Principles of Scientific Computing
Basic Numerical Analysis, I

Jonathan Goodman

last revised January 16, 2003

Manipulation of Taylor series expansions is one of the most common techniques in scientific computing. Most computational methods for differentiation, integration, and solution of differential equations are based directly on Taylor series. Series expansions not only lead to computational algorithms, but they also tell us how accurate these algorithms should be and predict properties of the error that are the basis for code validation and sophisticated adaptive computational software.

In this chapter we apply the Taylor series method for three specific problems, differentiation, integration, and interpolation. Numerical differentiation is the problem of finding (as accurately as necessary) a derivative of a function, given several values of the function itself. Numerical integration is the problem of computing the integral. Interpolation is the problem of evaluating a function at some value of its arguments given function values at other values of the arguments.

For all these purposes, we use Taylor series as *asymptotic expansions*. To see what this means, consider the expansion

$$e^x = 1 + x + \frac{1}{2}x^2 + \cdots + \frac{1}{n!}x^n + \cdots \quad . \tag{1}$$

One could use this formula to compute e^2 by taking $x = 2$ and adding up enough terms to get the desired accuracy. That is not what we do here. Instead, we fix the number of terms, for example

$$e^x \approx 1 + x + \frac{1}{2}x^2 \ , \tag{2}$$

and ask about the range of x values for which this approximation is accurate enough. In real problems, Taylor series beyond the first few may be difficult or impossible to compute. Fortunately, most of the methods in this chapter apply to any function that has a Taylor series expansion, in the asymptotic sense explained below. We rarely need explicit expressions for the coefficients.

As we stated in Chapter **??**, errors that arise from taking just a few terms of an infinite Taylor series are called *truncation errors*. For example, we truncate the infinite series (1) to get the three term approximation (2). This chapter is about truncation error. We generally neglect roundoff error, since truncation error is usually larger. We will have to revisit this assumption if our problem is ill posed, if our computational algorithm is unstable, or if the step size is too small.

For each of the problems discussed we will start with simple approximations of modest accuracy and proceed to methods of increasing complexity and accuracy. The simplest methods may be 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 second? However, 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.

We work with a *step size'*, which is generically called h but may have other names in specific contexts. The approximations become more accurate as h becomes smaller. The rate of improvement is the *order of accuracy*. More accurate approximations generally lead to faster computations. For example, in estimating $\int_a^b f(x)dx$, the region of integration, $[a, b]$ is divided into *panels* of size h. The smaller h, the more panels and longer it takes the computer to process them all. If a sophisticated integration method achieves 1% error with $h = .1$ while the simple one requires $h = .01$, the sophisticated method will be, maybe, five times faster – twice the cost per panel but ten times fewer panels. Whether this factor of five speedup is worth days of extra analysis and programming depends on the situation.