Prof. Raghuveer Parthasarathy
University of Oregon; Fall 2007
Physics 351

Numerically Solving Differential Equations: The Runge-Kutta Algorithm

DUE DATES FOR EXERCISES
P9 and P10: Friday, November 9, 2007; 5pm.

***Comment*:** Completing this assignment is a milestone – you'll find yourself with the amazing ability to numerically tackle differential equations that arise throughout physics, engineering, finance, etc. I'm exaggerating a bit, but not by much. Though there are subtleties and complications that we are not considering, the essential methodology is exactly that which you're discovering here. Be excited.

1 The Runge-Kutta Algorithm

In the last Programming Assignment we derived and examined the Verlet algorithm for numerically solving differential equations, which was valid for equations of the form $\ddot{x}(t) = G[x(t), t]$, e.g. an acceleration that depends on position and time, as arises for an undamped oscillator. Now we'll consider the more general form:

$\dot{x}(t) = f[x, t]$, plus knowledge of the initial condition: $x(t_0) = x_0$. (This doesn't seem more general, but it is, as we'll see below.)

There are many algorithms available for simulating the solution to the above differential equation. Just as for the Verlet algorithm, one can derive them by considering the Taylor series expansions of the relevant functions. Different methods employ different truncations and approximations. One quite powerful approach is the Runge-Kutta method[1], which we won't derive[2]. In it, one determines the value of x at time t_{n+1}, given the known value at time t_n, by evaluating functions at times in the interval $t_n < t < t_{n+1}$. Explicitly:

$$x_{n+1} = x_n + \frac{\Delta t}{6}\left(k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}\right), \text{ where}$$

$$k_{n1} = f(t_n, x_n)$$

$$k_{n2} = f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}\Delta t\, k_{n1}\right)$$

$$k_{n3} = f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}\Delta t\, k_{n2}\right)$$

$$k_{n4} = f(t_n + \Delta t, x_n + \Delta t\, k_{n3})$$

---

[1] The Runge-Kutta algorithm can be shown to have an error that is proportional to $(\Delta t)^5$. It is more robust than, for example, our Verlet algorithm, which had an error proportional to $(\Delta t)^4$ and other, simpler algorithms.

[2] Derivations can be found in any textbook on differential equations, for example *Elementary Differential Equations and Boundary Value Problems* by W. E. Boyce and R. C. DiPrima (Wiley, 2004).

2 Systems of first-order equations

The above $\dot{x}(t) = f[x,t]$, is a first-order differential equation, meaning that it involves a first derivative of *x*. We often care about second order equations, e.g. the differential equation corresponding to a damped oscillator: $\ddot{x}(t) = -\gamma \dot{x} - \omega_0^2 x$. (Writing more generally, $\ddot{x}(t) = h[x, \dot{x}, t]$, where *h* is some function.) This and other higher order differential equations can be turned into *systems* of first-order equations by simple substitutions: Just define a new variable, for example $y = \dot{x}$. We now have, considering our damped oscillator as an example, *two first-order* equations:

$$\begin{aligned} \dot{y}(t) &= -\gamma\, y - \omega_0^2 x \quad &= g(x, y, t) \\ \dot{x}(t) &= y &= f(x, y, t) \end{aligned} \tag{1}$$

In general, we can apply the Runge-Kutta algorithm to each of these first-order equations (together with the initial conditions for *x* and *y*). Explicitly:

$$\begin{aligned} x_{n+1} &= x_n + \frac{\Delta t}{6}\left(k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}\right) \\ y_{n+1} &= y_n + \frac{\Delta t}{6}\left(l_{n1} + 2l_{n2} + 2l_{n3} + l_{n4}\right) \end{aligned} \tag{2}$$

where

$$\begin{aligned} k_{n1} &= f(t_n, x_n, y_n) \\ k_{n2} &= f(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}\Delta t\, k_{n1}, y_n + \frac{1}{2}\Delta t\, l_{n1}) \\ k_{n3} &= f(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}\Delta t\, k_{n2}, y_n + \frac{1}{2}\Delta t\, l_{n2}) \\ k_{n4} &= f(t_n + \Delta t, x_n + \Delta t\, k_{n3}, y_n + \Delta t\, l_{n3}) \end{aligned} \tag{3}$$

and

$$\begin{aligned} l_{n1} &= g(t_n, x_n, y_n) \\ l_{n2} &= g(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}\Delta t\, k_{n1}, y_n + \frac{1}{2}\Delta t\, l_{n1}) \\ l_{n3} &= g(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}\Delta t\, k_{n2}, y_n + \frac{1}{2}\Delta t\, l_{n2}) \\ l_{n4} &= g(t_n + \Delta t, x_n + \Delta t\, k_{n3}, y_n + \Delta t\, l_{n3}) \end{aligned} \tag{4}$$

At first glance this looks like a mess, but notice that the form of most of the terms is very similar. Moreover, the protocol for evaluating Equation 2 to figure out the "next steps" of *x* and *y* is straightforward. We know *x* and *y* at time t_n; we use these and the functions *f* and *g* to evaluate Equations 3 and 4 to calculate k_{n1} and l_{n1}, we use these to calculate k_{n2} and l_{n2}, etc., until we have all the pieces with which to make use of Equation 2.

3 The damped oscillator

For our damped oscillator equation, "*g* " is only an explicit function of *x* and *y*, not t, i.e. $g(x, y, t) = g(x, y)$. Similarly, "*f* " is only a function of *y*, i.e. $f(x, y, t) = f(y)$. You can convince yourself that if we had a *driven* oscillator, *g* would be an explicit function of *x*, *y*, and t, while *f* would be unchanged. In the exercise P9, below, you will use the Runge-Kutta method to simulate the motion of a damped harmonic oscillator

Exercise

(P9, 10 pts.**) The damped oscillator.** Write a MATLAB program to implement the Runge-Kutta method to simulate the motion of a mass-on-a-spring, with spring constant k = 0.1 N/m, mass m = 1.0 kg, and damping coefficient b = 0.03 N s / m. Use the initial conditions $x_{(t=0)} = 6\,\text{m}$, and $\dot{x}_{(t=0)} = 0\,\text{m/s}$. Plot x vs. t over a time interval equal to 10 T, with $\Delta t = T/25$, where T is the period of the undamped oscillator (i.e. $2\pi/\omega_0$). Use symbols (e.g. circles, with plot(t,x,'o')) to plot the data. **(8 pts.)** On the same plot, graph the analytic solution to x(t) – see Problem Set 4, or its solution set, or both, in which this is calculated. **(2 pts.)**

Turn in your plot. (You don't need to turn in the MATLAB code, though you can if you want.)

Hints / Comments:

Unlike the Verlet assignment, I'm not providing a "ready-made" MATLAB program. You'll have to figure out how to write your own. I'll show you some fragments of my program (below) which may be of help. Also: Future assignments will make use of this Runge-Kutta algorithm. Don't panic, however, if you're unable to successfully write your program; I'll supply the "solution" to this exercise after it's due.

```
[... lines omitted...]
gamma = b/m;
w0 = sqrt(k/m);
[... lines omitted, calculating Deltat, Tfinal, etc. ...]
% Initial conditions
x(1) = 6.0;  % initial position, meters
y(1) = 0.0;  % initial velocity, m/s
t(1) = 0.0;  % initial time, seconds
[in the "for" loop]
    t(j) = Deltat*(j-1);
    kn1 = y(j-1);
    [... lines omitted, calculating ln1, kn2, ln2, etc...]
    kn4 = y(j-1) + Deltat*ln3;
    x(j) = x(j-1) + (Deltat/6.0)*(kn1 + 2*kn2 + 2*kn3 + kn4);
    [... lines omitted...]
```

(P10, 7 pts.**) Error.** Modify your program to calculate the error in the numerical calculation, defined as the mean-square-deviation of the numerical solution x(t) from the analytic solution x_a(t), i.e. $\chi^2 = \left\langle \left(x(t) - x_a(t) \right)^2 \right\rangle$. (The symbol χ is "chi," and χ^2 is the typical measure of error or uncertainty in many sorts of analyses.) Evaluate χ^2 for several values of Δt, and determine, e.g. by plotting both on a log-log plot, how χ^2 depends on Δt. (In other words, is χ^2 proportional to Δt, Δt^5, Δt^6, ... ?) *Read the following note!*

Note: *Exercise P10 is "above and beyond" the goals of this handout. I'm assigning it so that those "breezing through" the assignment will have something to ponder. Recall from the syllabus how these programming assignments are graded – not doing P10 will very likely have zero impact on your grade. Moreover, I won't be at all disappointed if you don't do P10. Really. I don't mind at all. Go study other things, or take a good nap.*