

# Using Runge-Kutta 4 Method to solve Newton's Equations of Gravity

Angus Brewster

March 2024

## Abstract

In this program Newtonian gravity is explored through the orbit of a projectile about the earth and moon. A brief history of Newtonian gravity and the numerical method used (RK4) is covered before their theory is discussed. The program uses a menu system to navigate between its different functions. The impact of changing the time-step and initial conditions of the projectile is investigated as well as finding the correct initial conditions for a round trip around both celestial bodies.

## 1 Introduction

Gravity is one of the fundamental forces that describe the standard model. While gravity is by far the weakest of the fundamental forces, it is no less important than the others. In fact on cosmological scales, gravity is the main determining force in the evolution of the universe. Gravity has been studied by many people throughout history, Greek Philosophers as early as 4<sup>th</sup> century BC, Romans in 1<sup>st</sup> century BC, Byzantines in 6<sup>th</sup> century BC, Indians in 7<sup>th</sup> century BC and many others. Modern physics uses Newtonian gravity developed by Sir Isaac Newton in 16<sup>th</sup> century AD. Newton published his laws of universal gravitation, over 3 books collectively known as "Philosophiæ Naturalis Principia Mathematica", which translates to "Mathematical Principles of Natural Philosophy" (or the Principia for short) [3]. Newton's laws are the basis of Newtonian Mechanics, and still apply in many fields of Physics today.

Some problems in physics can be solved analytically, where algebra can be manipulated to find a solution straight away, however most problems in physics require numerical methods to solve them. Numerical methods are effectively a series of trial and error attempts which use educated guesses to get closer to the true value. Runge-Kutta numerical methods are group of numerical methods developed in Germany in the 20<sup>th</sup> century AD by Carl Runge and Wilhelm Kutta [1]. Runge-Kutta 4<sup>th</sup> order or RK4 is the most commonly used of these, so called because its error is proportional to the step-size to the fourth power.

## 2 Theory

### 2.1 Newton's Equations of Gravity

Newtons "Mathematical Principles of Natural Philosophy" is split into 3 books, with many prepositions. Newton's Law of Gravitation is seen in preposition 7 of book 3, where he states "Gravity exists in all bodies universally and is proportional to the quantity of matter in each" (translated from Latin)[3]. Earlier in the Principia, Newton had proved that all planets "gravitate" and that the gravity between two planets was proportional to "the square of the distance between their centres" through geometric reasoning. Newton had also stated in his third law of motion that "To every action there is always opposed an equal reaction: or, the mutual actions of two bodies upon each other are always equal, and directed to contrary parts." [2]. From both of these propositions Newton stated the force,  $F$ , between 2 objects with masses  $M$  and  $m$  is equal and proportional to the inverse square of their distance  $r$ . From this, the modern equation is formed:

$$F = \frac{-GMm}{r^2}. \quad (1)$$

The negative sign is present due to the fact that both bodies are attracted to one another.

Newton's second law states "The alteration of motion is ever proportional to the motive force impress'd; and is made in the direction of the right line in which that force is impress'd" from this follows the modern algebraic interpretation:

$$F = ma. \quad (2)$$

By equating the two forces, the equations can be arranged as:

$$ma = \frac{-GMm}{r^2}, \quad (3)$$

As  $a$  is the second derivative of displacement, equation 3 can be expressed as,

$$m\ddot{r} = \frac{-GMm}{r^2}. \quad (4)$$

These equations work in one dimension with scalars, however if they are to be used two dimensions, they need to be two dimensional vectors, thus equation 4 becomes

$$m\ddot{\mathbf{r}} = \frac{-GMm}{|\mathbf{r}|^2} \hat{\mathbf{r}}. \quad (5)$$

Where  $\hat{\mathbf{r}}$  is the unit vector in the  $\mathbf{r}$  direction. As  $\hat{\mathbf{r}}$  is defined as  $\frac{\mathbf{r}}{|\mathbf{r}|}$ , equation 5 can be re written as,

$$m\ddot{\mathbf{r}} = \frac{-GMm}{|\mathbf{r}|^3} \mathbf{r}. \quad (6)$$

For a system with 3 masses, 2 stationary and 1 mobile the equation becomes:

$$m\ddot{\mathbf{r}} = \frac{-GM_1m}{|\mathbf{r}_1|^3} \mathbf{r}_1 + \frac{-GM_2m}{|\mathbf{r}_2|^3} \mathbf{r}_2, \quad (7)$$

Where  $M_1$ ,  $r_1$  and  $M_2$ ,  $r_2$  are the distance away and masses of stationary objects one and two respectively.

It is worth noting, in this code the force exerted on the moon and earth by the rocket is not accounted for due to the fact that it would be of negligible magnitude.

## 2.2 Runge-Kutta 4 Derivative Method

In the general case that  $\frac{d\vec{x}}{dt} = \vec{f}(t, \vec{x})$  the Runge-Kutta method solves the derivative with a time-step of  $h$ , with the following equations:

$$\vec{X}_{i+1} = \vec{X}_i + \frac{h}{6} [\vec{f}_1 + 2\vec{f}_2 + 2\vec{f}_3 + \vec{f}_4] \quad (8)$$

$$t_{i+1} = t_i + h \quad (9)$$

where

$$\vec{f}_1 = \vec{f}(t_i, \vec{x}_i) \quad (10)$$

$$\vec{f}_2 = \vec{f}\left(t_i + \frac{h}{2}, \left(\vec{x}_i + \frac{h\vec{f}_1}{2}\right)\right) \quad (11)$$

$$\vec{f}_3 = \vec{f}\left(t_i + \frac{h}{2}, \left(\vec{x}_i + \frac{h\vec{f}_3}{2}\right)\right) \quad (12)$$

$$\vec{f}_4 = \vec{f}(t_i + h, (\vec{x}_i + h\vec{f}_3)) \quad (13)$$

In order to solve equation 7 the RK4 method needs to be implemented four times: the  $x$  and  $y$  position and the  $x$  and  $y$  velocities. Instead of writing this in equations, it is more succinct in code:

```

1 def f1(t, x, y, vx, vy):
2     return vx
3
4 def f2(t, x, y, vx, vy):
5     return vy
6
7 def f3(t, x, y, vx, vy):
8     return ((-m_e * G * x) / ((x**2 + y**2)**(3/2))) + ((-m_m * G * (x - x_moon)) / (((x - x_moon)**2 + (y - y_moon)**2)**(3/2)))
9
10 def f4(t, x, y, vx, vy):
11     return ((-m_e * G * y) / ((x**2 + y**2)**(3/2))) + ((-m_m * G * (y - y_moon)) / (((x - x_moon)**2 + (y - y_moon)**2)**(3/2)))
12
13 k1x = f1(t[i], x[i], y[i], vx[i], vy[i])
14 k1y = f2(t[i], x[i], y[i], vx[i], vy[i])
15 k1vx = f3(t[i], x[i], y[i], vx[i], vy[i])
16 k1vy = f4(t[i], x[i], y[i], vx[i], vy[i])
17
18 k2x = f1(t[i] + dt/2, x[i] + dt*k1x/2, y[i] + dt*k1y/2, vx[i] + dt*k1vx/2, vy[i] + dt*k1vy/2)
19 k2y = f2(t[i] + dt/2, x[i] + dt*k1x/2, y[i] + dt*k1y/2, vx[i] + dt*k1vx/2, vy[i] + dt*k1vy/2)
20 k2vx = f3(t[i] + dt/2, x[i] + dt*k1x/2, y[i] + dt*k1y/2, vx[i] + dt*k1vx/2, vy[i] + dt*k1vy/2)
21 k2vy = f4(t[i] + dt/2, x[i] + dt*k1x/2, y[i] + dt*k1y/2, vx[i] + dt*k1vx/2, vy[i] + dt*k1vy/2)
22

```

```

23 k3x = f1(t[i] + dt/2, x[i] + dt*k2x/2, y[i] + dt*k2y/2, vx[i] + dt*k2vx/2, vy[i] + dt*k2vy/2)
24 k3y = f2(t[i] + dt/2, x[i] + dt*k2x/2, y[i] + dt*k2y/2, vx[i] + dt*k2vx/2, vy[i] + dt*k2vy/2)
25 k3vx = f3(t[i] + dt/2, x[i] + dt*k2x/2, y[i] + dt*k2y/2, vx[i] + dt*k2vx/2, vy[i] + dt*k2vy/2)
26 k3vy = f4(t[i] + dt/2, x[i] + dt*k2x/2, y[i] + dt*k2y/2, vx[i] + dt*k2vx/2, vy[i] + dt*k2vy/2)
27
28 k4x = f1(t[i] + dt, x[i] + dt*k3x, y[i] + dt*k3y, vx[i] + dt*k3vx, vy[i] + dt*k3vy)
29 k4y = f2(t[i] + dt, x[i] + dt*k3x, y[i] + dt*k3y, vx[i] + dt*k3vx, vy[i] + dt*k3vy)
30 k4vx = f3(t[i] + dt, x[i] + dt*k3x, y[i] + dt*k3y, vx[i] + dt*k3vx, vy[i] + dt*k3vy)
31 k4vy = f4(t[i] + dt, x[i] + dt*k3x, y[i] + dt*k3y, vx[i] + dt*k3vx, vy[i] + dt*k3vy)
32
33 x[i+1] = x[i] + dt/6 * (k1x + 2*k2x + 2*k3x + k4x)
34 y[i+1] = y[i] + dt/6 * (k1y + 2*k2y + 2*k3y + k4y)
35 vx[i+1] = vx[i] + dt/6 * (k1vx + 2*k2vx + 2*k3vx + k4vx)
36 vy[i+1] = vy[i] + dt/6 * (k1vy + 2*k2vy + 2*k3vy + k4vy)

```

Listing 1: RK4 implementation in code for gravity on 2 celestial bodies and rocket.

## 3 Part A

### 3.1 Menu

Before any parts of the program could be used, a menu system was required for them to be accessed through. In this program, the menu system is a looping function which calls other functions for plotting orbits. Once such a function has been called and run, the menu re-calls itself. The user also has the option to end the program, when the menu function will pass the if, else loop and pass to end the program. The menu system also has an input checking implementation, which checks that the input options are valid through the use of a while true loop that tests the value for an error.

### 3.2 Orbit

First, to test that the code was functioning correctly, a calculation of a well known orbit close to the earth was done. The orbit chosen was that of the ISS as it is well known and can be calculated with relative ease. The orbit was also done with a larger initial velocity to create an elliptical orbit rather than a circular one, this was to test that different initial conditions were indeed providing different trajectories.

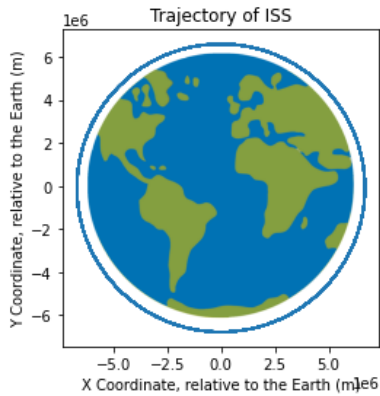


Figure 1: Simulation of the orbit of the ISS.

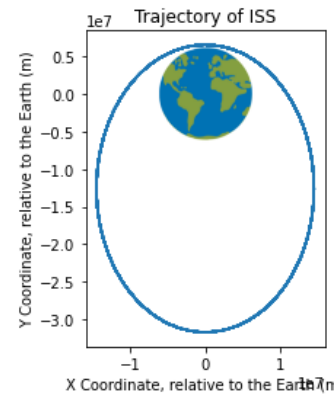


Figure 2: Simulation of near earth elliptical orbit.

Next, part a of the experiment was to plot the earth and moon's effects being taken into account in the calculation of the motion of the projectile. The rocket was given a distance from the centre of the earth equivalent to a low orbit of 6500km. Dependent on the initial velocity the rocket would either pass by the earth and the moon in an elliptical orbit or a figure 8 shaped orbit. The initial velocity also impacted whether the orbit was stable or not. If the orbit is stable it will be the same over an infinite amount of time. Below are 5 plots all with the same initial conditions with just the y velocity being altered.

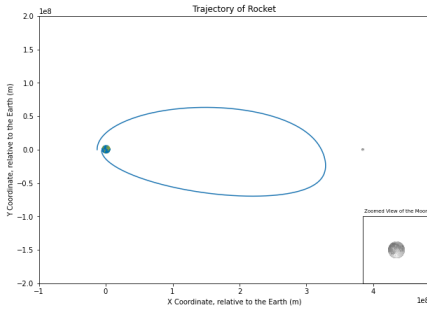


Figure 3: Rocket orbit,  
initial velocity: 7712.2m/s

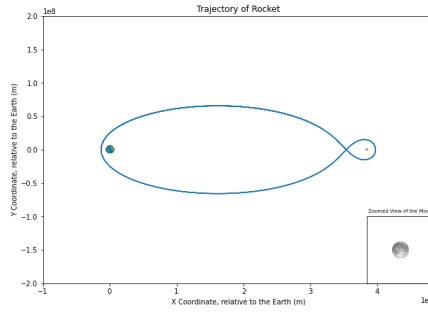


Figure 4: Rocket orbit,  
initial velocity: 7723.35m/s

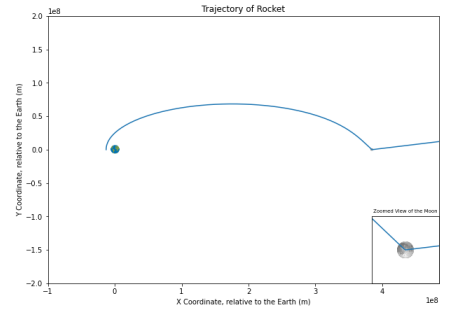


Figure 5: Rocket orbit,  
initial velocity: 7734.5m/s

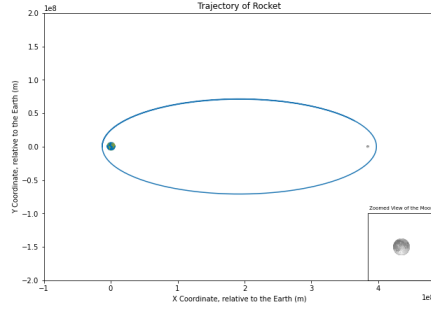


Figure 6: Rocket orbit,  
initial velocity: 7745.6m/s

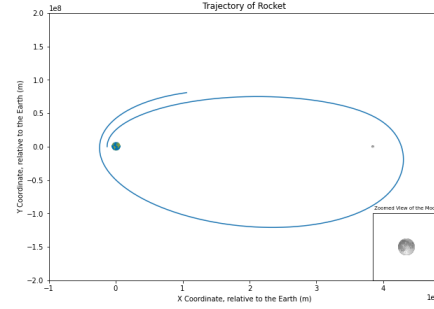


Figure 7: Rocket orbit,  
initial velocity: 7756.7m/s

Figure 4 and 6 are both stable orbits as they keep the same path regardless of time taken. There was an attempt to test for energy conservation, through summing the kinetic energy and gravitational potential energy. Unfortunately, whilst this could be calculated, it was not correlated with the time-step as it should be in a fourth order calculation. This indicates either an error in the methodology in calculating the error, or a problem with the calculation of the orbit.

The time-step,  $dt$ , was also varied:

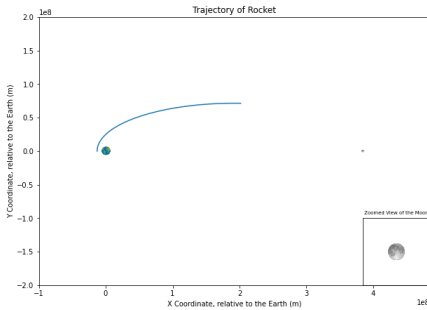


Figure 8: Rocket orbit,  
time-step: 0.1s

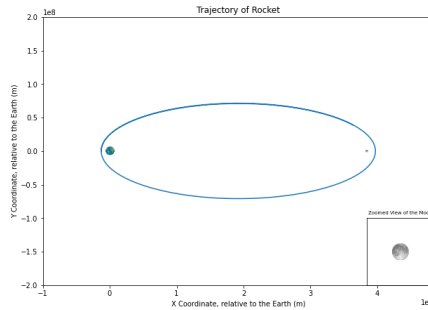


Figure 9: Rocket orbit,  
time-step: 1s

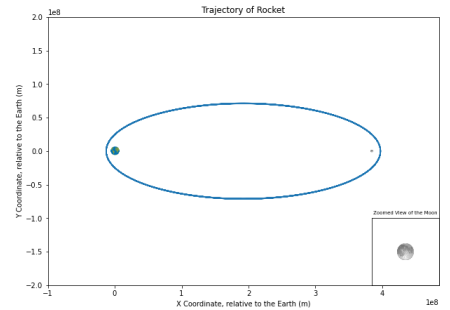


Figure 10: Rocket orbit,  
time-step: 10s

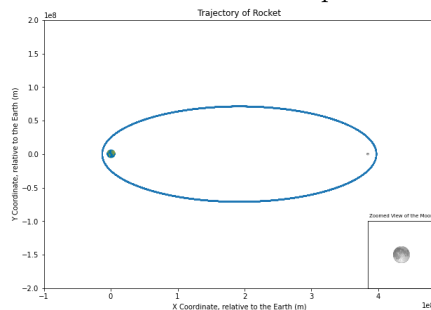


Figure 11: Rocket orbit,  
time-step: 100s

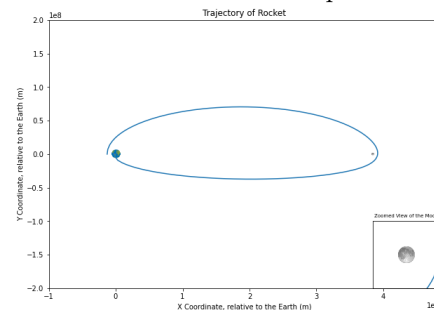


Figure 12: Rocket orbit,  
time-step: 1000s

From this we can see that if there is too small a time-step the orbit will not be able to complete without increasing the number of points calculated over. It can also be observed that if the time step is increased too far, the calculation becomes inaccurate as seen in figure 12

## 4 Part B

The goal of part B of the assignment was to have the rocket leave the earth's surface, orbit the moon and return to the earth. This was a case of testing multiple initial conditions through trial and error until the desired outcome was met. In order to check whether or not the rocket had returned to the earth, a test was done for each time step to check if the rocket's distance was more than the earth's radius. If the rocket had indeed returned to earth, the iteration stopped and there was a print statement stating as such, before plotting the graph. The print statement also stated the time the total journey took, for example for the initial position at  $-x$ , the journey took 64,0296 seconds. The initial coordinates for the rocket could be anywhere on the earth's radius (6371km from the origin) so there are many solutions. The starting test velocity chosen was the escape velocity of the earth at approximately 11km/s, the below solutions were found through trial and error.

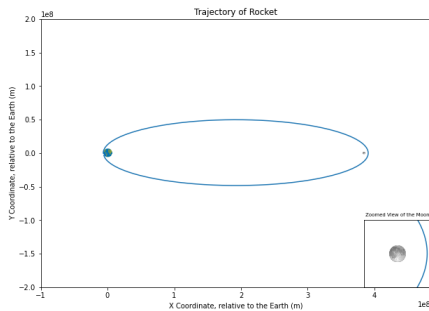


Figure 13: Returning rocket orbit, initially at  $-x$

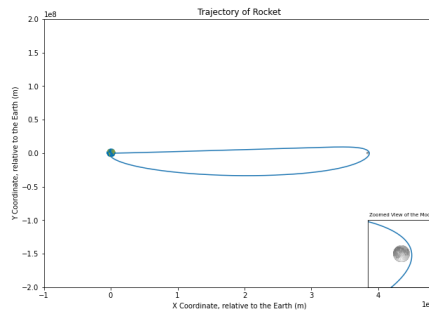


Figure 14: Returning rocket orbit, initially at  $+x$

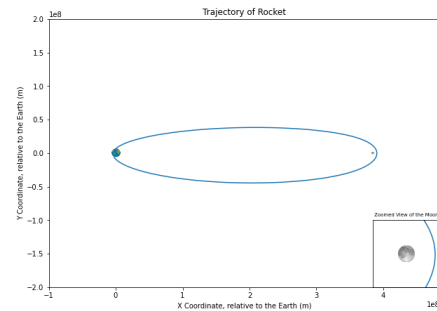


Figure 15: Returning rocket orbit, initially at  $x=0$

Depending on the magnitude of the initial speed, the rocket could get as close or far from the moon as desired.

## 5 Possible Improvements

The code could be improved in a number of ways.

- Faster iteration over the RK4 calculation
- A GUI would provide an easier to understand user interface
- Calculation of energy conservation

Faster iterations could be done over a number of methods. Firstly an overall function with 4 outputs rather than functions f1, f2, f3 and f4 would be more memory efficient and thus probably have faster calculation times. Using SciPy's built in `scipy.integrate.RK45` class could also yield faster iterations. The second of the problems could be solved by making an actual GUI built using something like Tkinter, resulting in a more intuitive user experience. A calculation of the energy conservation to test the accuracy of the program would be beneficial. Whilst this was attempted the results did not act as expected, so a proper implementation would be an improvement.

## References

- [1] J.C. Butcher. A history of runge-kutta methods. *Applied Numerical Mathematics*, 20(3):247–260, 1996.
- [2] Roderick W. Home. The third law in newton's mechanics. *The British Journal for the History of Science*, 4(1):39–51, 1968.
- [3] Isaac Newton. *Philosophiæ naturalis principia mathematica* / Autore Js. Newton, Trin. Coll. Cantab. Soc. Matheseos Professore Lucasiano, Societatis Regalis Sodali. Jussu Societatis Regiæ ac typis Josephi Streater. Prostat apud plures bibliopolas, Londini, 1687.