NONLINEAR DYNAMICS AND CHAOS

With Applications to Physics, Biology, Chemistry, and Engineering

STEVEN H. STROGATZ

PERSEUS BOOKS
Cambridge, Massachusetts

2.8 Solving Equations on the Computer

Throughout this chapter we have used graphical and analytical methods to analyze first-order systems. Every budding dynamicist should master a third tool: numerical methods. In the old days, numerical methods were impractical because they required enormous amounts of tedious hand-calculation. But all that has changed, thanks to the computer. Computers enable us to approximate the solutions to analytically intractable problems, and also to visualize those solutions. In this section we take our first look at dynamics on the computer, in the context of ***numerical integration*** of $\dot{x} = f(x)$.

Numerical integration is a vast subject. We will barely scratch the surface. See Chapter 15 of Press et al. (1986) for an excellent treatment.

Euler's Method

The problem can be posed this way: given the differential equation $\dot{x} = f(x)$, subject to the condition $x = x_0$ at $t = t_0$, find a systematic way to approximate the solution $x(t)$.

Suppose we use the vector field interpretation of $\dot{x} = f(x)$. That is, we think of a fluid flowing steadily on the x-axis, with velocity $f(x)$ at the location x. Imagine we're riding along with a phase point being carried downstream by the fluid. Initially we're at x_0, and the local velocity is $f(x_0)$. If we flow for a short time Δt, we'll have moved a distance $f(x_0)\Delta t$, because distance = rate $\times$ time. Of course, that's not quite right, because our velocity was changing a little bit throughout the step. But over a sufficiently *small* step, the velocity will be nearly constant and our approximation should be reasonably good. Hence our new position $x(t_0 + \Delta t)$ is approximately $x_0 + f(x_0)\Delta t$. Let's call this approximation x_1. Thus

$$x(t_0 + \Delta t) \approx x_1 = x_0 + f(x_0)\Delta t.$$

Now we iterate. Our approximation has taken us to a new location x_1; our new velocity is $f(x_1)$; we step forward to $x_2 = x_1 + f(x_1)\Delta t$; and so on. In general, the update rule is

$$x_{n+1} = x_n + f(x_n)\Delta t.$$

This is the simplest possible numerical integration scheme. It is known as ***Euler's method***.

Euler's method can be visualized by plotting x versus t (Figure 2.8.1). The curve shows the exact solution $x(t)$, and the open dots show its values $x(t_n)$ at the discrete times $t_n = t_0 + n\Delta t$. The black dots show the approximate values given by the Euler method. As you can see, the approximation gets bad in a hurry unless Δt is extremely small. Hence Euler's method is not recommended in practice, but it contains the conceptual essence of the more accurate methods to be discussed next.

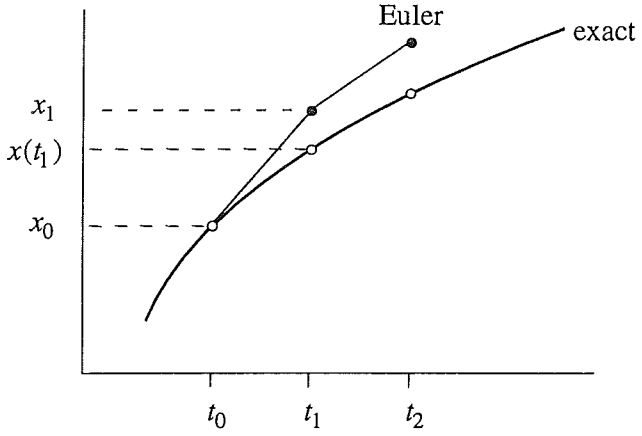

Figure 2.8.1

Refinements

One problem with the Euler method is that it estimates the derivative only at the left end of the time interval between t_n and t_{n+1}. A more sensible approach would be to use the *average* derivative across this interval. This is the idea behind the ***improved Euler method***. We first take a trial step across the interval, using the Euler method. This produces a trial value $\tilde{x}_{n+1} = x_n + f(x_n)\Delta t$; the tilde above the x indicates that this is a tentative step, used only as a probe. Now that we've estimated the derivative on both ends of the interval, we average $f(x_n)$ and $f(\tilde{x}_{n+1})$, and use that to take the *real* step across the interval. Thus the improved Euler method is

$$\tilde{x}_{n+1} = x_n + f(x_n)\Delta t \qquad \text{(the trial step)}$$

$$x_{n+1} = x_n + \tfrac{1}{2}\left[f(x_n) + f(\tilde{x}_{n+1})\right]\Delta t\,. \qquad \text{(the real step)}$$

This method is more accurate than the Euler method, in the sense that it tends to make a smaller ***error*** $E = |x(t_n) - x_n|$ for a given ***stepsize*** Δt. In both cases, the error $E \to 0$ as $\Delta t \to 0$, but the error decreases *faster* for the improved Euler method. One can show that $E \propto \Delta t$ for the Euler method, but $E \propto (\Delta t)^2$ for the improved Euler method (Exercises 2.8.7 and 2.8.8). In the jargon of numerical analysis, the Euler method is first order, whereas the improved Euler method is second order.

Methods of third, fourth, and even higher orders have been concocted, but you should realize that higher order methods are not necessarily superior. Higher order methods require more calculations and function evaluations, so there's a computational cost associated with them. In practice, a good balance is achieved by the ***fourth-order Runge–Kutta method***. To find x_{n+1} in terms of x_n, this method first requires us to calculate the following four numbers (cunningly chosen, as you'll see in Exercise 2.8.9):

$$k_1 = f(x_n)\Delta t$$
$$k_2 = f(x_n + \tfrac{1}{2}k_1)\Delta t$$
$$k_3 = f(x_n + \tfrac{1}{2}k_2)\Delta t$$
$$k_4 = f(x_n + k_3)\Delta t.$$

Then x_{n+1} is given by

$$x_{n+1} = x_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

This method generally gives accurate results without requiring an excessively small stepsize Δt. Of course, some problems are nastier, and may require small steps in certain time intervals, while permitting very large steps elsewhere. In such cases, you may want to use a Runge–Kutta routine with an automatic stepsize control; see Press et al. (1986) for details.

Now that computers are so fast, you may wonder why we don't just pick a tiny Δt once and for all. The trouble is that excessively many computations will occur, and each one carries a penalty in the form of ***round-off error***. Computers don't have infinite accuracy—they don't distinguish between numbers that differ by some small amount δ. For numbers of order 1, typically $\delta \approx 10^{-7}$ for single precision and $\delta \approx 10^{-16}$ for double precision. Round-off error occurs during every calculation, and will begin to accumulate in a serious way if Δt is too small. See Hubbard and West (1991) for a good discussion.

Practical Matters

You have several options if you want to solve differential equations on the computer. If you like to do things yourself, you can write your own numerical integration routines, and plot the results using whatever graphics facilities are available. The information given above should be enough to get you started. For further guidance, consult Press et al. (1986); they provide sample routines written in Fortran, C, and Pascal.

A second option is to use existing packages for numerical methods. The software libraries by IMSL and NAG have a wide variety of state-of-the-art numerical integrators. These libraries are well documented, reliable, and flexible, and can be found at most university computing centers or networks. The packages *Matlab*, *Mathematica*, and *Maple* are more interactive and also have programs for solving ordinary differential equations.

The final option is for people who want to explore dynamics, not computing. Dynamical systems software has recently become available for personal computers. All you have to do is type in the equations and the parameters; the program solves the equations numerically and plots the results. Some recommended programs are *Phaser* (Kocak 1989) for the IBM PC or *MacMath* (Hubbard and West