Kepler's Sieve:
Learning Asteroid Orbits from Telescopic Observations

A dissertation presented

by

Michael S. Emanuel

to

Institute of Applied Computational Science

in partial fulfillment of the requirements

for the degree of

Master of Science

in the subject of

Data Science

Harvard University

Cambridge, Massachusetts

May 2020

Dissertation Advisors:
Professor Pavlos Protopapas
Professor Christopher H. Rycroft

Author:
Michael S. Emanuel

Kepler's Sieve:
Learning Asteroid Orbits from Telescopic Observations

Abstract

A novel method is presented to learn the orbits of asteroids from a large data set of telescopic observations. The problem is formulated as a search over the six dimensional space of Keplerian orbital elements. Candidate orbital elements are initialized by randomly choosing the six elements independently at random, either by sampling from a known asteroid or from uniformly distributed angles. A statistical model describes the distribution of angular distances between the directions of observed detections to the predicted direction from the observatory site to a body with these orbital elements. This model yields a log likelihood function, which attains large positive values when the candidate orbital elements are near to the elements of a real asteroid viewed many times in the data. The candidate elements and the parameters describing the mixture distribution are jointly optimized using gradient descent. Computations are performed quickly and efficiently on GPUs using the TensorFlow library.

The methodology of predicting the directions of telescopic detections is validated by demonstrating that out of approximately 5.69 million observations from the ZTF dataset, 3.75 million (65.71%) fall within 2.0 arc seconds of the predicted directions of known asteroids. The search process is validated in multiple stages. First, I demonstrate that given an initial guess equal to a perturbation applied to the elements of asteroids present in the data, the search process is able to successfully recover the true orbital elements to a high degree of precision. Next, I demonstrate that a random initialization is able to converge to elements matching some known asteroids to high precision. Finally, I use the search on observations that do not match any known asteroids. I present orbital elements for [5] new, previously unknown asteroids.

Exact number of new asteroids presented

All code for this project is publicly available on GitHub at github.com/memanuel/kepler-sieve.

Contents

Acknowledgments

I would like to thank my advisor, Pavlos Protopapas, for suggesting this topic and for his consistent support, advice and encouragement.

I would like to thank Chris Rycroft, my secondary advisor, for guiding me through a paper in Applied Math 225 in which I explored numerical integrators for solving solar system orbits.

I would like to thank Matt Holman and Matt Payne from the Center for Astrophics (CFA) for their advice on state of the art solar system integrators.

Most importantly, I would like to thank my wife Christie and my children Victor and Renée for their love and support. The Covid-19 crisis struck just as my work on this thesis kicked into high gear. It would not have been possible to complete it without extraordinary understanding and support from them.

Introduction

Determining the orbits of asteroids is one of the oldest problems in astronomy. Classical methods are based on taking multiple observations of the same body through a telescope. For an object that is large and bright enough, the human eye can ascertain the continuity of the motion, i.e. that the data are multiple sightings of the same object. Once enough sightings have been obtained, orbital elements can be solved using traditional numerical methods, such as a least squares fitting procedure that seeks elements to minimize the sum of squares error to all of the observations.

State of the art techniques for solving this problem are remarkably similar in spirit to the classical method. Indeed, the first interstellar object, 'Oumuamua, was discovered when astronomer Robert Weryk saw it in images captured by the Pan-STARRS1 telescope on Maui. [1] [2] More automated methods also exist. Still, these methods are based on a search in the space of the observable data attributes: the time of observation (MJD), the right ascension (RA) and declination (DEC). The apparent magnitude or brightness (MAG) is the third important observed quantity available for telescopic detections. Two observations made close together in time at two points in the sky very near to each other have a relatively high probability of belonging to the same object. Such a pair of observations is called a "tracklet." Today's most automated approaches to identifying new asteroids from telescopic data are based on performing a greedy search of the observed data to extend tracklets. Once a tracklet is identified, the algorithm attempts to extrapolate the path where future detections of this object might be. After enough detections are strung together, a fitting procedure is tried to determine the orbital elements.

This is a solid technique and I do not mean to cast aspersion on it. In this paper, however, I

[1] Wikipedia - Oumuamua

[2] NY Times - Astronomers Race to Study a Mystery Object from Outside the Solar System

propose a new method which I believe has some significant advantages. Rather than searching in the space of the data, i.e. (MJD, RA, REC, MAG), I propose to instead search over the six dimensional space of Keplerian orbital elements $(a, e, i, \Omega, \omega, f)$. Why should we complicate things by searching implicitly, as it were, on the space of possible orbits, rather than the simpler and more direct method currently used? The main reason is to avoid a combinatorial explosion.

If you limit your search to candidate tracklets where you detect the same object multiple times in a short span of time, you are going to miss out any object that you detect only once or twice on a given night of observations. But if this same object were seen on multiple nights, possibly separated over multiple days or longer, it becomes very costly to propose enough candidate tracklets to pick them up. Indeed you will soon face a combinatorial explosion in the number of possible tracklets. A simplified model of the number of tracklets might be that we have a data set containing observations with a uniform density ρ per day per degree squared of sky, and we set a threshold τ in time and d in angular distance for how close a second observation must be to mark it as a candidate tracklet.

Here is a simple model showing the quadratic cost of enumerating candidate tracklets.
If you extend this further to tracks with 3 observations, the scaling gets even worse (cubic).
Let ρ be the average density of detections per day per degree of sky.
Let T be the number of days of observations in our data set.
Let τ be the threshold in days for 2 observations to be considered close enough in time to form a candidate tracklet.
Let Δ be the threshold angular distance in degrees for 2 observations to be considered close in the sky to form a candidate tracklet.
Let $A = 41,253$ be the number of square degrees in the sky. [3]
Let $N = T \cdot A \cdot \rho$ be the total number of detections in the data set.
Let $m = \tau \cdot \pi\Delta^2 \cdot \rho$ be the average number of observations that will be close enough to each candidate starting point of a tracklet.
Let $NT_2 = \dfrac{N \cdot m}{2!} = \dfrac{\tau}{T} \cdot \dfrac{\pi d^2}{A} \cdot \dfrac{\rho^2}{2!}$ be the total number of candidate trackets of size 2.
Let $NT_k = \dfrac{N \cdot m^{k-1}}{k!} = \left(\dfrac{\tau}{T} \cdot \dfrac{\pi d^2}{A}\right)^{k-1} \cdot \dfrac{\rho^k}{k!}$ be the total number of candidate trackets of size k.

[3] Wikipedia - Square Degrees in the Sky

We can see the bad news right away. The number of tracklets of size k, NT_k, scales as ρ^k. And the factors in the denominator don't bail us out. The number of possible ways m to extend a tracklet is going to be a large number well in excess of 1.

This is the principal motivation for searching in the space of orbital elements. While it's a large 6 dimensional space, its size is fixed. The cost of the search algorithm presented below scales linearly in the observation density ρ for each candidate element analyzed. The cost of the entire algorithm is therefore on the order of N^2, with no explosion as you consider tracklets larger than 2. The second major reason for searching in the space of orbital elements is that it permits the search algorithm to string together observations made far apart in time. This is a capability that eludes searches based on tracklets.

I summarize now the key steps in the search algorithm. The first step is to generate a set of candidate orbital elements. This is done with a very simple approach, one which can almost certainly be imporved on later: random initialization. For four of the orbital elements, a, e, i, and Ω, one of the $733,489$ catalogued asteroids is selected at random. Its orbital elements are used to populate these four. It is worth emphasizing that each element is initialized with a *different* random asteroid; the four elements in this part will almost never match one of the known asteroids across all four elements. The remaining two orbital elements, M (mean anomaly) and ω (argument of periapsis), are modeled to be distributed uniformly at random on the circle $[0, 2\pi)$. These are then converted to the representation using $(a, e, i, \Omega, \omega, f)$ using the rebound numerical integrator.

Once the candidate elements have been initialized, they are integrated numerically using the `rebound` library. This is considered to be the gold standard of their true orbits. This initial integration is then used to filter the data set of ZTF observations to a subset that are relevant for searching for orbits. A routine computes the direction $\mathbf{u}_{\text{pred}}$ in the barycentric mean ecliptic (BME) reference frame that an observer at a given observatory site on earth would have seen light leaving an object with the candidate elements at a given observation time (MJD). This quantity is computed at each unique observation time in the ZTF data set. A separate computation is performed once on all of the ZTF observations converting the observed triplets (MJD, RA, DEC) into vectors $\mathbf{u}_{\text{obs}}$, the direction of the observation in the BME frame. The angular distance between the predicted and observed direction is computed. A threshold (2.0 degrees) is applied, and all ZTF observations falling within this threshold are cached in memory of the search class.

During the main body of the search process, the elements will be adjusted by a small amount in each training round. These perturbed elements will have their orbits evaluated using the Kepler two body model. An implementation is performed on the GPU using TensorFlow that is fast and differentiable. The ground truth orbit is used to provide an adjustment term so that the predicted orbits will match the true orbits exactly when the perturbation is zero. The predicted orbit can therefore be considered to be a linearization of the true orbits based on the Kepler model.

The objective of the optimization function is based on the log likelihood of a statistical model for the distribution of distances between predicted and observed directions. A lemma will demonstrate that for directions uniformly distributed on the sphere, the squared distance over the threshold distance would be uniformly distributed on the interval $[0, 1]$. A mixture model is formulated, where the distance between every predicted and observed direction is modeled as a mixture of hits and misses. The misses are distributed uniformly on $[0, 1]$. The hits are distributed as a truncated exponential distribution. The decay parameter λ of this exponential process is associated with a resolution paramater R. This model is equivalent to assuming that some fraction h (for hits) of the detections are due to a real body with the candidate elements, and that the results of the detection will be normally distributed with a precision parameter equal to the resolution. During the search process, the threshold parameter is also updated. This dynamic threshold should not be confused with the original threshold of 2.0 degrees used to build the filtered training data.

The optimization process jointly optimizes the candidate orbital elements and three parameters in the mixture model: the assumed number of hits, the resolution R, and the threshold. Intuitively, we want the model to gradually tighten its focus, and adjust the orbital elements so they hit as many observations as closely as possible. But we *don't* want the model to get "faked out" by trying to get the elements closer to observations that belong to *other* asteroids. The model needs some way to update probabilities that each observation is a hit or a miss, which it does using the mixture model. Early on, the optimization will tries to get close to the central tendency of the data set. If the initialization was good, it will gradually tighten in the resolution and threshold parameters. The gradients will encourage the model to adjust the candidate orbital elements so that some of the observations, the ones it sees as highly probable hits, will be very close what is predicted by the candidate elements. The observations modeled as highly probable misses will

hardly contribute to the gradients of the candidate elements.

In practice, the optimization is carried out in alternating stages. In odd numbered stages, only the resolution parameters are tuned at a higher learning rate; in even numbered stages, both the resolution and orbital elements are adjusted together at a slower learning rate. There are some additional subtleties where the actual optimization function during the training of the mixture parameters has a term to encourage the model to shrink the resolution and threshold parameters. These will be discussed at greater length below.

As much as possible, I have sought to validate individual components of these calculations in isolation. My numerical integration of the planets is validated against results from NASA JPL (Jet Propulsion Library) using the superb Horizons system [4] I separately validated the numerical integration of the first 20 asteroids against positions and velocities obtained from Horizons.

GET EXACT NUMBER

The notion of a direction in space from an observer on earth is typically reported in telescopic data using a right ascension and declination. While these are convenient and standard for reporting observed data, they are not well suited to the approach taken here. All directions are represented internally in this project as a unit vector $\mathbf{u} = (u_x, u_y, u_z)$ in the Barycentric Eliptic Plane. These calculations were validated in isolation by querying the Horizons system for both the positions of and directions to known asteroids. It is vital that this calculation takes into account the finite speed of light. Treating light travel as instantaneous leads to errors that are catastrophically large in this context, on scales in the arc minutes rather than arc seconds.

The end to end calculation of a direction from orbital elements was verified indirectly as follows. I integrated the trajectories of all the known asteroids using a collection of orbital elements downloaded from JPL. I then computed the nearest asteroid number to each ZTF asteroid, and the distance between the predicted direction and observed direction. I reviewed the statistical distribution of these distances. I observed that out of approximately 5.69 million observations from the ZTF dataset, 3.75 million (65.71%) fall within 2.0 arc seconds of the predicted directions of known asteroids. I took this as overwhelming evidence that these calculations were

[4] NASA Horizons
I cannot say enough good things about Horizons. If you want an external "gold standard" of where an object in the solar system was or will be and a friendly user interface, Horizons is an excelent resource.

accurate.

To put this degree of precision in context, 1.0 arc second is a back of the envelope estimate of the precision with which a modern telescope can determine direection of an observation under ideal observational conditions. [5] If you were to use an approximation that observations were made at Earth's geocenter (i.e. you did not account for location of the observatorory on Earth's surface) you would already be making errors on the order of 3 arc seconds. If you were to perform your calculations using the sun's location as your coordinate origin rather than the solar system barycenter, you would make errors larger than 1.0 arc second. I know because I made both of these errors in earlier iterations before squeezing them out!

I tested the capabilities of the search process with an increasingly demanding set of search tasks. The first three search tasks involved recovering the elements of known asteroids. I took a batch of 64 asteroids that appeared most frequently in the ZTF data set. These asteroids were represented between 160 and 200 times in the data, where hits here are counted at a threshold of 2.0 arc seconds as before. Here is a summary of the tests I ran:

- Initialize search with correct orbital elements, but resolution $R = 0.5°$ and threshold $\tau = 2.0°$. All 64 elements were recovered to **???**
- Initialize search with small perturbation applied to orbital elements; a by 1.0%, e by 0.25%, i by 0.05°, remaining angles f, Ω and ω by 0.25°. 37 of 64 elements were recovered to **???**.
- Initialize search with large perturbation applied to orbital elements; a by 5.0%, e by 1.0%, i by 0.25°, remaining angles f, Ω and ω by 1.0°. 11 of 64 elements were recoverd to **???**. In some cases, a different (but correct) set of orbital elements was obtained; the perturbation was so large the search found a different asteroid.
- Initialize a search with **randomly initialized** orbital elements. Search against the subset of ZTF observations within 2.0 arc seconds of a known asteroid. This search converged on one set of orbital elements matching a real asteroid.

The last last test was significantly more demanding in that it did not rely on known orbital elements.

[5] Discussion with Pavlos Protopapas

The work encombpassed in the first three tests above can be seen as a way to indpendently validate a subset of the known asteroid catalogue. It can efficiently associate a large number of telescope observations with known asteroids, which could in turn be used to further investigate those asteroids. Analysis might include refining their estimated orbital elements, fitting the $H - G$ model of brightness (magnitude), or identifying some of them for further investigation if they meet criteria of interest, e.g. orbits that will approach near to Earth in the future.

The main thrust of this work, however, is not on refining the existing asteroid catalog, it is finding new asteroids. The final search I ran was against the subset of ZTF observations that did not match any of the known asteroids. Random initializations for orbital elements were tried. Most of these initializations fail to converge on elements with enough hits to match real asteroids in the data, but a small number do successfully converge. So far I have identified 10 asteroids with 8 or more hits. I have verified that none of the orbital elements modeled for these asteroids appear in the catalogue of known asteroids I obtained from JPL. I have also done an ad-hoc review of the ZTF records to ensure that they are plausibly belonging to the same object. I believe that these represent new and unkown asteroids, and plan to submit them to the Minor Planet Center for possible classification. **UPDATE WITH REAL NUMBERS**

The ultimate goal of this project is not to simply perform a one time search of a dataset to identify some new asteroids. The goal is rather to create a tool that will be of enduring use to astronomy community for solving the problem of searching for new asteroids given large volumes of telescopic data. To that end, I plan to consult with Matt Holman and his colleagues at the Minor Planet Center to see what refinements and improvements would be required to upgrade this from a tool I can use to one that is of wider use to the astronomy community.

Chapter 1

Integrating the Solar System

1.1 Introduction

The calculation of planetary orbits is arguably the canonical problem in mathematical physics. Isaac Newton invented differential calculatus while working on this problem, and used his theory of gravitation to solve it. In the important special case that one body in the system is a dominant central mass, and all other bodies are viewed as massless "test particles", then a simple closed form solution is possible. This formulation of the gravitational problem is often called the Kepler Problem, named after Johannes Kepler. Kepler first studied this problem and published his famous three laws of planetary motion, the first of which states that the planets move in elliptical orbits with the sun at one focus. This is a surprisingly good approximation for the evolution of the solar system, and the basis for the efficient linearized search over orbital elements developed in this thesis.

The two body approximation is not, however, sufficiently accurate for a high precision model of the past and future positions of the known bodies in the solar system. While the mass of the sun is much larger than that of the heaviest planet, Jupiter, the planets are sufficiently massive (and often closer to each other and other bodies of interest) that gravity due to their mass must also be accounted for. The modern approach to determining orbits in the solar system is to use numerical integrators of the differential equations of motion.

1.2 The REBOUND Library for Gravitational Integration

REBOUND is an open source library for numerically integrating objects under the influence of gravity. It is available on GitHub. It is a first rate piece of software and I would like to thank Matt Holman and Matt Payne for recommending it to me last year. At the end of Applied Math 225, I wrote a research paper in which I learned to use this library, extensively tested it on the solar system, and used it to simulate the near approach of the asteroid Apophis to Earth that will take place in 2029. In this project, I use REBOUND as the "gold standard" of numerical integration. Because of its important role, I describe below how the IAS15 integrator I selected works. [1]

The IAS15 integrator, presented in a 2014 paper by Rein and Spiegel, is a an impressive achievement. It a fast, adaptive, 15th order integrator for the N-body problem that is (amazingly!) accurate to machine precision over a billion orbits. The explanation is remarkably simple in comparison to what this algorithm can do. Rein and Spiegel start by writing the equation of motion in the form

$$y'' = F[y', y, t]$$

Here y is the position of a particle; y' and y'' are its velocity and acceleration; and F is a function with the force acting on it over its mass. In the case of gravitational forces, the only dependence of F is on y; but one of the major advantages of this framework is its flexibility to support other forces, including non-conservative forces that may depend on velocity. Two practical examples are drag forces and radiation pressure.

This expression for y'' is expanded to 7th order in t,

$$y''[t] \approx y_0'' + a_0 t + a_1 t^2 + \cdots + a_6 t^7$$

They next change variables to dimensionless units $h = t/dt$ and coefficients $b_k = a_k dt^{k+1}$:

$$y''[t] \approx y_0'' + b_0 h + b_1 h^2 + \cdots + b_6 h^7$$

The coefficients h_i represent relative sample points in the interval $[0, 1]$ that subdivide a time step. Rein and Spiegel call them substeps. The formula is rearranged in terms of new coefficients g_k

[1] REBOUND provides a front end to use multiple integrators. In this project, I make exclusive use of the default IAS15 integrator.

with the property that g_k depends only on force evaluations at substeps h_i for $i \leq k$.

$$y''[t] \approx y_0'' + g_1 h + g_2 h(h - h_1) + g_3 h(h - h_1)(h - h_2) + \cdots + g_8 h(h - h_1) \cdots (h - h_7)$$

Taking the first two g_i as examples and using the notation $y_n'' = y''[h_n]$,

$$g_1 = \frac{y_1'' - y_0''}{h_1} \qquad g_2 = \frac{y_2'' - y_0'' - g_1 h_2}{h_2(h_2 - h_1)}$$

This idea has a similar feeling to the Jacobi coordinates: a change of coordinates with a dependency structure to allow sequential computations.

Using the b_k coefficients, it is possible to write polynomial expressions for $y'[h]$ and $y''[h]$:

$$y'[h] \approx y_0' + hdt \left(y_0'' + \frac{h}{2} \left(b_0 + \frac{2h}{3} \left(b_1 + \cdots \right) \right) \right)$$
$$y[h] \approx y_0 + y_0' hdt + \frac{h^2 dt^2}{2} \left(y_0'' + \frac{h}{3} \left(b_0 + \frac{h}{2} \left(b_1 + \cdots \right) \right) \right)$$

The next idea is to use Gauss-Radau quadrature to approximate this integral with extremely high precision. Gauss-Radau quadrature is similar to standard Gauss quadrature for evaluating numerical integrals, but the first sample point is at the start of the integration window at $h = 0$. This is a strategic choice here because we already know y' and y'' at $h = 0$ from the previous time step. This setup now reduces calculation of a time step to finding good estimate of the coefficients b_k. Computing the b_k requires the forces during the time step at the sample points h_n, which in turn provide estimates for the g_k, and then feed back to a new estimate of b_k.

This is an implicit system that Rein and Spiegel solve efficiently using what they call a predictor-corrector scheme. At the cold start, they set all the $b_k = 0$, corresponding to constant acceleration over the time step. This leads to improved estimates of the forces at the substeps, and an improved estimates for the path on the step. This process is iterated until the positions and velocities have converged to machine precision. The first two time steps are solved from the cold start this way.

Afterwards, a much more efficient initial guess is made. They keep track of the change between the initial prediction of b_k and its value after convergence, calling this correction e_k. At each step, the initial guess is b_k at the last step plus e_k. An adaptive criterion is used to test whether the

predictor-corrector loop has converged. The error is estimated as

$$\widetilde{\delta b_6} = \frac{\max_i |\delta b_{6,i}|}{\max_i |y_i''|}$$

The index i runs over all 3 components of each particle. The loop terminates when $\widetilde{\delta b_6} < \epsilon_{\delta b}$; they choose $\epsilon_{\delta b} = 10^{-16}$. It turns out that the b_k behave well enough for practical problems that this procedure will typically converge in just 2 iterations!

The stepsize is controlled adaptively with an analogous procedure. The tolerance is set with a dimensionless parameter ϵ_b, which they set to 10^{-9}. As long as the step size dt is "reasonable" in the sense that it can capture the physical phenomena in question, the error in y'' will be bounded by the last term evaluated at $h = 1$, i.e. the error will be bounded by b_6. The relative error in acceleration $\widetilde{b_6} = b_6/y''$ is estimated as

$$\widetilde{b_6} = \frac{\max_i |b_{6,i}|}{\max_i |y_i''|}$$

These are similar to the error bounds for convergence of the predictor-corrector loop, but involve the magnitude of b_6 rather than its change δb_6. An immediate corollary is that changing the time step by a factor f will change b_6 by a factor of f^7.

An integration step is computed with a trial step size dt_{trial}. At the end of the calculation, we compute the error estimate $\widetilde{b_6}$. If it is below the error tolerance ϵ_b, the time step is accepted. Otherwise, it is rejected and a new attempt is made with a smaller time step. Once a time step is accepted, the next time step is tuned adaptively according to $dt_{\text{required}} = dt_{\text{trial}} \cdot \left(\epsilon_b / \widetilde{b_6}\right)^{1/7}$ Please note that while the relative error in y'' may be of order 7, the use of a 15th order integrator implies that shrinking the time steps by a factor α will improve the error by a factor of α^{16}.

1.3 A Brief Review of the Keplerian Orbital Elements

In his work on the two body problem and the orbits of the planets, Kepler defined six orbital elements that are still in use today. A set of orbital elements pertains to a body as of a particular instant in time, which is typically referred to as the "epoch" in this context. The data sources I've seen all describe the time as a floating point number in the Modified Julian Day (mjd) format. In particular, I obtained orbital elements for all the known asteroids from JPL small body orbital

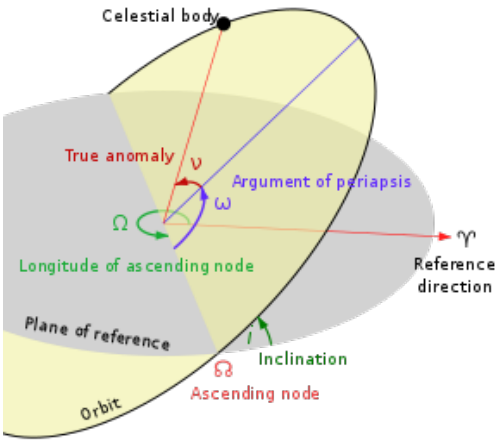

Figure 1.1: *Definition of the traditional Keplerian orbital elements orbital elements, courtesy of Wikipedia. Two parameters define the shape and size of the ellipse; two define the orientation of the orbital plane; and the last two orient the ellipse in its plane and the phase of the body on its ellipse.*

elements as of MJD 58600, corresponding to 27-Apr-2019 on the Gregorian calendar.

Here is a brief review of the definitions of these orbital elements

- a, the semi-major axis; named `a` in JPL and REBOUND
- e, the eccentricity; named `e` in both systems
- i, the inclination; named `i` in JPL and `inc` in REBOUND
- Ω, the longitude of the ascending node; named `node` in JPL and `Omega` in REBOUND
- ω, the argument of perihelion; named `peri` in JPL and `omega` in REBOUND
- f, the true anomaly; named `f` in REBOUND; not quoted directly by JPL
- M, the mean anomaly; named `M` in both systems
- mjd, the epoch as a Modified Julian Date

Distances are in A.U. in both JPL and REBOUND.

Angles are quoted in degrees in JPL and in radians in REBOUND.

These orbital elements have stood the test of time because they are useful and intuitive. They are ideal for computations, both theoretical and numerical, because in the case of the two body

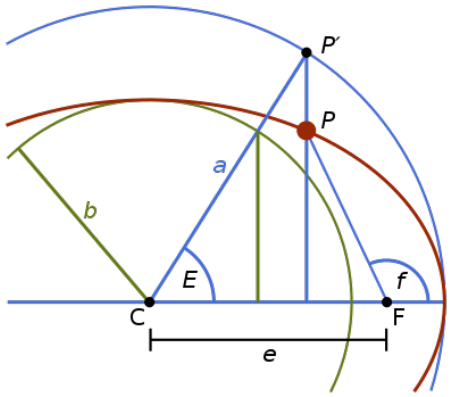

Figure 1.2: *Three Orbital Anomalies: Eccentric, Mean and True*

problem five of the six orbital elements remain constant. The careful reader will note that there are 8 entries in the table above, but I've described elements as coming six at a time. The epoch is considered to be the "seventh element" because in the Kepler two body problem, we can describe one body at different times, but it will have the same orbit. This point of view extends to the N-body problem, which is fully reversible; the same system can be described at at different moments in time. In practice, the orbital elements are often used to describe the initial conditions of all the bodies for an integration. The problem is then integrated numerically, possibly both forwards and backwards. Orbital elements can be reported for any body of interest.

A body orbiting the sun has six degrees of freedom. In Cartesian coordinates, there are three for the position and three for the velocity. In orbital elements, the first five are almost always $(a, e, i, \Omega, \omega)$. These five will remain constant for a body moving in the Kepler two body problem.

There is some variation in the choice of the sixth element, because different representations have different pros and cons. The true anomaly f is most convenient for transforming back and forth between orbital elements and Cartesian space. The mean anomaly M is most convenient for studying the time evolution of the system, because it changes linearly with time in the Kepler two body problem. The mean anomaly and true anomaly are related by the famous Kepler's Equation This relates the mean anomaly M to the eccentric anomaly E. The eccentric anomaly is yet another angle describing a body in orbit.

$$\tan\left(\frac{f}{2}\right) = \sqrt{\frac{1+e}{1-e}}\tan\left(\frac{E}{2}\right)$$

$$M = E - e\sin(E)$$

The linear evoluation of the mean anomaly, along with Kepler's equation, allows us to efficiently compute orbits for the Kepler two body problem. The relationship between the eccentric anomaly E and true anomaly f is a one to one function that can be evaluated fast on a computer. The mapping from eccentric anomaly E to mean anomaly M is also fast. The inverse mapping from M to E does not have a known analytical form. But it can be evaluated raplidly using Newton's Method with a reasonable initial guess. This is the method that I use to compute the orbits under the Kepler approximation.

1.4 Numerical Integration of the Planets in REBOUND

I have described above a library `REBOUND` that can efficiently integrate the solar system, and a data source Horizons that can be used to obtain accurate initial conditions for solar bodies. In principle integrating the solar system is a straightfoward exercise. In practice, there are quite a few details that need to be worked out before you can obtain reliably correct answers. You need to carefully specify the bodies you submit to Horizons. Horizons has separate identifiers for e.g. the barycenter of the Earth-Moon system, the Earth, and the Moon.

The module `horizons.py` contains functions used to query the Horizons AP. It also maintains a local cache with the results of prior queries; this yields significant savings in time because a typical horizons query using the Horizons API in `REBOUND` takes about one second. The main function in this module is `make_sim_horizons`. Given a list of object names and an epoch, it queries Horizons for their positions and velocities as of that date. It uses this data to instantiate a `REBOUND Simulation` object.

The module `rebound_utils.py` contains functions used to work with `REBOUND` simulations. It includes functions to build a simulation (`make_sim`). This will seek to load a saved simulation on disk if it is available, otherwise it will query Horizons for the required initial conditions. The function `make_archive` builds a `REBOUND SimulationArchive`. As the name suggets,

a `SimulationArchive` is a collection of simulation snapshots that have been integrated. This function also saves the integrated positions of the planets and test bodies as plain old `Numpy` arrays for use in downstream computations.

The module `planets.py` performs the numerical integration of the planets. To be more precise, it will integrate different collections of massive bodies in the solar system

- **Planets**: The Sun; The Earth and Moon as separate bodies; and the barycenters of the other seven IAU planets Mercury, Venus, Mars, Jupiter, Saturn, Uranus, and Neptune (10 objects)
- **Moons**: The 8 IAU planets, plus the following significant moons and Pluto (31 objects):
  Jupiter: Io, Europa, Ganymede, Callisto
  Saturn: Mimas, Enceladus, Tethus, Dione, Rhea, Titan, Iapetus, Phoebe
  Uranus: Ariel, Umbriel, Titania, Oberon, Miranda
  Neptune: Triton, Proteus
  Pluto: Charon
- **Dwarfs**: All objects in the solar system with a mass at least $1E-10$ Solar masses (31 objects):
  Planets: Earth, Moon, and barycenters of other seven planets
  Above 1E-9: Pluto Barycenter, Eris, Makemake, Haumea
  Above 1E-10: 2007 OR10, Quaoar, Hygiea, Ceres, Orcus, Salacia, Varuna, Varda, Vesta, Pallas
- **All**: All objects in the solar system with a mass at least $1E-10$ Solar masses (45 objects):
  All 8 planets (not barycenters)
  All the heavy moons above
  All the dwarf planets above

Each configuration above was integrated for a 40 year period spanning 2000-01-01 to 2030-12-31 and a time step of 16 days. I tested the integration by comparing the predicted positions of the 8 planets to the position quoted by Horizons at a series of test dates . The test dates are at 1 year intervals over the full 40 year span that is simulated. The best results were obtained by integrating smallest collection: Earth, Moon, and the barycenters of the other 7 planets. I was a bit surprised at this result and expected to do slightly better as the collection of objects became larger. Position errors are reported in AUs, with the root mean square (RMS) error over the 40 annual dates. I

Object Collection	Position Error	Angle Error
Planets	5.38E-6	0.79
Moons	1.35E-5	0.81
Dwarfs	5.38E-6	0.79
All	1.35E-5	0.81

Table 1.1: *Root Mean Square Error in Integration of Planets vs. Horizons*
Position Error: RMS error of 8 planets in AU.
Angle Error: RMS error in direction from planet to Earth geocenter, in Arc Seconds

also compute an angle error by comparing the instantaneous direction from each planet to Earth geocenter in the BME frame. I reported errors on this basis because on this problem, everything is done in terms of directions so precision eventually comes down to a tolerance in arc seconds.

While it might at first seem surprising that the results are worse for the more complex integrations including the moons, it's important to realize that this problem is intrinsically more difficult. Simulating the evolution of the barycenter of e.g. the Jupiter system is significantly easier than keeping track of the heavy moons and integrating them separately. Overall these results are excellent; over a span of 20 years in either direction, integrations are accurate on the order of 10^{-6} AU. The angular precision on the order of ~ 0.8 arc seconds is also excellent for such a long time span and well within the tolerance of this application.

After reviewing these results, I decided that the optimal strategy for the asteroid search problem was to treat the heavy bodies in the solar system as the smallest collection, shown on row 1. It is necessary to model the position of the Earth and Moon separately rather than the Earth-Moon barycenter, since our observatories are on the planet, not relative to the planetary system barycenter. However, the role of the other planets is only as a gravitational attractor that deflects the orbit of the Earth and the Asteroids. Speed is important in this application, so the smallest and fastest collection was the clear choice.

The second test of the integration extends the bodies under consideration from the planets to include ten test asteroids. I selected as the test asteroids the first 10 IAU numbered asteroids: Ceres, Pallas, Juno, Vesta, Iris, Hygiea, Egeria, Eunomia, Psyche, Fortuna. This test does not yet exercise the part of the code that insantiates asteroid orbits based on the bulk orbital elements files; that comes later. The asteroids here are initialized the same way as the planets, by querying

the Horizons API in `REBOUND`. (This method would not scale up to integrating all the asteroids though, because it is far too slow at about 1 second per asteroid.) The test protocol here was the same as for the planets. I compared the positions of these asteroids in the barycentric mean ecliptic frame predicted by my integartion at annual dates to the positions quoted by JPL. I also compared the instananeous angle from Earth geocenter to the asteroid.

Below are two charts summarizing the results. If we focus on a plausible window of ± 5 years around 2020, we can see that the selected planets integration is extremely accurate. Angular errors 5 years out are on the order of 0.5 arc seconds.

The charts and numerical outputs can be obtained at the command line by running

```
(kepler) $ python planets.py --test
```

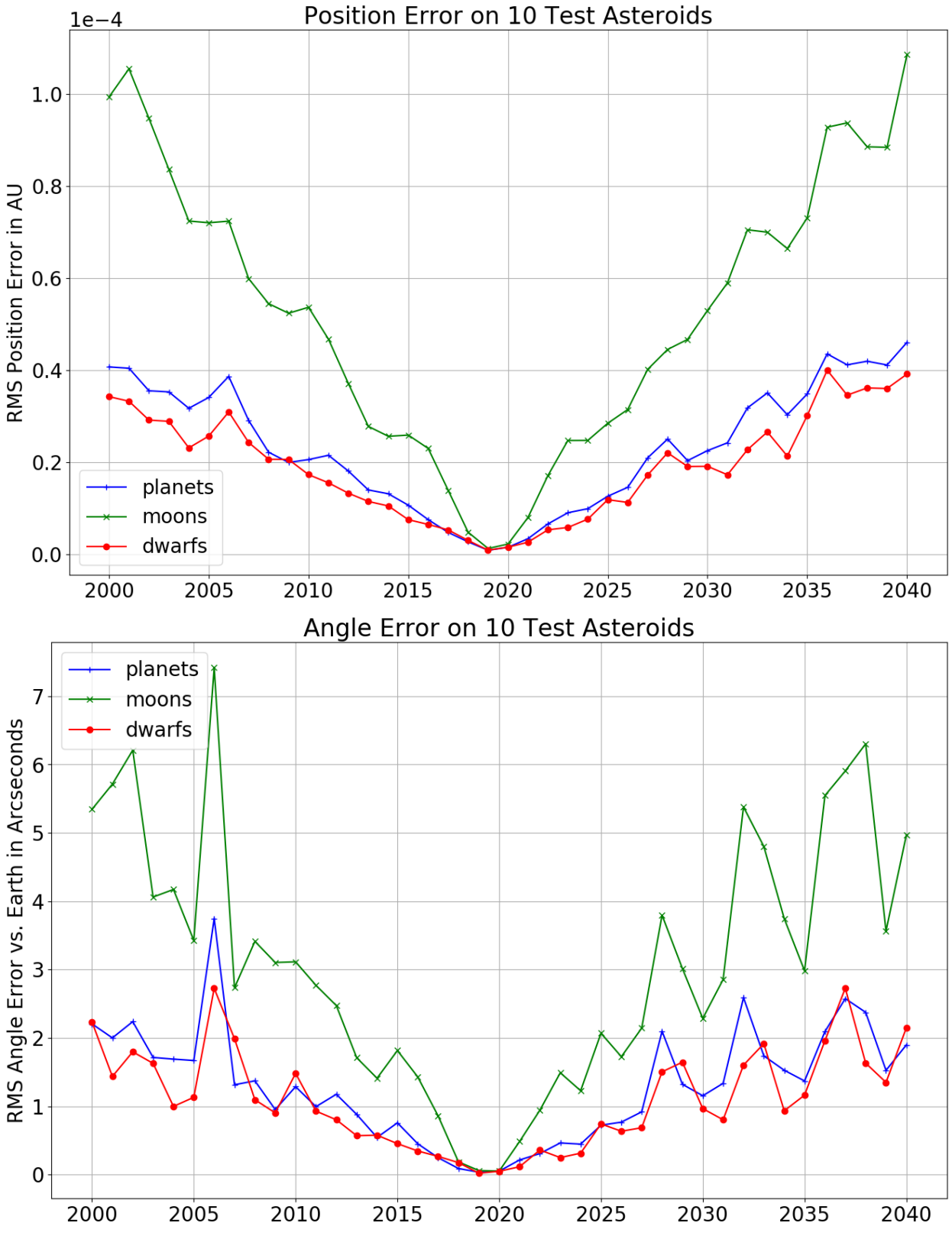

Figure 1.3: *Position and Angle Error of 10 Test Asteroids.*
My integration is compared to positions extracted from Horizons at 40 dates from from 2000 to 2040.
Initial conditions of the planets and asteroids are taken from the Horizons API as of MJD 58600 (2019-04-27).

(`kepler` is the name of the Anaconda environment with the necessary dependencies installed. An anaconda environment file `kepler.yml` is included in the `env` directory of the Git repository.)

1.5 Efficient Integration of All Known Asteroids

In the previous section I have described how to integrate the Sun and Planets where the initial conditions are obtained from Horizons using the API included in `REBOUND`. While this is a reliable and simple procedure that is ideal if you have only a few bodies to integrate, there is too much overhead in querying the Horizons API for it to be feasible for integrating all known asteroids. (If you spent 1 second on each of the 797,000 asteroids with available data, you would sit at your computer for over 9 days waiting for the job to complete, which it probably wouldn't because the angry folk at JPL would have probably blacklisted your IP address for spamming their server long before then.) Horizons includes two data files Small-Body Elements that are intended for bulk integrations and large scale analysis. The first two of these contain orbital elements for asteroids:

- Orbital elements for 541,128 asteroids with IAU numbers as of Epoch 58,600
- Orbital elements for 255,518 asteroids with designations but without IAU numbers; most elements have been updated to epoch 58,600 but some are older

I performed a bulk integration of the orbits of all 733,489 of these asteroids that had orbital elements as of the epoch 58600. In principle, I could have integrated all of the remaining asteroids with older elements forward in time to 58,600, but I had to draw the line somewhere. I took the fact that JPL didn't bother updating these older elements as evidence it wasn't a worthwhile use of time.

The module `asteroid_element.py` contains function for working with the orbital elements in these two data files. JPL follows different conventions than the defaults in `REBOUND`. Here is a brief comment abot the columns used

- **Num** is the IAU asteroid number; only available in the numbered asteroid file
- **Name** is the offial IAU name; only available in the numbered asteroid file
- **Designation** is the designation of known asteroids without IAU numbers

- **a** is the semi-major axis in AU
- **e** is the eccentricity (dimensionless)
- **i** is the inclination i quoted in degrees
- **w** is the argument of periapsis ω in degrees
- **Node** is the longitude of the ascending node Ω in degrees
- **M** is the mean anomaly M quoted in degrees
- **H** is the H parameter (mean brightness) in the H-G model of asteroid magnitude [2]BAA H-G Magnitude System
- **G** is the G parameter (sensitivity of brightness to phase angle)
- **Ref** is the name of a JPL reference integration

To use these elements in `REBOUND` is straightforward. Converting angles from degrees to radians is trivial. Distances are already quoted in AU. Converting from a mean anomaly M to a true anomaly f is not an obvious operation. Fortunately `REBOUND` allows you to instantiate an orbit with any legal combination of six elements, so I build the orbits from M and save a copy with f. The code to load the quoted orbital elements from JPL is in `load_ast_elt` in `asterod_element.py`. The function `load_data_impl` simply builds a Pandas DataFrame with all of the quoted elements. The function `ast_data_add_calc` adds the computed fields for the true anomaly f.

[2]https://www.britastro.org/asteroids/dymock4.pdf

There is one critical point to understanding these elements which is not at all obvious, so I will spell it out here. When quoting orbital elements of a body, they are always in reference to a dominant central mass, called the "primary". While it is often obvious from the context what that mass is, it is not always clear. In particular, two sound choices for the orbital elements of an asteroid are

- The central mass is the sun
- The central mass is the solar system barycenter

JPL quotes the asteroid orbital elements relative to the Sun, not the solar system barycenter. If you just plug in these elements to `REBOUND` without specifying a primary, it will default to the center of mass of all the massive bodies in the simulation that have been entered prior. Here are the most important lines of code to add each asteroid to the simulation:

```
sim_base = make_sim_planets(epoch_dt=epoch_dt)
primary = sim.particles['Sun']
sim.add(m=0.0, a=a, e=e, inc=inc, Omega=Omega, omega=omega, M=M, primary=primary)
```

The code to integrate all the known asteroids is in `asteroid_integrate.py`. At the risk of stating the obvious, when integrating asteroids, they are treated as massless "test particles." That is, they move under the influence of gravity from the massive particles in the simulation (here, the Sun and Planets), but they are *not* modeled as having their own gravity. This is an absolutely essential computational simplification. N massive bodies have $\binom{N}{2}$ gravitational interactions, so the cost of integrating them scales as $N^2/2$. If you add K massless bodies to the system, there are an additional $N \cdot K$ interactions, so the cost scales as $N^2/2 + N \cdot K$. I broke the asteroids into chunks of $K = 1000$, and used a collection of massive bodies with $N = 10$ (Sun, 8 Planets, Moon) A naive integration that treated them all as objects with gravitation that "just happened to be equal to zero" would cost $509,545$ calculations per time step. The efficient treatment by `REBOUND` of the asteroids as test particles reduces this cost to $10,045$, a savings of a factor of 488 (roughly $K/2$).

`make_sim_asteroids` build a `REBOUND` simulation initialized with the elements of asteroids with numbers in a given range. `calc_ast_pos_all` is the workhorse that integrates this simulation forwards and backwards in time over the requested date range. It saves a `REBOUND` simulation archive that allows the simulation to be loaded as of any time between the start and

end. It also saves plain old `numpy` arrays with the position and velocity of all the particles in the integration. As before, all asteroids had their motions integrated over a 40 year period spanning 2000 to 2040. The simulation archive is saved with an interval of 16 days; it will recover any requested date by performing a "short hop" integration from the nearest saved date. The `numpy` arrays are saved daily, because the intended use case is a naive cubic spline interpolation of positions and velocities. Even with all of the computational efficiency of `REBOUND`, this is a big computational job. According to the time stamps of the output files, running it took about 4:30 (four and a half hours); this was redlining a server with 40 high end Intel CPUs and 256 GB of RAM. The size of the data saved is 1.37 TB. (The `REBOUND` simulation archive is a fairly bulky format compared to the plain old data array; it's saving a lot of auxiliary information that is required to efficiently restart the simulation anywhere.) I ended up saving it to a network attached storage device on my network so it didn't overflow the main file system on the server where the job was running.

I tested the accuracy of this integration using a similar approach to the integration of the planets. I extracted the positions and velocities of 25 test asteroids from the Horizons API. The test asteroids were the first 25 numbered asteroids, starting with Ceres and ending with Phocaea. Comparisons were made at annual intervals in the 40 year period. Please note that while this test might look the same as the previous test of the asteroid integration, there is an important difference. The asteroids on the last test had their initial conditions determined by querying the same Horizons API. This time, only the initial conditions of the massive bodies were taken from Horizons. Initial conditions for the asteroids were based on the orbital elements in the two JPL files described above. I call thist the "soup to nuts" of the asteroid integration. It can be run from the command line by typing

```
(kepler) $ python asteroid_integrate --test
```

The real program is run from the command line with different arguments, e.g.

```
(kepler) $ python asteroid_integrate 10000 1000
```

would integrate a block of 1,000 asteroids starting at number 10000. Here are two charts presenting the root mean square position error and angle error to Earth geocenter for the 25 test asteroids.

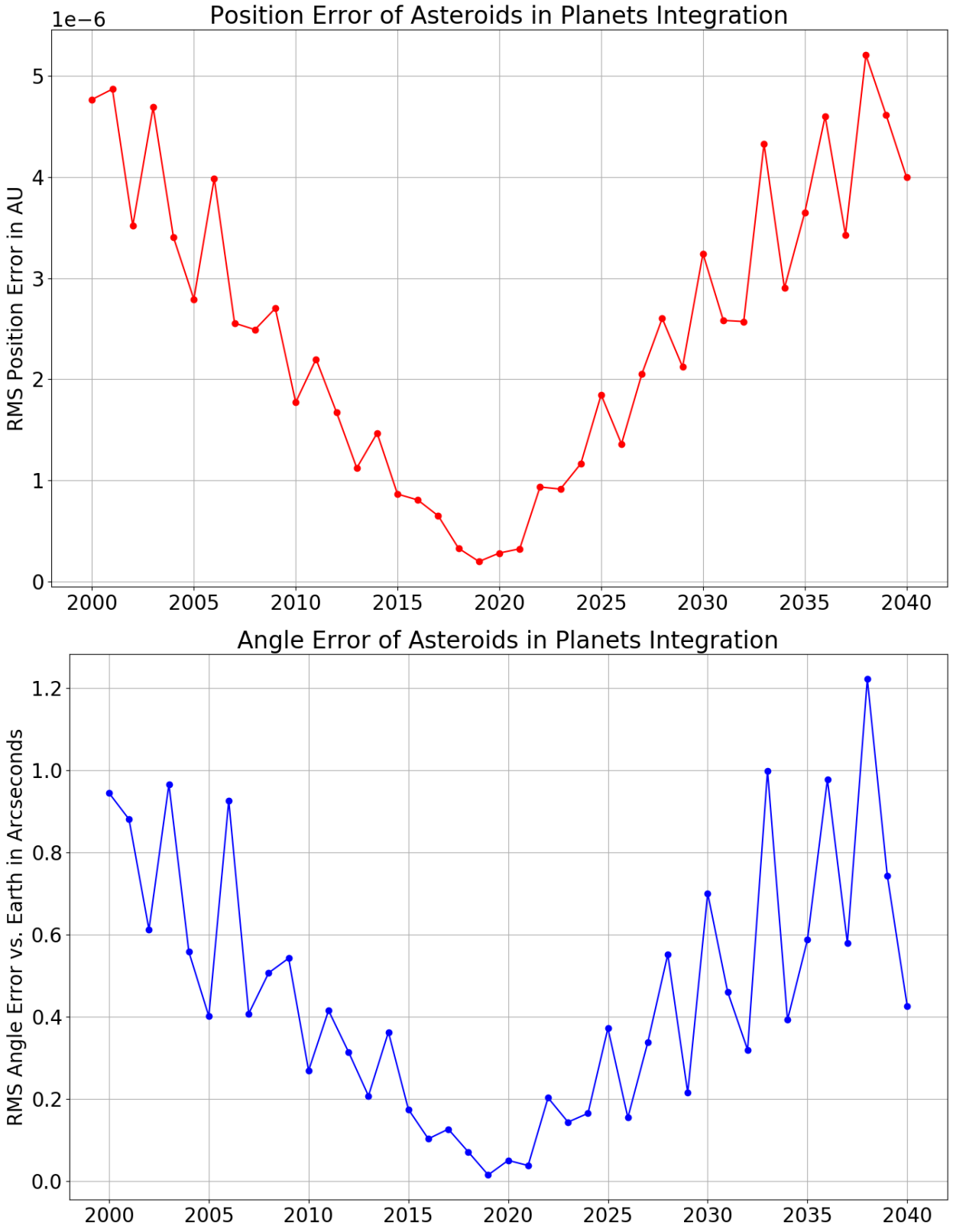

Figure 1.4: *Position and Angle Error of 25 Test Asteroids*
My integration is compared to positions extracted from Horizons at 40 dates from from 2000 to 2040.
Initial conditions of the planets are taken from the Horizons API as of MJD 58600 (2019-04-27).
Initial conditions of the asteroids are taken from orbital elements quoted in the Horizons small body elements file download as of MJD 58600.

I was very pleased with these results. Over a 40 year span, the error is at worse on the order of 5E-6 AU, with a RMS of 2.49E-6 AU. The angle errors over the full span have an RMS of 0.45 arc seconds, and peak around 1.0 arc second for dates 20 years in the past or future. In a more plausible 5 year band surrounding the epoch, accuracy is on the order of 0.2 arc seconds or better. Here is one anecdotal data point to put this degree of precision in context. On an earlier iteration, I foolishly did all calculations in the heliocentric rather than the barycentric frame. I was confused because the orbital elements were quoted in the heliocentric frame. When I finally switched to doing the computations in the barycentric frame (using the heliocentric elements only as a quotation mechanism for the initial conditions of the asteroids) I improved the accuracy by about 0.5 arc second.

To summarize this section: I have presented an efficient and high precision integration of almost all the known objects the solar system. The only dependency on Horizons in this calculation is the initial conditions of the Sun, Moon, and Planets. Over a span of 20 years this integration is accurate on the order of 2.5E-6 AU of position and 0.45 arc seconds of angle to Earth. I would like to take a step back here to comment on the progress that has been made in scientific computing and open source software that makes it possible for one person to perform such a large scale computation. When computations like this were first performed by NASA, they occurred on mainframes that were so expensive only the largest and best capitalized organizations could afford to buy them. Software to run these simulations was laboriously written and debugged, typically in `FORTRAN`, dating back to an era when programs ran on punch cards. As recently as the 1980s and 1990s, the task of integrating the orbits of 733,000 asteroids could realistically be undertaken by only a small handful of people with access to mainframe computers and extensive experience in both software engineering and mathematical physics.

Fast forward to today. High performance hardware is affordable and ubiquitous. Open source software has made a state of the art numerical integrator freely available to the public. NASA generously shares a treasure trove of valuable information along with excellent and user friendly APIs. A single motivated individual with a strong undergraduate background in physics and programming, but without specialized graduate level training, could plausibly have followed all the same steps shown here. I find that a truly remarkable achievement of the scientific community.

1.6 Integration of the Kepler Two Body Problem in Tensorflow

In the previous section we have seen how to efficently perform a numerical integration of the asteroids (or indeed any set of candidate elements). But for the search technique presented here to work, we require a much faster integration that is also differentiable. The integration algorithm used will be the semi-analytical solution to the Kepler two body problem. An efficient high speed implementation of this algorithm on GPU, including derivatives, is done in TensorFlow.

I will begin with a brief review of the analytical solution of the Kepler two body problem. This derivation loosely follows the treatment given at Wikiepedia - Kepler Problem, but I have added significant explanations and intermediate steps because I found the article too terse to fully explain the derivation. The assumption that the orbiting body is inifitesimally light compared to the central mass is often translated to a model that the body orbits under a central attractive field exerting a radial force $F(r)$ that always pulls towards the origin. Because this force is radial, it does not change the angular momentum $L = mv \times r$, which is a constant of the motion. As we recall from high school physics, a body with angular velocity ω has tangential velocity ωr and centripetal acceleration $\omega^2 r$. The angular momentum meanwhile in terms of m, r and ω is $L = m\omega r^2$ (put another way, the moment of interia is mr^2). The total acceleration of the orbiting body therefore has two terms: centripetal acceleration $\omega^2 r$, pointing towards the origin, and the radial acceleration d^2r/dt^2 pointing away from the origin. The force of gravity is $-GMm/r^2$ where G is the universal gravitational constant and M is the mass of the central body. Newton's equation $F = ma$ in this problem becomes

$$m\frac{d^2r}{dt^2} - m\omega^2 r = -\frac{GMm}{r^2}$$

We can divide out the mass of the test particle m to find

$$\frac{d^2r}{dt^2} - \omega^2 r = -\frac{GM}{r^2}$$

Let $h = \omega^2 r$ be the specific angular momentum. Since this is a constant, we can substitue $\omega = h \cdot r^{-2}$. The resulting equation with ω now eliminated as a variable is

$$\frac{d^2r}{dt^2} - h^2 r^{-3} = -GMr^{-2}$$

Since the angular momentum L is constant, a corollary is that the motion in confined to a plane. Let θ be the angle of the body in this plane of rotation. By the definition of angular velocity, $d\theta/dt = \omega$. We can also replace the differntial operator d/dt with a function of $d/d\theta$:

$$\frac{d}{dt} = \omega \cdot \frac{d}{d\theta} = hr^{-2}\frac{d}{d\theta}$$

We will now rewrite the equation with t eliminated in favor of θ:

$$\begin{aligned}
\frac{dr}{dt} &= hr^{-2}\frac{dr}{d\theta} \\
\frac{d^2r}{dt^2} &= \frac{d}{dt}\frac{dr}{dt} = hr^{-2}\cdot\frac{d}{d\theta}\left(hr^{-2}\frac{dr}{d\theta}\right) = hr^{-2}\frac{d}{d\theta}\left(hr^{-2}\right)\frac{dr}{d\theta} + hr^{-2}\cdot hr^{-2}\frac{d^2r}{d\theta^2} \\
&= h^2r^{-4}\frac{d^2r}{d\theta^2} - 2h^2r^{-5}\left(\frac{dr}{d\theta}\right)^2
\end{aligned}$$

In the second line, we applied the product rule to get the two terms in d^2/dt^2.

Now make this substitution for d^2/dr^2 to find

$$\begin{aligned}
h^2\cdot r^{-4}\frac{d^2r}{d\theta^2} - 2h^2r^{-5}\left(\frac{dr}{d\theta}\right)^2 - h^2r^{-3} &= -GMr^{-2} \\
r^{-2}\frac{d^2r}{d\theta^2} - 2r^{-3}\left(\frac{dr}{d\theta}\right)^2 - r^{-1} &= -\frac{GM}{h^2}
\end{aligned}$$

In the second line we muliply by the fraction r^2/h^2.

Now we introduce the essential change of variables that makes this problem analytically solvable. Let $u = r^{-1}$. Take the first two derivatives of u with respect to θ:

$$\begin{aligned}
\frac{du}{d\theta} &= -r^{-2}\frac{dr}{d\theta} \\
\frac{d^2u}{d\theta^2} &= \frac{d}{d\theta}\left(-r^{-2}\frac{dr}{d\theta}\right) = \frac{d}{d\theta}\left(-r^{-2}\right)\frac{dr}{d\theta} - r^{-2}\left(\frac{d}{d\theta}\frac{dr}{d\theta}\right) \\
&= 2r^{-3}\left(\frac{dr}{d\theta}\right)^2 - r^{-2}\frac{d^2r}{d\theta^2}
\end{aligned}$$

We can see that the two terms in the differential equation for r and θ are equal to $-du/d\theta$. Making this substitution as well as replacing r with u^{-1}, we find the simplified equation

$$\frac{d^2u}{d\theta^2} + u = \frac{GM}{h^2}$$

The anlytical solution to this equation is given by

$$u(\theta) = \frac{Gm}{h^2}\left(1 + e\cos(\theta - \theta_0)\right)$$

We can quickly verify that this is indeed a solution to the second order ODE since

$$\begin{aligned}
\frac{du}{d\theta} &= -\frac{Gm}{h^2} \cdot e\sin(\theta - \theta_0) \\
\frac{d^2u}{d\theta^2} &= -\frac{Gm}{h^2} \cdot e\cos(\theta - \theta_0) \\
\frac{d^2u}{d\theta^2} + u &= \frac{Gm}{h^2}\left(1 + e\cos(\theta - \theta_0) - e\cos(\theta - \theta_0)\right) = \frac{Gm}{h^2}
\end{aligned}$$

It was known in Kepler's time that this described an ellipse in the plane. The eccentricity e and phase angle θ_0 are determined by the initial conditions. The special case $e = 0$ describes a circular orbit.

The `REBOUND` library includes formulas for determining Cartesian coordinates from orbital elements. The formulas there were originally taken from Murray and Dermott's text on solar system dynamics [3]. In the `REBOUND` source code, the conversion formulas are in the file `tools.c`. I present them here in mathematical notation:

$$r = \frac{a \cdot (1 - e^2)}{1 + e\cos(f)}$$

$$\mu = G \cdot (M + m)$$

$$v_0 = \frac{\sqrt{\mu}}{a \cdot (1 - e^2)}$$

$$q_x = r \cdot \{\cos\Omega \cdot (\cos\omega \cdot \cos f - \sin\omega \cdot \sin f) - \sin\Omega \cdot (\sin\omega \cdot \cos f + \cos\omega \cdot \sin f) \cdot \cos i\}$$

$$q_y = r \cdot \{\sin\Omega \cdot (\cos\omega \cdot \cos f - \sin\omega \cdot \sin f) + \cos\Omega \cdot (\sin\omega \cdot \cos f + \cos\omega \cdot \sin f) \cdot \cos i\}$$

$$q_z = r \cdot \{(\sin\omega \cdot \cos f + \cos\omega \cdot \sin f) \cdot \sin i\}$$

$$v_x = v_0 \cdot \{(e + \cos f) \cdot (-\cos i \cdot \cos\omega \cdot \sin\Omega - \cos\Omega \cdot \sin\omega) - \sin f \cdot (\cos\omega \cdot \cos\Omega - \cos i \cdot \sin\omega \cdot \sin\Omega)\}$$

$$v_y = v_0 \cdot \{(e + \cos f) \cdot (\cos i \cdot \cos\omega \cdot \cos\Omega - \sin\Omega \cdot \sin\omega) - \sin f \cdot (\cos\omega \cdot \cos\Omega - \cos i \cdot \sin\omega \cdot \sin\Omega)\}$$

$$v_z = v_0 \cdot \{(e + \cos f) \cdot \cos\omega \cdot \sin i - \sin f \cdot \sin i \cdot \sin\omega\}$$

Here are the formulas used to go in the other direction, from Cartesian coordinates to orbital

elements:

$$
\begin{aligned}
\mu &= G \cdot (M + m) \\
r &= \sqrt{q_x^2 + q_y^2 + q_z^2} \\
v^2 &= v_x^2 + v_y^2 + v_z^2 & v &= \sqrt{v^2} \\
v_{\text{circ}}^2 &= \frac{\mu}{r} & v_{\text{circ}} &= \sqrt{v_{\text{circ}}^2} \\
a &= \frac{-\mu}{v^2 - 2v_{\text{circ}}^2} \\
h_x &= (q_y v_z - q_z v_y) & h_y &= (q_z v_x - q_x v_z) \\
h_z &= (q_x - v_y - q_y v_v) \\
h &= \sqrt{h_x^2 + h_y^2 + h_z^2} \\
v_{\text{diff}}^2 &= v^2 - v_{\text{circ}}^2 \\
r \cdot v_r &= (q_x v_x + q_y v_y + q_z v_z) & v_r &= (r \cdot v_r)/r \\
e_x &= \mu^{-1} \cdot (v_{\text{diff}}^2 \cdot q_x - r \cdot v_r \cdot q_x) & e_y &= \mu^{-1} \cdot (v_{\text{diff}}^2 \cdot q_y - r \cdot v_r \cdot q_y) \\
e_z &= \mu^{-1} \cdot (v_{\text{diff}}^2 \cdot q_z - r \cdot v_r \cdot q_z) & e &= \sqrt{e_x^2 + e_y^2 + e_z^2} \\
i &= \arccos\left(\frac{h_z}{h}\right) \\
n_x &= -h_y & n_y &= h_x \\
n &= \sqrt{n_x^2 + n_y^2} \\
\Omega &= \arccos 2(n_x, n, n_y) \\
E &= \arccos 2(1 - r/a, e, v_r) \\
M &= E - e \sin(E) \\
\omega + f &= \arccos 2(n_x q_x + n_y q_y, r, q_z) \\
\omega &= \arccos 2(n_x e_x + n_y e_y, e, e_z) \\
f &= (\omega + f) - \omega
\end{aligned}
$$

A few brief comments are in order. The conversion from orbital elements to Cartesian coordiantes is slightly messy but straightforward. It's an exercise in three dimensional geometry. The calculation of the inverse operation, from Cartesian coordinates to elements, is a bit more

involved. r is the distance of the object to the primary and v^2 its speed. v_{circ} is the circular velocity; if the object were moving at this velocity, tangentially to the primary with no radial velocity, it would be in a stable circular orbit. The semimajor axis a is a function of the object's total energy (which is negative for an object in a bound elliptical orbit). The vector $\mathbf{h} = (h_x, h_y, h_z)$ is our friend the specific angular momentum that we saw in the analytical solution to the Kepler problem. The quantity v^2_{diff} is the difference in squared velocity between this object and one moving in a circular orbit. It is used to determine the eccentricity vector $\mathbf{e} = (e_x, e_y, e_z)$. $\mathbf{n} = (h_y, h_x)$ is a vector that points along the ascending node $\hat{z} \times \mathbf{h}$. The function $\arccos 2(x, r, y)$ computes the arc cosine of x/r, but chooses a quadrant based on the sign of y. It can be thought of as a non-standard sister to the familiar library function $\arctan 2$. E is the eccentric anomaly, and the mean anomaly M is determined from E using Kepler's equation. The formulas presented here work in the non-degenerate case that the orbit is not in the $x - y$ plane. That case requires special handling, which `REBOUND` provides. Such special handling is much more challenging to code on GPU in TensorFlow than on CPU.

For those new to machine learning in Python, TensorFlow is a back end for efficient GPU computation. Keras is a set of interfaces that can in principle be implemented with multiple back ends. TensorFlow includes a reference implementation of Keras. The module `orbital_elemnt.py` includes custom Keras layers `ConfigToOrbitalElement` and `OrbitalElementToConfig`. They follow the recipes outlined above. The code looks a bit turgid compared to a C program; this style makes is sometimes required to make sure that the GPU computations are carried out exactly as desired.

The custom layer `MeanToEccentricAnomaly` computes the eccentric anomaly E given a mean anomaly M. As mentioned earlier, this must be done through numerical methods since there is no analytical solution. The numerical solution uses Newton's Method. Kepler's Equation gives us $M = E - e \sin(E)$. Define the function

$$F(E) = E - e \sin(E) - M$$

We solve $F(E) = 0$ using Newton's Method with the following simple iteration:

$$F'(E) = 1 - e\cos(E)$$
$$\Delta E = \frac{F}{F'E()}$$
$$E^{(i+1)} = E^{(i)} - \Delta E$$

This iterative update is carried out in the custom Keras layer `MeanToEccentricAnomalyIteration`. In standard computer programs, it is common to see iterations repeat until a convergence threshold is satisfied. In this case, the needs are a bit different. We want a function that will output answers close to within a tight tolerance of the right answer. But importantly, we also want functions that produce sensible derivatives when differentiated automatically using TensorFlow. Early trials showed that including any kind of branching or early termination logic was more trouble than it was worth. The poor performance of the GPU on conditional code made it slower, and the numerical derivatives occasionally gave wonky answers as inputs approached values where a the number of iterations changed. For this reason, I ended up setting a fixed number of 10 iterations. I verified that for the range of eccentricities allowed in candidate elements (I take at $e \leq 63/64 = 0.984375$) full convergence on 16 bit floating point is achieved. The custom layer `MeanToTrueAnomaly` converts M to the true anomaly f by first getting the eccentric anomaly E as described above, then using the relationship mentioned earlier between the mean and true anomalies, namely

$$\tan\left(\frac{f}{2}\right) = \sqrt{\frac{1+e}{1-e}}\tan\left(\frac{E}{2}\right)$$

The module `asteroid_model.py` contains the main classes used for modeling the position over time of a set of candidate orbital elements during the search process. This file includes a custom layer `ElementToPosition` that computes both the position and velocity of a set of orbital elements, again following the exact same logic above. The class `AsteroidPosition` is a keras Model. It is initialized with a vector of times, the MJDs as of which predicted positions are desired. `AsteroidPosition` is specialized for the problem of predicting all the position and velocities of a batch or orbital elements. It is *not* assumed that the observation times are shared in common across all the candidate elements in the batch. Internally, a list of the row lengths is maintained (i.e. the number of observation times for each of the canidate elements). Tensorflow

has a notion of a ragged tensor which is sometimes useful, but many of the calculations have to be done using flat tensors. When the class is initialized, it assembles arrays with the position and velocity of the sun in the barycentric mean ecliptic frame at all of the desired time points. This is done by calling the function `get_sun_pos_vel` which is defined in `asteroid_data`. This function in terms loads a saved integration done at daily resolution from the disk, and performs a cubic spline interolation. Experience has shown that this admittedly naive interpolation strategy is more than adequate to capture the position and velocity of the sun when the samples are daily. (The sun doesn't moves enough that if you disregard its motion, you can make errors on the order of 1E-3 AU, but it moves on small position and velocity scales.)

The main interface for this model accepts as inputs a set six vectors of length `batch_size`, representing the six orbital elements $(a, e, inc, \Omega, \omega, f)$. The wonderful feature of the Kepleriean orbital elements for two calculations in the two body problem now comes into play. Five of the six elements–all except the element describing the body's position on its orbital ellipse–remain constant through the motion. These are copied using `tf.repeat` to the desired shape. The mean anomaly M meanwhile has a linear dependence in time. The slope is called the mean motion and customarily written with N. The mean motion is given by $N = \sqrt{\mu/a^3}$ where a is the semi-major axis as usual, and $\mu = G \cdot (M + m)$. G is the universal gravitational constant, M is the mass of the sun, and m is the mass of the orbiting body. For this application, we model $m = 0$ so μ is a constant known at compile time. The mean anomaly as a function of time is just

$$M(t) = M_0 + N(t - t_0)$$

where M_0 represents the mean anomaly at the epoch, N the mean motion, and t_0 the MJD of the epoch. The mean anomaly M is extracted from the initial true anomaly f using the layer `TrueToMeanAnomaly` defined in `OrbitalElements`.

The `AsteroidPosition` class also has a method called `calibrate`. This method ensures that the predicted position and velocity at the set of calibrated orbital elements exactly match the output of the numerical integration. There are several ways this could have been done, including sophisticated approaches to compute the difference between the orbital elements predicted by the Kepler two body solution and the numerical integration. I opted for the simplest possible approach of just adding correct vectors dq and dv, because the goal of this piece of code is get

approximately correct positions and velocities as fast as possible. During the search process, as the orbital elements evolve, the asteroid position model is periodically recalibrated. At the learning rates used, errors due to the Kepler approximation breaking down over the small perturbations to the orbital elements are not a problem.

This class integrates approximate orbits and their derivatives to the orbital elements with remarkable speed. During training, a typical runtime per sample is around 350 microseconds. Samples here are an entire orbit with on the order of 5,000 distinct time points for each candidate element in the batch. That's fast! Numerical integrations on a CPU to full double precision are the gold standard for the right answer, but for the inner loop of a search process, they can't compete with an approximate GPU solution on speed.

Chapter 2

Predicting Directions from Positions

2.1 Introduction

2.2 Potential outcomes framework

Chapter 3

Searching for Asteroids

3.1 Introduction

Some people just cite papers in introductions for no reason.

3.2 Setup

3.3 Conclusion

References

[1] Hanno Rein, Shang-Fei Liu
REBOUND: An open-source multi-purpose N-body code for collisional dynamics.
Astronomy & Astrophysics. November 11, 2011.
arXiv: 1110.4876v2

[2] Hanno Rein, David S. Spiegel
IAS15: A fast, adaptive high-order integrator for gravitational dynamics, accurate to machine precision over a billion orbits.
Monthly Notices of the Royal Astronomical Society. Printed 16 October 2014.
arXiv: 1405.4779.v2

[3] Murray, C. D.; Dermott, S.F.
Solar System Dynamics
Cambridge University Press. 1999