# Computational Physics
# Homework 5

Kieran Finn

October 16, 2014

## 1 Problem 1

Consider the ODE for newtonian orbits. First solve exactly to find the shape of the orbit, then calculate using Runge-Kutta numerical methods. Compare the value at 0 and $2\pi$, which should be the same as the orbit is closed and see how well the algorithms do as a function of step size and order.

### 1.1 Description of program

I used the program *closure.py* as the main wrapper, although most of the calculation takes place inside the *runge_kutta.py* package. This package contains three functions called *Euler RK2step* and *RK4step* which perform a single iteration of the Runge-Kutta algorithms of order 1,2 and 4 respectively.

It also contains a subroutine *fixed_step*, which takes as inputs: the function $f \equiv \frac{dy}{dx}$, the starting and finishing values for x $x_0$ and $x_1$, the step size h, the starting value of y, the order of the Runge-Kutta method to use and a verbosity argument that allows a choice between outputting the final value and a list of all steps as well as options to print each step and pause in between each one. This subroutine first selects an RK method to use as an iterator, performs a few error checks, such as ensuring the steps will eventually lead from $x_0$ to $x_1$. It then follows the appropriate algorithm from $x_0$ to $x_1$ in steps of h. It should be noted that, since numpy makes very little distinction between arrays and single values, this subroutine can just as easily take ys which are vectors, just as long as the function $f$ is expecting a vector and will return a vector of the same size. In fact I have written the function for Newtonian orbits in a way that maintain y's vector structure rather than using the components separately since I believe this will be faster.

Two other wrappers are also provided, *plotable*, which formats the output of *fixed_step* in a way that's easy to plot, and *evaluate*, which is used to find the value of y at a particular value of x. Since this value may not be an integer number of steps from the starting value (e.g. if it is an irrational number like $2\pi$), this subroutine uses *fixed_step* to iterate to the nearest integer step to the desired x, then performs one final step with a step size exactly equal to the remaining distance.

The *closure* algorithm works as follows. The relevant parameters are input by the user, including the eccentricity, largest step size, smallest step size and a multiplicative factor to determine the other step sizes, $dh$, so that $h_{n+1} = h_n \times dh$. The code then iterates through the chosen step sizes and calculates the difference between $y(0)$ and $y(2\pi)$ using the *evaluate* function and each of the three orders. It then creats a plot which we can use to compare.

## 1.2 Experimetntal results

figure 1 shows the output of the program.

## 1.3 Theoretical results

### 1.3.1 Exact Solution

The ODE

$$\frac{d^2u}{d\phi^2} + u = \frac{GM}{h^2} \tag{1}$$

must be solved in two stages. First we solve the homogeneous equation

$$\frac{d^2u}{d\phi^2} + u = 0 \Rightarrow \frac{d^2u}{d\phi^2} = -u, \tag{2}$$

which is the equation of a simple harmonic oscillator and so has solution

$$u = B\cos(\phi - \phi_0) \tag{3}$$

We then solve the particular integral

$$\frac{d^2u}{d\phi^2} + u = \frac{GM}{h^2}, \tag{4}$$

which clearly has the solution

$$u = \frac{GM}{h^2}. \tag{5}$$

Thus the most general solution is

$$u = B\cos(\phi - \phi_0) + \frac{GM}{h^2}. \tag{6}$$

Since the closest approach (when $u = \frac{1}{r}$ is maximised) occurs when the argument of the cos is 0 and we want this to occur when $\phi = 0$ we can set $\phi_0 = 0$. We can also redefine $B = \frac{GM}{h^2}\epsilon$ to get the final expression for the solution.

$$u = \frac{GM}{h^2}(1 + \epsilon\cos\phi) \tag{7}$$

### 1.3.2 Analysis of errors

By looking at the plots we see that the errors made by all three methods follow very nice power laws through most of the range until roundoff error takes over (which happens for RK4 around $h = 0.002$). In fact we can be quantitative by fitting a straight line to the log plots to find the power law. By looking at figure 2 we see that the power laws followed are approximately

- Euler $\Delta \propto h$
- RK2 $\Delta \propto h^3$
- RK4 $\Delta \propto h^5$

Euler is behaving as expected, but the other two methods are converging quicker than we expect. This is probably due to the simple nature of this problem

If we instead look at the velocity, as shown in figure 3 we see the following power laws

- Euler $\Delta \propto h^2.3$
- RK2 $\Delta \propto h^2$
- RK4 $\Delta \propto h^4$

This time we see that it is Euler that is converging quicker than expected and the two RK methods are following their respective theoretical curves. This shows that, even when we have this simplistic solution, the numerical methods may do better than expected in one variable but not both.

The value of $\epsilon$ does not affect the convergence rate, as expected.

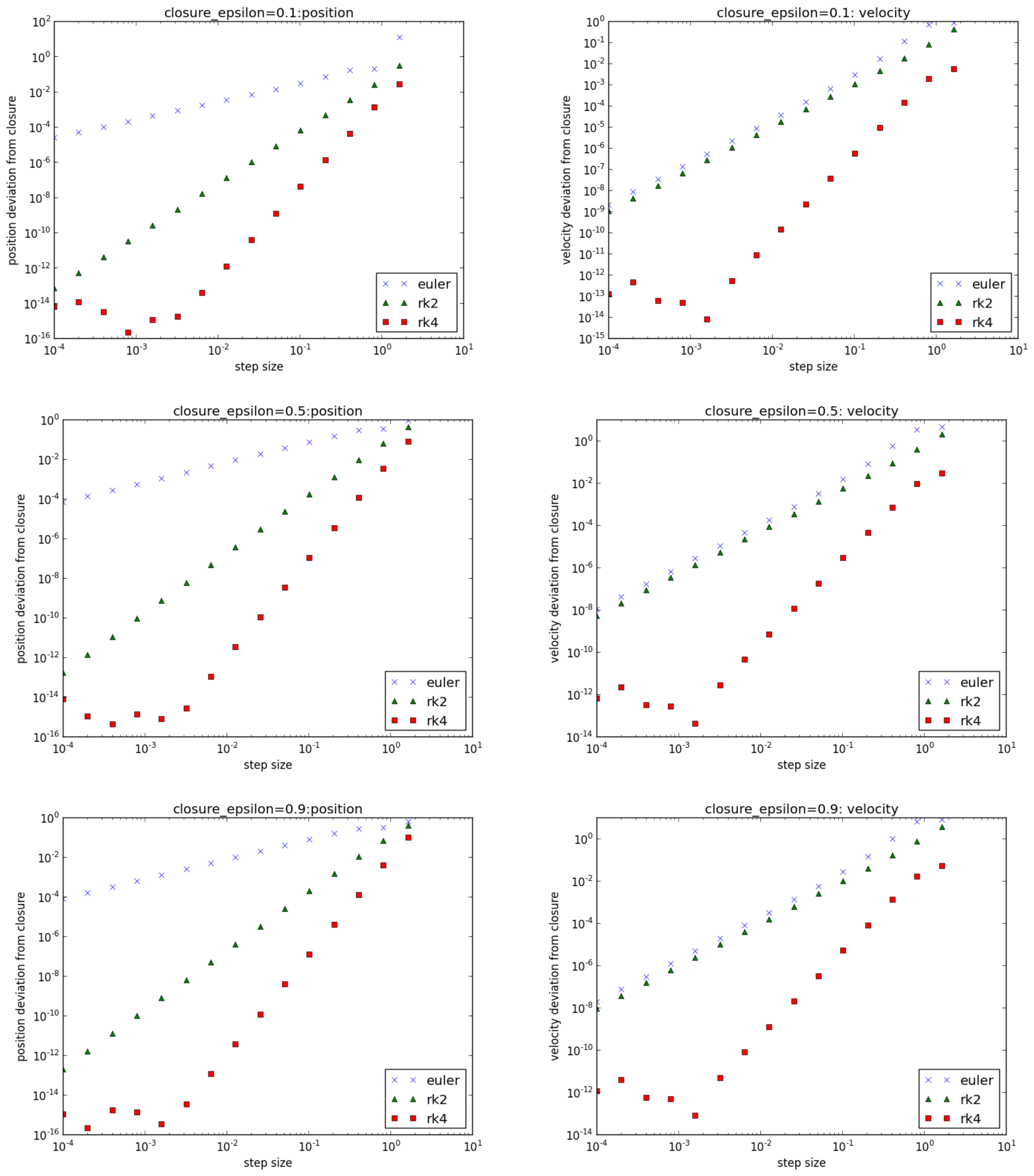

Figure 1: Deviation from closure verses step size, comparing different orders of Runge-Kutta methods.

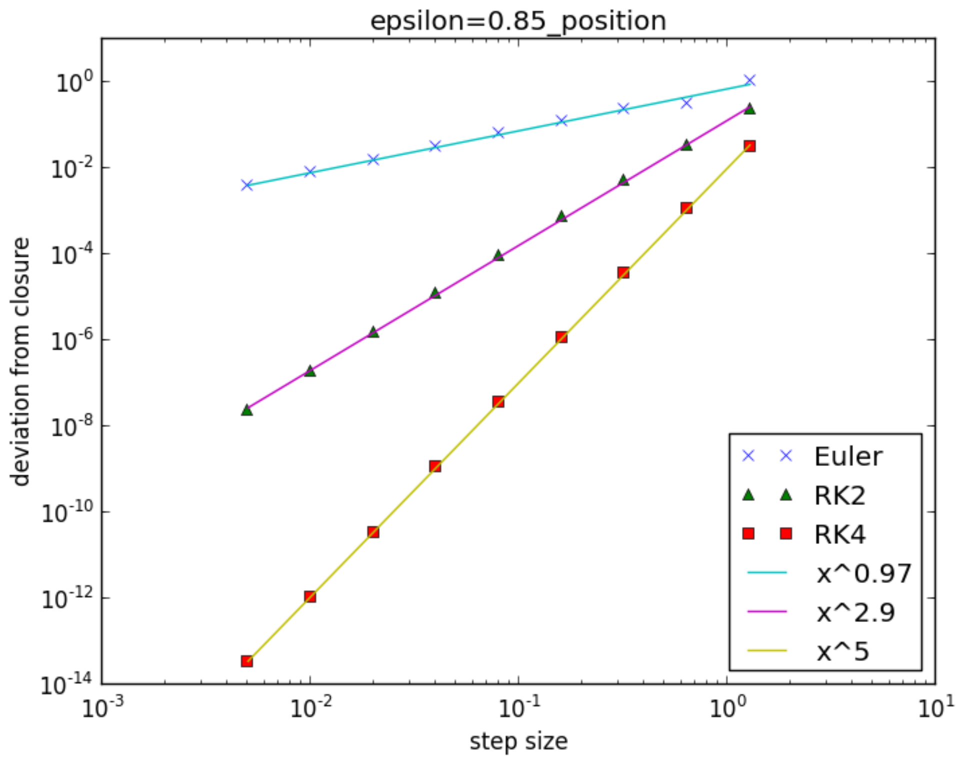

Figure 2: Fit a power law to the deviations in position. $\epsilon = 0.85$.

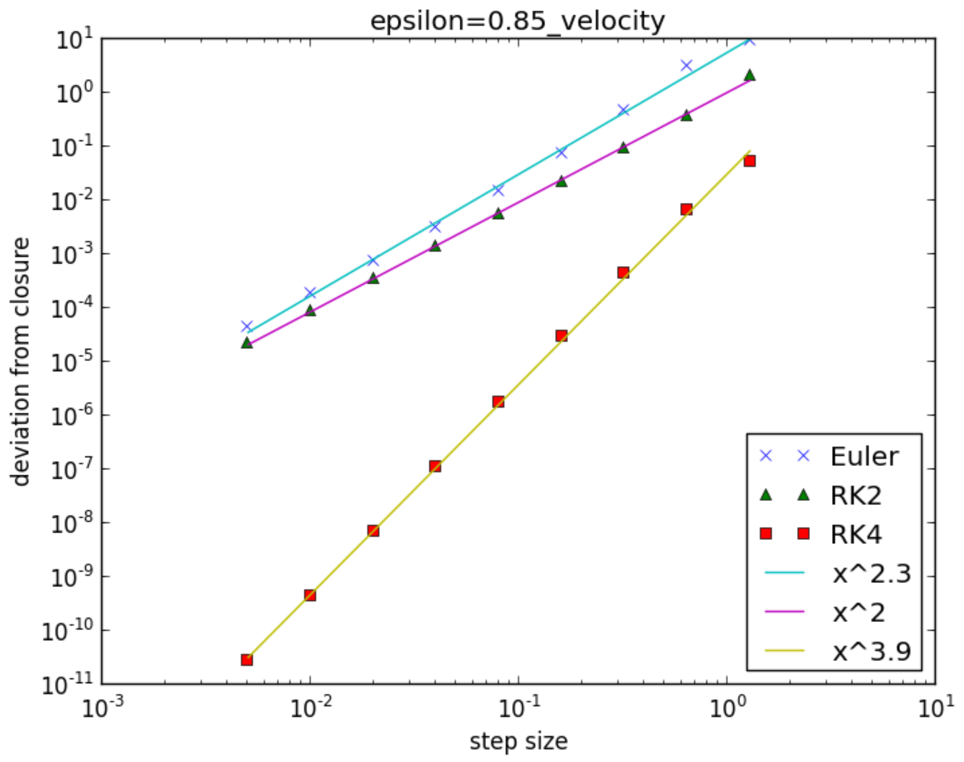

Figure 3: Fit a power law to the deviations in velocity. $\epsilon = 0.85$.

## 2 Problem 2

Generalise to GR and look at the orbital precession, comparing it to the theoretical result.

### 2.1 Description of the program

This part of the problem used the wrapper *relativistic.py* but, again, most of the functionality is in *runge_kutta.py*. In addition to defining a new function called *GR_orbit* to encode the actual equations I needed two new subroutines to find the exact location of the next perihelion. The first was *pass_y*, which simply implemented a Runge Kutta method (as detailed in the previous section) until the dependant variable passed a certain value. This was needed since otherwise the program would find the aphelion instead, which is the next point where $\dot{u} = 0$. *find_y* was used to find an exact value for x at which a condition for y was met. The subroutine iterates until the target y is passed, keeping an up-to-date list of the last 3 values of the function. Once the target is passed, it fits a parabola to these points and interpolates the exact location where the function reaches the target.

The wrapper *relativistic.py* scrolls through lambda and, for each value, runs *pass_y* to pass the aphelion and then *find_y* to find the perihelion and subtracts $2\pi$ to calculate the processional shift.

### 2.2 Theoretical results

#### 2.2.1 Proving the perihelion precession

The equation we want to solve is

$$\frac{d^2u}{d\phi^2} + u = \alpha + 3\frac{\lambda}{\alpha}u^2 \tag{8}$$

where $\alpha = \frac{GM}{l^2}$ and $\lambda = \left(\frac{GM}{lc}\right)^2$. We can simplify this enormously by defining

$$v \equiv \frac{u}{\alpha} \tag{9}$$

so that the equation becomes

$$\frac{d^2v}{d\phi^2} + v = 1 + 3\lambda v^2. \tag{10}$$

We know the solution for $\lambda = 0$ is

$$v_0 = 1 + \epsilon \cos\phi \tag{11}$$

so we choose to perturb about this solution so that

$$v = v0 + \lambda x \tag{12}$$

plugging this into equation 10 and keeping only terms of order $\lambda$ we find

$$\frac{d^2 v_0}{d\phi^2} + \lambda \frac{d^2 x}{d\phi^2} + v_0 + \lambda x = 1 + 3\lambda v_0^2 + O(\lambda^2) \tag{13}$$

but, since $v_0$ satisfies the equation with $\lambda = 0$ a lot of terms drop out and we have

$$\begin{aligned}
\frac{d^2 x}{d\phi^2} + x &= 3(1 + \epsilon \cos\phi)^2 && (14)\\
&= 3(1 + 2\epsilon\cos\phi + \epsilon^2\cos^2\phi) && (15)\\
&= 3(1 + 2\epsilon\cos\phi + \frac{\epsilon^2}{2} + \frac{1}{2}\cos(2\phi)) && (16)
\end{aligned}$$

Since this is a perturbation we don't care about the homogeneous solution and we can search for the particular integral straight away. The constant term can easily be satisfied by a constant x. The other two terms require a bit more thought but we can see that

$$\left(\frac{d^2}{d\phi^2} + 1\right)\cos(2\phi) = -4\cos(2\phi) + \cos(2\phi) = -3\cos(2\phi) \tag{17}$$

and

$$\left(\frac{d^2}{d\phi^2} + 1\right)\phi\sin\phi = -\phi\sin\phi + 2\cos\phi + \phi\sin\phi = 2\cos\phi \tag{18}$$

so that the equation can be solved by

$$x = 3 + \frac{3\epsilon^2}{2} + 3\phi\sin\phi - \frac{1}{2}\cos(2\phi) \tag{19}$$

so that the full solution (to order $\lambda$) is

$$v = 1 + \epsilon\cos\phi + \lambda\left(3 + \frac{3\epsilon^2}{2} + 3\phi\sin\phi - \frac{1}{2}\cos(2\phi)\right) \tag{20}$$

We know that the first perihelion occurs at $\phi = 0$ and that the second one occurs at $\phi = 2\pi + \Delta\phi$. Since $\Delta\phi$ is order $\lambda$ we see that any contribution coming from the $\cos(2\phi)$ term will be constant plus higher order in $\lambda$. But, since the constants affect the *value* of the perihelion and not its position these can be ignored and thus we consider

$$\begin{aligned} v &= K + \epsilon\cos\phi + 3\epsilon\lambda\phi\sin\phi && (21)\\ &\approx K + \epsilon(\cos\phi\cos(3\lambda\phi) + \sin\phi\sin(3\lambda\phi)) && (22)\\ &= K + \epsilon\cos(\phi - 3\lambda\phi) && (23) \end{aligned}$$

where we have used the small angle formula and are working to first order in $\lambda$.

Therefore the second perihelion occurs when

$$\begin{aligned} \phi(1-3\lambda) &= 2\pi && (24)\\ \Rightarrow \phi &= \frac{2\pi}{1-3\lambda} && (25)\\ &\approx 2\pi(1+3\lambda) + O(\lambda^2) && (26) \end{aligned}$$

and we see that the precession in orbit is

$$\Delta\phi = 6\pi\lambda. \tag{27}$$

## 2.3 Experimental results

Figure 4 shows the results of the program. We see that $\Delta\phi = 6\pi\lambda$ is a very good fit for most of the range. It deviates slightly for large $\lambda$ for two reasons. Firstly, the larger value means that perturbation theory no longer holds as well, but secondly because it is beginning to enter the regime where these orbits are no longer bounded, as happens too close to a black hole.

## 2.4 Mercury

The program *mercury.py* impliments this using real data to find the value $\Delta\phi$ lost each orbit. It then uses the known period of Mercury to calculate the rate of precession. The answer I get is

$$\frac{d\phi}{dt} = 43.011\text{seconds per century} \tag{28}$$

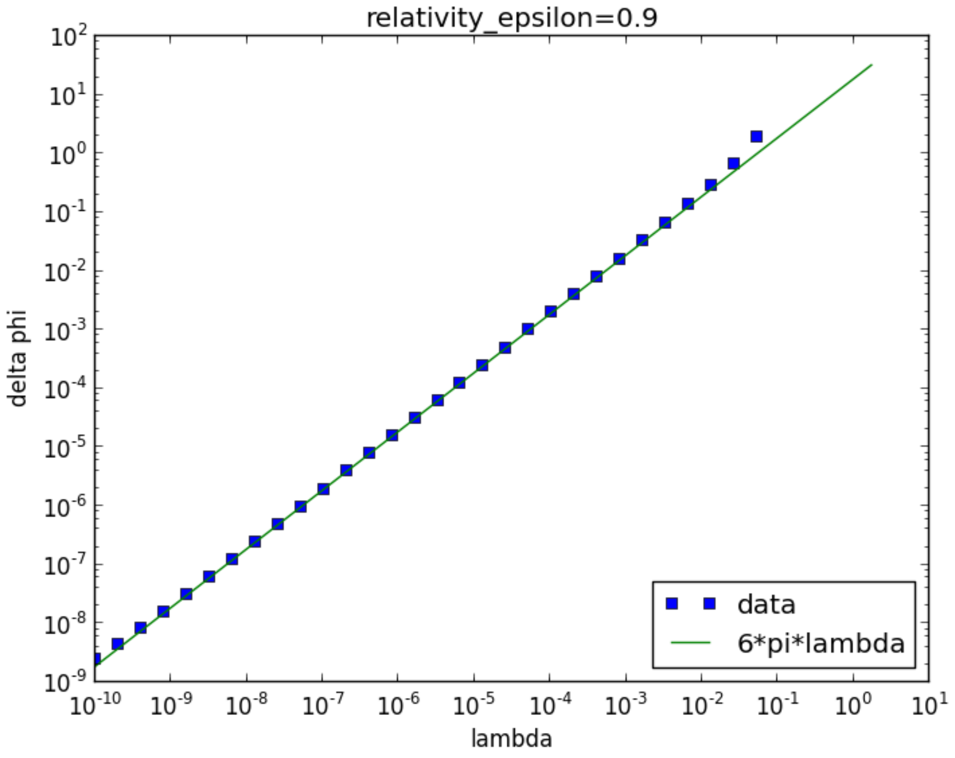

Figure 4: Perihelion precession in GR. Compared with theoretical prediction $\epsilon = 0.9$.