

Physics 305: Ch. 1

Solving ODEs II: Falling bodies

1.1 Higher order equations

Last week we learned numerical methods for solving first order differential equations,

$$\frac{df(t)}{dt} = \mathcal{H}(f(t), t) \quad (1.1)$$

But Newton's laws of motion, and in fact most of the fundamental differential equations in physics, are second order equations,

$$\frac{d^2 f(t)}{dt^2} = \mathcal{G}\left(t, f(t), \frac{df(t)}{dt}\right) \quad (1.2)$$

But this equation, and other higher order equations, can be turned into a system of first order equations by introducing auxiliary variables. For definiteness, think about $F = ma$ in one dimension. But the force may depend on the position, on the velocity, and maybe even separately on time:

$$\frac{d^2 x(t)}{dt^2} = \frac{1}{m} F\left(x(t), \frac{dx(t)}{dt}, t\right) \quad (1.3)$$

But now just think about $x(t)$ and $v(t)$ as separate variables, and this is

$$\begin{aligned} \frac{dx(t)}{dt} &= v(t) \\ \frac{dv(t)}{dt} &= \frac{1}{m} F(x(t), v(t), t) \end{aligned} \quad (1.4)$$

$$(1.5)$$

You can now use the techniques we learned last week — just think of $x(t)$ and $v(t)$ as a single thing, the set of variables describing the system. In the algorithms

we learned last week, every calculation of the function we called $f(t)$ is replaced by a calculation of both x and v . For example, the very simple Euler method now just says that each of the position and velocity change by their derivative. So, inside your loop over time, you would have a code fragment something like:

```
// this is just the updating fragment!!
// eps is the time step
// F is a function you supply. In many cases it
// won't depend on all of x, v and t, so it won't
// need three arguments
xnew = x + eps * v;
vnew = v + (eps/mass) * F( x, v, t );
x = xnew;
v = vnew;
```

Note that this time it was really important that I introduced the variable `xnew`. That's because when I called the function that computes the force in this case, I still needed to know the “old” x . That is, x evaluated at the beginning of the time step.

As a general comment, going to two variables describing your system instead of one gives you many more opportunities to foul up the distinction between old values, new values, values at the midpoint of the interval You have been warned.

Similarly, the second order Runge-Kutta method works just like before, except you remember that your system is described by two variables.

```
for( t = t_start ; ... ){
  // THIS TIME I'LL ONLY WRITE COMMENTS - YOU WRITE THE CODE
  // compute xhalf = x at the midpoint of the time step
  // compute vhalf = v at the midpoint
  // (be sure you use x at the beginning, if F depends on x)
  // use vhalf to compute xnew = x at the end of the interval
  // use xhalf, vhalf and maybe even t_half to compute the
  // force, and from that vnew = v at the end of the interval.
  // now set x and v to xnew and vnew respectively, and
  // back to top of loop
```

Again, think this through before you start writing code. Pay special attention to which position— x , x_{half} or x_{new} —is used where.

One of your homework problems involves motion in two dimensions. You can handle this by an obvious extension of the above — use four variables to describe your system: x , y , v_x and v_y .

1.2 Falling bodies

Near the surface of the earth the gravitational field is approximately constant. Every body feels a force $F = -mg$, where the minus sign is because the force is downwards. If this is the only force, you can all solve for the motion exactly. But with our new tools we can do something more interesting — we can study the motion of bodies falling in the atmosphere including the effects of drag forces. Actually, we will do two approximate forms for the drag force, because for the simplest one we can still get an exact solution to check our numerical solution.

Watch out! The sign convention in these notes is that $x(t)$, the height, increases as you go up, and $v(t)$ is negative if the object is going down. Also, we take g to be a positive number.

1.2.1 Elementary approximations for drag forces

Air resistance is a complicated thing, as anyone who has ever thrown a boomerang can attest, but we will settle for some simple approximations. The simplest thing we might imagine is “linear drag”, or a drag force proportional to the velocity.

$$F_{drag} = -Cmv \quad (1.6)$$

Here C is a constant. Then the acceleration is

$$\frac{\partial v(t)}{\partial t} = -g - Cv(t) \quad (1.7)$$

Now if an object falls through a uniform medium, it will eventually reach a “terminal velocity”, where the frictional and gravitational forces cancel. Clearly this happens at $v = -g/C$. Define the terminal velocity – note that it is positive — by $v_{term} = g/C$, and the equation becomes

$$\frac{\partial v(t)}{\partial t} = -g(1 + v(t)/v_{term}). \quad (1.8)$$

You should check that you understand this, including both positive and negative velocities.

Alas, although the linear drag model is simple and appealing, it isn’t very realistic. A much more realistic simple model is to suppose that the drag force is proportional to the square of the velocity,

$$|F_{drag}| = C_2v^2 \quad (1.9)$$

Again, we can express this in terms of the terminal velocity

$$\frac{dv(t)}{dt} = -g(1 \pm v^2(t)/v_{term}^2) \quad (1.10)$$

Here we must be very careful about the sign. Clearly, the drag force is supposed to act in the opposite direction from the motion. In the formula for linear drag, we just wrote $-v$ to get this, but v^2 is always positive. One way around this is to write

$$\frac{dv(t)}{dt} = -g \left(1 + v(t)|v(t)|/v_{term}^2 \right) \quad (1.11)$$

(In coding this, remember that in C the absolute value for floating point numbers is `fabs()`.)

1.2.2 Exact solution for falling body with linear drag

We can do an exact solution for the case of linear drag:

$$\begin{aligned} \frac{dy(t)}{dt} &= v(t) \\ \frac{dv(t)}{dt} &= -g \left(1 + \frac{v(t)}{v_{term}} \right) = \frac{-g}{v_{term}} (v_{term} + v(t)) \end{aligned} \quad (1.12)$$

The key point is that the velocity equation here doesn't depend on the position. So we can just solve the first order equation for the velocity as a function of time, and then integrate the answer to get the position.

Use the same trick as in the coffee cooling problem. Let

$$\tilde{v}(t) = v_{term} + v(t) \quad (1.13)$$

(Remember my conventions: v_{term} is positive, and a downward velocity is $v < 0$.) Then

$$\frac{d\tilde{v}(t)}{dt} = \frac{-g}{v_{term}} \tilde{v}(t) \quad (1.14)$$

Which is solved by

$$\tilde{v}(t) = \tilde{v}(0) \exp\left(\frac{-g}{v_{term}}t\right) \quad (1.15)$$

or

$$v(t) = -v_{term} + (v(0) + v_{term}) \exp\left(\frac{-g}{v_{term}}t\right) \quad (1.16)$$

You should always try to think of sanity checks for your answers. Does this answer make sense? Check it for large t :

$$\lim_{t \rightarrow \infty} v(t) = -v_{term} \quad (1.17)$$

Check it for small t : At $t = 0$ we just get $v(0)$, so that's OK. For small t , Taylor expand

$$e^{\frac{-g}{v_{term}}t} \approx 1 - \frac{g}{v_{term}}t \dots \quad (1.18)$$

so, to first order in t in the case where we start from rest ($v(0) = 0$), we get

$$v(t) \approx -v_{term} + v_{term} \left(1 - \frac{g}{v_{term}} t \right) = -gt \quad (1.19)$$

To find the height, use

$$y(t) = y(0) + \int_0^t v(t') dt' \quad (1.20)$$

If $v(0) = 0$ this is

$$\begin{aligned} y(t) &= y(0) + \int_0^t v_{term} \left(\exp\left(\frac{-g}{v_{term}} t'\right) - 1 \right) dt' \\ &= y(0) + v_{term} \left(\frac{-v_{term}}{g} \exp\left(\frac{-g}{v_{term}} t'\right) - t' \right) \Big|_0^t \\ &= y(0) - \frac{v_{term}^2}{g} \exp\left(\frac{-g}{v_{term}} t\right) + \frac{v_{term}^2}{g} - v_{term} t \end{aligned} \quad (1.21)$$

1.2.3 Notes on coding

- In the two dimensional motion problem, and in next week's assignment on orbital dynamics, you will need to find the magnitude of a two component vector. You could write `mag = sqrt(x*x + y*y);`
But C provides a handy little “hypotenuse” function that does this, `mag = hypot(x, y);`
- You can think of the `hypot()` function as finding the radius in polar coordinates from the x and y components in Cartesian coordinates. The other half of the conversion is to find the polar angle θ . We know that $\tan(\theta) = y/x$, so the `atan(y/x)` function almost does this job. But this doesn't get the quadrant right — `atan(x/y)` is the same as `atan((-x)/(-y))`. Again, the C math library provides a handy function: `atan2(y, x)`. This is the arctangent, except that the quadrant is determined from the signs of both x and y . **Note** the order of the arguments — y comes first. Remember that all trigonometry functions, including this one, use angles in radians.
- In the problem with two dimensional motion with quadratic drag, it becomes tricky to get the components right. Remember that the **magnitude** of the force depends on both components of the velocity. But you will need the x and y components of the drag force separately. Of course, the x component is just given by $C|v|^2 \cos(\theta)$. So, using the tricks above, you might find yourself writing something like:

```
vmag = hypot(vx,vy);
fx = C * cos( atan2( vy, vx ) ) * vmag*vmag;
```

Please don't. Notice that a fragment like

```
vmag = hypot(vx,vy);
fx = C * vx * vmag;
```

does the job more elegantly. You should convince yourself that this version also handles the sign of the velocity correctly.

- **A common error:** What is wrong with the following code? We want to use Euler's method for first order DE to make a graph of $y(t)$ as a function of t .

```
...
y = y_initial;
for( t=t_start; t<t_end-h/2.0; t+= h ){
    y_new = y + h* deriv(t,y);
    y = y_new;
    printf("%e\t%e\n",t,y);
}
```

Think about the first time the `printf` function is encountered. At this point, `t` will still be set at `t_start`, but `y` will no longer be equal to `y_initial`. Do you see how to fix this?