Exploring Projectile Motion and Orbital Simulations in Kerbal Space Programme

1.1 Introduction

A century ago, American physicist and engineer Robert Goddard had built the first liquid-fueled rocket, forever changing the course of human history through paving the way for modern spaceflight and exploration. A century later, I sit at my desk building virtual rockets to emulate humanity's successes in spaceflight on a computer game, Kerbal Space Program (KSP) where players manage a space program which aims to launch little green men called kerbals into space to explore the cosmos.

Figure 1: In-game screenshot

As someone who loves mathematics and applying it through physics, many mechanics of the game have intrigued me. I would like to tackle one of these mechanics in today's essay, specifically KSP's approach to modelling the projectile motion of rockets and other spacecraft.

1.2 Initial thoughts

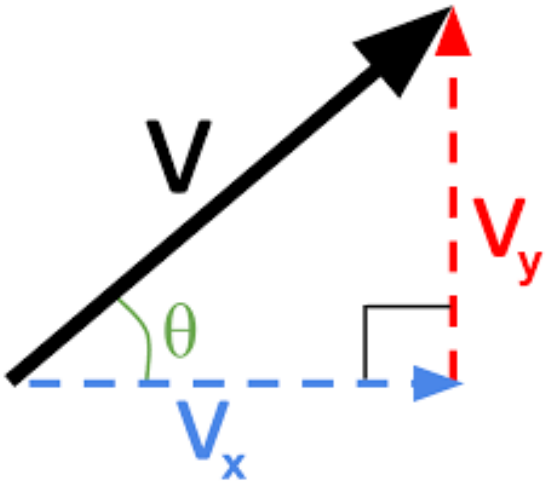

Figure 2: Illustration of resolving velocity vectors[1]

What initially came to mind that projectile motion can be modelled through simple quadratic equations by resolving a velocity vector into its horizontal and vertical components and applying one of the fundamental kinematic formulas:

$$x = \frac{1}{2}at^2 + ut$$

Where:
x is the displacement
a is the acceleration
t is the time
u is the initial velocity

This can be done to determine the horizontal and vertical displacement of a projectile at any point of time in its motion. Yet this method assumes ideal conditions where only the aforementioned variables matter. This is not feasible for video games like KSP where many different variables which continuously change come into play i.e., gravitational force, drag, and thrust.

1.3 Variables and Assumptions

For simplicity and my own sanity, we assume that the z axis remains constant, and thus limiting this exploration to 2 dimensions. Additionally we will not be taking into account any other aerodynamic forces other than drag (i.e Lift). Firstly,

Gravitational force (F_g)

is the attractive force that acts between any two objects with mass, which in our case is our spacecraft and KSP's analog for Earth, Kerbin. It always acts towards the centre of Kerbin and is defined by[2]:

$$F_g = \frac{GMm}{r^2}$$

Where,
G is the gravitational constant
M is the mass of Kerbin (assumed to be constant)
m is the mass of our rocket (decreases as fuel is burnt off)
r is the distance between the centre of Kerbin and our rocket (changes according to the rocket's position)

Drag (F_d)

is the resistive force is a type of friction force that acts opposite to the motion of objects moving through the air and its equation is[3]:

$$F_d = \frac{1}{2}\rho v^2 C_d A$$

Where,
ρ is air density (changes with altitude)
v is the current velocity (constantly changing)
C_d is the drag coefficient (assumed to be constant as the rocket's shape doesn't change)
A is the projected frontal area subjected to the airflow (assumed to be constant)

Thrust (F_t)

is the propelling force of the rocket and depends on the rocket engine used and the amount of throttle applied by the player.

1.4 Explicit Euler's Integration

Right now things may seem complicated with 9 different variables of which half of them are constantly changing. So how does the game convert all of this information to determine the path that our rocket takes?

With so many changing variables, the KSP takes it step by step. The game's physics engine updates at 50 instances a second and has a timestep (Δt) of 0.02 seconds between each update. At any particular update (n_{th} update), the game takes into account all of the variables and then calculates each of the aforementioned forces. With this the game determines the net force (F_{net}):

$$F_{net} = F_g + F_d + F_t$$

The game uses Newton's second law to calculate acceleration (a) which is equal to:

$$a = \frac{F_{net}}{m}$$

By solving the acceleration, other variables such as velocity (V) and displacement with respect the centre of the planet Kerbin (x) in the next update ($n+1_{th}$ update) can be derived:

$$V_{n+1} = V_n + a\Delta t$$

$$x_{n+1} = x_n + V_n\Delta t$$

Where Δt is the timestep of 0.02s

This method of solving differential equations is called explicit Euler's integration and uses the initial gradient to estimate the next value of a function over a small step size. In this case, every 0.02 seconds, the game is trying to solve the displacement of the spacecraft over time through solving the velocity (the rate of change of displacement) using the acceleration (the rate of change of velocity). Therefore, the velocity and position of the vessel is being updated in real time.

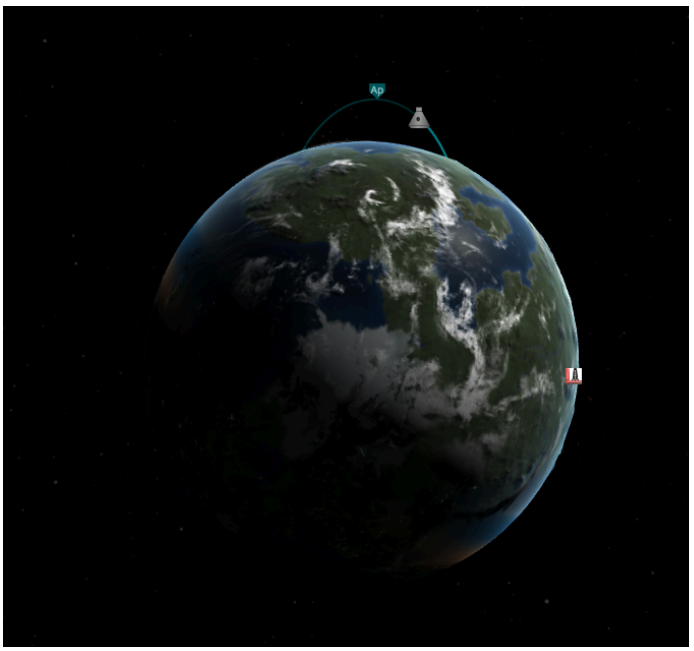

Figure 3: Simulated Trajectory In-game

Additionally, the game can plot the spacecraft's simulated trajectory by assuming the only force acting on it is gravity and determines its future position using Euler's integration.

1.5 Limitations

This works well for projectile motions near the earth however, being a game about space, most of the action happens once we reach orbit. In orbit, the explicit Euler's method runs into a big problem. Assuming the spacecraft reaches a circular orbit around Kerbin the total amount of energy it possesses is[4]:

$$E_{Total} = E_k + E_p = \frac{1}{2}mv^2 + \frac{1}{2}m\,\omega^2 x^2$$

Additionally due to the spacecraft's circular motion its acceleration can be modelled by:

$$a = -\omega^2 x$$

Therefore by using Euler's integration,

$$V_{n+1} = V_n - \omega^2 x_n \Delta t$$

$$x_{n+1} = x_n + V_n \Delta t$$

$$V_{n+1}^2 = (V_n - \omega^2 x_n \Delta t)^2$$
$$= V_n^2 - 2x_n V_n \omega^2 \Delta t + \omega^4 x_n^2 \Delta t^2$$

$$x_{n+1}^2 = (x_n + V_n \Delta t)^2$$
$$= x_n^2 + 2x_n V_n \Delta t + V_n^2 \Delta t^2$$

Therefore,

$$E_{n+1} = \frac{1}{2}m(V_n^2 - 2\omega^2 x_n V_n \Delta t + \omega^4 x_n^2 \Delta t^2 + x_n^2 \omega^2 + 2x_n V_n \omega^2 \Delta t + V_n^2 \omega^2 \Delta t^2)$$
$$= \frac{1}{2}mV_n^2 + \frac{1}{2}m\omega^2 x_n^2 + \frac{1}{2}m(\omega^4 x_n^2 \Delta t^2 + V_n^2 \omega^2 \Delta t^2)$$
$$= E_n + \frac{1}{2}m(\omega^4 x_n^2 \Delta t^2 + V_n^2 \omega^2 \Delta t^2)$$

Since m is a positive value, the term $\frac{1}{2}m(\omega^4 x_n^2 \Delta t^2 + V_n^2 \omega^2 \Delta t^2)$ is always positive therefore E_{n+1} is always increasing, an undesirable result since there are no external forces apart from gravity acting upon the spacecraft. In other words, the orbit becomes unstable with the orbital radius increasing over timesteps.

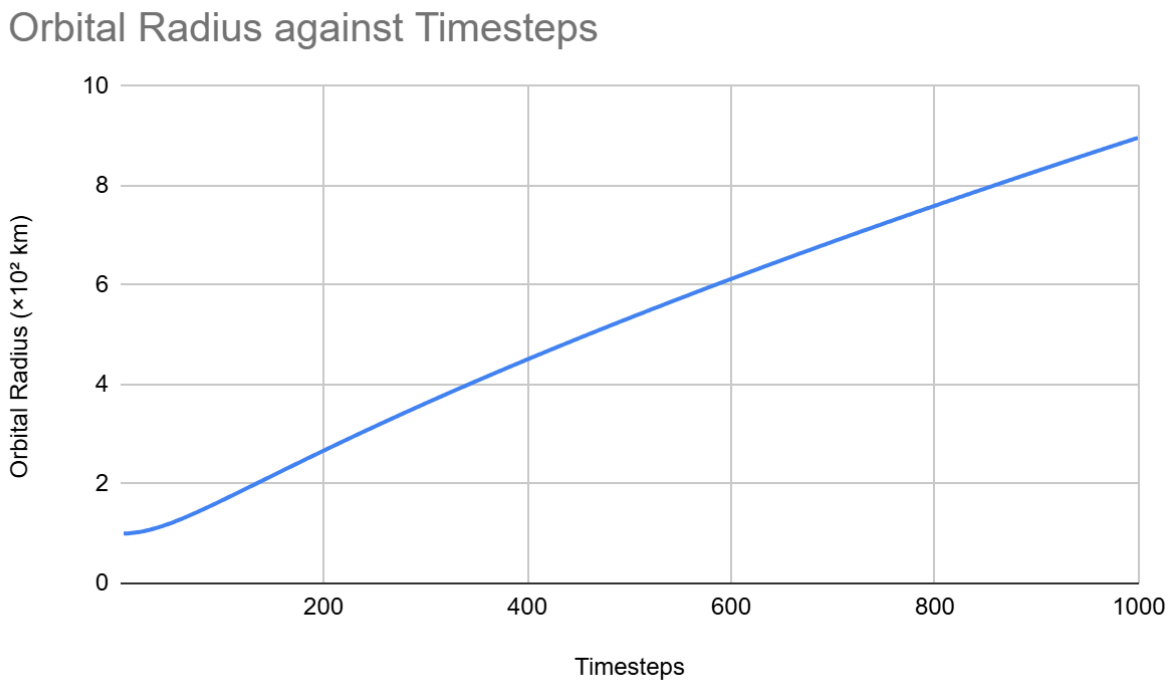

Figure 4: Graph of Orbital Radius against Timesteps using Explicit Euler's method

Using simplified values for constants (the gravitational constant, the mass of Kerbin and the initial orbital radius), I have plotted the orbital radius against timesteps in google sheets. As seen above, the orbital radius has an upward trend as the number of timesteps increases, demonstrating how the orbit would grow bigger and bigger and eventually spiral out of control.

1.6 Symplectic Euler's Integration

To correct this, the game applies a simple tweak to the explicit euler's method, changing the order of how the velocity and displacement are updated where the velocity in the next time step is used to calculate the new displacement of the spacecraft.

$$V_{n+1} = V_n + a\Delta t$$

$$x_{n+1} = x_n + V_{n+1}\Delta t$$

This is called symplectic Euler's method and this solves the aforementioned issue. Similarly, acceleration can be substituted for $-\omega^2 x$.

$$V_{n+1} = V_n - \omega^2 x_n \Delta t$$

$$\begin{aligned} x_{n+1} &= x_n + V_{n+1}\Delta t \\ &= x_n + (V_n - \omega^2 x_n \Delta t)\Delta t \\ &= x_n + V_n \Delta t - \omega^2 x_n \Delta t^2 \end{aligned}$$

$$\begin{aligned} V_{n+1}^2 &= (V_n - \omega^2 x_n \Delta t)^2 \\ &= V_n^2 - 2x_n V_n \omega^2 \Delta t + \omega^4 x_n^2 \Delta t^2 \end{aligned}$$

$$\begin{aligned} x_{n+1}^2 &= (x_n + V_n \Delta t - \omega^2 x_n \Delta t^2)^2 \\ &= (x_n + V_n \Delta t)^2 - 2(x_n + V_n \Delta t)(\omega^2 x_n \Delta t^2) + (\omega^2 x_n \Delta t^2)^2 \\ &= x_n^2 + 2x_n V_n \Delta t + V_n^2 \Delta t^2 - 2(\omega^2 x_n^2 \Delta t^2 + \omega^2 x_n V_n \Delta t^3) + \omega^4 x_n^2 \Delta t^4 \\ &= x_n^2 + 2x_n V_n \Delta t + V_n^2 \Delta t^2 - 2\omega^2 x_n^2 \Delta t^2 - 2\omega^2 x_n V_n \Delta t^3 + \omega^4 x_n^2 \Delta t^4 \end{aligned}$$

Therefore,

$$\begin{aligned} E_{n+1} &= \tfrac{1}{2} m V_{n+1}^2 + \tfrac{1}{2} m \omega^2 x_{n+1}^2 \\ &= E_n + \tfrac{1}{2} m(+ V_n^2 \omega^2 \Delta t^2 - \omega^4 x_n^2 \Delta t^2 - 2\omega^4 x_n V_n \Delta t^3 + \omega^6 x_n^2 \Delta t^4) \end{aligned}$$

Higher-order terms of order Δt^3 and Δt^4 are truncated, as their values become negligible due to the small timestep size. This leaves us with,

$$E_{n+1} = E_n + \tfrac{1}{2} m \omega^2 \Delta t^2 (V_n^2 - \omega^2 x_n^2) + \ldots$$

As such, theoretically symplectic euler's method would lead to the circular orbit remaining unchanged and hence stable as:

$$V_n^2 - \omega^2 x_n^2 = 0$$

Because in circular motion[5],

$$V_n = \omega x_n$$

Yet, due to Euler's method being used, the in-game spacecraft is moving in tiny little steps (straight lines) or in a massive polygon. Since the new velocity is used to update the displacement, the displacement is slightly overestimated. This causes a tiny erroneous increase in the displacement. Since,

$$V_{n+1} = V_n - \omega^2 x_n \Delta t$$

The greater displacement would lead to a smaller velocity in the next time step. Furthermore, since the displacement in the next time step is dependent on V_{n+1} where,

$$x_{n+1} = x_n + V_{n+1} \Delta t$$

The underestimated and hence smaller velocity in the next time step would lead to an underestimation of displacement in the next update creating a correction of the error. Therefore, one step slightly overestimates the displacement causing the next step to slightly underestimate it. This repeats in a cycle causing a slight self-correcting fluctuation in the orbit yet the orbit remains stable unlike when explicit Eulers is used.

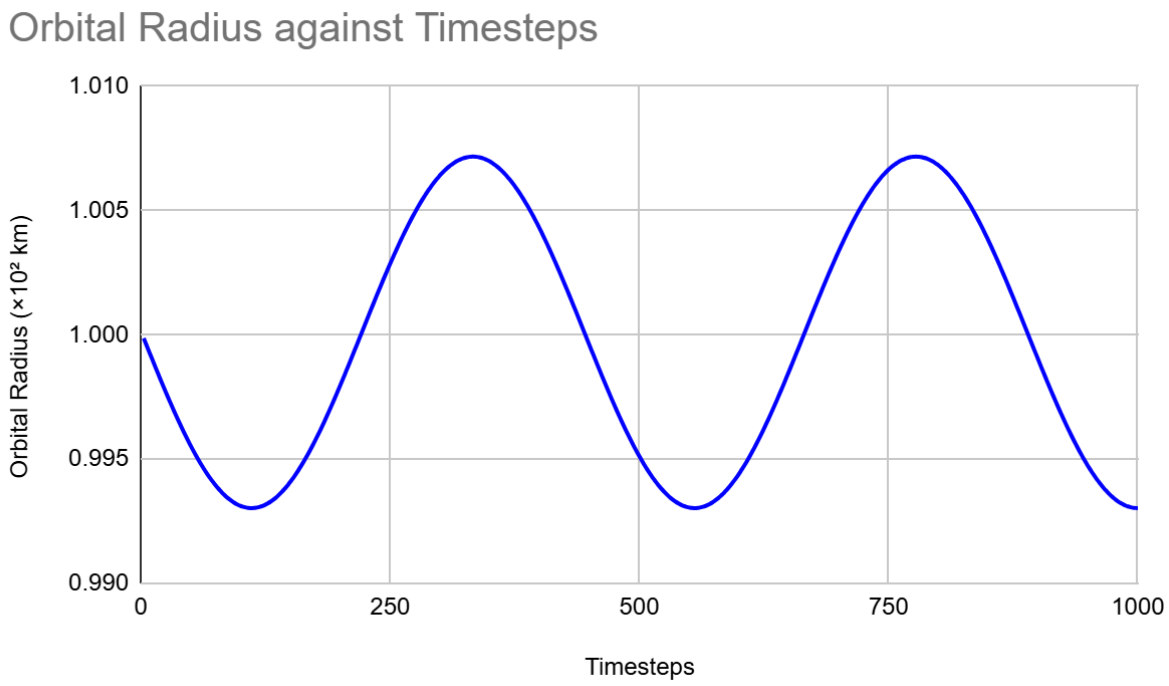

Figure 4: Graph of Orbital Radius against Timesteps using Symplectic Euler's method

Similar to before, simplified values were used for constants and the graph of orbital radius against timesteps was plotted in google sheets. Unlike before, there is no increasing trend in the orbital radius and hence the orbit is shown to be stable when simulated using Symplectic Euler's method. The sinusoidal pattern is due to the aforementioned reasons which are incorporated into the nature of Euler's method yet these deviations always return to the same value. As such, by using symplectic Euler's method, the game allows for the orbit to be stable and remain relatively unchanged over time albeit slightly fluctuating.

1.7 Conclusion

While symplectic Euler's method is not the most accurate method for simulating projectile or orbital motion, it is still able to achieve realistic results with a relatively low use of processing power. This is due to other more accurate methods such as Runge-Kutta 4, a higher order numerical method, requiring a significantly higher total number of calculations. As such, the use of symplectic Euler's method remains highly effective in the context of video games like Kerbal Space Programme.

Wrapping up, this exploration has been a fun way for me to apply the mathematics and physics that I had learnt in school. Additionally, I learnt a lot about how numerical methods such as explicit and symplectic Euler's integration allow not just Kerbal Space program but also many other video games to approximate physics in a real-time simulation.

Citations

1. Khan Academy. (2023). Khanacademy.org. https://www.khanacademy.org/science/ap-college-physics-1/xf557a762645cccc5:kinematics/xf557a762645cccc5:motion-in-2d/a/what-are-velocity-components
2. Hall, N. (2022, July 28). Drag Equation. Glenn Research Center | NASA; NASA. https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/drag-equation/
3. NASA. (2023, November 20). Weight Equation | Glenn Research Center | NASA. Glenn Research Center | NASA. https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/weight-equation-2/
4. 15.3: Energy in Simple Harmonic Motion. (2016, October 18). Physics LibreTexts. https://phys.libretexts.org/Bookshelves/University_Physics/University_Physics_(OpenStax)/Book%3A_University_Physics_I_-_Mechanics_Sound_Oscillations_and_Waves_(OpenStax)/15%3A_Oscillations/15.03%3A_Energy_in_Simple_Harmonic_Motion
5. OpenStax, & College, D. of P. and A. at D. (n.d.). 9.1 Rotation Angle and Angular Velocity. Pressbooks.bccampus.ca. https://pressbooks.bccampus.ca/douglasphys1107/chapter/9-1-rotation-angle-and-angular-velocity/