Georgia Institute of Technology

School of Physics

Final Report
PHYS 3266

Simulating Orbital Perturbations and Inferring their Sources

Written By:
-Joshua Brandt-
-Paul Vollrath-

Date: April 24, 2022

Abstract

[Abstract]

1 Introduction

1.0.1 History

In 1609, Johannes Kepler formulated his three laws of orbital motion after observing the trajectory of Mars in the night-sky. His laws determine the orbits of two interacting celestial objects, allowing one to predict the size, shape, and orientation of the orbits. It wasn't until 1687 when Isaac Newton created his Universal Law of Gravity that it was understood the exact mechanism by which orbits are formed. Newton's Law,

$$\vec{F}_g = \frac{Gm_1m_2}{r^2}\hat{r}$$

now forms the basis for the analysis and prediction of orbits. With the advent of this law came the realization, which Newton made himself, that the orbits of more than two bodies is a deterministic, yet chaotic, phenomenon: there is no equivalent for Kepler's Laws for three or more bodies. This "Three-body Problem" ultimatley became the foundation for Chaos Theory.

There is a bright-side however. Since gravity interacts between every pair of objects, the motion of a particular object carries with it information about the distance and mass of other objects. In this way, if you know the propeties of each body graviationally interacting, you can solve Newton's Laws for each, and numerically determine its orbit.

This indeed became an active area of astronomical research in the mid-1800s. Ever since Uranus was observationally discovered in 1781 by William Herschel, its position in the night-sky was tracked. Astronomers John Couch Adams and Urbain Le Verrier noticed slight deviations in the orbit of Uranus: perturbations not predicted by interacting with any known planets. Using these perturbations, both astronomers calculated where a new planet would have to be to cause the observed orbit. Within the first night of looking, equipped with Le Verrier's prediction, German astronmer Johann Galle saw Neptune through a telescope for the first time.

Ever since that first purposeful discovery of a planet, astronomers have been using this method to detect planets, in our own Solar System and in others. In fact, many believe that there is a yet undiscovered "Planet X" outside of the Kuiper Belt beyond apparent sight. They cite unaccounted perturbations in the known planets.

1.0.2 Goal

The goal of our project is to computationally implement this process. Taking in a perturbed orbit, can we give an accurate suggestion of the properties of the planet doing the perturbing? The idea is to simulate a set of known planets and stars, each time making a guess as to where and how large a new planet would need to be. Evolving the system over many years, we compute properties (See 1.0.4) of the orbit, and compare them to the observed (perturbed) properties. In a sort of Bayesian analysis, if our guessed planet gives an orbit that closley matches the perturbed orbit, we can say that the guessed planet has a high likliehood of having the properties of a real planet doing the perturbing.

1.0.3 Reference Frames

Planetary systems in general tend to form within a plane, which in our System is called the ecliptic. More specifically, the ecliptic is the plane of the Earth-Sun orbit. As a beggining, we are deciding to begin the project analyzing the Solar System, and all subsequent discussion will be based upon that fact.

The ecliptic, as we will treat it, is a stationary frame: the frame in which the Earth and Sun lie in at initialization. During our simulation, we track all positions with respect to the position at which the Sun was initialized. But that is not the frame where orbital analysis typically takes place. The barycentric frame is a frame whose origin follows the center of mass of the Solar System. In order to analyze our orbits, we store the positions in this frame.

1.0.4 Orbital Elements

2 Methods

2.1 Data Input

2.1.1 JSON Files

The data-entry to our code is done through Javascript Object Notation (JSON) files. There are two types of JSON files that we used: a configuration file, and a planet file. An example configuration file is shown below

```
{
  "Planets": ["Earth", "Sun", "Mercury", "Venus", "Uranus", "Jupiter",
  "Saturn", "Neptune", "Mars"],
  "dt": 3600,
  "Runtime": 60,
  "Ephemeris Date": "2001-01-10"
}
```

This contains all of the information needed to run a simulation. For each planet, the code will look for the corresponding planet file. *dt* represents the length of the time-step to use, the Runtime is how many real seconds to fun for, and the Ephermeris Date is which date you want to initialize the Solar System at (See 2.1.2)).

The planet files contain information on the stars/planets you want to use during the simulation. Below is Earth's

```
{
  "name": "earth",
  "mass": "<Find>",
  "iposition": "<Find>",
  "ivelocity": "<Find>",
  "Horizon ID": 399
}
```

The "Find" keyword is used when you're using a real planet which carries a unique "Horizon ID". Alternativley, you can manually enter information to make custom planets and stars.

The configuration file tells the code which planets to read in, and for each one it converts the information in the JSON file into an instance of our "Body" class. These bodies are manipulated by the Simulator to carry out the simulation.

2.1.2 Horizons API

[Talk about API]

2.1.3 The Body Class

Once all of the information about an object is known from either JSON data entry or the Horizons API, the code creates an instance of the Body class for each object participating in the simulation. Bodies have attributes including: mass, position, velocity, net force, kinetic energy which are used during the simulation; position and velocity histories which are lists storing all of the past positions of that body in the barycentric frame for later use; and semimajor axis, semiminor axis, inclination, and eccentricity which are set after the orbit is analyzed. Their methods include: adding force, removing force, and setting position & velocity. Body objects follow through each step of the code: created at the beggining, manipulated during the simulation, and analyzed after; the code takes in the raw material and produces the product in the form of Body classes.

2.2 Simulating Orbits

[Talk about code]

2.2.1 Verlet Method

[Talk about verlet method]

2.3 Orbit Analyzer

The job of the orbit analyzer is to take in the position history of each planet and compute the orbital elements. We chose to focus on the inclination, the semi-major axis, and eccentricity (implictly requring the semi-minor axis). This is because these elements focus on the fundamental physical properties of the orbit, while others are more dependent on arbitrary reference directions (inclination depends on the "arbitrary" choice of the ecliptic plane, yet that choice is less arbitrary than the vernal equinox). We believe that with these few elements, we could capture significant and comparable information about the orbits. Of course, there were downsides to this choice. All planets lie roughly in the ecliptic plane, each with an inclination less than 10 degrees, and all orbits are mostly circular (save Mercury with $e = 0.2$). This means that in two of the orbital elements, we expect to see barley any deviation, yet the semi-major axis obviously varies widley. It would then perhaps make sense to weight the semi-major axis more highly as a comparative measure: that is the dominant discriminating factor.

2.3.1 Inclination

The inclination ι is the tilt of an orbit with respect to the ecliptic plane; by definition, Earth has $\iota = 0$. The way we computed the inclination first involved fitting a plane to

the orbit in question. The goal is to find the coefficients of the form

$$Ax + By + Cz = D$$

We can do some simplification given that the Sun is necessarily in the plane of each orbit (Kepler's First Law) and the Sun is (very nearly) at $(0, 0, 0)$. So we know $D = 0$. Letting $C = 1$ without loss of generality simplifies the problem to just finding A and B such that

$$Ax + By = -z$$

Each orbit is stored in the form of n coordinates, so constructing

$$A = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \quad \vec{x} = \begin{bmatrix} A \\ B \end{bmatrix} \quad \vec{b} = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$$

We can solve for $\vec{x}$ using least squares

$$A^T A\vec{x} = A^T\vec{b}$$

utilizing numpy's method to obtain the plane.

We know the normal vector to the ecliptic is $n =< 0, 0, 1 >$ and the normal vector of the orbit's plane is $< A, B, 1 >$. The inclination is simply the angle between these two vectors

$$\cos(\iota) = \frac{< 0, 0, 1 > \cdot < A, B, 1 >}{\sqrt{(A^2 + B^2 + 1)}} = \frac{1}{\sqrt{(A^2 + B^2 + 1)}}$$

2.3.2 Semi-major Axis

The semi-major axis for an inclined orbit is difficult to calculate since an ellipse must be fitted to data outside of the x-y plane. To make the problem simpler, we first apply a transformation to the data to put it all in the x-y plane. To do this, we re-write each coordinate in terms of a basis where the plane of the orbit is the new x-y plane, whose normal vector becomes the new z direction. To accomplish this, we first need the normal vector to the orbital plane (which we have from the previous section), and two linearly independent vectors in the plane of the orbit. Since we have an analytic expression for the plane, this is simple to do analytically as well. All we need to do is to rewrite the plane in vector-parametric form, as the two-dimensional span of the vectors in question, always orthogonal to the normal (which we already know).

$$x = -\frac{B}{A}y - \frac{1}{A}z$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \cdot \vec{n} = 0 \implies \begin{bmatrix} -\frac{B}{A}y - \frac{1}{A}z \\ y \\ z \end{bmatrix} \cdot \vec{n} = 0 \implies \left(\begin{bmatrix} -\frac{B}{A} \\ 1 \\ 0 \end{bmatrix} y + \begin{bmatrix} -\frac{1}{A} \\ 0 \\ 1 \end{bmatrix} z \right) \cdot \vec{n} = 0$$

Which gives our new basis as the column space of the transition matrix

$$\begin{bmatrix} -\frac{B}{A} & -\frac{1}{A} & A \\ 1 & 0 & B \\ 0 & 1 & 1 \end{bmatrix}$$

Since it is best practice to use an orthonormal basis, we did this by applying the QR decomposition to the transition matrix, where the column space of Q is an orthonormal basis of the column space of the transition matrix.

$$\begin{bmatrix} -\frac{B}{A} & -\frac{1}{A} & A \\ 1 & 0 & B \\ 0 & 1 & 1 \end{bmatrix} = QR$$

With Q being our orthonormal transition matrix.

The coordinates in the new basis, $\vec{x}_{\text{new}}$ is the solution of

$$Q\vec{x}_{\text{new}} = \vec{x}_{\text{old}}$$

since we're expecting to transition potentially millions of points, we need a fast way of solving this system repeatedly for each new $\vec{x}_{\text{new}}$ The best way to do this is to LU decompose Q, fulfilling the consistent row operations to create two triangular matricies which can be easily solved using Scipy's LU-Solver. So we solved

$$LU\vec{x}_{\text{new}} = \vec{x}_{\text{old}}$$

for each orbit point.

With that done, we can know analyze the orbit within its own plane. The general equation of an ellipse

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

can be simplified without loss of generality to

$$Cy^2 + Bxy + Dx + Ey + G = -x^2$$

We solve for these coefficients the same way as before, using least squares. With these in hand, we can solve for the semi-major axis a and semi-minor axis b in terms of these coefficients according to

$$a, b = \frac{-\sqrt{2(AE^2 + CD^2 - BDE + (B^2 - 4AC)F((A + C) \pm \sqrt{(A - C)^2 + B^2}))}}{B^2 - 4AC}$$

2.3.3 Eccentricity

The eccentricity is easily calculated with the standard formula

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

2.4 Data Output

[Output files]

3 Results

[Results]

3.1 Errors

[Talk about eccentricity errors]

4 Conclusion

[Conclusion]