Principles of Scientific Computing
Basic Numerical Analysis, I

Jonathan Goodman

last revised January 16, 2003

Among the most common computational tasks are differentiation, interpolation, and integration. The basic methods used for these operations are *finite difference* approximations for derivatives, *low order polynomial* interpolation, and *panel method* integration. *Taylor series* is the mathematical tool used to design methods for these three problems.

Even the errors in approximate differentiation or integration have Taylor series expansions, which tell us how the error depends on the computational *step size*, h. These *asymptotic error expansions* play a central role in computational software. A *convergence study* verifies the correctness of a code by checking that the errors depend on the computational parameters as theory says they should. *Richardson extrapolation* combines several computations with different h values to produce a more accurate overall answer. *Adaptive* algorithms adjust h until the Richardson estimate suggests that the error is below a specified tolerence. All these applications, ranging from basic to sophisticated, rest on simple manipulation of Taylor series.

Errors in approximate differentiation and integration that may be interpreted as neglecting terms in a Taylor series are called *truncation error.* We usually ignore roundoff error when discussing truncation error. Roundoff is usually negledgible compared to truncation error in practical computations that have both. An exception discussed below is the limit, due to roundoff, on the accuracy of finite difference approximations to derivatives, which is much larger than ϵ_{mach}.

For each of the problems discussed, there is a range of methods. The simplest methods are robust and easy to program. These are often adequate for casual computing. Why spend more time optimizing a code than possibly could be saved in a computation that takes the computer less than a minute? However, sophisticated methods can be much more accurate and faster. The basic operations of integration and differentiation are often at the hearts of computational algorithms whose running times are a serious concern. Time discovering and implementing more complex and accurate approximations may be repaid amply.

1 Taylor series and asymptotic expansions

Taylor series describe the behavior of a function about a point. The Taylor series if a function f about the point x is

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \cdots + \frac{h^n}{n!}f^{(n)}(x) + \cdots \quad . \tag{1}$$

For now, it may help to think of x as a fixed point and h as the varaible. The approximation formed by taking the first $n+1$ terms on the right will be written

$$f(x+h) \approx F_n(x,h) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(x) \tag{2}$$

These are progressively more accurate descriptions of the behavior of f near x. The first one, $f(x+h) = F_0(x,h) = f(x)$, just says that for small h, the value of $f(x+h)$ is not very different from the value of $f(x)$, which is to say that f is continuous at x. The next approximation, $F_1(x,h) = f(x) + hf'(x)$ is a linear function of h that takes into account the slope of the graph at x. The tangent line approximation to f near x is a better approximation than the flat line approximation, F_0. An even better approximation is $f(x+h) \approx F_2(x,h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x)$, which takes into account the curvature of the graph at x.

The errors of basic differentiation and integration methods depend on the errors in the Taylor series approximations $F_n(x,h)$. The error is the difference between $f(x+h)$ and $F_n(x,h)$. It is simpler at first to examine the difference between $F_n(x,h)$ and $F_{n-1}(x,h)$

The series (1) *converges to f* if

$$F_n(x,h) \rightarrow f(x+h) \quad \text{as } n \rightarrow \infty. \tag{3}$$

Generally speaking, if you can write a formula for $f(x)$ then the expansion will converge for sufficiently small h. According to (3) we improve the accuracy of the Taylor series approximation (2) by increasing n, taking more terms.

The accuracy also improves if we make h smaller without changing n. We will see that for small enough h, the error is roughly proportional to a power of h:

$$f(x+h) - F_n(x,h) \approx C_{n+1}h^{n+1} \; . \tag{4}$$

The power of h on the right side is the *order of accuracy*. Higher order of accuracy, larger n, generally implies less error. For example, if $h = .1$, then h^3 is ten times smaller than h^2. Unless C_3 is ten times larger than C_2, the *third order* approximation (order of accuracy $= 3$, $n = 2$) will be more accurate than the *second order* approximation ($n = 1$). Higher order approximations gain accuracy at a greater rate than low order ones. For example, (4) implies that the third order approximation improves by a factor of 8 if we reduce h be a factor of 2 while the second order approximation improves by only a factor of 4.

We can make a heuristic derivation of the error estimates (4) using the series (1):

$$f(x+h) - F_n(x,h) = \frac{1}{(n+1)!}f^{(n+1)}(x)h^{n+1} + \cdots \quad .$$

We factor out h^{n+1}, which is a common factor of every term on the right, and get:

$$\begin{aligned} f(x+h) - F_n(x,h) \quad &= \quad h^{n+1}\left(\frac{1}{(n+1)!}f^{(n+1)}(x) + \frac{1}{(n+2)!}f^{(n+2)}(x)h + \cdots\right) \\ &= \quad h^{n+1}G_{n+1}(x,h) \; , \end{aligned}$$

where G_{n+1} is the quantity in big parentheses on the top line. If the sum defining G_{n+1} converges, then we can take

$$C_{n+1} = \max_{|h| \leq h_0} |G_{n+1}(x,h)| \ ,$$

and the inequality (4) will be true.

A better tool for rigorous analysis is the Taylor series *remainder theorem*, which is the formula:

$$f(x+h) - F_n(x,h) = \frac{h^{(n+1)}}{(n+1)!} f^{(n+1)}(\xi) \ . \tag{5}$$

This ξ depends on x, h, and n in an unknown way, except that ξ is between x and $x+h$. If h is negative, then $x+h$ is to the left of x, but still ξ is between. For (5) to hold for a particular n, it is only necessary that the derivative of order $n+1$ exists for all ξ between x and $x+h$ (including at the the endpoints, x and $x+h$). For example, if $f(x) = 0$ for $x < 0$ and $f(x) = x^3$ for $x \geq 0$, then the second derivative of f exists for all x, and therefore $f(x+h) - (f(x) - hf'(x)) = \frac{h^2}{2} f''(\xi)$. However, the Taylor series for this f about $x = -1$ has all terms equal to zero, so the convergence (3) does not hold for $h = 2$. The approximation (2) does not even make sense for $n = 4$ and $x = 0$, since the fourth derivatvie of f is not defined for $x = 0$.

If $f(n+1)$ exists in the necessary range, than we may take

$$C_{n+1} = \frac{1}{(n+1)1} \max_{|h| \leq h_0} \left| f^{(n+1)}(x+h) \right| \ ,$$

and (4) will hold. The remainder theorem approach may be preferred because it does not rely on the convergence of the Taylor series as $n \to \infty$. Since our n will typically not be very big, what happens as $n \to \infty$ should be irrelevant.

2 Software tips

2.1 Write flexible and verifiable codes

The main way to verify correctness of a code it to check that it gets the right answer. For this reason, it is helpful to build codes general enough to calculate a variety of answers, some of which we already know. Maybe the problem you want to solve will strain the computer resources so that a convergence study is impossible. Hopefully, the same code can solve an easier problem, perhaps differing only in values of parameters, that for which a convergence study is possible.

2.2 Report failures

A code should not fail silently. If a procedure is unable to do what it is asked to do, it must report this in some way. Most procedures used in a computation can fail on some problems they would be exposed to. Most user want know that the answer is wrong if it is. As a principle of programming practice, every procedure should have some way to communicate failure, either to the calling procedure (preferred for serious codes) or directly to the user.

There are several ways to report failure. The simplest is just to print an error message, such as:

```
cout << "Procedure Integrate failed because n = " << n
     << " was larger than N_MAX = " << N_MAX << endl;
```

Notice that the message told the user the name of the routine, the reason for the problem, and the offending number.

3 Further reading

There several good old books on classical numerical analysis. One favorite is *Numerical Methods* by Germund Dahlquist and Åke Björk. Others are *not sure* by Gene Isaacson and Herb Keller and ????. Particularly interesting subjects are the symbolic calculus of finite difference operators and Gaussian quadrature.

There are several applications of convergent Taylor series in scientific computing. One example is the *fast multipole method* of Leslie Greengard and Vladimir Rokhlin.

4 Exercises

1. We want to study the function

$$f(x) = \int_0^1 \cos\left(xt^2\right) dt \quad . \tag{6}$$

Step 1: Write a procedure (or "method") to estimate $f(x)$ using a panel integration method with uniformly spaced points. The procedure should be well documented, robust, and clean. Robust will mean many things in future assignments. Here is means: (i) that it should use the correct number of panels even though the points t_k are computed in inexact floating point arithmetic, and (ii) that the procedure will return an error code and possibly print an error message if one of the calling arguments is out of range (here, probably just $n \leq 0$). It should take as inputs x and $n = 1/\Delta t$ and returns the approximate integral with that x and Δt value. This routine should be written so that another person could easily substitute a different panel method or a different integrand by changing a few lines of code.

Step 2: Verify the correctness of this procedure by checking that it gives the right answer for small x. We can estimate $f(x)$ for small x using a few terms of its Taylor series. This series can be computed by integrating the Taylor series for $\cos(xt^2)$ term by term. This will require you to write a "driver" that calls the integration procedure with some reasonable but not huge values of n and compares the returned values with the Taylor series approximation.

Step 3: With $x = 1$, do a convergence study to verify the second order accuracy of the trapezoid rule and the fourth order accuracy of Simpson's rule. This requires you to write a different driver to call the integration procedure with several values of n and compare the answers in the manner of a convergence study. Once you have done this for the trapezoid rule, it should take less than a minute to redo it for Simpson's rule. This is how you can tell whether you have done Step 1 well.

Step 4: Write a procedure that uses the basic integration procedure from Step 1, together with Richardson error estimation to find an n that gives $f(x)$ to within a specified error tolerance. The procedure should work by repeatedly doubling n until the estimated error, based on comparing approximations, is less than the tolerence given. This routine should be robust enough to quit and report failure if it is unable to achieve the requested accuracy. The input should be x and the desired error bound. The output should be the estimated value of f, the number of points used, and an error flag to report failure. Before applying this procedure to the panel integration procedure, apply it to the fake procedure `fakeInt.c` or `fakeInt.C`. Note that these testers have options to make the Richardson program fail or succeed. You should try it both ways, to make sure the robustness feature of your Richardson procedure works. Include with your homework, output illustrating the behavior of your Richardson procedure when it fails.

Step 5: Here is the "science" part of the problem, what you have been doing all this coding for. We want to test an approximation to f that is supposed to be valid for large x. The supposed approximation is:

$$f(x) \sim \sqrt{\frac{\pi}{8x}} + \frac{1}{2x}\sin(x) - \frac{1}{16x^2}\cos(x) + \cdots \quad . \tag{7}$$

Make a few plots showing f and its approximations using one, two and all three terms on the right side of (2) for x in the range $1 \leq x \leq 1000$. In all cases we want to evaluate f so accurately that the error in our f value is much less than the error of the approximation (2). Note that even for a fixed level of accuracy, more points are needed for large x. Why?