration is shown on the next page.

[‡]Ideally, one would exploit the fact that some grid points used to compute $T(h)$ are also required for $T(h/2)$, etc., thus limiting the number of new function evaluations required at each step.

```
 function R = romberg(f, a, b, max_k)
% Compute the triangular extrapolation table for Romberg integration
% using the composite trapezoid rule, starting with h=b-a.
% f:     function name (either a name in quotes, or an inline function)
% a, b:  lower and upper limits of integration
% max_k: the number of extrapolation steps (= number of columns in R, plus one.)
%        max_k=0 will do no extrapolation.
% Example: R = romberg('sin',0,pi,1,6)

 R = zeros(max_k+1);
 for j=1:max_k+1
     R(j,1) = trapezoid(f,a,b,2^(j-1));
 end
 for k=2:max_k+1
    for j=k:max_k+1
        R(j,k) = (4^(k-1)*R(j,k-1)-R(j-1,k-1))/(4^(k-1)-1);
    end
 end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% demonstration of calling romberg.m
% based on integrating sin(x) from 0 to pi; the exact integral is 2.

 a = 0; b = pi;
 max_k = 6;
 R = romberg('sin',a,b,max_k);

% The ratio of the difference between successive entries in column k
% should be approximately 4^k.  This provides a simple test to see if
% our extrapolation is working, without requiring knowledge of the exact
% answer.  See Johnson and Riess, Numerical Analysis, 2nd ed., p. 323.

 Rat = zeros(max_k-1,max_k-1);
 for k=1:max_k-1
    Rat(k:end,k) = (R(1+k:end-1,k)-R(k:end-2,k))./(R(2+k:end,k)-R(1+k:end-1,k));
 end
```