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

Numerically Solving Differential Equations: The Verlet Algorithm

DUE DATES FOR EXERCISES
P7: Friday, October 19, 2007; 5pm.
P8: Friday, October 26, 2007; 5pm.

Note: *I expect that most people in Phys. 351* ***have not*** *taken a course on differential equations. It's certainly not a prerequisite. Also, even if you've taken such a course, it may not be of much help, since many such courses don't discuss computational methods. So: relax, and read on...*

1 Differential Equations

As we've seen in class, physical systems can often be described by differential equations. Sometimes, as for a mass-on-a-spring, we can analytically solve the differential equation, determining, for example, the motion of the mass as a function of time, $x(t)$. This is the ideal situation – and fortunately, there are lots of important physical systems amenable to exact solutions. Sometimes, however, we cannot find an analytic solution. Then we must turn to numerical (i.e. computational) solutions.

A differential equation relates derivatives of some variables. For example:

$\ddot{x}(t) = -\frac{k}{m}x(t)$ describes the motion of a mass m connected to an ideal spring of spring constant k, neglecting gravity, friction, etc., as we saw in class. This is a "second order" differential equation, since it involves a second derivative of x. We can view this equation as a set of instructions: if we know x at some time t_0, we then know the acceleration at time t_0, from the above equation. Provided that we also know the velocity $\dot{x}$ at time t_0, we can determine from elementary kinematics where the object will be at some slightly later time.

The essence of any numerical scheme to solve differential equations consists simply of the above procedure. We can't consider a continuum of times, but we can consider known conditions at time t_0, to determine x at some nearby time t_1. We can then use this information to calculate x at some slightly later time t_2. And so on. We'll describe this more explicitly shortly.

There are **many** methods for numerically solving differential equations. They differ in their accuracy (i.e. their approximation to the true, continuum solution), their applicability to particular types of differential equations, and in the amount of computing power they require. In this lesson, we'll examine one of them. You should **go through each line of math yourself**, recopying it if it helps you understand.

Much of this discussion is from
Computational Physics, by J. M. Thijssen (Cambridge Univ. Press, Cambridge, 1999)
and from Wikipedia's article on *Verlet Integration*.

2 The Verlet Algorithm

Consider a differential equation that has the form

$\ddot{x}(t) = G[x(t), t]$ -- i.e. the function F can depend explicitly on x and t, but not, for example on $\dot{x}$. (Our above mass-spring equation satisfies this condition.) Let's Taylor expand x around time t_0:

$x(t_0 + \Delta t) = x(t_0) + \Delta t\, x'(t_0) + \frac{1}{2}\Delta t^2\, x''(t_0) + \frac{1}{6}\Delta t^3\, x'''(t_0) + O(\Delta t^4)$, where "$O$..." means "terms of order Δt^4 and higher, which we're going to ignore..." (Given the practice you've had recently, the above equation should be very comprehensible!) Let's also expand "backwards:"

$x(t_0 - \Delta t) = x(t_0) - \Delta t\, x'(t_0) + \frac{1}{2}\Delta t^2\, x''(t_0) - \frac{1}{6}\Delta t^3\, x'''(t_0) + O(\Delta t^4)$. Adding these two:

$x(t_0 + \Delta t) + x(t_0 - \Delta t) = 2x(t_0) + \Delta t^2\, x''(t_0) + O(\Delta t^4)$, from which:

$x(t_0 + \Delta t) = 2x(t_0) - x(t_0 - \Delta t) + \Delta t^2\, x''(t_0) + O(\Delta t^4)$, or, using the form of the differential equation above,

$$x(t_0 + \Delta t) = 2x(t_0) - x(t_0 - \Delta t) + \Delta t^2\, G[x(t_0), t_0] + O(\Delta t^4) \qquad \textbf{Equation 1}$$

Therefore, if we know the position at time $t_0 - \Delta t$ (i.e. one step ago) and time t_0 (now), we can determine the position at time $t_0 + \Delta t$ (i.e. one step in the future)!

We can also calculate the velocity by subtracting our forward and backward expansions:

$$v(t_0) = \frac{x(t_0 + \Delta t) - x(t_0 - \Delta t)}{2\Delta t} + O(\Delta t^2)$$

2 Implementing the Verlet Algorithm: A Simple Harmonic Oscillator

Let's apply the Verlet algorithm (Eq. 1) to a simple harmonic oscillator, e.g. a mass on a spring. The differential equation is

$\ddot{x}(t) = G[x(t), t]$, with $G[x(t), t] = -\frac{k}{m}x(t)$, as discussed above. An analytic solution can consider arbitrary k and m, but the computer requires more concreteness: Let's say k = 0.1 Newton/meter and m = 1 kg. These units set the units of x and t to be meters and seconds, respectively. (Note that only the ratio k/m matters – if you don't understand this, try considering either analytically or computationally different k and m that have the same ratio and see how x(t) looks.)

Let's consider time from t=0 seconds to t=100 seconds.

We'll need to fix some **initial conditions**. These will determine the initial two x values that form the "starting point" of our Verlet algorithm – see Equation 1. Let's say $x(t = 0) = 0, v(t = 0) = 2\ m/s$. We'll also need to fix the time step, Δt (named `Deltat` in the program below) – this can take some thought, as we'll see. For now, we'll just choose Δt = 0.5 seconds. Our first x position is $x(0) = 0$, our second is $x(\Delta t) = x(t = 0) + \Delta t\, v(t = 0) = \Delta t\, v(t = 0)$. (In determining this, we're neglecting terms of order Δt^2 and higher). Write the following program in the editor window, **thinking about what each line does,** and save it as an m-file with some name. Then, run it.

```
% verletSHO.m
% Numerical solution to the differential equation of motion for a
```

```
% simple harmonic oscillator, using the Verlet algorithm)
% m x'' = -k x, i.e. G[x(t),t] = -k x / m
%
% Raghuveer Parthasarathy
% Oct. 4, 2007

clear all
close all

x(1) = 0.0;  % initial position, meters
v(1) = 2.0;  % initial velocity, m/s
Deltat = 0.5;  % time increment, s
x(2) = x(1) + v(1)*Deltat;   %We'll explicitly write x(1), even though
               % it's zero here, in case we ever want to change
               % our initial conditions.  (Otherwise, we might get
               % confused!)
k = 0.1;  % Newtons / meter
m = 1.0;  % kilograms
Tfinal = 100.0;  % ending time, seconds
t = 0:Deltat:Tfinal;  % an array of all the time values -- starts at 0
N = length(t);  % "length" gives the number of elements in an array
for j=3:N;
    x(j) = 2*x(j-1) - x(j-2) + Deltat*Deltat*(-1.0*k/m)*x(j-1);
    v(j-1) = (x(j) - x(j-2))/ (2*Deltat);
end
v(N) = (x(N)-x(N-1))/Deltat;  % Why? Because v(N) is not
                              %set by the above For loop
figure; plot(t, x, 'ko:'); grid on;  % points and a dotted line
xlabel('Time, sec. ');
ylabel('x, meters')
```

You should find that MATLAB calculates and then plots $x(t)$ for this oscillator. Does it look like a sine function?

Exercises

[*Note*: If you've read and implemented the above text, P7 will take very little time. P8 takes a bit more work.]

(P7, 8 pts.**) Time steps.** For the differential equation above, we know the analytic solution, which I'll denote as $x_a(t)$. First, for the k, m values and initial conditions given above ($x(t=0)=0, v(t=0)=2\ m/s$), determine the equation for $x_a(t)$. Consult your lecture notes to review the use of initial conditions.

(a, 4 pts.**)** Modify the program above to also plot $x_a(t)$. Create a separate time array – you can call it `ta` -- that is `ta = 0:0.1:100`. Calculate the `xa` array. Just as in the previous Programming handout, use "hold on" to plot `xa vs ta` as a solid blue line in the same figure as the Verlet solution to the differential equation above. Turn in your plot. Do the exact and numerical solutions look similar? (By the way, the MATLAB command to take a square root is "`sqrt`," e.g. `c = sqrt(d)`.)

(b, 4 pts.**)** Modify the above program to make Deltat a user-input variable, rather than having its value fixed in the program, using the input command (see the previous handout). Run the program with Deltat = 3.0 and Deltat = 6.0 seconds, generating the numerical solution to the differential equation and also

plotting the exact solution xa(t) from part (a). For each of these, turn in your plot. Mr. K. suggests trying Deltat = 0.0001 seconds – why does he think this is a good idea? Why might it not be a good idea? (You don't have to try it – just think about it and write down your conclusions. If you do try it, be sure to look at Section 11 of the last handout.)

(P8, 8 pts.**) A swinging pendulum.** In class we showed that for small angles, θ, a swinging pendulum exhibits simple harmonic motion. You may wish to revisit your notes, and make sure you can derive this result yourself. What happens if θ is not small? We no longer have an analytic solution to the differential equation of motion, but we can numerically approximate the solution. Consider a pendulum of length 0.5 meters. Consider the initial conditions to be that $\theta(t=0)=\theta_0$ radians, where we'll test particular values of θ_0, and $\dot{\theta}(t=0)=0$ radians per second. First, write down the analytic solution to the differential equation of motion for the pendulum for these particular initial conditions *using the small angle approximation* – we'll call this solution $\theta_a(t)$. (You determined this in Problem Set 3.) Next, write a program to use the Verlet algorithm to solve the pendulum's differential equation *without* the small angle approximation – i.e. keeping the $\sin(\theta)$ intact. Graph the resulting $\theta(t)$ and $\theta_a(t)$ on the same plot. Let the time array span t=0 to `Tfinal`, where `Tfinal` is 6 times the period of oscillation of a simple pendulum. You'll have to figure out a good `Deltat` value to use. Do this for θ_0=3, 20, and 40 degrees (don't forget to convert these to radians!). Turn in the plots and your program listing.