Ordinary Differential Equations
—
Elliptical Motion

Keegan W$^{\text{m.}}$ Karbach

March 16, 2020

Introduction

The dynamics of orbiting bodies have been of scientific interest for many centuries. Ancient astronomers seeked explanation for the movement of certain stars in the sky, a question which would remain unanswered until Newton's formulation of the theory of gravity. Indeed, it is historically appropriate for us to study these dynamics using the orbit of Halley's comet as the correspondence of Halley with Newton was instrumental in Newton's publication of the *Principia Mathematica* – included therein, derivations of Kepler's Laws.

Central force problems, such as the ones described by Newton and Halley, arise in many physical situations, as many natural forces act centrally such as electromagnetism and gravity. Additionally, the modeling of orbital dynamics is of great import; much mathematical work in the 18th and 19th century was applied to create accurate data for orbiting stellar bodies for navigational purposes. In modern times, accurate computational modeling of orbital dynamics is paramount in the aerospace sector as the number of artificial satellites continues to rapidly increase.

At their most basic, central force problems describe the motion of a particle in a central potential field with a radially-acting force. These trajectories describe conic sections, both bound and unbound depending on the initial parameters of the particle. When examining naturally occurring phenomena, two body problems are often encountered where the objects orbit a single point (the barycenter) and the inter-body force acts on the vector between the two bodies – these problems can be simplified into single body problems with a change of reference frame.

When examining orbital dynamics, as in this project, we can assume a single body problem (the mass of the orbiting body is much less than the mass of the sun so we treat the sun as immobile). We begin by considering the centrally-acting force of gravity:

$$\mathbf{F} = -\frac{GmM}{|\mathbf{r}|^3}\mathbf{r}$$

where $\mathbf{r}$ and m are the position vector and mass of the orbiting body, respectively, M is the mass of the sun, and G is the gravitational constant. This gives us a constant acceleration with which we can apply the numerics that were developed in Project 2 for parabolic motion: Euler's Method and the Euler-Cromer method. With trials on known problems, we can see that the Euler method quickly becomes unstable due to magnification of numerical error, as does the Euler-Cromer method when applied with larger time steps. This motivates the development of more refined methods, namely the fourth-order Runge-Kutta routine. Further refinement for this type of problem leads to a heuristic algorithm, adapting the time step at each iteration both to minimize the running time and the error below a certain bound.

1 Problem 3.2

Prove conservation of angular momentum for the Kepler problem using the Euler-Cromer method.

The claim is that, for the Euler-Cromer method

$$\begin{aligned}\mathbf{v}_{n+1} &= \mathbf{v}_n + \tau\mathbf{a}(\mathbf{r}_n)\\ \mathbf{r}_{n+1} &= \mathbf{r}_n + \tau\mathbf{v}_{n+1}\end{aligned}$$

angular momentum

$$\mathbf{L} = \mathbf{r} \times (m\mathbf{v})$$

is conserved.

Proof.
First we need to break the vectors into components:

$$\begin{aligned}\mathbf{L} &= \langle x, y\rangle \times m\langle u, v\rangle\\ &= m(xv - yu)\hat{\mathbf{z}}\end{aligned}$$

Now, when this is introduced into the Euler-Cromer equations:

$$\begin{aligned}\mathbf{L}_{n+1} = \mathbf{v}_{n+1} \times \mathbf{r}_{n+1} &= m(x_{n+1}v_{n+1} - y_{n+1}u_{n+1})\hat{\mathbf{z}}\\ &= m(x_n v_n - y_n u_n) + m\tau\left(x_n a_n^{(x)} - y_n a_n^{(y)}\right)\\ &= \mathbf{L}_n + m\left(\mathbf{r} \times \mathbf{a}(\mathbf{r}_n)\right)\end{aligned}$$

But for a central force, $\mathbf{a}(\mathbf{r})$ and $\mathbf{r}$ are co-linear. Hence, their cross product is zero and the second term drops from the above equation.

$$\therefore \qquad \mathbf{L}_{n+1} = \mathbf{L}$$

□

2 Problem 3.6

The Euler-Cromer method must use a small time step for the more elliptic orbits. For a system analogous to Halley's comet, find the largest value of τ for which the total energy is conserved to $\sim 1\%$ per orbit using a variety of aphelion velocities.

We first begin with validation of the `orbit` program. Working from the code given in [1], we can port it to python and duplicate Figure 3.3 – shown here as Figure 1. This code will do for a small set of data, but takes far too long to execute when utilizing the small time steps required for more elliptical orbits. Remediation for this is performed by vectorizing the code and calling a function for the method that has been optimized to bypass Python compilation in favor of C++. This implementation is validated by comparison to Figure 3.3 from [1]. This routine also

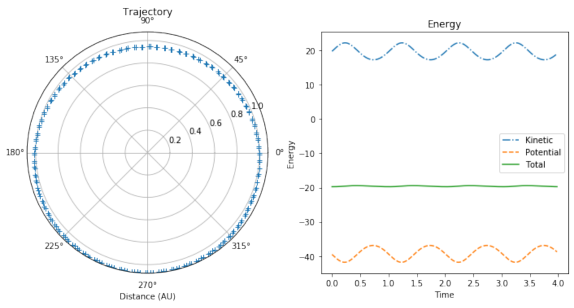


Figure 1: Validation of the Euler-Cromer routine. Note that this runs over four full rotations, the orbit closes (unlike the Euler Method using the same parameters), and there is a slight variation in the total energy.

validates my stopping criterion, ending the iteration when the method reaches aphelion and is within an error margin from zero degrees in the polar plot. The margin term was included to ensure stoppage in one full period as opposed to half a period or multiple periods. This validation is illustrated in Figure 2.

```
for i in range(nStep):
  ...
  r, v, time = eulcro(r, v, time, tau, GM)

  if i > 1 / tau:
      if (rplot[i] < rplot[i - 1]) and np.abs(thplot[i]) < margin:
          break
```

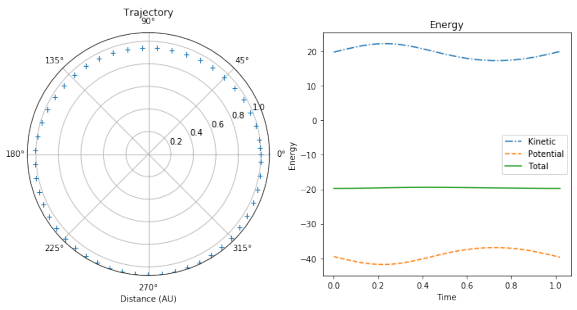


Figure 2: Validation of optimized Euler-Cromer routine. This runs strictly over one full period and will calculate relative percent deviation in total energy both over the entire period and at the aphelion.

Using this implementation, we can effectively calculate the deviation of total energy over a single orbit. Two different methods were used to ensure conservation of energy within one percent:

```
dev_E = np.abs((totalE[0] - totalE[-1])/np.mean(totalE)) * 100
dev_E_abs = np.abs((np.max(totalE) - np.min(totalE))/np.mean(totalE)) * 100
```

The first method calculates the total energy at the closure of the orbit while the second calculates the percent change of the minimum and maximum energies over the entire orbit. As total energy *should* be constant throughout the entire orbit, I used the deviation over the entire orbit for my calculation. The reason for the discrepancy between the two deviation calculations is a numerical artifact near the perihelion of the orbit – due to the increased velocity when the body approaches the sun, a smaller time step is needed to fully resolve the body's path. This becomes computationally expensive when using a non-adaptive method, leading to our later development of an adaptive method.

Figure 3 illustrates the decrease in time step necessary to keep the total energy conserved to 1% as orbital velocity decreases and the orbit becomes more elliptical. Our goal is to ascertain the time step required to track Halley's comet. Using the above method, the time step necessary for the given level of energy conservation was calculated over a range of aphelion velocities: $V_0 = [1, .9, .8, ..., .1]$ in units of AU/yr. These results were plotted on a logarithmic scale and an interpolation function was created over the range in question. The initial velocity of Halley's comet was calculated from the data given in Table 3.1 [1] using the velocity equation at aphelion $(r = Q)$.

$$v = \sqrt{GM\left(\frac{2}{r} - \frac{1}{a}\right)}$$

where $a = q/(1-\epsilon)$ and $Q = (1+\epsilon)q/(1-\epsilon)$. The corresponding value for the necessary time step was obtained from the interpolation function.

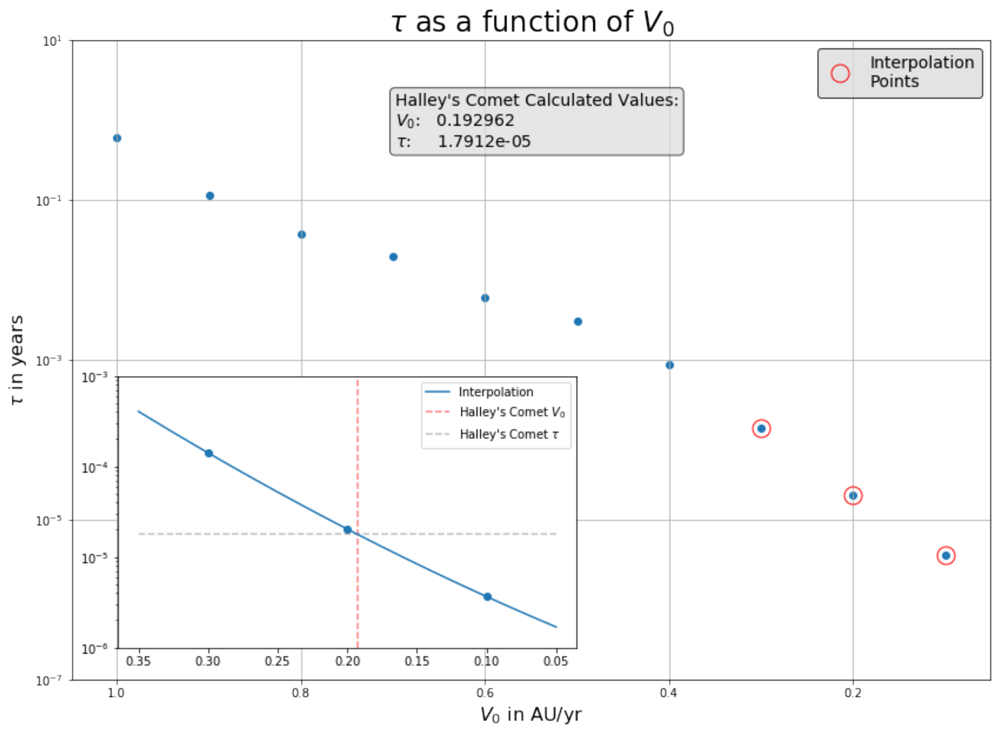


Figure 3: Maximum τ with periodic relative deviation in total energy limited to 1%.

After obtaining the value for the time step, the solution was validated using the Euler-Cromer implementation and parameters for Halley's comet – shown in Figure 4. Note that the deviation in total energy is around .5%. Further analysis could be performed in a similar fashion using smaller steps in the vector of initial velocities or more points in the interpolation range, however these would both be prohibitively computationally expensive for this problem.

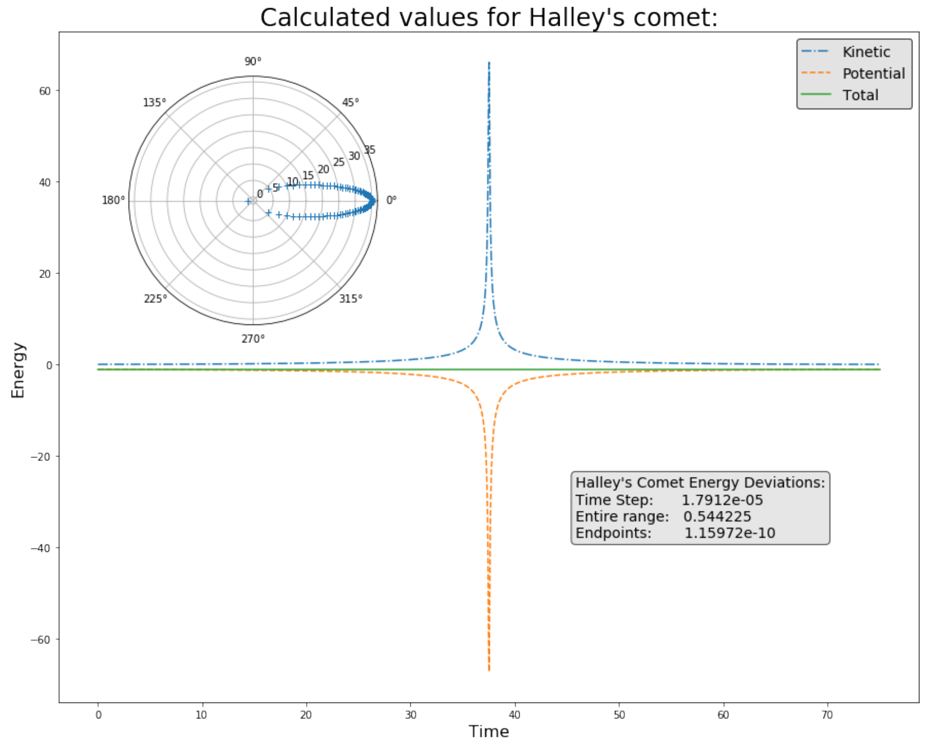
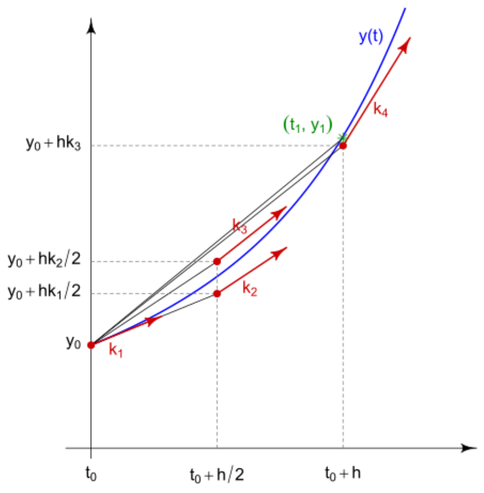


Figure 4: Validation of orbit for time step obtained from Figure 3

3 Problem 3.11



Repeat the analysis from Problem 3.6 using a fourth-order Runge-Kutta method and compare with the Euler-Cromer method.

The computational inefficiency of the Euler-Cromer for this problem is evident – for highly elliptical orbits, a minute time step is necessary, leading to a very large number of data points and a long computation time. The primary reason for this shortcoming is the fact that the Euler-Cromer method, despite its increased numerical stability in this problem, is still of $\mathcal{O}(\tau)$ in terms of global truncation error. What we need is a method that will mitigate this error.

As opposed to a single estimation of the slope at $t = t_0 + \tau$, the fourth-order Runge-Kutta method takes a weighted estimate of slopes measured at four places, the beginning and end of

Hilber Traum, ***Slopes used by the classical Runge-Kutta method***, 2017 the time step as well as two measurements at the half-way point. This design makes the method globally fourth order, $\mathcal{O}(\tau^4)$, a substantial improvement! This allows us to track an orbital body with a much larger time step than we would need for the same parameters using the Euler-Cromer method.

$$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \frac{1}{6}\tau[\mathbf{F}_1 + 2\mathbf{F}_2 + 2\mathbf{F}_3 + \mathbf{F}_4]$$

$$\text{where} \begin{cases} \mathbf{F}_1 = \mathbf{f}(\mathbf{x}, t) \\ \mathbf{F}_2 = \mathbf{f}(\mathbf{x} + \frac{1}{2}\tau\mathbf{F}_1, t + \frac{1}{2}\tau) \\ \mathbf{F}_3 = \mathbf{f}(\mathbf{x} + \frac{1}{2}\tau\mathbf{F}_2, t + \frac{1}{2}\tau) \\ \mathbf{F}_4 = \mathbf{f}(\mathbf{x} + \tau\mathbf{F}_3, t + \tau) \end{cases}$$

The implementation of this method is very general: we set up a function as above and can then call any function to calculate the derivatives as necessitated by the problem. Hence, two new functions are defined:

```
def rk4(x,t,tau,derivsRK,param):
    half_tau = 0.5*tau
    F1 = derivsRK(x,t,param)
    t_half = t + half_tau
    xtemp = x + half_tau*F1
    F2 = derivsRK(xtemp,t_half,param)
    xtemp = x + half_tau*F2
    F3 = derivsRK(xtemp,t_half,param)
    t_full = t + tau
    xtemp = x + tau*F3
    F4 = derivsRK(xtemp,t_full,param)
    xout = x + tau/6.*(F1 + F4 + 2.*(F2+F3))
    return xout
def gravrk(s,t,GM):
    r = np.array([s[0], s[1]])
    v = np.array([s[2], s[3]])
    accel = -GM*r/la.norm(r)**3
    deriv = np.array([v[0], v[1], accel[0], accel[1]])
    return deriv
```

Following the same analysis as was done for the previous problem, we arrive at the results in Figure 5. There is a similar functional form to the relationship between the time step and the initial velocity of the orbiting body, but there is a difference of two orders of magnitude between the curves. To accurately track Halley's comet using the Euler-Cromer method, we required a time step of 1.79×10^{-5} whereas with the Runge-Kutta method requires a time step of only 9.57×10^{-3}. Due to the larger step size, the computation time is substantially decreased for this improved method as well.

Again, the solution is validated by running a single orbit using the fourth-order Runge-Kutta routine with the parameters calculated from the above analysis. These results are shown in Figure 6.

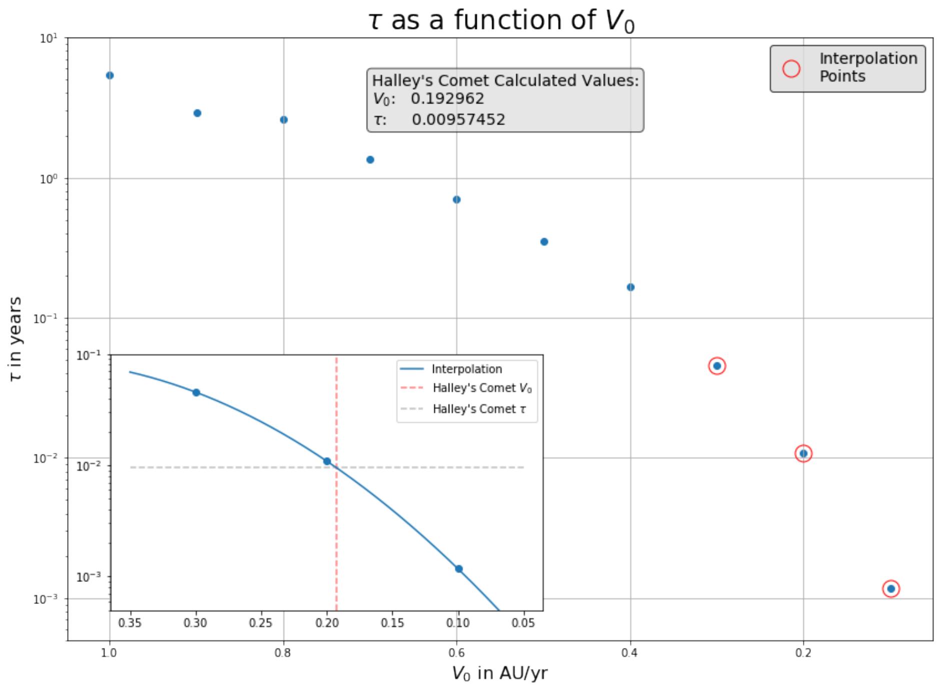


Figure 5: Maximum τ with periodic relative deviation in total energy limited to 1%.

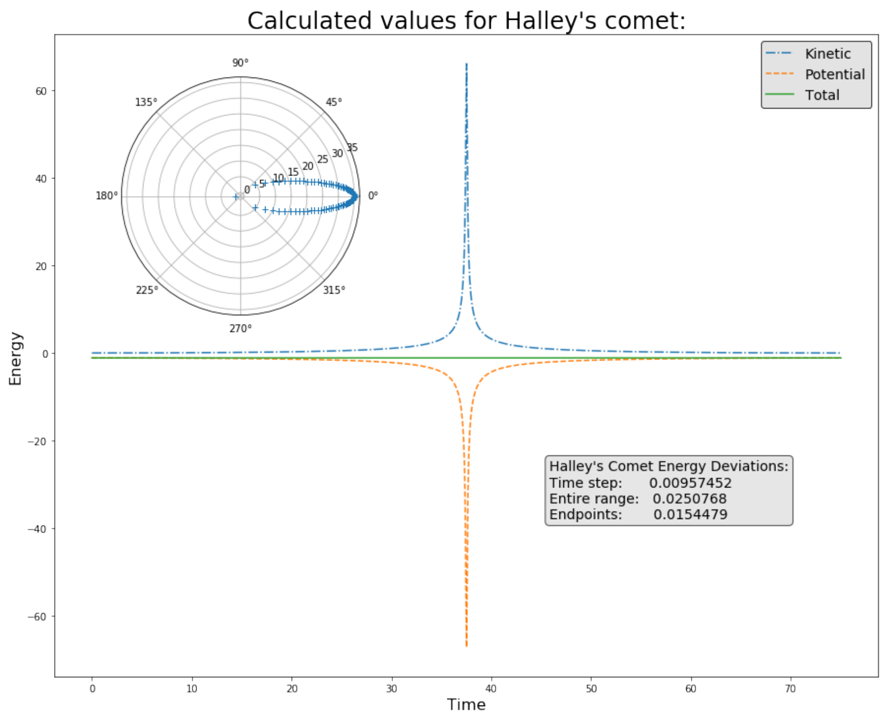


Figure 6: Maximum τ with periodic relative deviation in total energy limited to 1%.

4 Problem 3.14

Consider the central force

$$\mathbf{F}(\mathbf{r}) = -\frac{GMm}{r^3}\left(1 - \frac{\alpha}{r}\right)\mathbf{r}$$

where α is a constant. Using an adaptive Runge-Kutta method, show that the orbit precesses as $360(1-a)/a$ degrees per revolution given

$$a = \sqrt{1 + G\left(\frac{Mm^2\alpha}{L^2}\right)}$$

The previous problems have illustrated that computational efficiency is of great import in problems requiring a large number of time steps – switching from a second-order to a fourth-order integrator saved us considerable computation time. Despite this refinement, we are still presented with a concern. Having a constant time step creates many data points around the aphelion. The stopping criteria that were used to determine to what percent energy was conserved are concerned with the data near the perihelion and as such, these points dictate the size of the data necessary to accurately plot the orbit. If we could somehow change the time step at each iteration, we could better optimize the data such that the error is roughly equivalent over the entire range, obviating the unnecessary clustering of data points at the far end of the orbit.

This is the motivation for the development of an adaptive Runge-Kutta routine, a heuristic routine that, in its simplest form takes each time step twice, once as a full step, then, independently, as two half-steps. Upon first inspection, this may seem counter-intuitive: it seems as if we are now taking twelve function evaluations as opposed to four as each step of the method requires four. However, we are benefiting from the increased accuracy of the half-step and both starting points are equivalent, so the actual overhead is a factor of 1.375. With this slight increase in computational overhead, we halve our local truncation error as well as substantially decreasing the overall size of the data necessary to compute an accurate orbit. This is an acceptable bargain.

The routine given in [1] was translated into Python and validated with the results shown in Figure 7. The versatility of the Runge-Kutta routine is again illustrated in the fact that we only need to define a new function to calculate the derivatives at each time step. Given the above central force, this function is defined as

```
def grav2(s,t,GM):
    alpha = -.013117283951
    r = np.array([s[0], s[1]])
    v = np.array([s[2], s[3]])
    accel = -(GM*r/la.norm(r)**3) * (1-(alpha/la.norm(r)))
    deriv = np.array([v[0], v[1], accel[0], accel[1]])
    return deriv
```

where `alpha` has been calculated to show $n = 8$ precessions with closure. This leads to the equation

$$\frac{360^\circ}{8} = \frac{1-a}{a}$$
$$a = \frac{8}{9}$$
$$\frac{8}{9} = \sqrt{1 + G\left(\frac{Mm^2\alpha}{L^2}\right)}$$
$$\frac{8}{9} = \sqrt{1+16\alpha}$$
$$\alpha \approx -.013117283951$$

For values defined as $GM = 4\pi^2$, $\{r_0, v_0\} = \{1, \pi/2\}$, and $m = 1$.

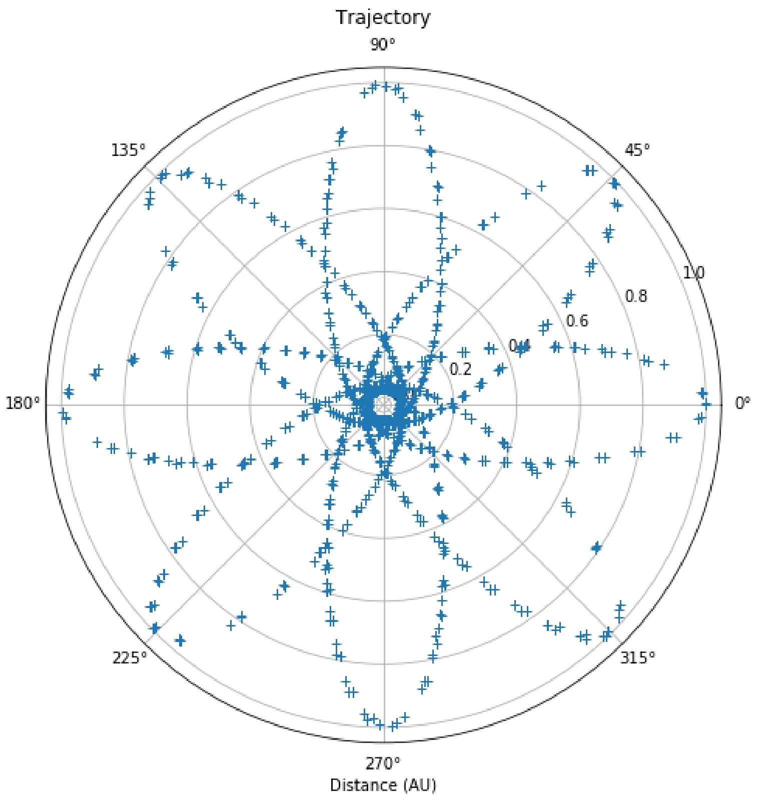


Figure 7: Look! I made a flower with math!.

This leads to the results in Figure 8, clearly showing an orbit the precesses precisely 45° each rotation. Note that 1000 steps were taken, which leads to 24 closed orbits. A fun trick to see where the apogees fall in terms of degree:

```
maxes = [int(rplot[0])]
for i in range(len(rplot)):
    if i>2:
        if rplot[i] < rplot[i-1] and rplot[i-1] > rplot[i-2]:
            maxes.append(i-1)
thplot = np.asarray(thplot)
maxes = np.asarray(maxes)
print(thplot[maxes]*180/np.pi)
```

Which leads to an array of $[0.45, 43.60, 91.69, 133.26, -178.69, ...]$. The slight discrepancy from expected values can be attributed to the error allowance in our adaptive routine although we can visually confirm closure of the orbits as there are more than eight plotted.

5 Problem 3.14

Add a drag force to the comet – $\mathbf{F}_d = C|\mathbf{v}|\mathbf{v}$*, fixing the constant* C *such that* $|\mathbf{F}_g(\mathbf{r}_1)| = 100|\mathbf{F}_d(\mathbf{v}_1)|$*. Show that the kinetic energy, averaged over an orbit, increases with time.* We have refined the simple model of an orbital body substantially; let us apply these refinements to a more complex model. Looking back to Project 2, a good place to start would be to add a drag force to the comet. But why? A comet travels through the vacuum of space, what is it interacting with to cause drag? The answer lies in the tail of the comet. Solar wind causes a radiation pressure on comets, leading to the ablation of material from the surface and causing the comet's tail.

We need to first find the drag coefficient:

$$|\mathbf{F}_g(\mathbf{r}_1)| = 100|\mathbf{F}_d(\mathbf{v}_1)| \longrightarrow GM\frac{\mathbf{r}_0}{\mathbf{r}_0^3} = 100\left(C|\mathbf{v}_0|\mathbf{v}_0\right)$$
$$C = .01\left(\frac{GMr_0^{-2}}{v_0^2}\right)$$

For values defined as $GM = 4\pi^2$, and $\{r_0, v_0\} = \{1, \pi/2\}$ this leads to $C = .16$. Defining a new function to calculate the derivatives given a constant drag force as

```
def gravdrag(s,t,GM):
    c = .16
    r = np.array([s[0], s[1]])
    v = np.array([s[2], s[3]])
    accel = -GM*r/la.norm(r)**3 - c*la.norm(v)*v
    deriv = np.array([v[0], v[1], accel[0], accel[1]])
    return deriv
```

Allows us to calculate a trajectory, shown in Figure 9.

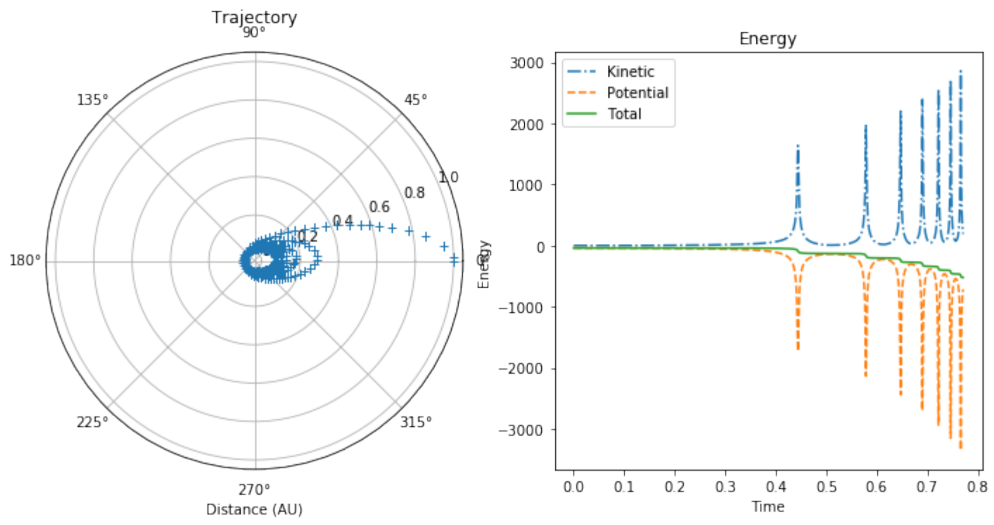


Figure 8: Comet experiencing drag force.

On this plot, we can visually confirm that the kinetic spikes appear to be increasing over each orbit while the total energy decreases. This indicates a decaying orbit, as is verified in the polar plot of the trajectory. This, however, doesn't answer the question. To determine the average energy over an orbit, a similar trick can be used as was used in the previous problem. Picking out the energy spikes allows us to divide the data into distinct orbits over which we can use a vectorized mean function. Plotting the average energies of the orbits at the temporal end points of the orbits, as in Figure 10, we can now say for certain that the average kinetic energy increases over each orbit.

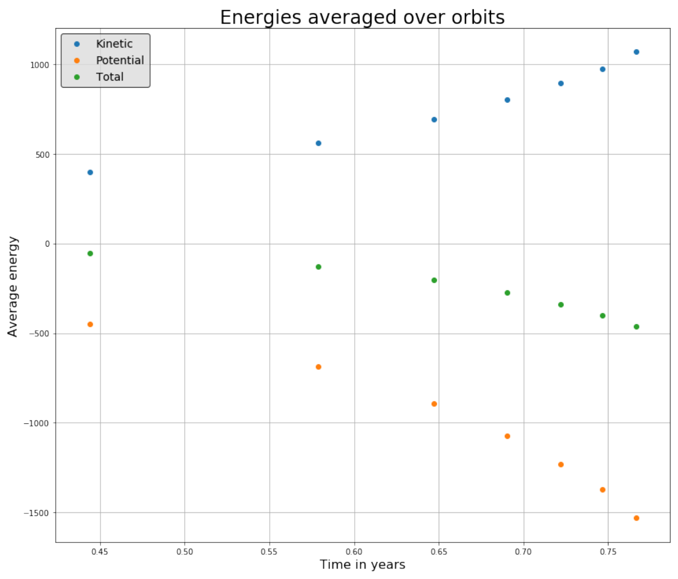


Figure 9: Orbital average energies.

Conclusion:

In this project, we have further explored finite difference method implementations for ordinary differential equations as applied to central force problems. In this exploration, we began with the Euler method, which was unstable, prompting the implementation of a method that would conserve energy over an entire orbit. The Euler-Cromer method (conditionally stable) allowed us to accurately track Halley's comet but was prohibitively inefficient. We then looked to a method of a higher order – the Runge-Kutta fourth-order scheme. This workhorse allowed the same accuracy in tracking highly-elliptical orbits while doing so much more efficiently and with more stability than the Euler-Cromer method. Finally, we refined our methods further by introducing an adaptive time step to the Runge-Kutta routine: the pinnacle of this project.

Armed with the ability to analyze multiple orbits efficiently, we then began to explore more complex physical model. We first modeled precession, an important physical phenomenon involved in many central force phenomena. We then added a constant drag force on the body, hearkening back to earlier explorations in projectile motion. These cases illustrated the power of our implementation of the Runge-Kutta method, as we needed simply to change the acceleration term in the derivative calculation function to apply the method to an entirely new problem.

This project amplified the necessity for error and stability analysis for numerical methods; the magnification of local truncation error is what obviated the use of the Euler Method and, indeed, even the Runge-Kutta routine will become unstable with too large a time step. (Preview for stability criteria!) Much of this analysis was performed 'inline', that is I saw most of these phenomena while running various code cells with different parameters and, as such, aren't seen in this paper or in the Code Appendix.

In terms of programming, the implementations were rather straightforward, even the Runge-Kutta although this required much more research into its derivation for increased theoretical understanding. I continue to explore efficiency, significantly increasing the efficiency of my Euler-Cromer implementation through use of broadcasting and vectorizing, but had trouble applying this concept to the two flavors of Runge-Kutta. Given more time to fiddle, I'm sure I could have squeezed out even more performance from these two important methods.

Appendices

A: Code

References

[1] Central force problems. URL `https://www.reed.edu/physics/faculty/wheeler/documents/Classical.pdf`.

[2] Alejando L Garcia. *Numerical methods for physics.* 2015. ISBN 9781514136683.

[3] Volker Hohmann. *Numerical Methods for Physicists.* University of Oldenburg, 1 edition, Apr 2004.