Modelling the orbit of Comet Halley

IB Mathematics Exploration

Introduction

I have been interested in astronomy since a very young age. I would keep a log of the phases of the moon and I would stay until night to see meteor showers when I heard they were coming in the news. In recent years I learned an array of things. However, I was intrigued when I learnt about modelling in mathematics followed by how the planets orbit the way they do in physics, since then I have been wanting to model an orbit of some astronomical object. Seeing how multiple areas of mathematics which are taught at a theoretical level can come together to solve practical real-world problems has always been mesmerising to me.

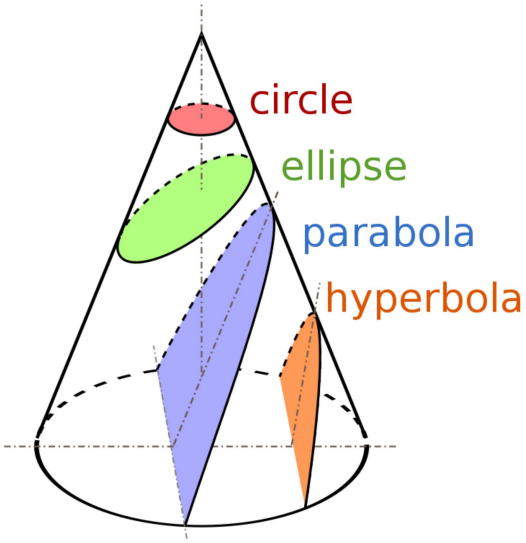
While exploring the different astronomical objects in the universe, I stumbled across a comet named Halley, I quickly recalled being taught about the comet in my history lesson about its history in the Norman invasion of England in 1066, however, at the time I was only taught about the history of the comet, not the astronomical importance that the comet had. It was one of the first comets to be hugely studied among astronomers around the world. I found out that Halley is the only known periodic comet that is regularly visible to the naked eye from Earth, and the only naked-eye comet that can appear twice in a human lifetime. It last appeared in the inner parts of the Solar System in 1986 and will next appear in mid-2061. I was astonished to find out that one day I could possibly see the comet of such historical importance with my own eyes! Also finding out that it took such a long time of approximately 70 years to orbit the sun, the area it would cover in one orbit around the sun was a measure that I couldn't comprehend in my mind. Therefore, I set out an aim to model the orbit of the comet Halley using different methods and finally, find out how much area would it sweep out in its one orbit around the sun.

Aim

As established in the introduction, this exploration aims to model the comet Halley's orbit and find out the total area it covers in its one full orbit around the sun. I will use two approaches, firstly by assuming that the orbit of the comet is perfectly ellipse and finding an equation of an ellipse to model its orbit then perhaps integrating it to find the total area covered. Secondly, I will be modelling the orbit using laws of physics to calculate the forces acting on the comet as it orbits through the solar system. This will let me determine if the orbit is indeed elliptical and show discrepancies if any. I think that the forces from other planetary bodies should affect the comet's orbit. Therefore, I will find a general function to find the area covered by the comet over a period of time and later add the area covered throughout the orbit to get the total area covered. Finally, I will compare the results from both methods to see if my assumptions are valid.

Upon researching I found out that NASA doesn't provide the specific data I need on the orbit of the comet publicly, therefore, I will use the properties of the comet's orbit that they do provide and using physics and the help of computer science, model the path it takes in order to find the variables that I need to find the area it covers over the full orbit.

Conic Sections



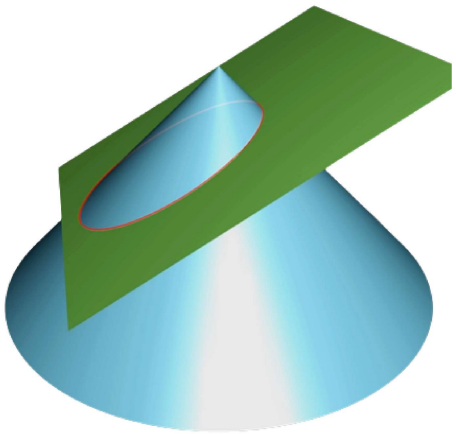
.

A **conic section** is the intersection of a plane and a right circular cone. By changing the angle of the plane the intersection can be: a **circle**, an **ellipse**, a **parabola**, or a **hyperbola** as shown in the figure.

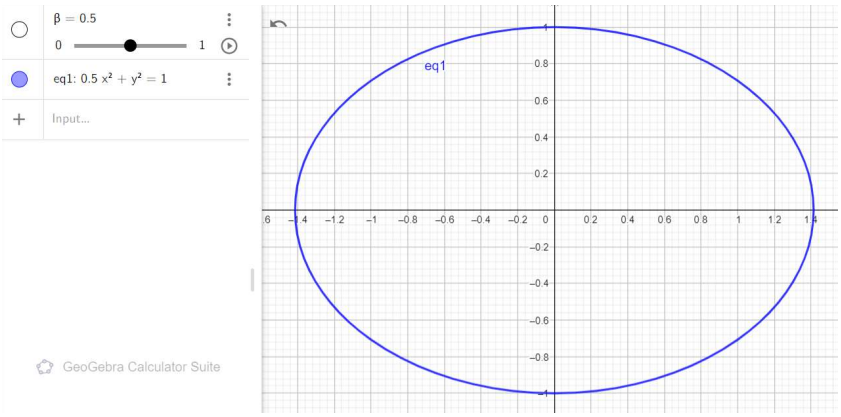
Curiously, in astronomy, using the Newtonian approach in solving two-body problems to trace a path an astronomical object orbits always corresponds to one of the four conic sections. If two stars are gravitationally bound, then their orbits are always either elliptical or circular. If a comet falls from a great distance, with no initial velocity the path is parabolic, whereas if a fly-by occurs where the initial velocity is significant, the path is hyperbolic.

Ellipse

Of the various conic sections, the one I wanted to explore the most was the **ellipse** as it's the most common orbit trajectory in our solar system and beyond. Most importantly, it was the one that **comet Halley** followed; the orbit that I'm exploring.

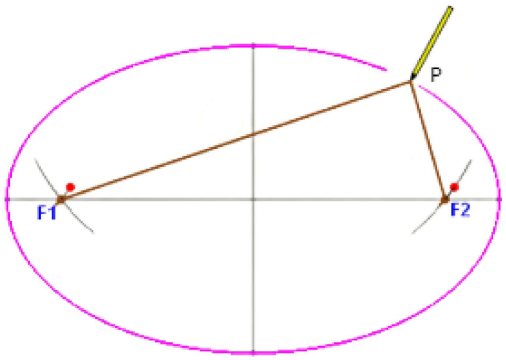


There are 3 major ways to define an ellipse; one of them is slicing a cone with a plane at an angle, an angle that is smaller than the slope of the cone itself. The curve of points where the plane and cone intersect forms an ellipse. I always thought that slicing a cone would produce a shape bulge towards one side like an egg, but turns out it creates a perfectly symmetrical shape, an ellipse!



Another way to create an ellipse is to take a circle and stretch it out in one dimension by a constant β as shown in the figure to the left. I took a unit circle and stretched it by a factor β.

Equation of an Ellipse



Lastly, another method to define an ellipse *(most common)* is as follows: pin one end of the rope to the paper $(F1)$ and pin the other end $(F2)$ on the paper some distance from $(F1)$ in such a way that the rope is not taut. Then pull the rope taut using the pencil and move the pencil around (while keeping the rope taut) to draw an ellipse. $F1$ and $F2$ now are the foci of the ellipse.

$$|PF1| + |PF2| = \sqrt{(P_x - F1_x)^2 + (P_y - F1_y)^2} + \sqrt{(P_x - F2_x)^2 + (P_y - F2_y)^2} = constant$$

This is the defining factor of an ellipse.

When I extend the string horizontally, it's the longest diameter of the ellipse, which is called the major axis $\therefore constant = 2a$

Similarly, the shortest diameter is called the minor axis.

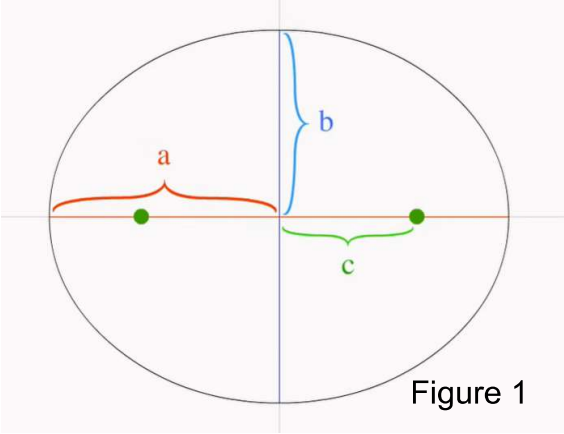


Figure 1

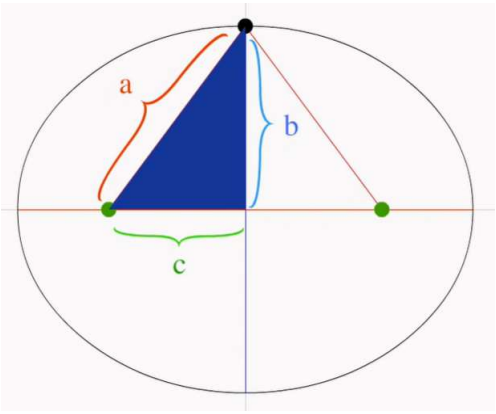
$Let,$

$a = \frac{1}{2} major\ axis$

$b = \frac{1}{2} minor\ axis$

$c = distance\ from\ centre\ to\ one\ of\ the\ foci$

$\therefore major\ axis = 2a$

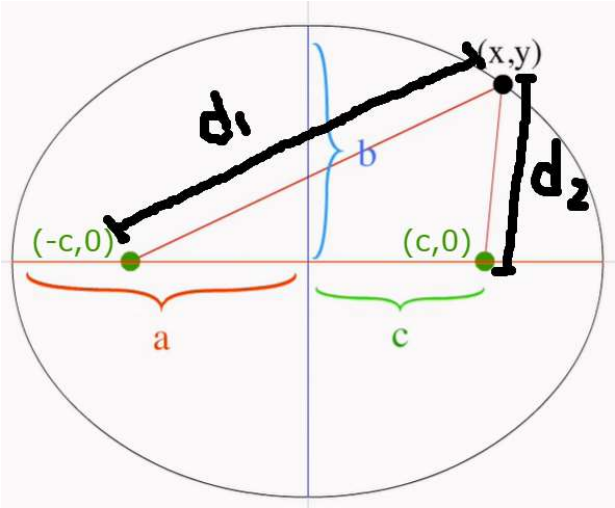


Extending the string evenly at the top, I formed a right angle triangle with sides a, b, c

According to the pythagorean theorem, $b^2 + c^2 = a^2$

Making b the subject of the formula I got:

$$b^2 = a^2 - c^2 \quad \textbf{[eq1]}$$



Length of the string is equal to the length of the major axis, $\therefore d_1 + d_2 = 2a$

Using the distance between two points formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

(Substituting into the distance equation)

$$\sqrt{(x + c)^2 + (y - 0)^2} + \sqrt{(x - c)^2 + (y - 0)^2} = 2a$$

Simplifying this with a series of algebraic steps I get:

$$\sqrt{(x + c)^2 + (y)^2} + \sqrt{(x - c)^2 + (y)^2} = 2a$$

$$\sqrt{(x - c)^2 + (y)^2} = 2a - \sqrt{(x + c)^2 + (y)^2}$$

Squaring both sides to get rid of the square root:

$$\left(\sqrt{(x - c)^2 + (y)^2}\right)^2 = \left(2a - \sqrt{(x + c)^2 + (y)^2}\right)^2$$

Expanding the parenthesis

$$(x - c)^2 + y^2 = 4a^2 - 4a\sqrt{(x + c)^2 + y^2} + (x + c)^2 + y^2$$
$$x^2 - 2xc + c^2 + y^2 = 4a^2 - 4a\sqrt{(x + c)^2 + y^2} + x^2 + 2xc + c^2 + y^2$$

Simplifying the equation I get,

$$-2xc = 4a^2 - 4a\sqrt{(x + c)^2 + y^2} + 2xc$$

$$4a\sqrt{(x + c)^2 + y^2} = 4a^2 + 2xc + 2xc$$

$$4a\sqrt{(x + c)^2 + y^2} = 4a^2 + 4xc$$

$$a\sqrt{(x + c)^2 + y^2} = a^2 + xc \quad \textit{[Dividing by 4]}$$

Squaring both sides again to eliminate the square root

$$\left(a\sqrt{(x+c)^2+y^2}\right)^2 = \left(a^2+xc\right)^2$$

$$a^2x^2 + 2a^2xc + a^2c^2 + a^2y^2 = a^4 + 2a^2xc + x^2c^2$$
$$a^2x^2 + a^2c^2 + a^2y^2 = a^4 + x^2c^2$$

$$a^2x^2 - x^2c^2 + a^2y^2 = a^4 - a^2c^2$$

Factorising

$$x^2(a^2 - c^2) + a^2y^2 = a^2(a^2 - c^2)$$

Dividing both sides by $a^2(a^2 - c^2)$

$$\frac{x^2(a^2-c^2)}{a^2(a^2-c^2)} + \frac{a^2y^2}{a^2(a^2-c^2)} = \frac{a^2(a^2-c^2)}{a^2(a^2-c^2)}$$

Eliminating the common factors

$$\frac{x^2}{a^2} + \frac{y^2}{(a^2-c^2)} = 1$$

As showed in **eq1** earlier, $b^2 = a^2 - c^2$

Substituting for b^2, I got the following equation of an ellipse.

$$\therefore \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad \textbf{[eq2]}$$

Now that I have an equation of an ellipse, I can change its parameters to model the path the comet Halley follows.

Comet Halley

Comet Halley *(1P/Halley)*, is a periodic comet visible from Earth every 75–76 years. The comet is named after an English astronomer Edmond Halley, who looked into claims of a comet approaching Earth in 1531, 1607 and 1682. He concluded that these three comets were actually the same comet returning over and over again, and predicted that it would return again in 1758. It was the first comet to be recognised as periodic. Unfortunately, Halley didn't live to see the comet's correctly-predicted return, but the comet did return as he predicted and so the comet was given his name.

As the properties like distance covered by the ellipse are going to be astronomical, I decided to work in astronomical units to make calculations easier to follow.
$1\ AU\ =\ 149597871\ km$

Now, I needed to find the missing variables a and b to model the elliptical orbit of this comet using the equation derived above **eq2**:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Properties known for the comet Halley:
$perihelion\ =\ 0.586\ AU$ *(the point in the orbit of a planet, asteroid, or comet at which it is closest to the sun)*
$aphelion\ =\ 35.082\ AU$ *(the point in the orbit of a planet, asteroid, or comet at which it is furthest from the sun)*
semi-major axis $a\ =\ 17.83\ AU$
eccentricity $e\ =\ 0.967$
(source: ssd.jpl.nasa.gov)

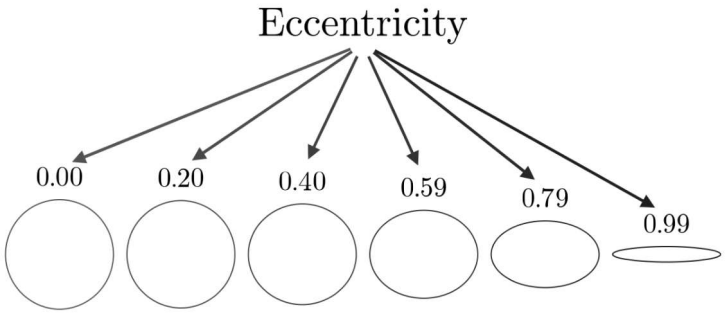
Eccentricity

Eccentricity is the “squisification” of the circle. It’s defined as $e\ =\ \frac{c}{a}$ (e being the eccentricity),
For $0\ <\ e\ <\ 1$ we obtain an ellipse, for $e\ =\ 1$ a parabola, and for $e\ >\ 1$ a hyperbola.

From **eq1** above, we know that $b^2 = a^2 - c^2$
$\therefore c = \sqrt{a^2 - b^2}$

$e\ =\ \frac{\sqrt{a^2-b^2}}{a}$ *(Substituting c into the equation)*

$e\ =\ \sqrt{1 - \frac{b^2}{a^2}}$



As we know the eccentricity (e) and the semi-major axis (a) of the comet, I can use that to find the value of the semi-minor axis (b) by substituting the known values in the equation below.

$$0.967 = \sqrt{1 - \frac{b^2}{17.83^2}}$$

$$b = 4.54\,AU$$

Substituting the values of a and b, I got this equation of the ellipse that models the orbit of the comet Halley.

$$\therefore \frac{x^2}{17.83^2} + \frac{y^2}{4.53^2} = 1$$

Finding the area

Now that I modelled the orbit that the ellipse follows, I found the **total area** covered by the comet in its full orbit around the sun from the ellipse equation model. I used integration to find the area by integrating the formula of the ellipse as derived above **eq2:** $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

Making y the subject of the formula I get: $y = b\sqrt{1 - \left(\frac{x}{a}\right)^2}$

Simplifying it further, I now have an equation in terms of y which I can integrate: $y = \frac{b}{a}\sqrt{a^2 - x^2}$

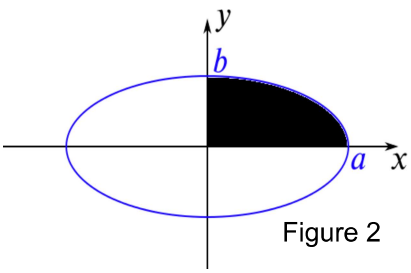


Figure 2

As shown in **figure 2**, I could see that the ellipse is divided into four quadrants. So I thought of calculating the area of 1 quadrant and multiplying the result by 4 to get the area of an ellipse.

$$A = 4\int_0^a y\,\mathrm{d}x \qquad (let\ A\ be\ the\ total\ area)$$

$$A = 4\int_0^a \frac{b}{a}\sqrt{a^2 - x^2}\,\mathrm{d}x \qquad \textit{(Substituting for y)}$$

$$A = 4\frac{b}{a}\int_0^a \sqrt{a^2 - x^2}\,\mathrm{d}x$$

As $\frac{b}{a}$ is a constant, to make the integration easier, I took it out of the integral which is allowed by the properties of integration

Substituting $x = a\,sin(t)$ **[eq3]** *(simplifies the equation later using trig identities)*

$$dx = a \cdot cos(t)\,dt$$

Using definite integrals by substitution, changing the limits:

$x = a$ changes to $t = \frac{\pi}{2}$ (as **eq3** becomes: $a = asin(t),\ t = sin^{-1}(1) = \frac{\pi}{2}$)

$x = 0$ changes to $t = 0$ (as **eq3** becomes: $0 = asin(t),\ t = sin^{-1}(0) = 0$)

The following trig identities will come in use in the following steps

$cos2x = cos^2x - sin^2x$ *(Trig identity)*

$sin^2x + cos^2x = 1$ *(Trig identity)*

$cos2x = cos^2x - (1 - cos^2x)$ *(Substituting $(1 - cos^2x)$ for sin^2x)*

$cos^2x = \frac{cos2x}{2} + \frac{1}{2}$ *(Making cos^2x the subject as required)*

Making changes to the limits and substituting for x and dx gives us the following,

$$A = 4\frac{b}{a}\int_0^{\pi/2}\sqrt{a^2 - a^2\sin^2(t)}\cdot a\cdot\cos(t)\,\mathrm{d}t$$

$$A = 4\mathrm{b}\int_0^{\pi/2}\sqrt{a^2\left(1-\sin^2(t)\right)}\cdot cos(t)\,dt$$

(Cancelling out the term a (as it's constant I took it out of the integral) & taking a^2 as a factor)

$$A = 4\mathrm{ba}\int_0^{\pi/2}\sqrt{\cos^2(t)}\cdot cos(t)\,dt$$

(Trig identities substitution)

$$A = 4\mathrm{ba}\int_0^{\pi/2}\cos^2(t)\,dt$$

Simplifying the integration by substitution as cos^2x is equivalent with $\frac{cos2x}{2} + \frac{1}{2}$ as shown above

$$A = \int_0^{\pi/2}\frac{\cos(2x)}{2} + \frac{1}{2}\,\mathrm{d}x$$

$$A = 4\mathrm{ab}\left[\frac{\mathrm{t}}{2} + \frac{\sin2\mathrm{x}}{4}\right]_0^{\pi/2}$$

(Using GDC)

$$A = 4ab\left(\frac{\pi}{4}\right)$$

$$A = \pi ab$$

Using the equation of ellipse that I found earlier:

$$\frac{x^2}{17.83^2} + \frac{y^2}{4.53^2} = 1$$

And substituting the values for a and b into $A = \pi ab$, I got the **total area** of the orbit covered by the comet Halley in one orbit around the sun.

$$Area \approx \pi\cdot(17.83)\cdot(4.53) \approx 253.75\,AU^2 \approx 5.68\times10^{24}\,m^2$$

Use of approximation signs as I don't know the properties of the comet Halley that I used with high significant figures. The values I got were really high but as expected due to the orbit being highly elliptical as I found earlier. Just to help me visualise how much the area covered was, I tried to see approximately how many squares of length $5\,AU$ $(5\ times\ the\ average\ distance\ from\ sun\ to\ earth)$ would fit in that area covered by the comet. The area covered by this imaginary square would be $2.24\times10^{22}m^2$. To see how many of these would fit in the total area covered by the comet:

$\frac{5.68\times10^{24}\,m^2}{2.24\times10^{22}m^2} \approx 250$ squares of length $5AU$ would fit.

Although a length of $5\,AU$ was still an astronomical number, at least it helped me get a sense of how huge the area covered by the comet Halley was.

Physics modelling

I am now modelling the orbit of the comet Halley from a physics perspective, modelling the path of the comet without any presumptions of its orbit; different to what I did in the previous method where I postulated that the orbit is elliptical. To be able to model the comet's orbit precisely, I planned to simulate the solar system which would enable me to see the orbit the comet would take due to all the forces of attraction from the other astronomical objects such as mainly the sun but also the other planetary bodies in the solar system which I thought would affect the orbit. I would finally then be able to find out the area it covers in its orbit.

Background knowledge

I thought of using what's called an n-body simulation where I measure the force of attraction between an object (planet, comet, etc) and the sun, but also including the force of attraction between the other planets in the solar system as that was why I wanted to use this approach. To measure the force of attraction, I used Newton's law of universal gravitation.

$$F = \frac{GMm}{R^2}$$

$F = force\ of\ attraction$

$G = 6.67428\ Nm^2kg^{-2}$ *(Gravitational constant)*

$M = mass\ of\ the\ bigger\ object$

$m = mass\ of\ the\ smaller\ object$

$R = distance\ between\ the\ centres\ of\ the\ objects$

[Eq4]

It states that every particle attracts every other particle in the universe with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between their centres as apparent in the formula. I thought even though the gravitational force on the comet from the sun will be substantially higher than the other objects, it might be noticeable at times when it's really close to a big planet like Jupiter in its orbit as the $F \propto Mm$ & $F \propto \frac{1}{R^2}$, the big mass M of Jupiter (although just 0.1% of the sun's mass) and the radius getting smaller when it gets close to it would make the force of attraction significant. For that reason, I thought of choosing to use the n-body simulation method as it would find out the distortions caused by the other planet's gravitational force on its presumed elliptical orbit.

After I found the force of attraction, it allowed me to find the acceleration (a) caused by the resultant force so that I could find the new velocity of the comet. To do that, I used Newton's Second Law. He described it as the resultant force to be the change in momentum (p) over the change in time (t).

$$F_{net} = \frac{dp}{dt}$$

*__momentum__ is the quantity of motion of a moving body, measured as a product of its mass and velocity. $\vec{p} = \mathrm{m}\vec{\mathrm{v}}$

$\frac{dp}{dt} = \frac{m \cdot dv}{dt}$ (assuming the mass of the comet remains constant, we get the following by substituting for dp)

$a = \frac{dv}{dt}$ (Acceleration is defined as the change in velocity over change in time)

$\therefore F_{net} = ma$ **[eq5]**

With these two equations, I could find out the forces of attraction acting on the comet by the sun and the other planetary objects. Then finding the resultant force by adding all the force vectors would give me the acceleration on the comet.

Now for the simulation, I needed some properties of each object like their mass, distance from the sun and average velocity. Also, an ephemeris (data giving the positions of a celestial object at regular intervals throughout a period.) of the planets so that I had a point from where to start the simulation from.

Planetary Fact Sheet - Nasa
Converted some units $1AU = 149597871km$

Table 1

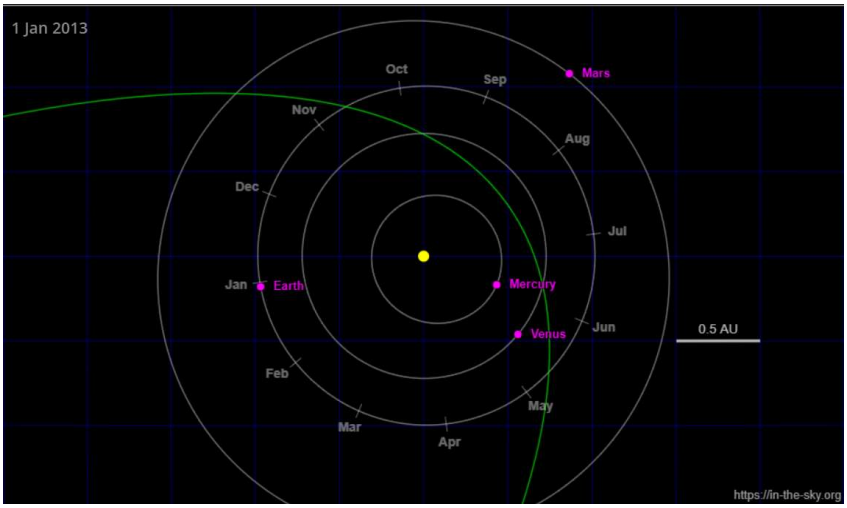
	MERCURY	VENUS	EARTH	MARS	JUPITER	SATURN	URANUS	NEPTUNE	PLUTO
Mass $(10^{24}kg)$	0.33	4.87	5.97	0.642	1898	568	86.8	102	0.013
Diameter (km)	4879	12,104	12,756	6792	142,984	120,536	51,118	49,528	2376
Distance to Sun (AU)	0.38704	0.7232	1.000	1.524	5.20395	9.57233	19.1647	30.18091	39.481
Perihelion (10^6km)	46	107.5	147.1	206.7	740.6	1357.6	2732.7	4471.1	4436.8
Aphelion (10^6km)	69.8	108.9	152.1	249.3	816.4	1506.5	3001.4	4558.9	7375.9
Orbital Period $(days)$	88	224.7	365.2	687	4331	10,747	30,589	59,800	90,560
Orbital Velocity (km/s)	47.4	35	29.8	24.1	13.1	9.7	6.8	5.4	4.7
Orbital Eccentricity	0.206	0.007	0.017	0.094	0.049	0.052	0.047	0.01	0.244

Sun: $mass = 1.989 \times 10^{30}\ kg$

Halley Comet:

$mass = 2.2 \times 10^{14}\ kg$
$perihelion = 0.586\ AU$
$aphelion = 35.082\ AU$
$orbital\ velocity = 54.5\ km$
$orbital\ eccentricity = 0.9672$
$orbital\ period = 76.1\ years$

The figure shows the position of planets in 2013. This will help me as I know where to place the planets when I first start the simulation. 2013 was the first time that I can remember seeing a meteor for the first time so being a significant day for me, I chose that year to start the simulation from that point on.



— 1P/Halley

provided by in-the-sky.org, compiled from nasa.gov

Explanation of the Simulation

Calculating the force and finding the change in velocity and updating the position of each object on each timestep *(an interval, eg: 1s)* manually would require an enormous amount of time and effort as the comet has an average orbital period of 75 years and I would need to calculate it for its entire orbit as I wanted to find the total area it covered. Instead, being a computer enthusiast and a student, I wrote a python program to do the calculations for me. Writing the code wasn't easy and took time and effort but it meant that I would have precise results in a short amount of time once the code was complete and I could reduce the timestep to smaller numbers to increase the accuracy; meaning that instead of calculating the forces month by month I could calculate it second by second. Following are the steps of calculations that were required and done in the simulation.

Steps

I worked in a 2D xy-plane. I set the position of the sun to be at the centre $(0, 0)$, I did this because it made finding the distance between the objects and the sun much easier as it would just be the magnitude of the position vector of the object.

Using values from **Table 1**, I put the starting position of the planets as their perihelion starting with their average velocity. As mentioned earlier, there is no secondary data available for the velocities of objects at a certain point in time or at a certain point in their orbit so I had to use the average. This however didn't affect the results much as I used their maximum velocity as their starting velocity and as their starting position was their perihelion, it is when the velocity would be at the maximum, this helped reduce the systematic error by quite a lot

Firstly, I found R, the distance between the two objects to substitute in **eq4**. I used the pythagorean theorem to find that as we know their x and y coordinates (being their perihelion).

$$distancebetween = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Now, I knew all the variables required for **eq4**, $F = \frac{GMm}{R^2}$

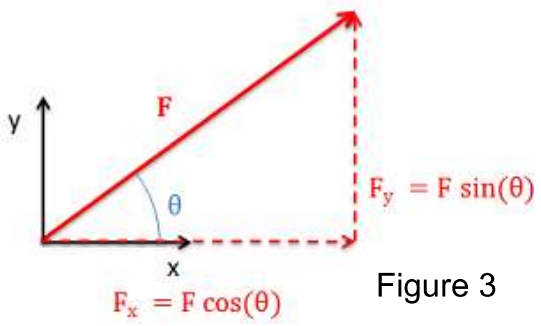
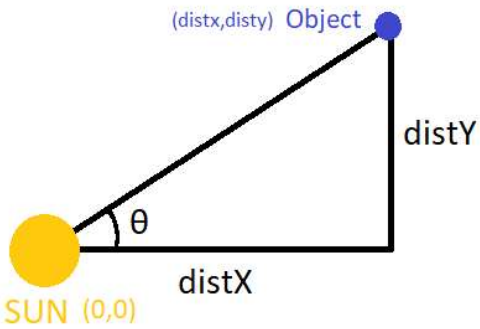


Figure 3

Substituting the values into the formula, I got the resultant force, now I need to find the x and y components that make up the resultant force. This is because I want to be able to move the position of the object in both x and y directions (as I am working in a xy-plane) and to do that I need to break the resultant force down into the two components. This could be done using trigonometry as shown in **figure 3.**

I know the $distX$ (x distance from the sun) and $distY$ (y distance from the sun) to the sun of the object because as previously mentioned, the sun's position is $(0, 0)$, so the object's x and y position is the distance from the sun. Therefore, to find the angle θ, I took the $arctan$ as shown:



$$\theta = tan^{-1}\left(\frac{distY}{distX}\right)$$

Now as I have the angle θ, I used that to find the x and y component of the resultant force using trigonometry as shown in **figure 3.**

$$F_x = F \times cos(\theta) \quad (let\ F_x\ be\ the\ x\ component\ of\ the\ F)$$

$$F_y = F \times sin(\theta) \quad (let\ F_y\ be\ the\ y\ component\ of\ the\ F)$$

Now that I have the force separated into x and y component, I updated the velocity of the object in both the directions (x and y); I essentially added the acceleration due to the force of attraction to the previous velocity the comet had as I knew what the starting velocity of the comet was. Using the **eq5** to find the acceleration and adding to the previous velocity, we get the new velocity of the comet as shown below.

let $velocity_x$ and $velocity_y$ be the velocity of the comet and $mass_{comet}$ be the mass of the comet

Rearranging **eq5** to be in terms of acceleration: $acceleration = \frac{Force}{mass}$

$$newvelocity_x = velocity_x + \frac{F_x}{mass_{comet}}$$

$$newvelocity_y = velocity_y + \frac{y}{mass_{comet}}$$

We know that $displacement = velocity \times time$

So finally, to find the new position coordinate of the object, I calculated the displacement in the specified $timestep$ in each x and y component and added it to its previous positions.

$$newdistX = distX + (newvelocity_x \times timestep)$$
$$newdistY = distY + (newvelocity_y \times timestep)$$

The code I wrote [see **appendix** for source code] would now calculate this for each object against all the other objects mentioned in **table 1** and iterate it over and over for the specified $timestep$ which I firstly set to $1s$. So it would now calculate the change in the velocity and position of the objects every second by second.

Finally, after starting the objects from their perihelion with their maximum velocity, the following **figure 3** shows what the simulation looked like. Also, I added images to the objects and background of our milky way to make it look good. Looking at **figure 3**, I postulated that the comet would in fact follow an elliptical path. But I wasn't certain yet as the comet hadn't completed a full orbit around the sun in the simulation.

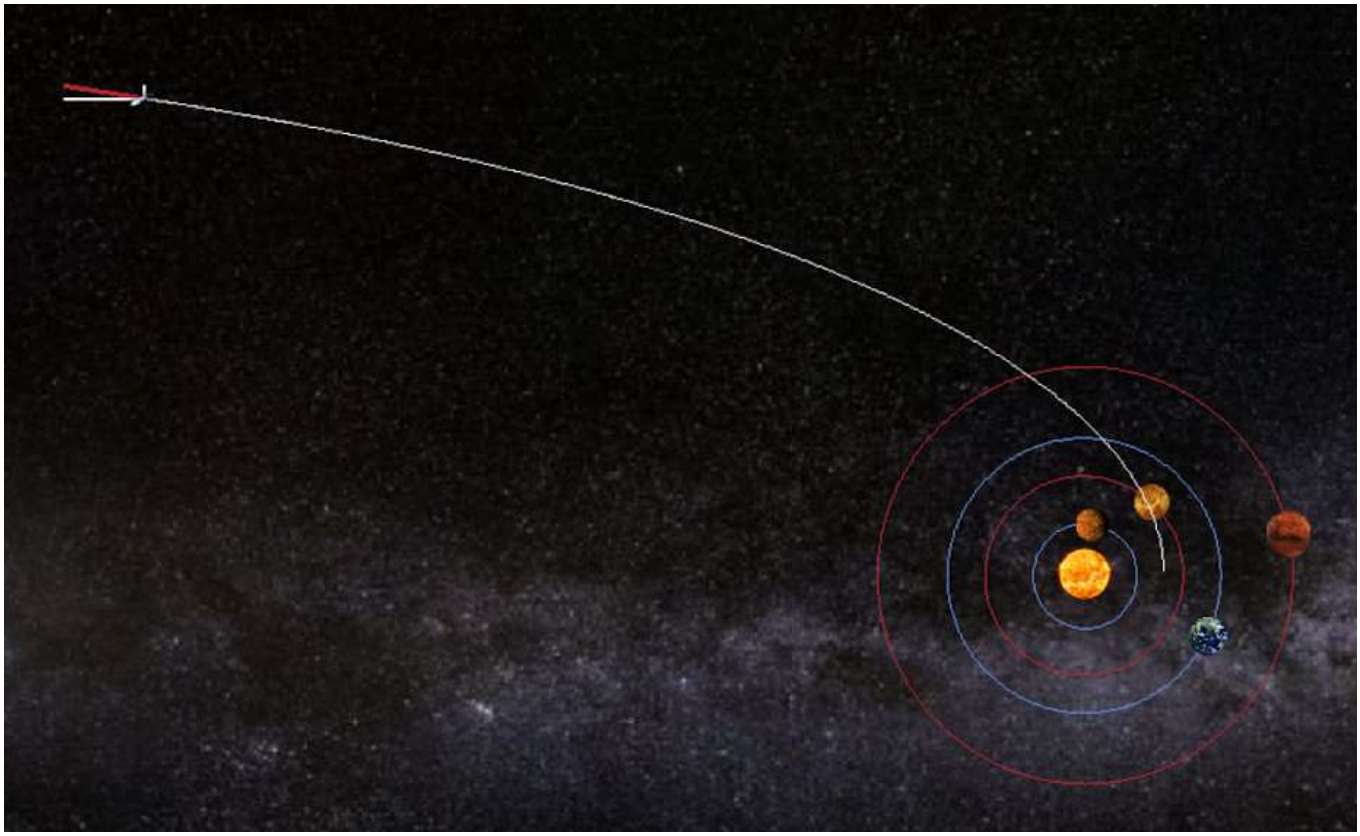


Figure 3 Orbits to scale, objects apparent size not to scale *(all images used in the simulation referenced in the bibliography)*

Running this for the first time was a pleasing experience as I could see the forces of attraction were being calculated and the positions of the objects were changing. Now for the comet to complete the full orbit, it would take some time as my computer could only handle roughly 120 iterations per second, i.e. every second in real-time, $120s$ would pass in the simulated space. Now that would be a pretty long time to wait for the comet to complete an orbit knowing that it had an eccentricity of 0.9672 and an average of 75 years of the orbital period. So I decided to shut off the graphical element of the simulation, it would still do the number crunching and save the output but not show the results graphically, which is quite an intensive task computationally. Doing that drastically improved the performance to about 3300 iterations per second. But still with comet Halley's orbital period of $\approx 2.37 \times 10^9\ seconds$ That meant it would still take roughly a week of computation for the comet to complete an orbit!

After pondering for a while, I hypothesised that the change in the velocity of the comet over such a tiny period of a second over second wouldn't be significant. To verify the hypothesis, I collected the data for 1 day of simulated time.

After looking at the data collected, the maximum velocity reached by the comet was of – $54511\ ms^{-1}$ and the minimum velocity reached was of – $54550\ ms^{-1}$ over the span of the day, which was a percentage difference of 0.0018%. Also recognising that the comet was starting from its perihelion, where the changes in velocity are the most drastic in the orbit in a short period of time compared to its full orbit and seeing that the difference is not big enough to affect the results too much, I decided to increase the $timestep$ for which the calculations were iterated to $1\ day\ (86400\ s)$, i.e. I would assume that the velocity remains constant for the period of $1\ day$. This reduced the time taken to compute its full orbit from roughly a week to less than $10s$. Also, the change in velocity over a period of a day would decrease as the comet approached its aphelion as the comet is at its peak velocity at perihelion. This meant that the compromise in the $timestep$ would be less and less and less of an issue.

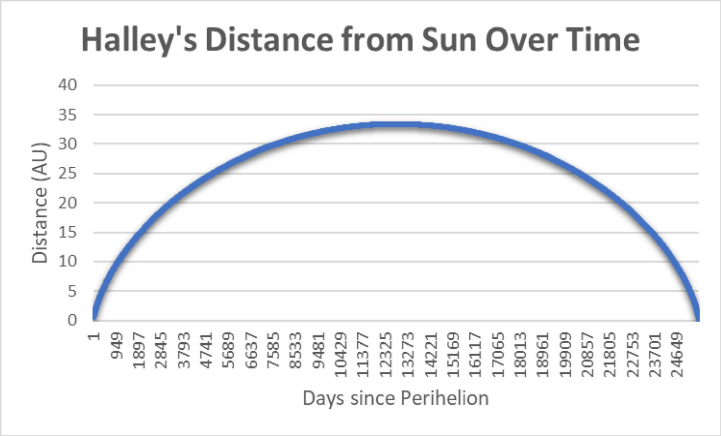
Data Generated From Simulation

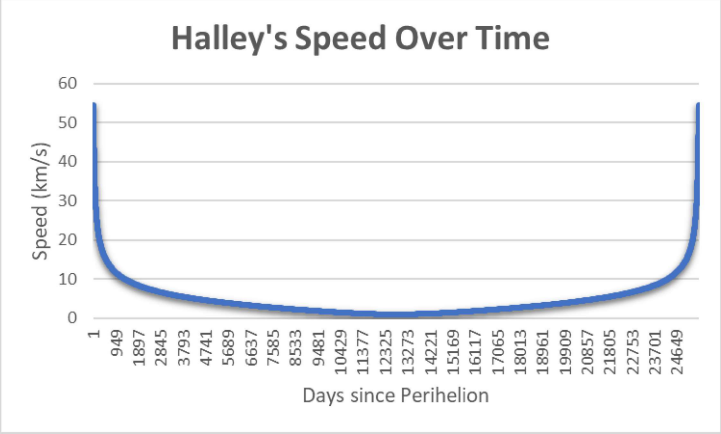
Following are the 10 of the roughly 26000 data points generated by the simulation for the entire orbit of the comet. Each data point with the interval ~of 7 years is shown for the comet [see **appendix** for more]

Time Since Perihelion	Velocity (km/s)	Halley's Distance from sun (AU)
$(1\ day)$	-54.5704	0.586
7 years $(2555\ days)$	7.076465	17.367
14 years $(5100\ days)$	4.277478	25.182
21 years $(7665\ days)$	2.662522	29.933
28 years $(10220\ days)$	1.511195	32.570
35 years $(12775\ days)$	0.940312	33.422
42 years $(15330\ days)$	1.495699	32.608
49 years $(17885\ days)$	2.640719	29.996
56 years $(20440\ days)$	4.244778	25.287
63 years $(22995\ days)$	7.005839	17.540

*Years are close approximations as numbers of days in a year is assumed to be 365, which is not always the same

Graph Representing the full dataset generated of Halley's one orbit around the sun





Halley's distance from the sun increases as it reaches the aphelion, and starts to decrease as it approaches the perihelion; the graph shows the same behaviour. Upon first look at the graph, it looked like a 2-degree polynomial would fit but it didn't fit well as the velocity is not constant, its high when it's near the perihelion but slows when it's at aphelion so as the simulation starts from the perihelion, the derivative of this curve stays relatively flat for a while towards the midpoint.

I need to point out that this is **not** the derivative of the graph above as the graph above represents the distance from the sun, not the displacement. I made the mistake of thinking that at first but then I realised it soon later.

The graph again is as expected, the acceleration due to the force of attraction when the comet is closer to the sun is higher and lower when it's at aphelion as described above by **eq4**, which means that the velocity would be higher when it's closer to the sun as demonstrated by the graph

Finding the Area

Now that I have the required data generated by the simulation; velocity and the distance from the sun over its full orbit, I can finally find the area covered by the comet in its orbit around the sun.

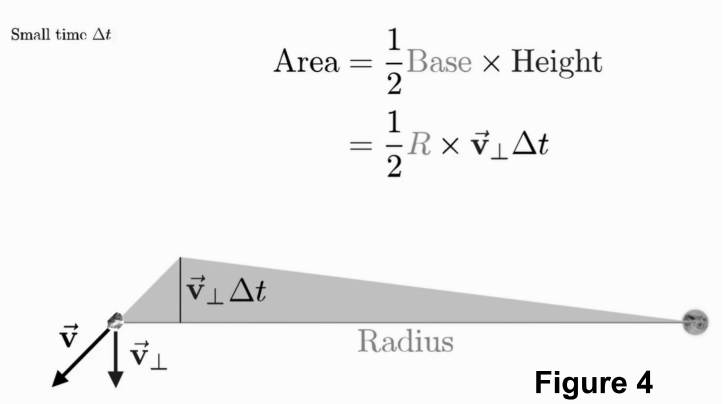


Figure 4

The area covered by the comet in a small change in time (Δt) can be approximated to a triangle as shown in **figure 4.** This will get more and more accurate as $\lim_{\Delta t \to 0}$

So what I needed was to find the area of that triangle.

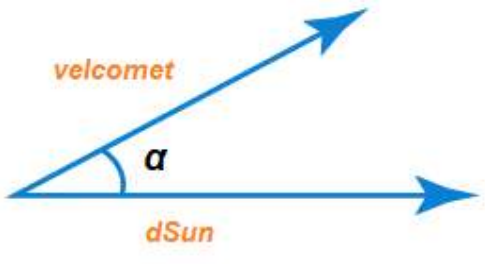
As shown in the **figure 4,** I need to find the component of the object's velocity perpendicular to the the line to the sun $(v_\perp)$**.** As I have two vectors $\vec{v}$ (velocity of the comet) and $\overrightarrow{dsun}$(vector of distance to the sun, *as mentioned previously: the sun is at the position* $(0, 0)$ *so the distance vector is the position vector of the object*), I remembered that I can use the vector dot product to find the angle (α) between two vectors**.** And as shown in **figure 4**, finding the angle will then enable me to find the velocity component perpendicular to the sun using trigonometry which will finally enable me to find the area covered in Δt

$$\vec{v} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\overrightarrow{dsun} = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$$

Using the well-known vector dot product formula:

$$\alpha = cos^{-1}\left(\frac{\vec{v} \cdot \overrightarrow{dsun}}{|\vec{v}||\overrightarrow{dsun}|}\right)$$



$$\alpha = cos^{-1}\left(\frac{x_1 \cdot x_2 + y_1 \cdot y_2}{\sqrt{x_1^{\,2} + x_2^{\,2}}\sqrt{y_1^{\,2} + y_2^{\,2}}}\right)$$

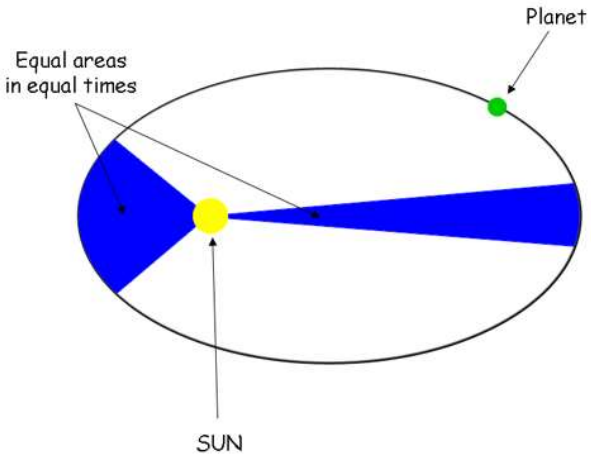
$$v_\perp = |\vec{v}| \cdot sin(\alpha) \quad \textbf{[eq6]}$$

Finally, I substituted the variables found into the following equation to find the area of a triangle in a Δt

$$Area = \lim_{\Delta t \to 0} \frac{1}{2}|\overrightarrow{dsun}| \times v_\perp \Delta t \quad \textbf{[eq7]}$$

Equal areas in equal time law

While researching about finding areas that objects cover around the sun, I came across a German astronomer Johannes Kepler who had a law to relate areas covered with time which was exactly what I was doing above, so I explored it further.



Kepler's second law mentions that a line joining a planet and the Sun sweeps out equal areas during equal intervals of time as shown in the figure to the right.

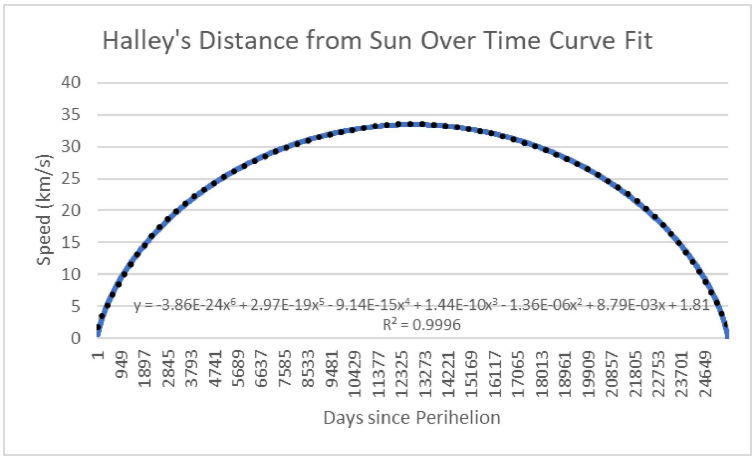
As the aim of finding out if factors like the gravitational force from other planetary bodies affect the orbit of comet Halley and to find out if comet Halley's orbit was really elliptical, I set out an experiment to verify if Kepler's second law of planetary motion really holds and see if the area covered is really the same in equal times as I was sceptical that the law would hold for smaller $timesteps$. To do that I used the formula to calculate the area that I previously derived **eq7** for each small change in time, to see if I get a constant value for the area covered for values throughout the orbit of the comet.

My conjecture is that the product of the magnitude of $dsun$ and $v_{\perp}$ in **eq7** should be a constant if the area is dependent on the Δt. I set out a plan to see if the conjecture was correct experimentally by actually calculating it for values of $dsun$ and $v_{\perp}$ that I generated from the simulation using real physics; not just a presumption that I accepted as I did for the first method of trying to model the orbit using the equation of an ellipse where it presumes that no external forces act on the comet and it follows a perfectly elliptical orbit; which needs to hold true for Kepler's second law to be true. My conjecture is that for more precise and precise calculations (smaller $timesteps$, accurate measurements like properties of the data in **table 1** with high significant figures), the law starts to fall apart.

To carry out this experiment, I needed to find a function that fits the $dSun$ as a function of time and a function that fits the $v_{\perp}$ so that I could have a function to calculate the area covered in a set time anywhere in its orbit. As I had the data from the simulation for every single day in its orbit, I tried to fit a polynomial to it by using technology as it was a huge dataset to manually try to curve it.

Halley's distance from sun curve fit

Looking visually at first, I thought finding an equation to fit this curve seemed simple as a 2-degree polynomial but it didn't work due to reasons pointed out previously. So, I loaded the distance data over time on Excel's curve fitting feature and chose the polynomial fitting tool, I kept increasing the degree of polynomial until I got a good enough fit which was a degree of 6. The following is the function with input t (days since perihelion) which outputs the distance from the sun for the comet.

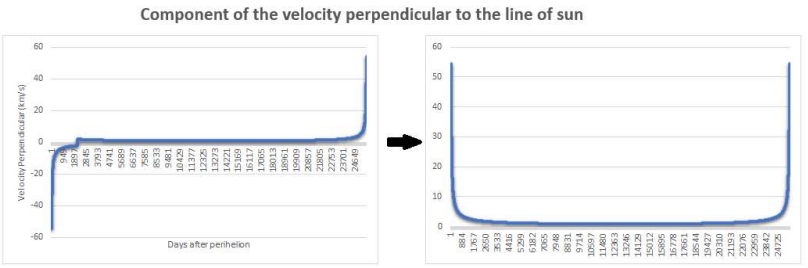


... - *Fitted curve equation superimposed*

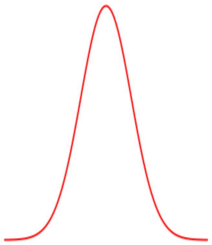
$$distFromSun(t) = -3.86 \times 10^{-24} t^6 + 2.97 \times 10^{-19} t^5 - 9.14 \times 10^{-15} t^4 + 1.44 \times 10^{-10} t^3 - 1.36 \times 10^{-6} t^2 + 8.79 \times 10^{3} x + 1.81 \ \ (units: AU) \ \ [where \ 0 < t \leq 25593]$$

Halley's velocity perpendicular to the line of sun fit

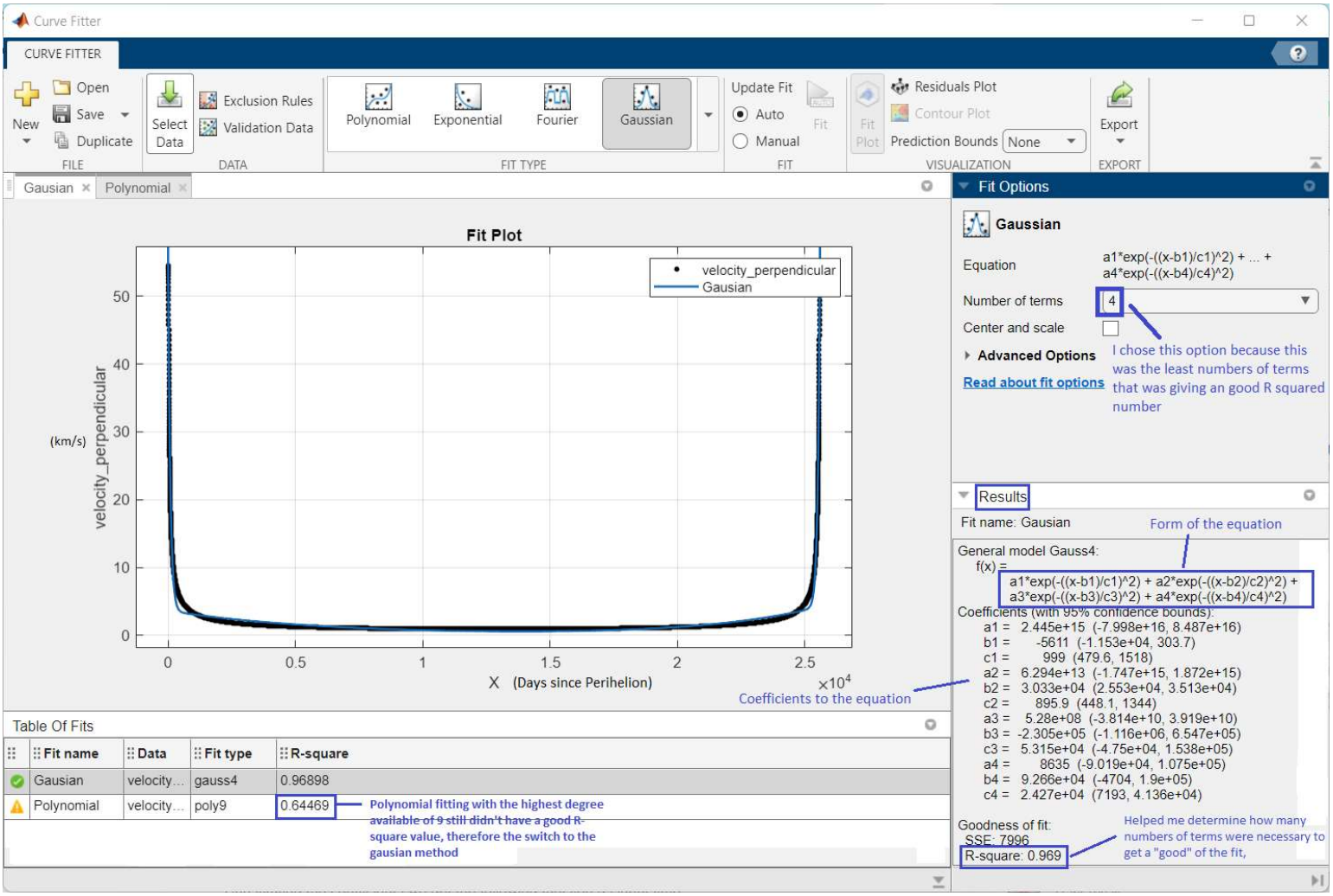
Similarly, I tried to curve fit for the component of the velocity perpendicular to the line of the sun which was found using **eq6** in the simulation required to find the area covered. To make the fit easier, I decided to use the absolute value, this is because the curve would then be smooth as shown in the figure to the left and be much easier to find a curve fit for. This won't have an effect on the area covered as we just need the magnitude of it as shown in **eq7**.



I thought that I would be able to find the fit of the function with an n degree polynomial, however, trying it in Excel didn't give a good R^2 value *(R-Squared is a statistical measure of fit that indicates how much of the data is explained by the model. So, if it's 0.50, then approximately half of the output can be modelled by the inputs, therefore we want it to be high)*. I researched some other curve fitting methods available, then, I came across the Gaussian function while I was exploring the use of Matlab's curve fitting tool.



Gaussian curves are like Normal distribution curves *(it's just that normal distribution has a standard deviation of 1 and mean of 0)*. They have a sharp rise and a sharp fall, combining multiple of these can give us the curve we are looking for which also falls and rises sharply as shown in the figure. Although there is so much more to the Gaussian curve, it is outside the scope of this exploration, so I am just going to leverage the fact that a gaussian curve has a sharp rise and a sharp fall which is the case for the velocity. Hence, I tried the curve fit using Matlab as shown below.



Using Matlab's curve fitting tool as described in the figure, Substituting the coefficients into the form of the equation, we get the following function that models the velocity component perpendicular to the sun.

*$(e\ -\ euler\ number,\ not\ the\ eccentricity)$

$t\ is\ the\ days\ since\ perihelion$

$$velocityPerpendicular(t)\ =\ 2.445\times10^{15}\cdot e^{-\left(\frac{t+5611}{999}\right)^2}\ +\ 6.294\times10^{13}\cdot e^{-\left(\frac{t-3.033\cdot10^4}{895.9}\right)^2}\ +$$

$$5.28\times10^{8}\cdot e^{-\left(\frac{t+2.305\cdot10^5}{5.315\cdot10^4}\right)^2}\ +\ 8635\cdot e^{-\left(\frac{t-9.266\cdot10^4}{2.427\cdot10^4}\right)^2}\quad (units:\ kms^{-1})\quad [where\ 0<t\le25593]$$

Now, with the help of these two curve fits, I now have a function to find the area covered in a Δt in any day in the comet's orbit around the sun. We can with the help of labels in the graph above see that the fit is not perfect, but the R-squared value of 0.969 as calculated by Matlab should be enough for my purposes as values over 0.90 are considered to be good. With this curve fits, I now have a function to find the area covered on any day of the orbit of comet Halley

let t be the days since perihelion (just as it has been throughout)

let Δt be $86400s$ $(1\ day)$ (to calculate the area covered from t to $t\ +\ 1\ day$)

$$Area(t)\ =\ \frac{1}{2}\ distFromSun(t)\ \times\ velocityPerpendicular(t)\ \times\ \Delta t$$

Firstly, after finding the function for $distFromSun(t)$ and $velocityPerpendicular(t)$, I typed the equation into Excel and passed the input t as values in column A, then I calculated the area covered each day in Halley's full orbit using Excel as shown below to see if my conjecture was valid.

SUM | =0.5*(C4*B4*SIN(D4*(PI()/180))*86400)

	A	B	C	D	E	F	G	H	I
1	Days since Perhelion	Velocity (m/s)	Halley's Distance from Sun (m)	Angle Between vectors	Area Covered in 1 Day (m²)				
2	1	-54570.42	87665597897	88.43	-2.07E+20				
3	2	-54551.37	87663450473	89.95					
4	3	-54493.07	87785773336	91.46	=0.5*(C4*B4*SIN(D4*(PI())/180))*86400)				
5	4	-54396.24	88031868893	92.97	-2.07E+20				
6	5	-54262.00	88400346637	94.48	-2.07E+20				
7	6	-54091.90	88889156904	95.97	-2.07E+20				
8	7	-53887.85	89495639358	97.44	-2.07E+20				
9	8	-53652.02	90216583646	98.89	-2.07E+20				
10	9	-53386.83	91048299177	100.32	-2.07E+20				
11	10	-53094.86	91986690804	101.72	-2.07E+20				
12	11	-52778.77	93027337174	103.10	-2.07E+20				
13	12	-52441.27	94165568815	104.44	-2.07E+20				
14	13	-52085.04	95396543368	105.75	-2.07E+20				
15	14	-51712.69	96715315937	107.03	-2.07E+20				
16	15	-51326.74	98116903025	108.27	-2.07E+20				
17	16	-50929.56	99596339086	109.48	-2.07E+20				
18	17	-50523.40	1.01149E+11	110.65	-2.07E+20				
19	18	-50110.30	1.02769E+11	111.78	-2.07E+20				
20	19	-49692.16	1.04453E+11	112.88	-2.07E+20				

seconds in a day

converting to radians as Excel requires the input to be angles in radians for the sin function

*angle between the component of the object's velocity perpendicular to the line of the sun and the position vector of the object, which is the distance oector from the sun as sun is at origin

Only 20 days of the calculations are shown in the Excel figure above due to the limit on how much I could fit in the screenshot, although I did calculate the area covered for all the days in the orbit of which the results are shown below as a graph.

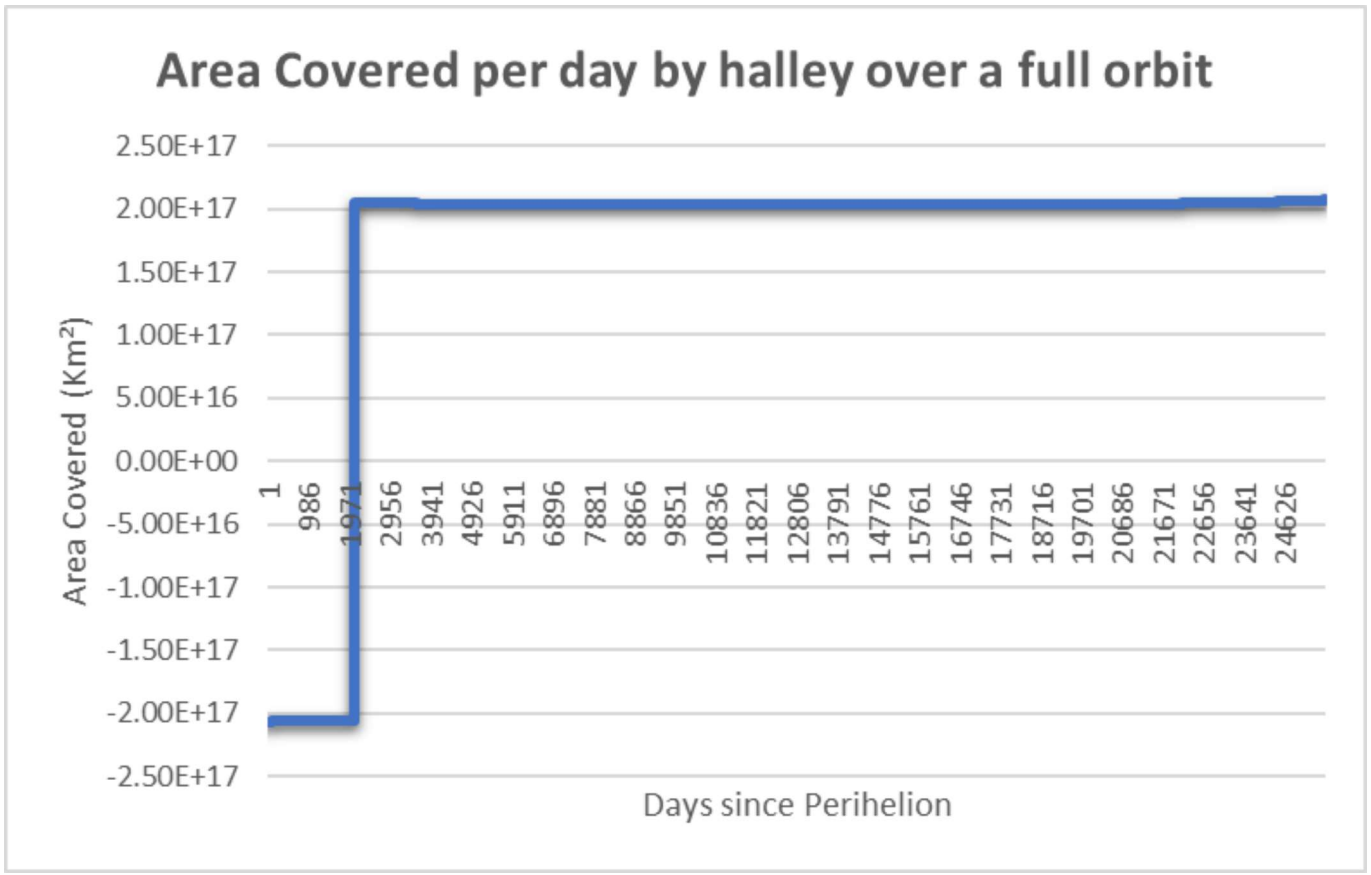
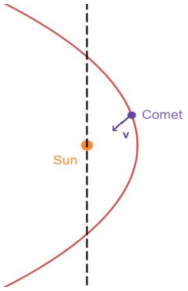


Figure 4

Explanation as to why the area is negative at some point in the orbit in the graph



To the right of the dotted line drawn vertically at the position of the sun in the diagram to the left, the area covered is negative as the resultant velocity acting upon the comet is negative as we can see it's pointed to the left. Substituting a negative value for velocity in **eq7** is what gives rise to this negative area that I was getting.

Looking at the graph above, The absolute area covered per day remains relatively almost constant throughout the orbit of the comet. This validates the equal time equal area law set out by Kepler to a certain extent. Although we can see there are some discrepancies on the graph above, the changes are really small but indeed validates my conjecture as at some points in its orbit, the planetary objects exert force on the comet, which is small compared relatively to the force from the sun but is enough to slightly change its orbit and affect the area covered. This was a great result as it confirmed what I believed to be the case at the start of the exploration that planetary objects might alter the orbit of the comet.

Now finally to calculate the **total area** covered by the comet over its full orbit around the sun using the physics modelling, I computed the following sigma notation.

(25593 is the days in Halley's full orbit)
$let\ \ \Delta t\ be\ 86400s\ (1day)$

$$Total\ Area\ Covered\ = \sum_{t=1}^{25593} \frac{1}{2} distFromSun(t) \cdot velocityPerpendicular(t) \cdot \Delta t$$

$$Total\ Area\ Covered \approx 5.22 \times 10^{24} m^2$$

Table showing how the Δt affects the area

Value for Δt	Area covered by Halley in a full orbit
$2629745s\ (1\ month)$	$\approx 4.57 \times 10^{24} m^2$
$604800s\ (1\ week)$	$\approx 5.14 \times 10^{24} m^2$
$86400s\ (1\ day)$	$\approx 5.22 \times 10^{24} m^2$
$3600s\ (1\ hour)$	$\approx 5.23 \times 10^{24} m^2$

I saw that the values were converging to around $5.23\times 10^{24} m^2$ as I decreased the Δt to lower values, this was expected as I approximated the area to a triangle, it was supposed to get better with a small change in time which it has as verified by the table above. I tried lowering the Δt even more but at that point, it was starting to generate gigabytes worth of data and take up lots of memory to the point my computer couldn't handle it. Also, as I explained earlier, there were some discrepancies on the graph in **figure 4**. I used Excel's standard deviation function to check the spread between the values for the area covered per day and I was happy to see that the standard deviation was getting smaller and smaller as I decreased the Δt as I was expecting it to.

Comparisons of results from different models

	Area covered by Halley in one orbit around the sun
Integration of equation of an ellipse	$\approx 5.68\times10^{24}\ m^2$
Physics modelling	$\approx 5.23 \times 10^{24} m^2$

I calculated percentage difference between the two values I found for the total area covered around a full orbit around the sun for comet Halley using two different methods and it was ~8%. Using two completely different approach, one where I assumed that the orbit is perfectly elliptical which only required me to know some properties of the comet like the aphelion, the eccentricity of the orbit and the semi-major axis to fit it into an equation and integrate it to modelling an entire solar system with the knowledge combined from mathematics, physics and computer science and getting the area covered by the two methods to be in the same order of magnitude and only with an percentage difference of 8%, I was really ecstatic. Seeing how multiple areas of mathematics can come together to neatly solve a problem has always been mesmerising to me and it's moments like this when my appreciation of the field grows.

I need to point out that the value of area covered derived by the equation of an ellipse is just a generalisation of an average area it would cover over an orbit as the orbits slightly change cycle over cycle as we only know that the average orbital period is of 75–76 years. This means that the $aphelion$ and the $perihelion$ would be slightly different on each orbit it makes around the sun giving rise to slightly different areas covered in each orbit.

Although, I certainly do have some systematic errors as the data provided by NASA were not to a high significant figures. Also, I assumed the mass of the comet to be constant throughout the orbit, but in each orbit around the Sun, the comet loses some mass as some of the mass the comet is made up of is ice which evaporates when it gets closer to the sun, at its perihelion. It's estimated that Halley loses an average of $5.7\ x\ 10^{11}\ kg/orbit$ which is quite a lot. This made me want to calculate how long it would be around for.

$5\ x\ 10^{14}\ kg\ (total\ mass) / 5.7\ x\ 10^{11}\ kg/orbit\ (lost) = 95\ orbits$

$95\ orbits \times (\sim 70\ years/orbit) \approx 7,000\ years$

I was grateful that it would still be around for my grandkids and generations to come and on every orbit it made through the sun, it would bring along all the historic moments associated with it.

Conclusion

Although using the formula of the ellipse was straightforward, the physics approach modelling was more accurate as it took into account more factors such as the perturbation caused to its orbit by other planetary orbits. Also, I am very proud of having explored the physics model as it has more applications. It can be generalised to fit other astronomical objects that follow any of the other orbits such as hyperbola and parabola from the conic sections which could be an extension for this exploration. Also, modelling using the equation of an ellipse only works for objects following elliptical orbit and doesn't take into account other factors that may alter the orbit. However, it can be useful in time for its simplicity as you can predict the orbit of an object really quickly if you don't require high precision, like just seeing if there would be any close encounters of two objects or finding out its whereabouts after a certain period of time.

Also, I had to start with the position of all the objects from their perihelion as there was no open-source data available telling me what velocities the objects were orbiting at a certain position along their orbits. Although the effect of it was not as apparent in this smaller time frame in my simulation, the model would start to drift away and be chaotic as the time passed as more and more interactions would occur.

Moreover as mentioned earlier, having a powerful computer would mean that I would be able to lower the Δt even more which meant I would get more precise results. Additionally, my simulation only considered interactions on a 2D plane due to computation limits, however, in the real world the orbits are tilted at a slight angle so a 3D simulation would have made it more accurate.

Finally, there was something that I hadn't considered, Einstein's theory of general relativity which was published in the 1920s and revolutionised our understanding of space-time and added to the works of Newton's law of universal gravitation published by Newton; the law on which my simulation was based. The theory of general relativity says that the observed gravitational effect between masses results from their warping of spacetime. Experiments and observations show that Einstein's description of gravitation accounts for several effects that are unexplained by Newton's law, such as minute anomalies in the orbits of Mercury and other planets. In Newton's description of gravity, the gravitational force is caused by matter. More precisely, it is caused by a specific property of material objects: their mass. In Einstein's theory and related theories of gravitation, the curvature at every point in spacetime is also caused by whatever matter is present. This could be something that I would look into as an extension to this exploration, by investigating the orbits of other objects outside of our solar system, perhaps a planet orbiting a neutron star where the relativistic effects of the curvature of the space time would be higher due the neutron star having a higher mass. But altogether, I learnt new things from the exploration and was successful in answering the questions and observations that I had at the beginning. This exploration has made me understand in great detail about the orbit of the comets and planetary objects in general and remind me how different areas of mathematics come together to solve a problem in an ever captivating way.

Bibliography

History of the comet: https://www.history.com/news/a-brief-history-of-halleys-comet-sightings
https://en.wikipedia.org/wiki/Halleys_Comet

Properties of the comet:
https://theskylive.com/halley-info
https://ssd.jpl.nasa.gov/tools/sbdb_lookup.html#/?des=1P&view=VOP

https://astronomy.swin.edu.au/cosmos/c/Conic+sections

Kepler's law: https://openstax.org/books/university-physics-volume-1/pages/13-5-keplers-laws-of-planetary-motion

Theory of general relativity: https://en.wikipedia.org/wiki/General_relativity

Medias

Conic sections: https://en.wikipedia.org/wiki/File:Conic_Sections.svg

Ellipse: https://en.wikipedia.org/wiki/Ellipse#/media/File:Ellipse-conic.svg

Construction of ellipse image: https://math.stackexchange.com/a/1097006

Equal areas in equal time: https://www.aanda.org/images/stories/PressRelease/PRaa200411/glo_kepler2.gif

Planets: https://solarsystem.nasa.gov/resources/490/our-solar-system/

Software used: Desmos, Geogebra, to draw graphs. Matlab, Excel for curve fitting and data processing, Python to create the simulation

Appendix

```
# Source code used for simulation with syntax highlighting to make it easy to read
import pygame
import math
import time

pygame.init()

WIDTH, HEIGHT =  1920, 1080 # setting the window size
WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Planet Simulation")

# defining the colors
WHITE = (255, 255, 255)
YELLOW = (255, 255, 0)
BLUE = (100, 149, 237)
RED = (188, 39, 50)
DARK_GREY = (80, 78, 81)

FONT = pygame.font.SysFont("comicsans", 16)

# loading the planet images
sun_image = pygame.image.load("media/sun.png")
mercury_image = pygame.image.load("media/mercury.png")
venus_image = pygame.image.load("media/venus.png")
earth_image = pygame.image.load("media/earth.png")
mars_image = pygame.image.load("media/mars.png")
jupiter_image = pygame.image.load("media/jupiter.png")
saturn_image = pygame.image.load("media/saturn.png")
uranus_image = pygame.image.load("media/uranus.png")
neptune_image = pygame.image.load("media/neptune.png")
pluto_image = pygame.image.load("media/pluto.png")
comet_image = pygame.image.load("media/comet.png")



class Planet:
    AU = 149.6e6 * 1000 # astronomical units in meters
    G = 6.67428e-11 # gravitational constant
    SCALE = 90 / AU  # scale; decrease the number 90 to zoom out and increase the number to
zoom in
    TIMESTEP = 3600*24 # 1 day in seconds
```

```
    image = pygame.image.load('media/sun.png') # making the default image of the object a
sun

    # initialising the planets with passed in parameters
    def __init__(self, x, y, radius, color, mass, max_lines_draw):
        self.x = x
        self.y = y
        self.radius = radius # doesn't affect the calculations, just the apparent size for
the graphics
        self.color = color
        self.mass = mass

        self.orbit = []
        self.sun = False
        self.distance_to_sun = 0
        self.distance_to_sun_x = 0
        self.distance_to_sun_y = 0

        self.x_vel = 0
        self.y_vel = 0

        self.max_lines_draw = max_lines_draw

    # draws to the screen, not a necessay part of the simulation, just for visualization
    def draw(self, win):
        x = self.x * self.SCALE + WIDTH / 2
        y = self.y * self.SCALE + HEIGHT / 2

        # comment to not draw lines, however, the program will run slower over time #
        if len(self.orbit) > 2: # only draw lines if there are more than 2 points in the
orbit
            updated_points = []
            for point in self.orbit:
                x, y = point
                x = x * self.SCALE + WIDTH / 2
                y = y * self.SCALE + HEIGHT / 2
                updated_points.append((x, y))


            # drawing the lines of the orbit path (the ellipse that we can see)
            updated_points = updated_points[-self.max_lines_draw:]
            pygame.draw.lines(win, self.color, False, updated_points, 1)

        # draw the resultant velocity
        if not self.sun:
```

```
            if self.radius == 10: # if comet halley... as it has radius 10
                # resultant  # 0.004 is the scale of the velocity to be shown in the screen
                pygame.draw.line(win, RED, (x, y), (x + self.x_vel * 0.004, y + self.y_vel *
0.004), 3)

                # only draw the x velocity
                pygame.draw.line(win, WHITE, (x, y), (x + self.x_vel * 0.004, y), 2)
                # only draw the y velocity
                pygame.draw.line(win, WHITE, (x, y), (x, y + self.y_vel * 0.004), 2)

                # draw a line to the center of the sun
                # pygame.draw.line(win, DARK_GREY, (x, y), (WIDTH / 2, HEIGHT / 2), 2)


        # scale the image with self.radius
        win.blit(pygame.transform.scale(self.image, (self.radius * 2, self.radius * 2)), (x
- self.radius, y - self.radius))

        # rendering the distance of the objects to the sun if it is not the sun
        if not self.sun:
            distance_text = FONT.render(f"{round(self.distance_to_sun/1000, 1)}km", 1,
WHITE)
            win.blit(distance_text, (x - distance_text.get_width()/2, y -
distance_text.get_height()/2))

    # calculates the force of attraction betwwen the other objects, other being the
paramater
    def attraction(self, other):
        other_x, other_y = other.x, other.y
        distance_x = other_x - self.x
        distance_y = other_y - self.y
        distance = math.sqrt(distance_x ** 2 + distance_y ** 2)

        if other.sun:
            self.distance_to_sun = distance
            self.distance_to_sun_x = distance_x
            self.distance_to_sun_y = distance_y

        # calculates the force of attraction using the formula
        force = self.G * self.mass * other.mass / distance**2
        theta = math.atan2(distance_y, distance_x) # calculating the angle
        # seperating the angle into x and y components
        force_x = math.cos(theta) * force
        force_y = math.sin(theta) * force
        return force_x, force_y
```

```
    # calculates the force of attraction between the objects and updates the position
    def update_position(self, planets):
        total_fx = total_fy = 0
        for planet in planets:
            if self == planet:
                continue

            fx, fy = self.attraction(planet)
            total_fx += fx
            total_fy += fy

        # updates the velocity of the object using f=ma
        self.x_vel += total_fx / self.mass * self.TIMESTEP
        self.y_vel += total_fy / self.mass * self.TIMESTEP

        # updates the position of the object
        self.x += self.x_vel * self.TIMESTEP
        self.y += self.y_vel * self.TIMESTEP
        self.orbit.append((self.x, self.y))

    # just an function that draws any line if wanted
    def draw_line(self, win, color, x_in, y_in):
        x = self.x * self.SCALE + WIDTH / 2
        y = self.y * self.SCALE + HEIGHT / 2
        pygame.draw.line(win, color, (x, y), (x + x_in * 0.004, y+ x_in * 0.004), 5)




def main():
    run = True
    clock = pygame.time.Clock() # pygame clock

    sun = Planet(0, 0, 20, YELLOW, 1.98892 * 10**30, 0) # inalizing the sun's class with
parameters
    sun.sun = True

    # initializing planets with parameters from the fact sheet table presented above in the
exploration
    mercury = Planet(0.38709893 * Planet.AU, 0.0, 11, BLUE, 3.3011 * 10**23, 88)
    mercury.image = mercury_image
    mercury.y_vel = -47.36 * 1000
```

```
    venus = Planet(0.72333566 * Planet.AU, 0.0, 12, RED, 4.867 * 10**24, 225)
    venus.image = venus_image
    venus.y_vel = -35.02 * 1000

    earth = Planet(1.0 * Planet.AU, 0.0, 13, BLUE, 5.972 * 10**24, 365)
    earth.image = earth_image
    earth.y_vel = -29.78 * 1000

    mars = Planet(1.52371034 * Planet.AU, 0.0, 15, RED, 6.4171 * 10**23, 687)
    mars.image = mars_image
    mars.y_vel = -24.07 * 1000

    jupiter = Planet(5.20336301 * Planet.AU, 0.0, 27, BLUE, 1.8982 * 10**27, 4333)
    jupiter.image = jupiter_image
    jupiter.y_vel = -13.06 * 1000

    saturn = Planet(9.53667594 * Planet.AU, 0.0, 24, RED, 5.6846 * 10**26, 10760)
    saturn.image = saturn_image
    saturn.y_vel = -9.68 * 1000

    uranus = Planet(19.19126393 * Planet.AU, 0.0, 18, BLUE, 8.6832 * 10**25, 30690)
    uranus.image = uranus_image
    uranus.y_vel = -6.80 * 1000

    neptune = Planet(30.06992276 * Planet.AU, 0.0, 19, RED, 1.0241 * 10**26, 60195)
    neptune.image = neptune_image
    neptune.y_vel = -5.43 * 1000

    # well added pluto as it has some imapct on the grativational force as it's twice as
massive as the comet.
    pluto = Planet(39.48211675 * Planet.AU, 0.0, 5, BLUE, 1.3022 * 10**22, 90560)
    pluto.image = pluto_image
    pluto.y_vel = -4.74 * 1000

    # comets
    comet_halley = Planet(0.586 * Planet.AU, 0, 10, WHITE, 2.2 * 10**14, 27500) # x value
perihelion
    comet_halley.image = comet_image
    comet_halley.y_vel = -54.55 * 1000

    # planets = list of planets
    planets = [sun, mercury, venus, earth, mars, jupiter, saturn, uranus, neptune, pluto,
comet_halley]
```

```
    bg_img = pygame.image.load('media/background.jpg').convert() # just to make it look nice
    bg_img = pygame.transform.scale(bg_img,(WIDTH,HEIGHT))

    counter = 0 # counter that stops the progrm after a certain amount of time dt

    distance_between = []
    area_covered = []
    speed = []
    to_save = [] # list to be filled with data to be saved


    while run: # runs until terminated, so does the precess below this line on each step

        clock.tick(1000000) # essentially the framerate but doesn't reach the limit in my
computer

        # start = time.time() # incase want to time the program
        WIN.blit(bg_img,(0,0))

        for event in pygame.event.get(): # close button
            if event.type == pygame.QUIT:
                run = False

        for planet in planets: # updates the position of each planet
            planet.update_position(planets)
            planet.draw(WIN) # draws the result in the screen

        object = comet_halley # dynamic, i can change of which i want to calculate!
        object2 = sun

        # calculates the resultant speed of the object
        speed_calc = math.sqrt(object.x_vel ** 2 + object.y_vel ** 2) # changing to km/s

        # if object.x_vel and object.y_vel are in same direction, then the comet's resultant
speed is positive
        if object.x_vel > 0 and object.y_vel > 0:
            speed_calc = abs(speed_calc)

        # if object.x_vel and object.y_vel are in opposite direction, then the comet's
resultant speed is negative
        elif object.x_vel < 0 and object.y_vel < 0:
            speed_calc = -abs(speed_calc)
```

```
        speed.append(speed_calc) # saves the speed of the comet to the list

        speed_vector = ((object.x, object.y),(object.x + object.x_vel, object.y +
object.y_vel))
        # find resultant_speed_vector
        resultant_speed_vector = speed_vector[1][0]-speed_vector[0][0],
speed_vector[1][1]-speed_vector[0][1]


        distance_to_sun = object.distance_to_sun # well the sun is the focus
        # distance to sun vector for object
        distance_to_sun_vector = ((object.x, object.y),(object.x + object.distance_to_sun_x,
object.y + object.distance_to_sun_y))
        distance_to_sun_vector = distance_to_sun_vector[1][0]-distance_to_sun_vector[0][0],
distance_to_sun_vector[1][1]-distance_to_sun_vector[0][1]

        # angle between the two vectors resultant_speed_vector and distance_to_sun_vector
        # using the dot product
        angle = math.acos((resultant_speed_vector[0]*distance_to_sun_vector[0] +
resultant_speed_vector[1]*distance_to_sun_vector[1])/((math.sqrt(resultant_speed_vector[0]**
2 + resultant_speed_vector[1]**2))*math.sqrt(distance_to_sun_vector[0]**2 +
distance_to_sun_vector[1]**2)))
        # convert to degrees
        angle = angle * (180 / math.pi)

        # calculating velocity_perpendicular_to_sun
        sin_angle = math.sin(angle * (math.pi / 180))

        # resultant vector of resultant_speed_vector
        resultant_speed_v = math.sqrt(resultant_speed_vector[0]**2 +
resultant_speed_vector[1]**2)

        velocity_perpendicular_to_sun = resultant_speed_v * sin_angle # Resultant x sin
theta
        area_swpet = 0.5*(distance_to_sun*velocity_perpendicular_to_sun*Planet.TIMESTEP)
        area_covered.append(area_swpet)

        # saves the data to a list to be saved
        to_save.append(str(speed_calc) + "," + str(distance_to_sun) + "," + str(angle) + ","
+ str(velocity_perpendicular_to_sun) + "," + str(area_swpet) + ",")

        distance_between.append(distance_to_sun)


        pygame.display.update() # updates the screen
```

```
        # end = time.time() # incase i want to time the simulation

        # following saves frames from x to y frame to a file
        # if counter > 24500 and counter < 25500:
        #   # save pygame screenshot with counter as filename
        #   pygame.image.save(WIN, "screenshots/background/comet_halley_" + str(counter) +
".png")

        if counter > 25593: # when to stop the program, this is one orbit days of the comet
halley as given my nasa.gov
            break

        counter += 1

    pygame.quit()

    # saving all the data to a single file as a csv
    f = open("to_save.csv", "w")
    for i in to_save:
        f.write(str(i)+"\n")
    f.close()

main() # starts the program
```

.

DATA GENERATED FROM SIMULATION WITH INTERVALS OF EVERY 219 DAYS FOR HALLEY (LIMITED DUE TO THE HUGE AMOUNT OF DATA PRODUCED)

DAYS SINCE PERIHELION	VELOCITY (km/s)	DISTANCE FROM SUN (METRES)	α	$V_{\perp}$
1	-54.5704	8.77E+10	88.433	-54.55
220	-21.6631	5.09E+11	154.2802	-9.40115
439	-16.3275	8.33E+11	159.4237	-5.73838
658	-13.7617	1.10E+12	161.5977	-4.34439
877	-12.1349	1.33E+12	162.8247	-3.5834
1096	-10.9657	1.54E+12	163.6054	-3.09509
1315	-10.0623	1.73E+12	164.1318	-2.7513
1534	-9.3306	1.91E+12	164.4949	-2.4943
1753	-8.71791	2.07E+12	164.7438	-2.294
1972	-8.19221	2.23E+12	164.9074	-2.13309
2191	7.73265	2.37E+12	165.0049	2.000721
2410	7.324827	2.51E+12	165.0492	1.889731
2629	6.95841	2.64E+12	165.0491	1.795212
2848	6.625772	2.76E+12	165.0106	1.71369
3067	6.321154	2.88E+12	164.938	1.642644
3286	6.040116	3.00E+12	164.8342	1.580173
3505	5.779171	3.10E+12	164.7016	1.524813
3724	5.535529	3.21E+12	164.5418	1.475416
3943	5.306932	3.30E+12	164.356	1.431061
4162	5.091523	3.40E+12	164.1452	1.391007
4381	4.88776	3.49E+12	163.9098	1.354647
4600	4.694352	3.58E+12	163.65	1.321483
4819	4.510203	3.66E+12	163.3656	1.291106
5038	4.334378	3.74E+12	163.0563	1.263175
5257	4.166074	3.82E+12	162.7212	1.237412
5476	4.004591	3.89E+12	162.3593	1.213581
5695	3.849322	3.96E+12	161.9689	1.191492
5914	3.699734	4.03E+12	161.5482	1.170988
6133	3.555363	4.10E+12	161.095	1.151938
6352	3.415807	4.16E+12	160.6064	1.134236
6571	3.280715	4.22E+12	160.0796	1.117785
6790	3.149777	4.27E+12	159.5112	1.102497
7009	3.022719	4.33E+12	158.8975	1.088291
7228	2.899295	4.38E+12	158.2344	1.075089
7447	2.779291	4.43E+12	157.5171	1.062822
7666	2.662522	4.48E+12	156.7404	1.051423
7885	2.548829	4.52E+12	155.8984	1.040831
8104	2.438084	4.57E+12	154.9842	1.030989
8323	2.330185	4.61E+12	153.9903	1.021841
8542	2.225054	4.64E+12	152.9079	1.013338
8761	2.122643	4.68E+12	151.7272	1.005434
8980	2.02293	4.72E+12	150.4365	0.998088
9199	1.925921	4.75E+12	149.0228	0.991266
9418	1.831657	4.78E+12	147.4708	0.984936
9637	1.740212	4.81E+12	145.7629	0.979076
9856	1.651704	4.83E+12	143.8791	0.973665
10075	1.5663	4.86E+12	141.7964	0.968691
10294	1.484221	4.88E+12	139.4889	0.964143
10513	1.405758	4.90E+12	136.9281	0.960015
10732	1.331279	4.92E+12	134.083	0.956301
10951	1.261235	4.94E+12	130.9216	0.952998
11170	1.196169	4.95E+12	127.4124	0.950097
11389	1.13672	4.96E+12	123.5278	0.94759
11608	1.083614	4.97E+12	119.2481	0.945467
11827	1.03765	4.98E+12	114.567	0.943718
12046	0.999667	4.99E+12	109.499	0.942333
12265	0.970483	4.99E+12	104.085	0.941306
12484	0.950821	5.00E+12	98.39602	0.94063

DAYS SINCE PERIHELION	VELOCITY (km/s)	DISTANCE FROM SUN (METRES)	α	$V_{\perp}$
12922	0.94196	5.00E+12	86.61318	0.940315
13141	0.953021	5.00E+12	80.76667	0.940673
13360	0.974081	4.99E+12	75.11069	0.941375
13579	1.004571	4.99E+12	69.74154	0.942428
13798	1.043747	4.98E+12	64.72608	0.943836
14017	1.090783	4.97E+12	60.10132	0.945608
14236	1.14484	4.96E+12	55.87834	0.947754
14455	1.205126	4.95E+12	52.04874	0.950283
14674	1.270929	4.93E+12	48.59116	0.953208
14893	1.341628	4.92E+12	45.47718	0.956542
15112	1.416696	4.90E+12	42.67521	0.960296
15331	1.495699	4.88E+12	40.15339	0.964481
15550	1.578279	4.86E+12	37.88127	0.969106
15769	1.664154	4.83E+12	35.83056	0.97418
15988	1.753107	4.80E+12	33.97573	0.979709
16207	1.844976	4.78E+12	32.29403	0.985705
16426	1.93965	4.74E+12	30.76554	0.992182
16645	2.037059	4.71E+12	29.37284	0.999159
16864	2.137168	4.68E+12	28.10085	1.006659
17083	2.239974	4.64E+12	26.93653	1.014716
17302	2.345504	4.60E+12	25.86854	1.023362
17521	2.453811	4.56E+12	24.88713	1.032642
17740	2.564974	4.52E+12	23.98378	1.042605
17959	2.679097	4.47E+12	23.15106	1.053305
18178	2.796311	4.42E+12	22.38254	1.064804
18397	2.916775	4.37E+12	21.67245	1.077165
18616	3.040676	4.32E+12	21.01575	1.090461
18835	3.16823	4.27E+12	20.40791	1.104766
19054	3.299687	4.21E+12	19.84493	1.120163
19273	3.435335	4.15E+12	19.32323	1.136742
19492	3.575492	4.09E+12	18.83962	1.154599
19711	3.720519	4.02E+12	18.3913	1.173842
19930	3.870821	3.95E+12	17.97569	1.194587
20149	4.026862	3.88E+12	17.59047	1.216963
20368	4.189175	3.81E+12	17.23364	1.241122
20587	4.358381	3.73E+12	16.90344	1.267241
20806	4.535194	3.65E+12	16.5986	1.295546
21025	4.720443	3.57E+12	16.31815	1.326306
21244	4.915091	3.48E+12	16.06151	1.359854
21463	5.120263	3.39E+12	15.82854	1.3966
21682	5.33729	3.29E+12	15.61938	1.437042
21901	5.567752	3.19E+12	15.43465	1.481797
22120	5.813555	3.09E+12	15.27533	1.531625
22339	6.077017	2.98E+12	15.14283	1.587475
22558	6.360999	2.87E+12	15.03914	1.650544
22777	6.669084	2.75E+12	14.96679	1.722352
22996	7.005839	2.62E+12	14.92927	1.804889
23215	7.377194	2.49E+12	14.9311	1.900788
23434	7.791039	2.35E+12	14.97854	2.01365
23653	8.258156	2.21E+12	15.08048	2.148571
23872	8.793763	2.05E+12	15.25	2.313034
24091	9.420144	1.89E+12	15.50769	2.518642
24310	10.17145	1.71E+12	15.88649	2.784255
24529	11.10377	1.52E+12	16.44063	3.142607
24748	12.31903	1.30E+12	17.26879	3.656962
24967	14.02985	1.07E+12	18.58273	4.470943
25186	16.79054	7.96E+11	20.9577	6.005617
25405	22.90168	4.62E+11	26.92896	10.37184