BPhO Computational Challenge: Solar System Orbits

Constantin & Vladimir Filip

August 14, 2023

Abstract

In this report, we aim to describe the mathematical basis and technical implementation of our orbital models, including the formulae used (and their derivations where appropriate), the techniques used to calculate orbital values and the development of our PyQt6 desktop application.

Introduction

We used Python with Matplotlib to develop the models and plot the graphs in each of the seven tasks. We then integrated task 7 with both the 2D and 3D animations from tasks 3 and 4, in addition to animating the spirographs, to produce a full PyQt6 desktop application. We then extended our models to other single-star planetary systems, as well as including the real-time calculation of parameters related to the motion of the planets within their orbit.

We have aimed to make the desktop application as customisable as possible, allowing the user to select any combination of planets from the available systems to plot with a centre planetary object of their choice.

1 Why we chose Python

Due to the mathematically intensive nature of the orbital calculations, we chose to use Python because the wide array of scientific and mathematical libraries that come with Python. For instance, we used NumPy for carrying out vectorised operations on entire arrays of data. In addition, the Matplotlib framework is highly customisable and has excellent support for 3D visualisations, as well as being easy to integrate into the powerful PyQt framework.

2 Implementation of the orbital models

2.1 Plotting the elliptical orbits

The polar form of the equation for an ellipse, with its origin at one of the two foci, is the following:

$$r(\theta) = \frac{b}{1 - \varepsilon \cos\theta}$$

where θ is the true anomaly (the angle between a point on the ellipse and the origin the focus) in *radians*, b is the semi-minor axis and ε is the eccentricity of the orbit.

To plot these orbits in two dimensions, we used the NumPy module to generate 1000 linearly spaced values of θ between 0 and 2π. We then substituted into the formulae above using the value of the semi-major axis and eccentricity of each of the planets, where $b = a(1 - \varepsilon^2)$ and converted the polar coordinates to their Cartesian form using the conversions $x = r\cos\theta$ and $y = r\sin\theta$.

To plot the orbits in three dimensions, we calculated the same values of r for each of the orbits since the planets orbit in the same 2D plane. Using the inclination angle of each of the planets' orbits, we converted the polar coordinates to three dimensional Cartesian coordinates using the conversions $x = r\cos\theta\cos\beta$, $y = r\sin\theta$ and $z = rcos\theta\sin\beta$, where β is the inclination angle.

2.2 Animating the orbits

For animating the planets, we could not use linearly spaced angles since this would plot the positions of the planets at the same points in their orbits rather than their position at a particular time. Instead, we modified the algorithms to use linearly spaced values of time t, converting each value of t to the corresponding value of θ *for each orbit* using the formula $\theta = \frac{2\pi t}{P}$, where P is the orbital period. This angle is known as the mean anomaly, which describes the angle that the planet would have moved through in a circular orbit with the same period (see Modelling Assumptions). The values of t were calculated based on the orbit of the outermost planet, generating 500 linearly spaced values of t for each period of the planet with the largest orbit.

HowthiswasimplementedinMatplotlib

2.3 Changing the centre of the orbits

The equations above describing the orbital path only apply for elliptical orbits, so when setting another planetary object as the frame of reference, we needed to consider its motion relative to the Sun rather than attempt to define the orbits of the other planets in the system relative to that planet. In general, we applied the following transformation:

$$x' = x - r_c \cos\theta_c, \quad y' = y - r_c \sin\theta_c$$

where x' and y' are the transformed x and y coordinates, r_c is the distance of the centre planet to the Sun at a time t and θ_c is the orbital angle of the centre angle at t. This transformation can also be applied to when the planets orbit around the Sun by setting r_c to 0.

For the 3D orbits, we applied the identical transformation to the z coordinates of the planets.

3 Orbit angle vs. time

Kepler's Second Law states that a planet sweeps out equal areas in equal intervals of time:

$$r^2 \frac{d\theta}{dt} = \frac{1}{2}\sqrt{G(m+M)(1-\varepsilon^2)a}$$

which is constant, where r is the distance to the Sun, A is the area and θ and is the true anomaly. This differential equation can be solved to give time as a function of the true anomaly:

$$t = P(1-\varepsilon^2)^{\frac{3}{2}} \frac{1}{2\pi} \int_{\theta_0}^{\theta} \frac{d\theta}{(1-\varepsilon\cos\theta)^2}$$

We then used Simpson's rule to evaluate this integral. However, since Simpson's rule uses an even number of strips, we used Simpson's 3/8 rule to account for when the strip width produces an odd number of strips:

$$\int_a^b f(x)dx \approx \frac{3}{8}h\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right]$$

We used Simpson's 3/8 rule to evaluate three of the total number of strips, which left an even number of strips that could be evaluated fully using Simpson's rule.

Once we calculated the true anomaly for the entire duration of the planet's orbit over linearly spaced values of θ, we used the built-in cubic spline function of the scipy.interpolate module to estimate the true anomaly at a given time.

4 Implementation of the spirographs

4.1 Plotting the spirographs

For the spirographs, we adapted the previous models to be able to modify the step for the time at which a line is plotted between a pair of planets. We found that 70 divisions per orbit of the outermost planet gave the most aesthetic spirograph result.

4.2 Animating the spirographs

When animating the spirographs, a new line was plotted at each step in time; we also animated the planets and their orbit to show the position of the two planets whenever a new line segment was plotted.

5 Developing the PyQt6 application

The app consists of a tab layout with 3 tabs. Each tab consists of a single containerised widget (inheriting from the QWidget class) that contains all the child widgets and bindings to carry out a specific function. The code for all three of these widgets can be found in the `pages.py` file of the code repository.

5.1 The 'Orbits' tab

This is where the animated orbit simulation is displayed, along with real-time statistics about a chosen planet in the animation. It works by extracting the orbit parameters (including the planets to be rendered, the centre of orbit and orbit speed) from the `OrbitSimSettings` class (more detail given later on) and initialising the back-end animation classes (`Animation2D` if a 2D orbit simulation is requested or `Animation3D` if 3D), passing on the settings as arguments. The animation classes then modify the canvas present in the Orbits page at fixed time intervals. The full source code for the contents of the 'Orbits' tab is contained in the `OrbitsPage` class in `pages.py`

5.2 The 'Orbit Settings' tab

This tab is hidden as its sole purpose is to affect the behaviour of the 'Orbits' tab. It is therefore accessible through a settings button in the 'Orbits' tab. This page is where the parameters of the orbit simulation can be altered or reset to default through individual child widgets that directly alter a single parameter. These parameters are the following:

- Star system, one out of the following: the Solar System, Tau-Ceti, HD-219134 and Proxima Centauri

- Centre of orbit, one out of all the celestial objects in the chosen star system
- Objects to show, any of the celestial objects in the chosen star system that have not been marked as the centre of orbit
- View type, 2D or 3D
- Orbit time, the time taken for the outermost planet in the simulation to complete one full orbit in seconds
- Number of orbits, the number of full orbits the outermost planet makes over the full simulation. This was implemented for the full extent of orbits to be adequately displayed when a planet has been chosen as the centre of orbit.

These parameters are stored in a static field `SETTINGS` of the `OrbitSimSettings` class, which is edited in-place at runtime whenever the user chooses any different parameters. Both the 'Orbits' page and the 'Orbit Settings' page contain an instance of `OrbitSimSettings`, and therefore any changes carried out in 'Orbit Settings' are immediately transferred to 'Orbits'.

The full source for the contents of the 'Orbit Settings' tab is contained in the `OrbitsPageSettings` class in `pages.py`

5.3 The 'Spirograph' tab

Much like the 'Orbits' tab, this tab also contains a canvas on which an animation is drawn and initialises the correct animation class, passing on the simulation parameters as arguments. However, the parameters of this animation are edited through child widgets present on the same page, in order to prevent the user from having to switch between pages only to customise a few parameters. In this case, the animation is real-time spirograph generation. The editable spirograph generation parameters consist of the following:

- Star system, to determine the choices for the two planets used in the spirograph
- Planet 1 and Planet 2, the two planets used to create to spirograph
- Speed, a choice between 'slow', 'medium' and 'fast'. Each of these choices correspond to pre-set Δt values that are proven to consistently yield coherent spirographs. The spirograph animation class that handles all calculations can work with any numerical value for Δt, but we opted for pre-set values simply to improve the generator's consistency.
- N, the number of orbits of the outermost planet

Once these parameters are inputted, and the 'Evaluate' button is clicked, the spirograph will be drawn line-by-line, with the number of completed orbits and the number of lines drawn being displayed in real-time.

5.4 Additional features

- Real-time calculation of linear velocity, angular velocity, distance to centre and current (x, y) coordinates.
- Dynamic rendering of the orbit simulation; in 3D orbits the resulting graph can be zoomed, panned and rotated dynamically so that the user can view the orbits from any angle they wish.

6 Implementing additional planetary systems

We extended our models to include the Tau Ceti, HD 219314 and Proxima Centauri solar systems through the use of enums in Python to allow for the customisation of planetary parameters. In particular, we chose HD 219314 as the two innermost planets in this solar system (HD 219134 b and HD 219134 c) have very large inclination angles (85.05^o and 87.28^o respectively), which we thought would produce especially interesting 3D simulations.

7 Technical considerations

7.1 Floating point errors

Due to the large number of arithmetical operations involved in the calculation and animation of these orbits, there is a significant impact from floating point errors due to the inaccurate representation of floating point numbers in Python. Therefore, we used the Decimal module for the exact representation of all constants and results of intermediary calculations, converting values to floats only when passed to Matplotlib functions.

7.2 Optimisation

Before animating the orbits, we initialised the line objects in Matplotlib with empty x and y values, and calculated all of the orbital values before the creation of the animation itself. As a result, the line objects only needed to be populated with pre-calculated data at a specific frame number, which Matplotlib then individually plotted as they were updated rather than clearing and replotting the entire graph, helping to significantly improve performance.

However, we found that as the number of lines plotted by the spirograph increased, the animation decreased in performance, as the library itself is limited in the speed at which it can determine which lines need to be plotted. A future refinement would be to incorporate a simple physics engine which can render the planets and the line segments more efficiently.

8 Modelling assumptions

Our models assume the following:

- Zero force of gravity between planets in the same planetary system.
- The star remains stationary at the left focus of the planets' orbits.
- No relativistic effects
- All planets start at the same point in their orbits (the aphelion).

In addition, the angle used in the calculation of the position of a planet as a function of time is known as the *mean anomaly*, which describes the angular distance travelled by a planet as though it moved in a circular orbit of the same orbital period. However, the polar equation used to describe the planets' orbits take the origin as the left focus rather than the centre of the elliptical orbit, so the mean anomaly is not the appropriate input for the polar equation. A more accurate value would be given by the *true anomaly*, which is the angle between the line joining the planet to the focus and the semi-major axis (which corresponds to the angle θ used in the orbital equations above). The true anomaly is calculated using the mean anomaly and another angle known as the *eccentric anomaly* using a numerical method.

Appendix

A Derivation of formula for linear velocity of a planet in an elliptical orbit in terms of distance to star

The total energy of a planet is given by:

$$E_{tot} = -\frac{GMm}{2a}$$

where G is the gravitational constant, M is the mass of the star, m is the mass of the planet and a is the length of the semi-major axis. This value is always constant as planets orbit in a closed system, so conservation of energy applies. E_{tot} is also equal to the sum of K and U, where K is the kinetic energy of the planet and U is the potential energy of the planet. K is given by

$$K = \frac{1}{2}mv^2$$

and U is given by

$$U = -\frac{GMm}{r}$$

where r is the distance to the star. Equating these expressions for E_{tot}:

$$-\frac{GMm}{2a} = \frac{1}{2}mv^2 - \frac{GMm}{r}$$

$$\Rightarrow \frac{2GM}{r} - \frac{GM}{a} = v^2$$

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

B Derivation of formula for linear velocity of planet relative to another planet

Since the formula in Appendix A only gives the magnitude of the linear velocity for elliptical orbits, the velocity of a planet relative to another planet cannot be worked out using this formula. In this approach, the velocity vectors of the centre planet and the orbiting planet relative to the star are calculated; the velocity vector of the orbiting planet relative to the centre planet is the difference of these two velocity vectors.

The parametric equation of an ellipse is

$$r(t) = (a\cos t, b\sin t)$$

We can differentiate this to calculate the gradient at a particular value of t:

$$r'(t) = (a\sin t, -b\cos t)$$

Since $x = r\cos\theta$ and $y = r\sin\theta$,

$$r\cos\theta = a\cos t \Rightarrow t = \arccos\left(\frac{r\cos\theta}{a}\right)$$

Similarly,

$$t = \arcsin\left(\frac{r\sin\theta}{b}\right)$$

Substituting into the derivative of $r(t)$, which gives the velocity vector:

$$\hat{v} = \frac{-ar}{b}(\sin\theta)\mathbf{i} + \frac{br}{a}(\cos\theta)\mathbf{j}$$

Since $\overrightarrow{v} = v\hat{v}$:

$$\overline{v} = \frac{-avr}{b}(\sin\theta)\mathbf{i} + \frac{bvr}{a}(\cos\theta)\mathbf{j}$$

Therefore, the linear speed v relative to a planet O is $|\overline{v} - \overline{v_O}|$.

9 References

1. Libretexts (2022) \2.2: The Ellipse," Physics LibreTexts [Preprint]. Available at: https://phys.libretexts.org/Bookshelves/Astronomy__Cosmology/Celestial_Mechanics_(Tatum)/02%3A_Conic_Sections/2.02%3A_The_Ellipse.

2. Wikipedia contributors (2023) \Ellipse," Wikipedia [Preprint]. Available at: https://en.wikipedia.org/wiki/Ellipse.

3. Wikipedia contributors (2023b) \Simpson's rule," Wikipedia [Preprint]. Available at: https://en.wikipedia.org/wiki/Simpson%27s_rule.

4. French, A. (2023) Solar System Orbits. Available at: https://www.bpho.org.uk/bpho/computational-challenge/BPhO_CompPhys_Challenge_2023_briefing.pdf (Accessed: August 14, 2023).