N-body RK4 Orbit Integrator

23770
Department of Physics, University of Bath, Bath BA2 7AY, United Kingdom

Abstract
This report outlines the use of a python program as a means for scientific enquiry in the investigation of dynamical systems, specifically the orbit of astrophysical bodies as defined by a second order ordinary differential equation (ODE). A simple RK4 numerical integration routine was developed and applied to a two-body problem before extending to an adaptive implementation and applying to N-bodies. The orbit of Halley's Comet around the Sun was found to be in agreement with physical intuition, in addition to more complex N-body orbits such as those of Jupiter and Saturn.

Introduction

Runge-Kutte (RK) methods refer to the subset of numerical integration methods of varying accuracy which, unlike the Euler method, do not require any explicit derivatives. The gradient of the function, *f(t)*, at a given point is extrapolated to estimate the value of the function at a later timestep, $f(t+h)$, performing a Taylor series expansion around the gradient at the mid-point $f(t+h/2)$ instead to provide a better estimate. This basis for the second-order RK method is extended to higher orders by taking combinations of Taylor expansions around various midpoints which arrange to cancel higher order terms, providing increasingly accurate solution to the ODE.

In the case of a fourth-order RK method, which is used in the program due to its balance of accuracy and simplicity, the overall calculation is accurate to terms of order h^4 whilst each individual step has an error of order h^5. An adaptive step size aims to have the error per interval roughly constant and is achieved by calculating the step error for two timesteps compared to the single second-order estimate. This has the effect of improving both the efficiency and accuracy of the routine by sampling a greater number of points where the function varies more rapidly and fewer where it does not. If the step error is less than the tolerated relative error, it is safe to increase the step size and continue with the routine, whereas a value greater than the relative error tolerance requires a reduction in the step size and repetition of the current step.

For simulation of a two-body system, the Sun, of mass M, is taken to be stationary at the origin, with an orbiting comet, of mass m, at position vector $\vec{r}$ in the z = 0 plane such that

$$m\frac{d^2\vec{r}}{dt^2} = -\frac{GMm}{r^2}\frac{\vec{r}}{r}. \qquad (1)$$

This can be extended to model the interaction of N-bodies by principle of linear superposition, varying which mass is taken to be the 'central' mass and recalculating the position vector between the two bodies. By transforming the coordinate system to a centre-of-mass frame, subtracting position and velocity vectors for the centre-of-mass for each data point, it is also possible to simulate more complicated systems such as a binary star system with an orbiting exoplanet.

The aims of this investigation are to successfully implement a numerical integration routine which is capable of accepting any method or system to be solved, specifically an N-body astrophysical system in order to model the time evolution of orbits given a set of initial conditions.

Method / Implementation

The program consists of two main classes: Body() and Simulation().

Instances of type Body are initialised for each individual astrophysical body under consideration, with state variables for the body name, mass and initial position/velocity vectors. If an instance of Body has Astropy units attached, these are converted to a 'G = 1' system with scales suitable on the order of solar systems. There is also a check within the class __init__ to flag the body as a central if it is initially at position [0,0], assuming this to be the centre of the coordinate system. Other Body class methods include 'getters' and 'setters' which allow for the internal state of a single instance to be retrieved for use elsewhere or updated from outside the class. This design choice allows for the program to be easily extended from the two-body central-mass case by instantiating as many as required and passing a list of objects to the Simulation class.

An instance of Simulation extracts relevant information from the list of interacting bodies as state variables which are later used in RK4 calculations. One must also set the differential equation solver to be used by the RK4 routine, applied to a system of interacting bodies, which essentially allows for generalisation to any second order ODE. Two external solver functions have been implemented, and in such a way that nbody_solve() is an extension on two_body_solve(), looping over the interacting bodies and summing each individual contribution by means of linear superposition. The bulk of the simulation is performed by the class method run() which is called on the simulation object and passed parameters for the total simulation time, initial timestep and adaptive stepsize. Astropy units are again appropriately handled and the corresponding RK4 routine is set by class method set_method() depending on the value of the Boolean 'adaptive' which could in fact have simply been a check for whether a relerr was specified.

The main section of code in run() initialises the simulation history based of the initial conditions of all the bodies before looping over the simulation time and appending to the history for each body at each timestep. The nested for-loop was initially performed entirely within the nbody_solve function; however, this required a large vector containing x, y, vx, vy data for all the bodies in turn to be passed through to RK4 at once and then some computationally expensive slicing performed which is not ideal considering this N-body solver already scales as $O(N^2)$. Having the first loop over the bodies within run() was also necessary for (attempted) implementation of the centre of mass frame which in retrospect could have benefitted from a 'System' class composed of all the interacting bodies and a class method to calculate the system centre of mass at each timestep.

The implementation of an adaptive step size actually takes the form of calculating a greater number of points in faster moving sections of the orbit based on the relative accelerations, rather than the error in RK4 as described in the introduction. This, along with any RK4 implementation, works well for the two-body problem but is not appropriate for multiple bodies as they would need to advance by same amount each step (this would be extremely computationally expensive to calculate for each body and then perform the step to advance the simulation in time). Another alternative for adaptive step sizes would be to implement an embedded Fehlberg method (RKF45) or calculation of the total system energy (which would provide an indication of the error in RK4 as it lacks long-term stability). One could instead employ a numerical integration with time-reversal symmetry, such as the leap-frog or Verlet methods which are often used in orbital simulations and arguably more appropriate, as this symmetry is important in the context of conservation of total system energy at each timestep.

Testing primarily took the form of so-called "caveman debugging", utilising well-positioned print statements, as it was not possible to write unit tests and set up a proper continuous integration workflow and debugging environment given the time constraints. It was also useful to compare the output of the simulation class method plot() to determine whether issues were in manipulation of the data for plotting or in the actual data itself, and the use of velocity colour bar was helpful as it allows for visual comparison of the data with physical intuition.

Results & Discussion

All results have been obtained by starting simulations at aphelion.

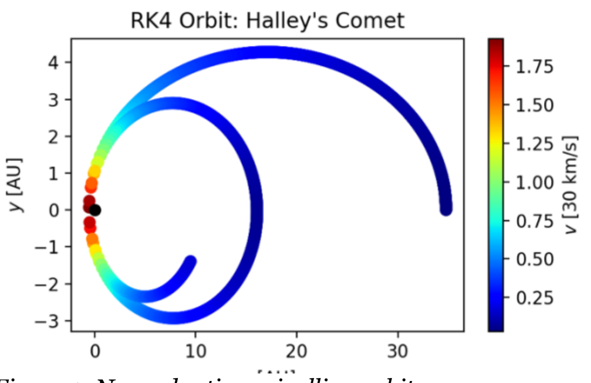

Figure 1: Non-adaptive spiralling orbit.

The orbit of Halley's Comet around the Sun was seen to be approximately elliptical using a non-adaptive step size of 0.5 years over a total simulation time of 65 years, as shown in Fig. 1, however it soon deviates and begins to spiral as the sampling is too coarse at regions of high velocity.

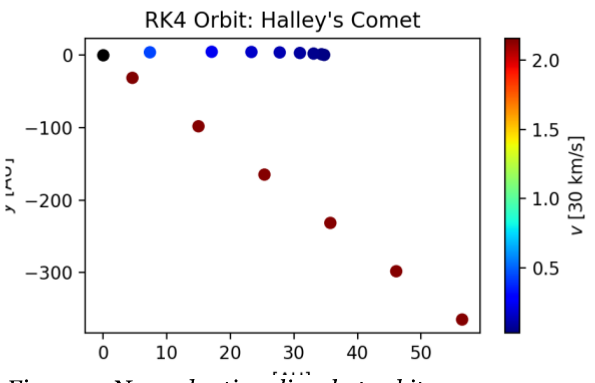

Figure 2: Non-adaptive slingshot orbit.

A similarly incorrect orbit is shown in Fig. 2 using a non-adaptive step of 5 years over the same period, however usage of an adaptive step with these conditions yields the more expected result in Fig. 3 which clearly demonstrates an elliptical orbit where the velocity of the comet also aligns with physical intuition.

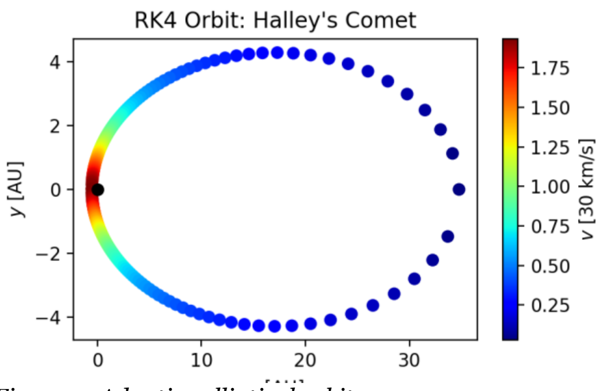

Figure 3: Adaptive elliptical orbit.

Similarly, but demonstrating the use of nbody_solve(), the orbits of 'Planet 1' and 'Planet 2' provided, with masses 10^{-3} and $4{\times}10^{-2}$ solar masses respectively and initial velocities calculated by another external function assuming initially circular orbits, elliptical orbits are found.

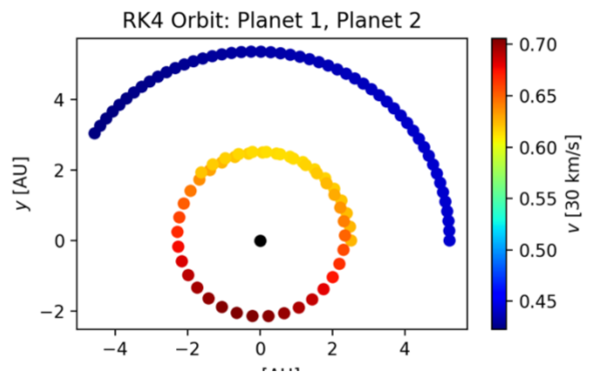

Figure 4: Initial slight deviation in N-body orbit.

A slightly more circular orbit of total simulation time 5 years with steps of size 0.1 years is shown in Fig. 4, whilst the simulation in Fig. 5 runs over 12.2 years in steps of 0.25 years and clearly demonstrates the interaction of the two bodies, in addition to the influence of the Sun at the centre.

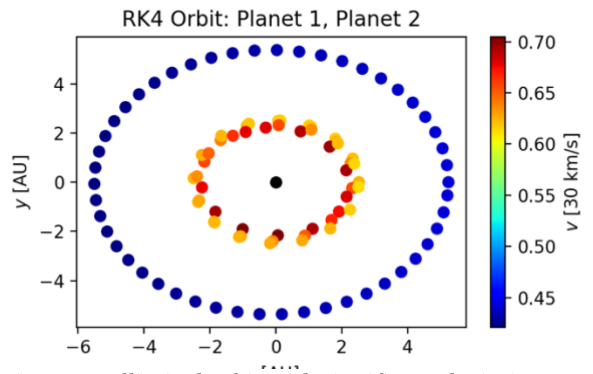

Figure 5: Elliptical orbit with significant deviation.

This is as expected given the relative size of the planets, such that Planet 1 feels a greater influence from Planet 2 than it provides itself and clearly begins to deviate more significantly over time.

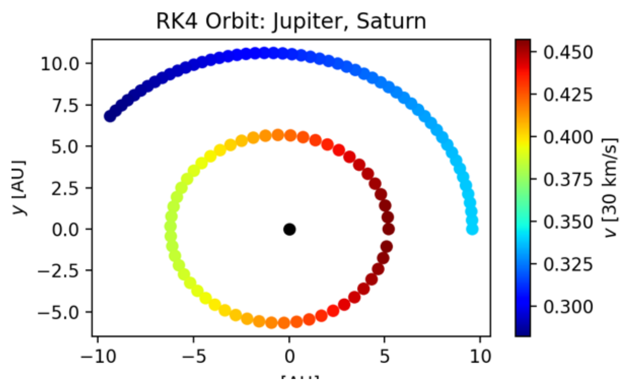

Figure 6: Initial Jupiter-Saturn orbit more elliptical.

The orbits of Jupiter and Saturn are slightly different in that they each appear to have a greater influence on one another, as shown by the variation in velocities and deviation from initially circular orbits in Fig. 6 and Fig. 7.

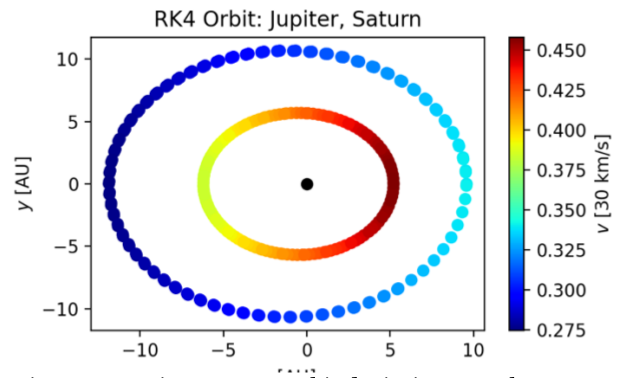

Figure 7: Jupiter-Saturn orbit deviations at a later stage.

The error in simulation is insignificant enough over low time frames, such as those investigated here, to produce meaningful results; however, this short-term stability does break down at larger scales.

Conclusion

This computational investigation does allow for predictions about the likely motion of bodies in a solar system, with the adaptive RK4 two-body implementation working particularly well. It has also been possible to simulate non-adaptive N-body orbits, however a different numerical integration routine such as the Verlet method would be more appropriate. It was also be useful to have finished an implementation capable of calculating the system energy at each step and also transforming to a frame based on the centre of mass to models systems where the initially central body is not fixed.