HW 3 Computational Physics

September 6, 2005 Due September 13

If you work in groups please hand in one solution per group, showing all names.

The equation of motion for a particle subject to both gravity and air resistance is:

$$\frac{d^2\mathbf{r}}{dt^2} = \mathbf{a}, \text{ where } \mathbf{a} = -g\hat{\mathbf{k}} - \alpha v \mathbf{v}. \tag{1}$$

Here $\mathbf{v}$ is the velocity, $v = |\mathbf{v}|$ the speed, and the particle is moving in the xz plane. Equation (1) uses a simple approximation for the force due to air resistance. In this assignment you will write your own fourth-order Runge-Kutta code to solve this equation of motion numerically.

The starting point is to write the coupled second-order differential equations above as a larger set of coupled first-order equations:

$$\frac{dy}{dt} = f(y), \tag{2}$$

where

$$y = (\ x,\ z,\ v_x,\ v_z\ ), \quad f(y) = (\ v_x,\ v_z,\ a_x,\ a_z\ ). \tag{3}$$

Here the acceleration is $a_x = -\alpha v v_x, a_z = -g - \alpha v v_z$. Using a notation

$$t_n = n\tau, \qquad y_n = y(t_n), \qquad n = 0, 1, 2, \ldots, n_{\max} - 1,$$

the fourth-order Runge-Kutta algorithm is

$$\begin{aligned} h_1 &= \tau f(y_n, t_n) \\ h_2 &= \tau f(y_n + h_1/2, t_n + \tau/2) \\ h_3 &= \tau f(y_n + h_2/2, t_n + \tau/2) \\ h_4 &= \tau f(y_n + h_3, t_n + \tau) \\ y_{n+1} &= y_n + \frac{h_1}{6} + \frac{h_2}{3} + \frac{h_3}{3} + \frac{h_4}{6} \end{aligned}$$

for $n = 0, 1, 2, \ldots, n_{\max} - 2$.

In all numerical work three steps are important:

- Be modular: break your problem into pieces which can go into separate functions.
- Debug each module: make sure each function works by testing cases for which you know the answer. Then if possible make sure the whole program works by testing simple cases.
- Think about the results.

This assignment works through this procedure. **Please discuss your results for each part separately rather than adding an analysis at the end.**

1. As part of your introduction, explain equations (2) and (3). Also find analytically the terminal velocity for the equation of motion (1). Find the value of α for which the analytic terminal velocity is 120 km/h.

2. Write a function `f_air` which, given the vector y, computes $f(y)$. Snippet:

```
def f_air(y,t):
  #RHS of the first-order coupled ODE's
  #Input t is a particular time.
  #Input y = array( ( x, z, vx, vz ) ) at that time.
  #Output:  array( ( vx, vz, ax, az ) ) at that time.
  x,z,vx,vz = y
  ...
  f = array( ( vx, vz, ax, az ) )
  return f
```

   Make sure that `f_air(t,y)` gives the correct answer for some inputs `t, y`.

3. Write a function that implements the fourth-order Runge Kutte method with inputs and outputs as explained in the comments below. Put your new function and its driving main code in the same file below `f_air`. Snippet:

```
def rk4(f,y0,t):
  #Implements 4th order Runge Kutte.
  #Input y0 is an array that contains the initial conditions.
  #  (Notice that y0 can have any arbitrary length.)
  #Input t is an array of times.
  #Output:  array y with shape len(t) x len(y0), where
  #  y[n,:]  contains y(tn) for all computed times.
  y = zeros( (len(t), len(y0)), Float)   # Allocate space
  y[0,:]  = y0                           # Put IC into row 0.
  ...
  return y

#Main code
g=...; alpha=...; y0 = ...; tau = ...; tfinal = ...
t = arange(0,tfinal+tau,tau)
y = rk4(f_air,y0,t)
```

   Debug using a small `nmax` (*e.g.*, use `tau=0.1`, `tfinal=0.5`). Go through the cycle of edit-save-run until your code runs without producing any error messages, and produces a reasonable-looking output. (You can't actually be sure it's working correctly until you work through a real test case, as in the next step.)

4. You need to ensure that your code is correct. This step is critical. Even after your code runs without error messages, it could be giving nonsensical answers (if, for example, the algorithm is faulty, or if `tau` is too big). The best possible check is to make sure that your code gives the correct result in some limit where you know the answer. Here a good test case is provided by eliminating drag.

   (a) From elementary physics, figure out $y(t)$ in the absence of air resistance (*i.e.*, when $\alpha = 0$), given arbitrary initial conditions $y(0)$.

   (b) See if your code gives the same answer in this limit. Do this by setting `alpha=0`. Choose initial conditions `y0`, a step size `tau`, and a final time `tfinal`. Compare your analytic and numerical results graphically. Plot your analytic and numerical z vs. x on the same figure. Then on four additional figures plot $x(t)$, $z(t)$, $v_x(t)$, and $v_z(t)$, each as a function of time. In each case plot both the analytical and numerical results (*i.e.*, two curves per plot). Label the plots clearly, including units.

5. Now at last you're ready to run your code for a real problem. (Once you finish step 4, this step takes only a few keystrokes.) Shoot an object with an initial velocity of 30 km/hr at an angle 30° above the horizontal, with the value of α calculated in step 1. Calculate the trajectory for 20 seconds. Once you have chosen a good step size (see below), construct five plots as described in step 4; on each figure show the numerical solution (including air resistance) and the analytic solution (without air resistance).

   Choosing a step size `tau` is crucial for accuracy. If your step size `tau` is too big your answer will be inaccurate due to truncation (*i.e.*, step-size) error. If `tau` is too small your code will take too long to run, and your answer may be inaccurate due to round-off error. How can you tell what's the right step size? That's a hard question to answer. Be guided by your physical intuition to pick a reasonable value of `tau`. See what you get, and then try again with half this value. If your answers don't change much, your step size is probably small enough. Your experience with the Euler algorithm in class suggests that you need a very small value (10^{-4}s or less). However, the RK4 algorithm is very good, and `tau` need not be nearly this small. Try a few values to see what happens (say, `tau` = 10 s, 5 s, 1 s, 0.1 s). Print the final position and velocity for each to see the effect of step size.

6. A final code check is provided by letting time run long enough that your particle reaches terminal velocity. Is the numerically-obtained terminal velocity correct?