Janel Williams
05/20/2019
Final Project

Modeling the orbits of exoplanetary systems in Visual Python

Introduction

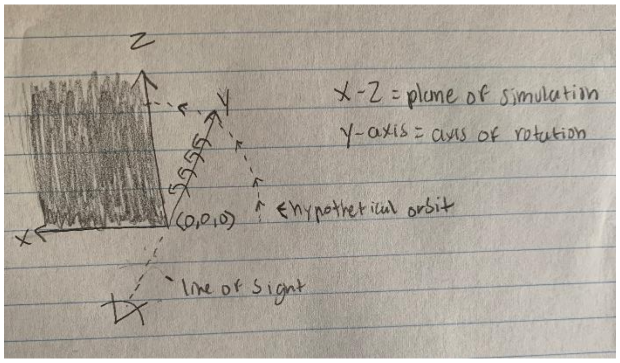
There have been many successful attempts in modeling the orbits of planets within our solar system. With the discovery of more exoplanets, these efforts have shifted to but as more planetary systems are being discovered, these efforts are shifting to modeling the orbits of Earth-like exoplanets. Python[1]and VPython[2] are two of many programming languages that are used to do such modeling. In my project, I aim to use the updated versions of the documentation for Python and VPython to simulate the orbits of a select planetary system within the habitable zone[3]. Overall my project consisted of three primary objectives:

1. Calculate the motion of exoplanets revolving around a host star using Python
2. Calculate the reflex motion[4] of the selected host star using Python
3. Track the movement of the exoplanets and host star and create visual displays using VPython

Methods

In order to simulate orbits, I had to choose a planetary system. In order to choose a planetary system, I had to consider some constraints that came with using a programming language like VPython. Because I opted to use the database NASA Exoplanet Archive[5] to retrieve data for my simulation, I was also operating under the constraints of the site. Even though the archive stores data for thousands of systems, they don't always share the same types of data. For instance, some planets will have their masses and velocities reported while others don't, leaving blank space on search results. Therefore, the first few steps in making a simulation required that I conceptualize about the placement of the system itself within a window generated by VPython, and possible values I'd need to accurately portray the orbit of the exoplanets about the star.

Conceptualization of VPython



In VPython, three dimensional objects can have a defined position and velocity. The three dimensionality of the objects implies that their respective positions and velocities have components in the x, y, and z direction. The way the viewer of the simulation perceives the motion of the object within this space 3D space is dependent on the trajectory of object with respect to the origin.

In other words, the best trajectory for viewing the simulation is based on the line of sight of the viewer. Looking at a window generated by VPython the x-axis is perpendicular to the

[1] http://spiff.rit.edu/classes/phys440/lectures/ellipse/ellipse.html
[2] https://www.glowscript.org/docs/VPythonDocs/index.html
[3] https://www.nasa.gov/ames/kepler/habitable-zones-of-different-stars
[4] Haswell, C. A. (2010). Our Solar System from afar. In *Transiting exoplanets* (pp. 31-36). Cambridge Univ. Press.
[5] https://exoplanetarchive.ipac.caltech.edu/

viewer's line of sight, the y-axis is parallel to the line of sight, and the z-axis sits at a 90° angle with respect to the x-y plane The ideal way of viewing the orbits would be to view them "face on". So, the simulated orbits must rotate about the y-axis on the x-z plane (see figure 1).

Once the preferred plane was identified, I began searching for the best way to characterize the changes in the position and velocity vectors to begin modeling orbits. The simplest way to do so was to analyze each planet independently within the astrocentric frame when the star is at rest.

Within this frame, the orbit of the planet has at least three constants: the semi-major axis a, the orbital period P, and the eccentricity e. These were the first three values that I recorded as being essential for my displays.

In addition, because my displays involve simulating the way moving bodies interact, Newton's Law of Gravitation[6] will apply. According to Newton's Law of Gravitation, the force of gravity acts between any two objects separated by a defined distance r. The force is proportional to the mass and inversely proportional to the square of r:

$$F_G = \frac{G(M_* + M_p)}{r^2}$$

Where G is the gravitational constant, M_* is the mass of the host star, and M_p is the mass of the planet. The gravitational force exerted on a planet by a star should be equal to the centripetal force. The centripetal force is given by F_c:

$$F_c = mr\omega^2,$$

where ω is the angular velocity of the planet.[7] Exoplanets and their host stars are separated by a semi – major axis, so I assumed that $a = r$ in the Law of Gravitation and the centripetal force equation. Therefore, the resulting expression after the substitution is:

$$F_G = F_c,$$

$$\frac{G(M_* + M_p)}{a^2} = M_p a\omega^2.$$

When solving for ω the resulting equation is:

$$\omega = \sqrt{\frac{G(M_* + M_p)}{M_p a^3}}$$

The angular velocity is the same for every point on the object when the object is rotating about an axis. The velocity, v for the object is proportional to the distance from the axis of rotation. Thus, the angular velocity is also given by this equation:

$$\omega = \frac{v}{r}.$$

[6] https://opentextbc.ca/physicstestbook2/chapter/newtons-universal-law-of-gravitation/

[7] http://physicsnet.co.uk/a-level-physics-as-a2/further-mechanics/circular-motion/

The assumption $a = r$ still applies here, so solving for v actually gives:

$$v = \omega a.$$

When the appropriate substitutions are made, the velocity of an exoplanet orbiting a star is theoretically:

$$v = \sqrt{\frac{G(M_* + M_p)}{M_p a^3}} \times a.$$

Using the above equation allowed me to define the initial velocity for each planet. Since the planets would be moving about the y-axis I would need to define the initial velocity for the y component of each planet's velocity vector.

Before making further calculations, I've conceptualized enough to officially select my exoplanet system. Using the aforementioned database, I was able to search for a host star that had planets that had a semi major axis, orbital period, eccentricity, planet mass and inclination that had values greater than >0. I also chose to search for systems that had more than 3 planets detected. The exoplanetary system that fit most of my parameters was K2 -266. The exact parameters for the system and other information are listed in the table 1.

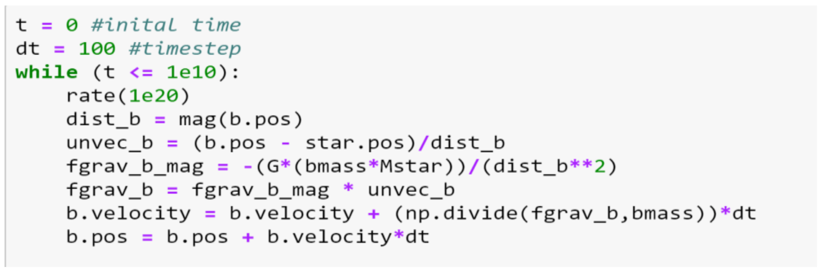
Parameters used to model the orbits of Planetary System K2-266

	Orbital Period (days)	Semi Major Axis (km)	Inclination (radians)	Eccentricity	Gravitational Force[1]	Initial Velocity (km/s)	Mass (kg)
K2-266 (host)							1.3720 e^30
Planet b	.658524	1953776.0	1.314581	.044	1.6187 e^24	216.4925	6.7481 e^25
Planet c	7.814	10157840.0	1.540776	.042	1.5329 e^21	94.9466	1.7274 e^24
Planet d	14.697	15483600.0	1.561371	.047	2.1446 e ^22	76.9031	5.6148 e^25
Planet e	19.482	15483600.0	1.561970	.043	1.2999 e^22	70.0056	4.9562 e^25

Table 1: Above are basic parameters for beginning to model orbits. The units for the semi major axis, inclination, and Mass were converted from astronomic units (au), degrees, and Jupiter masses (Mjup) to kilometers, radians, and kilograms respectively. This one done so that all of my units would be as direct and consistent as possible.

The semi major axis was used to define the initial position of the planets with respect to the origin, where the star rests. The velocity vectors of each planet were updated in my code by using the gravitational force, and the positions of the planets were updated using those new velocity values. A new set of coordinates for the position of the planet was found every type Python read through a loop (see figure 2 for a sample code).

Updating the Velocity and Positional Vector of Planet b



```
t = 0 #inital time
dt = 100 #timestep
while (t <= 1e10):
    rate(1e20)
    dist_b = mag(b.pos)
    unvec_b = (b.pos - star.pos)/dist_b
    fgrav_b_mag = -(G*(bmass*Mstar))/(dist_b**2)
    fgrav_b = fgrav_b_mag * unvec_b
    b.velocity = b.velocity + (np.divide(fgrav_b,bmass))*dt
    b.pos = b.pos + b.velocity*dt
```

Figure 2: An initial time t and a time step dt are defined. The loop will execute when t is less than or equal to 1e10 seconds. The defined rate tells VPython what the maximum amount of loops it can execute per second. Under these constraints, I define the direction (unit vector aka unvec_b) of the orbit, which is given by the product of the magnitude of planet b's position and gravitational force. That value is used to update b.velocity, which is the velocity vector for planet b. That's then used as a function to update the position of planet b, which is denoted by b.pos. Each planet had defined functions under this loop, so are all processed simultaneously.

Once I was able to create a code for the planetary orbits, I was ready to start on my second goal: to model the reflex motion of the star. The reflex motion of the star is dependent on the motion of planets. The relationship between the two is given by the barycentric velocity of the star:

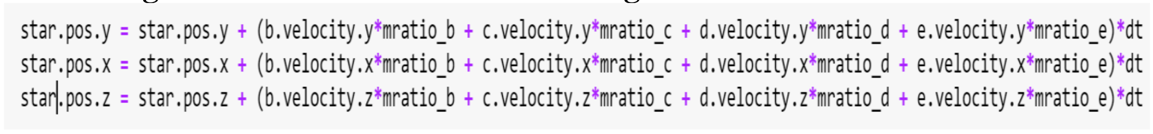
$$v_* = \frac{Mp}{Mp + M_*} v.$$

The velocity of the star, v_* can be found by calculating the product of the astrocentric velocity and the mass ratio. Since $M_p \ll M_*$, the mass ratio can be reduced and the barycentric velocity becomes:

$$v_* = \frac{M_p}{M_*} v.$$

Similar to the motion of the planets, the star has defined components in the x, y, and z direction. Since the movement of the star is dependent on the motion of the planets, the total velocity for the star in each direction will be the sum of the orbits of each planet. The processes is summed up in figure 3 below. Specific parameters for the star are organized in table 2.

Calculating the Reflex Motion of the Star using vectors



```
star.pos.y = star.pos.y + (b.velocity.y*mratio_b + c.velocity.y*mratio_c + d.velocity.y*mratio_d + e.velocity.y*mratio_e)*dt
star.pos.x = star.pos.x + (b.velocity.x*mratio_b + c.velocity.x*mratio_c + d.velocity.x*mratio_d + e.velocity.x*mratio_e)*dt
star.pos.z = star.pos.z + (b.velocity.z*mratio_b + c.velocity.z*mratio_c + d.velocity.z*mratio_d + e.velocity.z*mratio_e)*dt
```

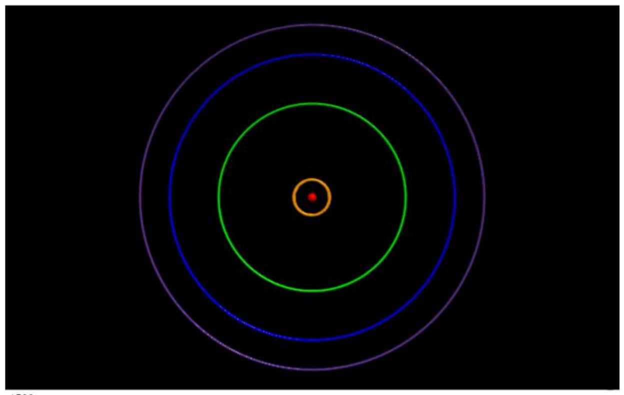
Figure 3: The position of the star in the y, x, and z direction (denoted by star.pos.y) is updated based on the sum of the product of the velocities and their mass ratios. As seen, only like components are being added together.

The last and primary goal was to create visual simulations. After defining the motion of the planets and the motion of the star within a loop, VPython was able to display them into a window. A VPython function called "gcurve" was used to track the motion of each object in the system with respect to the y – axis.

Results and Discussion

Figures 4.1 and 4.2, are screen shots of the generated depictions of the orbits (4.1) and the radial velocity curve in the y – direction. Each planet is color coded so that they are easily distinguishable upon a glance. As expected, the planet with the smallest orbital period and semi - major axis has a much higher radial velocity amplitude due to the tremendous gravitational. My intention was to model the elliptical orbits of the planet, due to the methods I used to define a rate of change in the position and velocity vectors I had to assume a circular orbit. That being said, even if I was able to create an ellipse orbit, the planets have such a low eccentricities that they may have appeared circular to the viewer even if I managed to incorporate the shape in my equations.

Duo Simulation of K2 - 266



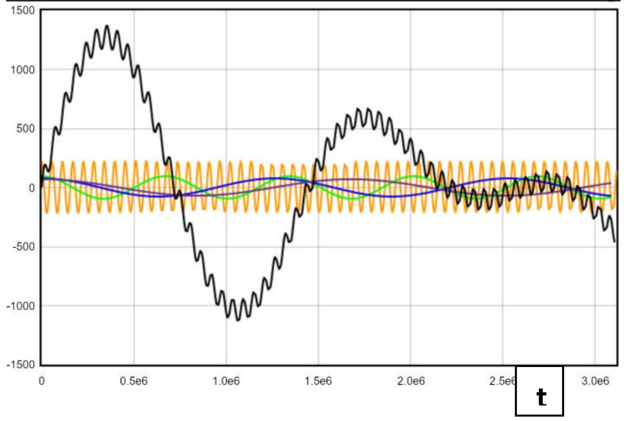


Figure 4.1 (top) and 4.2 (bottom): Figure 4.1 depicts the visual display of the orbits of the planets. Each planet, and their orbital trails are color coded. Planet b is orange, planet c is green, planet d is blue and planet e is purple. The same color scheme is used to differentiate between the planets on figure 4.2. This portion of the image represents the radial velocity of the planets as well as the change in the position star (shown by the black curve) as they rotate around the y – axis with respect to time, t. In other words, the position of the star is plotted on top of the velocity curves of each of the planets.

Figure 4.2 shows the change in the position of the planets and star in the y direction as a function of time, t. The value of t, again, is defined by the loop parameters shown in figure 2. The unusual shape of the star (the black curve), is due to the fact that the star's position is oscillating with each orbital period of the planets. The exact motion can by seen by zooming into the star (figure 5).

Reflex Motion of host star K2 - 266

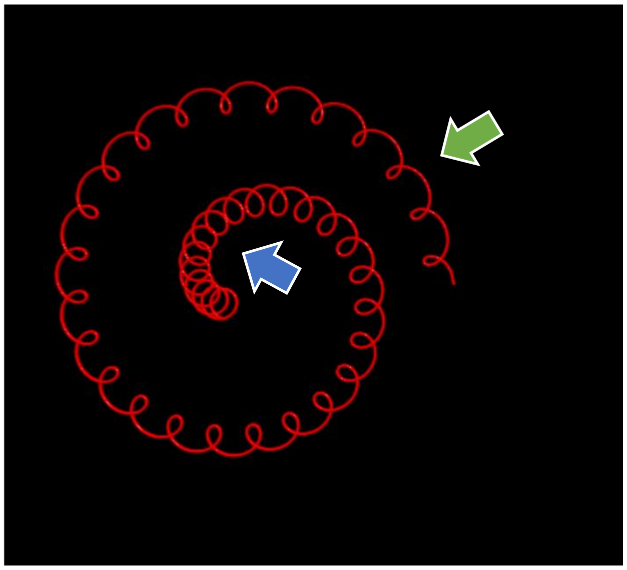


Figure 5:
To the left is an image of the trail made by the star as it orbits around a barycenter. As the planets go through more and more periods, the reflex motion of the star can be seen widening as the simulation goes on. I didn't measure the diameter of the loops or length of the red loops, but given the way stars oscillate as they move with their planets I could infer the star's velocity was decreasing as the planets finished their orbits. It would take more time to go around the larger loops (by the blue arrow) that it ended on than the smaller ones that were created when the simulation just started (by the green arrow).

Conclusion and Reflection

Overall, I was able to successfully model the orbits of the exoplanets revolving around star K2 - 266. In retrospect, it would have been interesting to define all the velocities, positions, and other related functions using the radial velocity components in the astrocentric and inertial frame.

Under an acrocentric frame, the motion of a single planet, or its velocity *v,* has components in the x, y, and z direction :

$$v_x = -\frac{2\pi a}{P\sqrt{1-e^2}}(\sin(\theta + \omega_{OP})) + esin(\omega_{OP})),$$
$$v_y = -\frac{2\pi\ a\cos i}{P\sqrt{1-e^2}}(\cos(\theta + \omega_{OP})) + ecos(\omega_{OP})),$$
$$v_z = -\frac{2\pi asini}{P\sqrt{1-e^2}}(\cos(\theta + \omega_{OP})) + ecos(\omega_{OP})).$$

Which use the semi – major axis *(a)*, the orbital period *(P)*, the eccentricity *(e)*, the inclination *(I)*, the true anomaly (θ), and the pericentre (ω_{OP}) (Haswell, 2010). In order to use this equation, I would have to define the true anomaly as a function of time. This would also allow me to model the star's radial velocity more efficiently. Since the motion of the system would be directed away from the line their lie of site, the radial velocity of the star as a result of a single planet acting on it would be defined as such:

$$\mathrm{V(t)} = V_{0,y} = -\frac{2\pi\ \mathrm{a}M_p\cos i}{(M_p + M_*)P\sqrt{1-e^2}}(\cos(\theta(t) + \omega_{OP})) + ecos(\omega_{OP})).$$

The trick, however, would be to define $\theta(t)$ in a way that python can read.[8] Upon further research it seems that this would be given by something called the eccentric anomaly (E), which changes with respect to time and utilizes the periods of the planets. Using these components would help me create orbits with more accurate shapes.

With my current code, it may be possible to use the timing of the loop to emulate elliptical orbits. The unit vectors of the gravitational force are being updated using the position of the star. If the star's position is being changed based on the velocities of the planets, then their motion are linked.

Unfortunately, I didn't have much time to delve into this idea as it would require me to re-do my entire code. If I'm in need of a more accurate simulation of planetary orbits in the future, I'll likely revisit the method.

[8] http://farside.ph.utexas.edu/teaching/celestial/Celestialhtml/node33.html

Code:
Code Documentation