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 randomly. An objective function is formulated based on log likelihood that rewards candidate elements for getting very close to a fraction of the observed directions. 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 on known asteroids by demonstrating the successful recovery of their orbital elements after initialization at perturbed values. A search is run 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.

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 planets 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

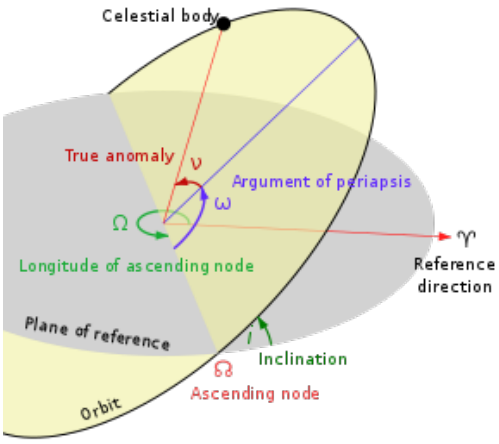

Figure 1.1: *Definition of the traditional Keplerian orbital elements orbital elements, courtesy of Wikipedia. Two parameters define the shape and size of the ellipse; two define the orientation of the orbital plane; and the last two orient the ellipse in its plane and the phase of the body on its ellipse.*

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
- t_0, 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

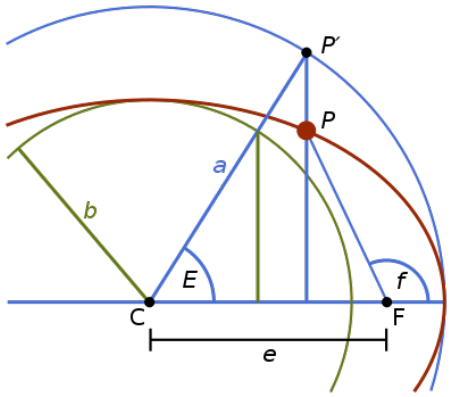

Figure 1.2: *Three Orbital Anomalies: Eccentric, Mean and True*

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 t_0 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 from 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 four 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):
  Earth: Moon (Luna)
  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 the other seven IAU 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 and heavy asteroids 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

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

errors are reported in AUs, with the root mean square (RMS) error over the 40 annual dates. I 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. [2] 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:

[2] For readers less familiar with astronomy, one AU (astonomical unit) is a unit of length that was defined to match the mean distance from the Earth to the Sun. It is official defined as equal to 1.495 878 707 E11 meters; see Wikipedia - Astronomical Unit. It is extremely convenient for solar system integrations since it leads to distances on the scale of around 1 to 10, and for this reason it is the default distance unit in `REBOUND`.

Ceres, Pallas, Juno, Vesta, Iris, Hygiea, Egeria, Eunomia, Psyche, Fortuna. This test does not yet exercise the part of the code that instantiates 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`. 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 integration 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
```

(`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.)

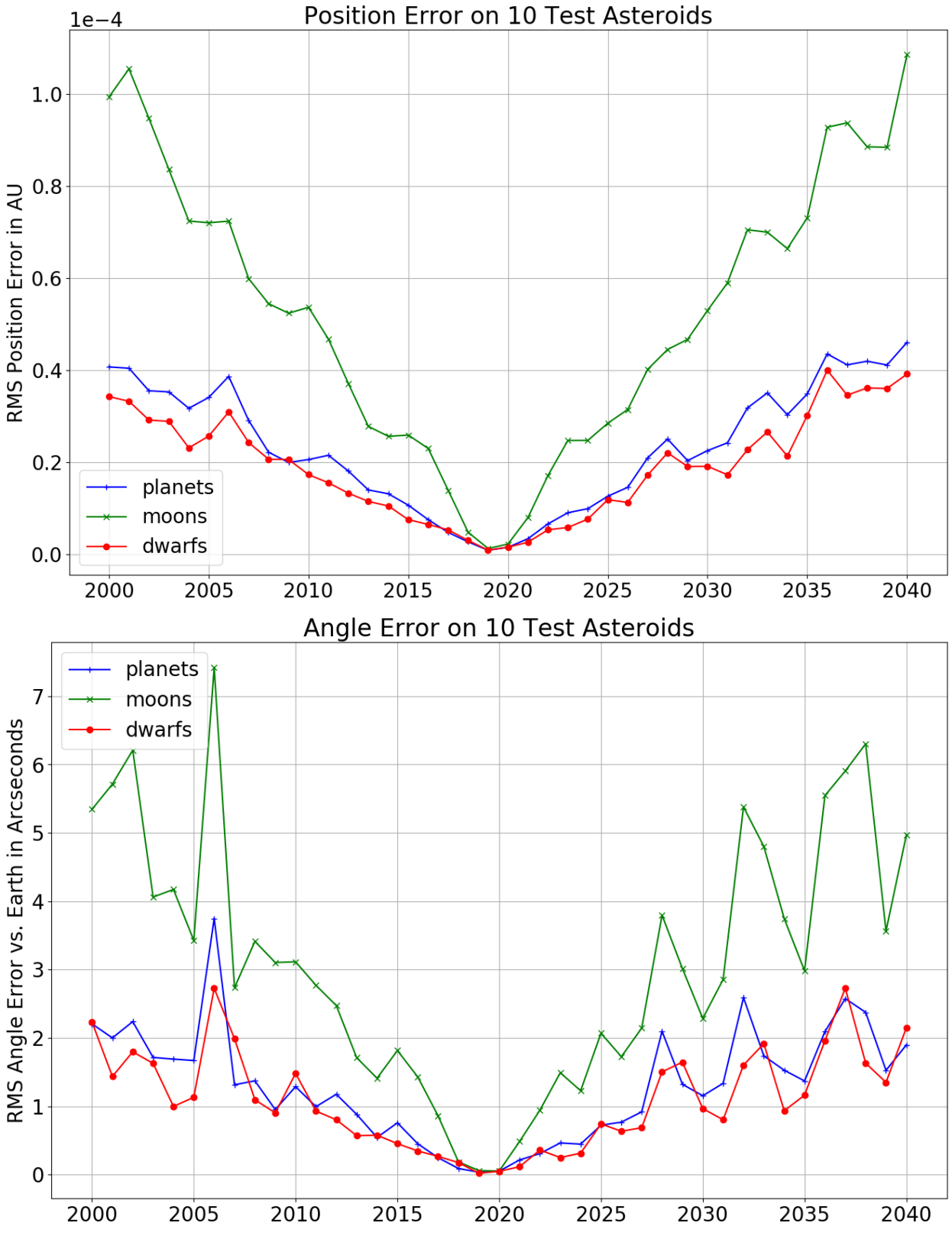

Figure 1.3: *Position and Angle Error of 10 Test Asteroids.*
My integration is compared to positions extracted from Horizons at 40 dates from from 2000 to 2040.
Initial conditions of the planets and asteroids are taken from the Horizons API as of MJD 58600 (2019-04-27).

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 [3]
- **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 true anomaly f.

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 clear from the context what that mass is, it is not always. 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:

[3] BAA H-G Magnitude System

```
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, Moon and Planets), but they are *not* modeled as having their own gravity. This is an 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` builds 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. I call this the "soup to nuts" test of the asteroid integration. While it 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. 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 10,000. Here are two charts presenting the root mean square position error and angle error to Earth geocenter for the 25 test asteroids.

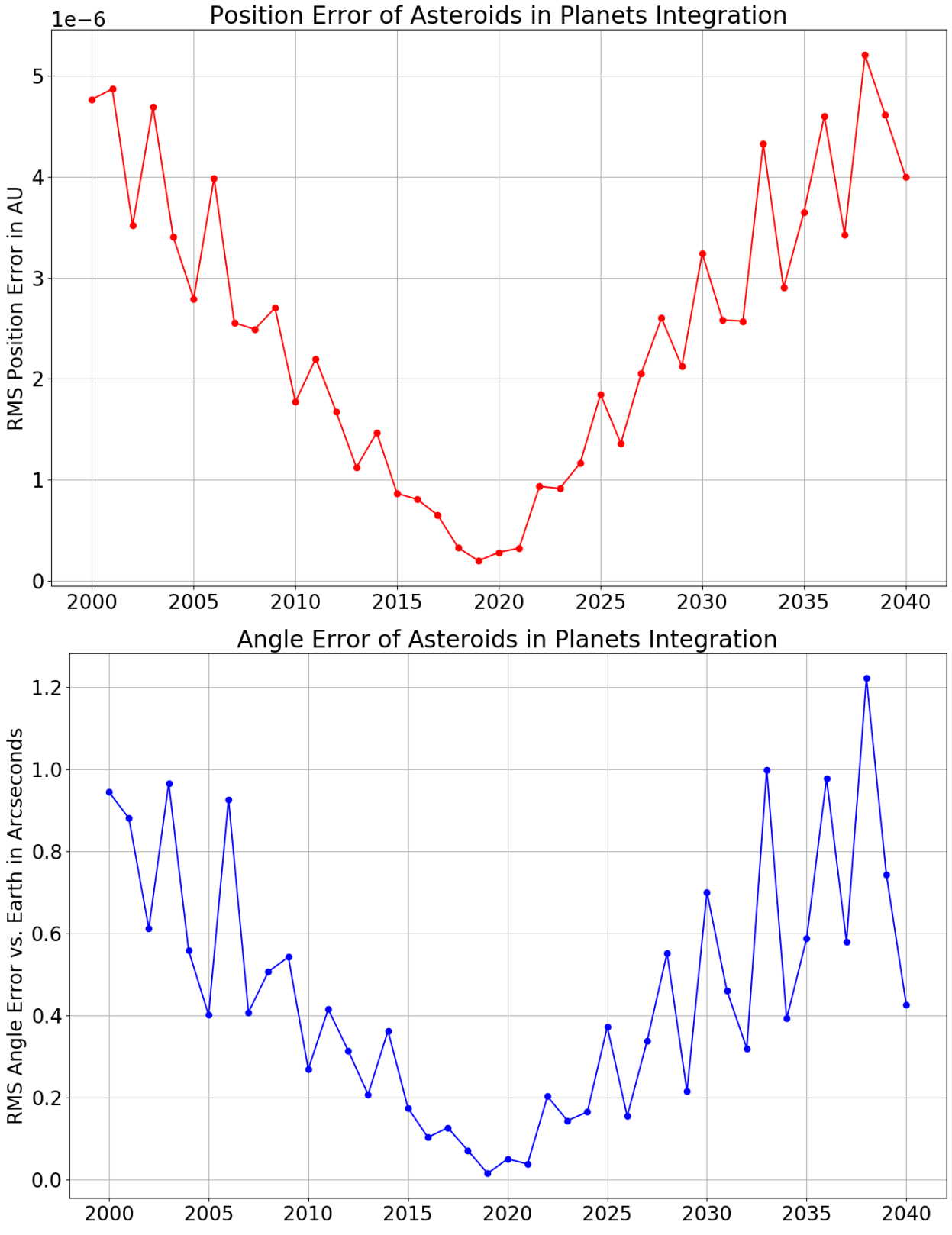

Figure 1.4: *Position and Angle Error of 25 Test Asteroids*
My integration is compared to positions extracted from Horizons at 40 dates from from 2000 to 2040.
Initial conditions of the planets are taken from the Horizons API as of MJD 58600 (2019-04-27).
Initial conditions of the asteroids are taken 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 0.6 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. They cost millions to tens of millions of dollars in today's money. 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 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 follow it. The assumption that the orbiting body is inifitesimally light compared to the central mass is often translated to a model where the body moves in 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 vector $\mathbf{L} = m\mathbf{v} \times \mathbf{r}$, 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 $\omega^2 r$. [4] The angular momentum magnitude meanwhile in terms of m, r and ω is $L = m\omega r^2$ (put another way, the moment of inertia 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 substitute

[4] In this discussion, ω refers to an angular velocity $\omega = d\theta/dt$, not the orbital element with the argument of perhelion. There are only so many Greek letters available, and I follow the conventions of mathematical physics in this paper whenver possible, even at the cost of the occasional clash of letters.

$\omega = h \cdot r^{-2}$. 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 vector $\mathbf{L}$ 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:

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

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

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

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 and vice versa. 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:

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

$$r^2 = q_x^2 + q_y^2 + q_z^2 \qquad r = \sqrt{r^2}$$

$$v^2 = v_x^2 + v_y^2 + v_z^2 \qquad v = \sqrt{v^2}$$

$$v_{\text{circ}}^2 = \frac{\mu}{r} \qquad v_{\text{circ}} = \sqrt{v_{\text{circ}}^2}$$

$$a = \frac{\mu}{2v_{\text{circ}}^2 - v^2}$$

$$h_x = (q_y v_z - q_z v_y) \qquad h_y = (q_z v_x - q_x v_z)$$

$$h_z = (q_x v_y - q_y v_x) \qquad h = \sqrt{h_x^2 + h_y^2 + h_z^2}$$

$$v_{\text{diff}}^2 = v^2 - v_{\text{circ}}^2$$

$$r \cdot v_r = (q_x v_x + q_y v_y + q_z v_z) \qquad r^2 \cdot v_r = r \cdot (r \cdot v_r)$$

$$e_x = \mu^{-1} \cdot (v_{\text{diff}}^2 \cdot q_x - r^2 \cdot v_r \cdot v_x) \qquad e_y = \mu^{-1} \cdot (v_{\text{diff}}^2 \cdot q_y - r^2 \cdot v_r \cdot v_y)$$

$$e_z = \mu^{-1} \cdot (v_{\text{diff}}^2 \cdot q_z - r^2 \cdot v_r \cdot v_z) \qquad e = \sqrt{e_x^2 + e_y^2 + e_z^2}$$

$$i = \arccos\left(\frac{h_z}{h}\right)$$

$$n_x = -h_y \qquad n_y = h_x$$

$$n = \sqrt{n_x^2 + n_y^2}$$

$$\Omega = \arccos 2(n_x, n, n_y)$$

$$E = \arccos 2(1 - r/a, e, v_r)$$

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

$$\omega + f = \arccos 2(n_x q_x + n_y q_y, r, q_z)$$

$$\omega = \arccos 2(n_x e_x + n_y e_y, e, e_z)$$

$$f = (\omega + f) - \omega$$

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. It turns out that simply ignoring this issue works adequately in practice for this application; mostly we are are converting from elements we identify to Cartesian coordinates, which is always numerically stable.

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_element.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 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 positions 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 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 moves enough in the BME 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 $(a, e, i, \Omega, \omega, f)$. The wonderful feature of the Kepleriean orbital elements for two calculations in the two body problem now comes into play. The first five orbital elements 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 output shape of the predicted positions and velocities are both $[N_{\text{obs}}, 3]$, where N_{obs} is the number of observations. These can be rearranged into ragged tensors of shape $[B, N_k, 3]$ where B is the batch size and N_k is the number of observations for candidate element k in the batch. The class makes available to its consumers the derivatives of these outputs with respect to the orbital elements.

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 offset 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.

The `AsteroidModel` 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.

1.7 Conclusion

I have presented in this chapter a high quality integration of the solar system. I have integrated a collection of heavy objects–the Sun, Moon, and 8 planets– sufficient to generate accurate predictions of the positions of an asteroid. I have integrated the orbits of all 733,489 known asteroids at a daily time step over 40 years. All of these results have validated against NASA using the Horizons API to extremely tight tolerances: positions are accurat on the order of 10^{-6} AU and angles between bodies to better than 1 arc second.

Finally, I have demonstrated a high performance implementation of the Kepler two body orbit on TensorFlow in the `AsteroidPosition` Keras layer. This can be calibrated to the numerically integrated orbit so it will be highly accurate for orbital elements near those it was calibrated against. This TensorFlow implementation runs extremely fast on the GPU and is capable of integrating on the order of 5000 time steps on the order of 300 microseconds.

Chapter 2

Predicting Directions from Positions

2.1 Introduction

In this chapter we will relate the integrated path of an asteroid to its appearance to an observer on Earth. We will review the equatorial coordinate system, which describes the location of an object in the sky with the parameters right ascension (RA) and declination (DEC). We will develop the calculations to transform between a direction from earth represented as a pair (RA, DEC) to a direction represented as a unit vector $\mathbf{u} = (u_x, u_y, y_z)$ in the barycentric mean ecliptic frame. We will explore the ZTF dataset of telescopic observation from the Palomar observatory in California. We will compute the direction $\mathbf{u}$ from which an observer at Palomar would have seen an object at a given observation time as a function of its predicted position $\mathbf{q}$ and velocity $\mathbf{v}$ at that observation time. This calculation will account for the time required for light to reach the observatory ("light time") and for the location of the observatory on the surface of the earth as distinct from geocenter ("topos adjustment"). We will run this calculation on all the known asteroids, computing the direction they would have appeared in the sky if they had been visible at Palomar. We will use this calculation to associate each ZTF obervation with the nearest asteroid to it in the sky.

Finally, we will study the statistical distribution of angular distance between the ZTF observations and the nearest asteroid. We will show that 65.71% of these observations fall within 2.0 arc seconds of the direction predicted for one of the 733,489 catalogued asteroids. We will compare this to the theoretical distribution of angular distances to the nearest asteroid if the predicted directions were distributed uniformly on the sphere. We will show that such a high

preponderance of "hits" is wildly unlikely and conclude that the asteroid catalogue has correct orbital elements for the objects being detected, and that the combined tolerance of the instruments and this calculation apparatus is on the order of 2.0 arc seconds.

2.2 A Brief Review of Right Ascension (RA) and Declination (DEC)

How can we describe the direction of an object we see in the sky? It is a question that dates back to the first astronomers in ancient times. The simplest and most intuitive coordinate system is the topocentric coordinate system, which uses the local horizon of an observer on the surface of the Earth as the fundamental plane. In this coordinate system, an object in the sky is described in terms of an altitude (sometimes called an elevation) and an azimuth. The altitude is how many degrees the object is above the visible horizon, so it will be between 0° and 90°. The topocentric coordinate system is intuitive and easy to use for an observer standing on the surface of the earth and looking at the sky. If I wanted to do some amateur astronomy with my kids and know where to point an inexpensive telescope, these are the coordinates I would want.

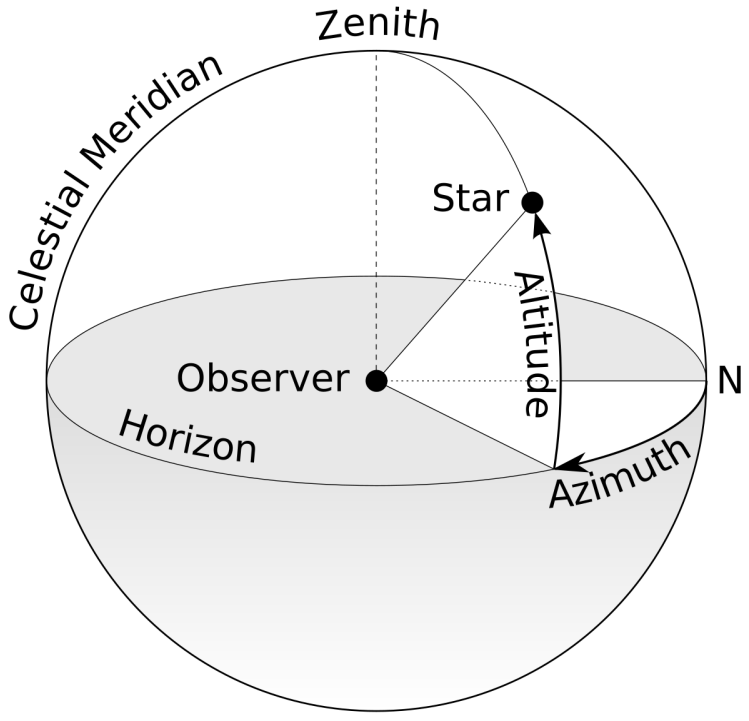

Figure 2.1: *The topographic coordinate system, courtesy of Wikipedia*

But the topcentric coordinate system is poorly suited to sharing observational data between astronomers. It is a different reference frame depending on where you are located on the Earth and the time in the evening. For this reason, astronomers dating back to ancient times developed the equatorial coordinate system. This coordinate system draws an imaginary sphere around the earth. The x axis of this coordinate frame points from the Sun to the center of the Earth at the Vernal Equinox of a specific date (typically J2000.0 these days). The y axis is 90 degrees to the East and the z axis points to the North pole. This is much easier to understand with a picture than with words:

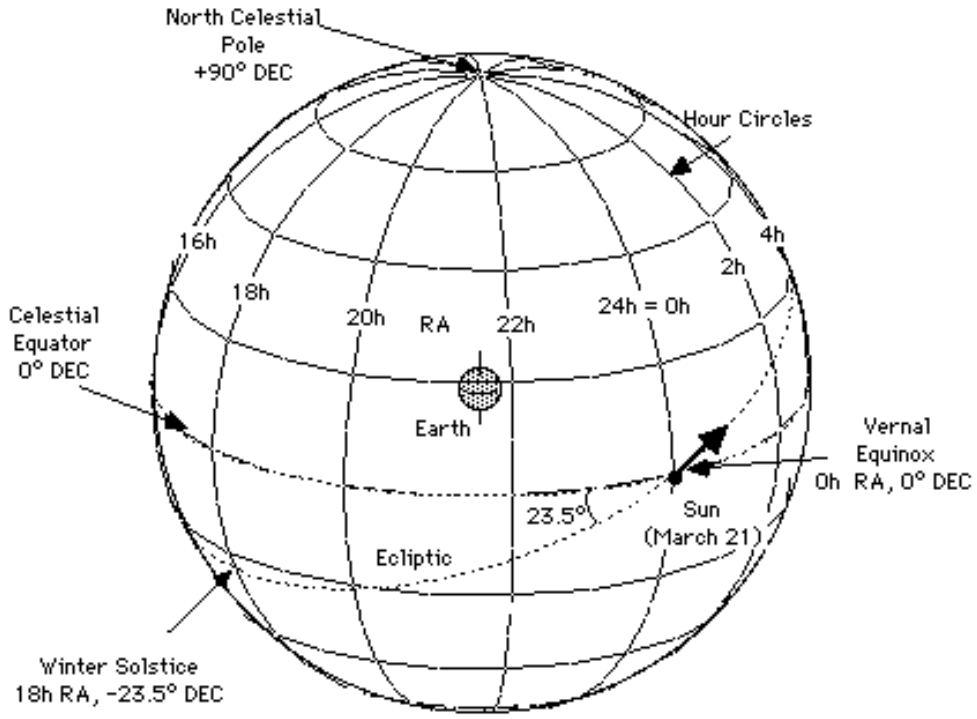

Figure 2.2: *The celestial sphere, courtesy of cool cosmos.*

The celestial sphere was originally conceived as having its z axis pointing along the axis of the Earth's rotation, i.e. from the South Pole to the North Pole. This is not a good definition though for the purposes of modern astronomy, because the Earth's rotational axis is not a fixed direction in the barycentric mean ecliptic frame. The Earth's rotational axis changes over time due to two separate effects: precession and nutation. Precession is a low frequency, predictable effect that

in analagous to wobbling movement of a top's rotational axis. It is typically covered in a first semester undergraduate physics course on mechanics. The precession of the Earth's axis is caused by mainly by the gravitational influence of the Sun and Moon, and has a period of approximately 25,772 years. Nutation is a high frequency effect, also caused mainly by the gravitational foreces of the Sun and Moon. In fact, the only difference bewteen the two effects is in the way we model and understand them. Precession isolates the mean perturbation over time; it's a slow, steady drift that is easy to predict. Nutation describes the much smaller, high frequency wobbles (on the order of tens of arc seconds) that are the residual after precession is accounted for.

Because of the instability of the Earth's rotational axis, the definition of the directions were updated to refer to the mean ecliptic as of a date, rather than Earth's rotational axis when the measurement is taken. Detailed calculations of the precession and nutation can then be used to adjust measurements taken on different dates. Ineed, the `astropy` astronomy package has the capability to perform these calculations. The diagram of the equataorial coordinate system clarifies this more modern definition. The vernal equinox is the moment when the Earth's orbit crosses the ecliptic.

This definition of a celestial coordinate system has in turn been superseded by a still more accurate system: the Interational Celestial Reference System, ICRS. The ICRS is based on an elegant idea. The goal of a celestial reference frame is to provide directions that are as near to fixed as possible in the reference frame of the solar system. Astronomers realized that since objects that are very far away from the Milky Way have orientations that are effectively fixed, the most precise way to define directions was based on large quantities of astronomical observations of these objects. In particular, radioastronomy (observations of stars in the radio wave frequency) is used. This is the basis of the ICRS and the frame of reference it defines: the International Celestial Refrence Frame (ICRF). The ICRF is a 3 dimensional coordinate system whose origin is the solar system barycenter (center of mass). The X, Y and Z axes are set by convention to line up very closely to the traditional definition of the J2000.0 frame. The orientation of the coordinate axes is based on the measured positions of 212 extragalactic objects, mainly quasars. These objects are so far away from Earth and the Milky Way galaxy that they are considered "fixed points" in space. [1]

[1] U.S. Naval Observatory - ICRS

Modern telescopes can achieve extremely accurate RA/Dec measurements by calibrating against the measured directions to these objects.

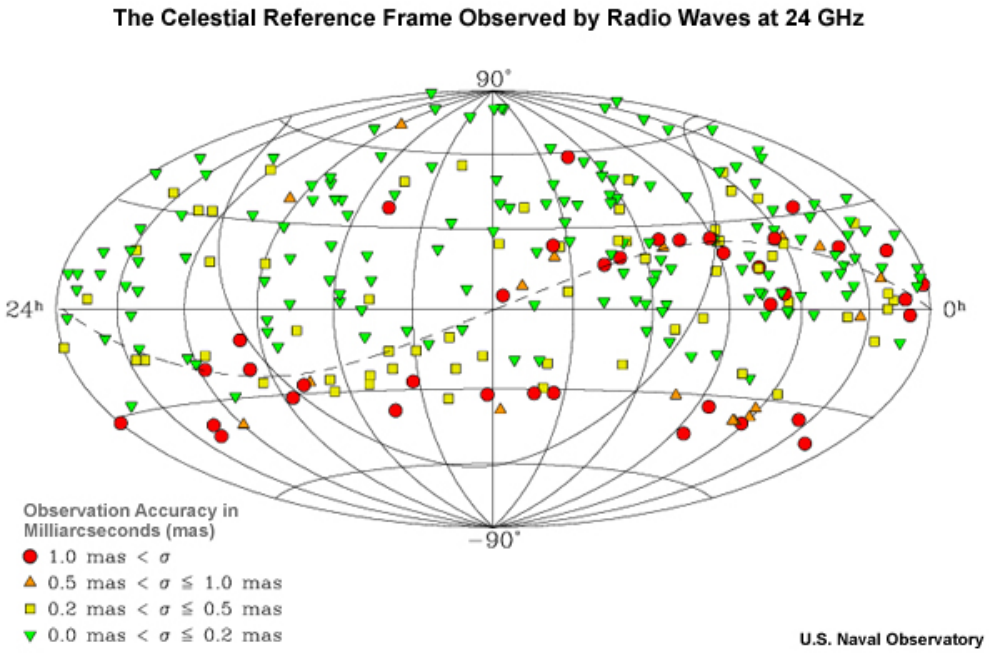

Figure 2.3: *The International Celestial Reference Frame (ICRF), courtesy of U.S. Naval Observatory.*

The main conclusion of this short section is that even an apparently simple idea, the direction from an observer on Earth to an object seen in the sky, is quite subtle if you want to take reproducible measurements that are accurate on the order of arc seconds. The ICRS is accurate on the order of a handful of milliarcseconds, meaning that uncertainty in the coordinate frame is not a meaningful contributor to errors for any calculations in this thesis. Fortunately this is a mature and well studied problem in astronomy, and it is addressed well by the `astropy` package. Rather than trying to reinvent the wheel, I use the `astropy` as much as possible in the next section to relate RA/Dec measurements to directions $\mathbf{u}$ in the barycentric mean ecliptic frame.

2.3 Mapping Between RA/Dec and Direction $\vec{u}$ in the Ecliptic Frame

In the previous section we have explored the ICRS, which provides the definition of the RA/Dec measurements quoted by astronomers. In this section, I review the definition of the barycentric mean ecliptic frame that is used for all calculations in this thesis. Then I explain the functions that

are used to perform the conversions between RA/Dec and directions in the ecliptic frame.

The barycentric ecliptic frame is the natural choice for computations done in relation to integrations of the solar system. It defines the xy plane to containing the mean ecliptic (ellipse of the Earth's orbit around the Sun). We take the mean ecliptic because the ecliptic varies slowly over time; the rotation of the Earth around the Sun in any given year would not be contained *exactly* in a plane, but there is a mean plane that comes closest in the least square sense to hitting all the points. The z axis is the unique direction that is orthogonal to this plane. Within the xy plane, the x axis is oriented from the Sun to Earth geocenter at the vernal equinox. In particular, the x axis is the direction from the Earth to the Sun at the vernal equinox as of the epoch, which is currently J2000.0 (JD 2451545.0, approximately January 1, 2000 12:00 UTC on the Gregorian calendar).

Here the vernal equinox is not defined with its ancient interpretation as the day when the day and night have equal lengths, but by the modern astronomical definition as the intersection of two planes: the mean ecliptic plane described above, and the mean equator as of that date. The idea of the mean equator as of a date is that as described above, the Earth's equator undergoes small wobbles at high frequency due to nutation. The mean equator as of a date disregards these wobbles, and instead considers only the trend of the equator, which changes slowly due to precession. This definition has the advantages of being more precisely measurable and changing more slowly than the true equator, making it the standard.

The module `ra_dec.py` handles conversions between RA/Dec and the direction $\mathbf{u}$ in the barycentric mean ecliptic (BME) frame. The two workhorse functions are named `radedc2dir` and `dir2radec`. The hard work in getting all of this to work comes in understanding the ideas of the coordinate systems and developing tests. Once you know what everything means and how to test that the implementation is right, it amounts to just a few lines of code. The `astropy` `SkyCoordinate` class is aware of both the ICRF and the BME frames. In order to transform between the two coordinate frames, we need only instantiate a `SkyCoordinate` of the required type, and ask `astropy` to transform it for us. It's so easy I will include a code snippet to give the flavor; this is from `radec2dir`:

```
obs_icrs = astropy.SkyCoord(ra=ra, dec=dec, obstime=obstime, frame=ICRS)
obs_ecl = obs_icrs.transform_to(BarycentricMeanEcliptic)
u = obs_ecl.cartesian.xyz
```

How can we test that all of this is working? By comparing my calculations to those done

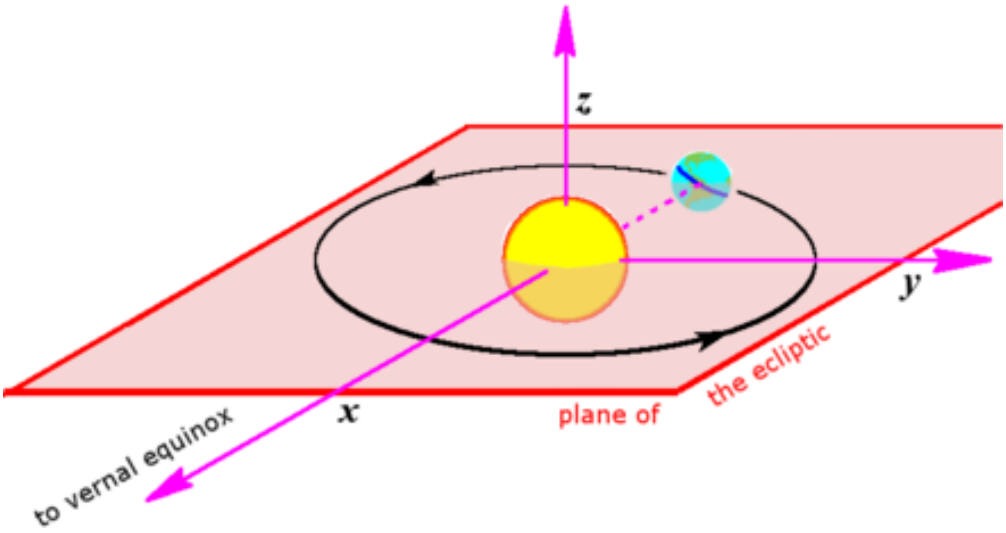

Figure 2.4: *The Heliocentric Ecliptic Frame, courtesy of Wikipedia*
The Barycentric Ecliptic Frame is analogous, but the origin is the solar system barycenter rather than the Sun.

by the JPL. At first I struggled to reconcile my results. I downloaded the computed position of Earth and Mars from JPL as well as the RA/Dec predicted by JPL for an imaginary observer at Earth geocenter. When I compared the JPL results to my own, they did not match, I eventually realized that the JPL calculation is accounting for light delay. In the next section, I wil explain this calculation and demonstrate that my results are consistent with JPL.

2.4 Computing a Direction $\vec{u}$ from Position $\vec{q}$ and Velocity $\vec{v}$

Suppose we have computed the position $\mathbf{q}_{\text{earth}}$ and $\mathbf{q}_{\text{ast}}$ where we believe the Earth and an asteroid are at the same time t. Can we compute the direction $\mathbf{u}$ from the Earth to the asteroid by simply subtracting the poitions and normalizing them, i.e. by

$$\mathbf{u} \stackrel{?}{=} \frac{\mathbf{q}_{\text{ast}} - \mathbf{q}_{\text{earth}}}{\|\mathbf{q}_{\text{ast}} - \mathbf{q}_{\text{earth}}\|}$$

The answer is no! This fails to account for the finite speed of light. The photons arriving at the observatory at the observation time t_1 were not emitted at t_1; they were emitted in the past, at time t_0. It's straightforward to calculate a correction that is accurate to first order once we draw a picture. It requires that we know both the predicted position and the predicted velocity of the

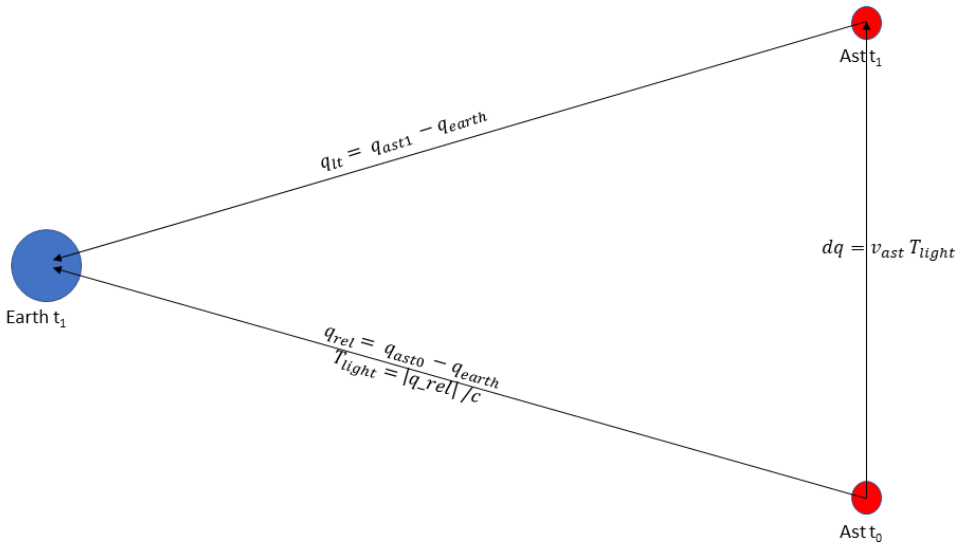

Figure 2.5: *Computing the Apparent Direction from Earth to an Asteroid, Accounting for the Speed of Light*

asteroid at time t_1.

$$\begin{aligned}
\mathbf{q}_{\text{rel}} &= \mathbf{q}_{\text{ast}} - \mathbf{q}_{\text{earth}} \\
T_{\text{light}} &= \|\mathbf{q}_{\text{rel}}\| / c \\
\Delta\mathbf{q}_{\text{ast}} &= \mathbf{v}_{\text{ast}} \cdot T_{\text{light}} \\
\mathbf{q}_{\text{lt}} &= \mathbf{q}_{\text{rel}} - \Delta\mathbf{q}_{\text{ast}} \\
\mathbf{u} &= \mathbf{q}_{\text{lt}} / \|\mathbf{q}_{\text{lt}}\|
\end{aligned}$$

Here c is the speed of light as usual.

One slightly counterintuitive fact is that we don't need to know or account for the speed of the Earth, or compute the relative velocity of the asteroid to the Earth. Why is this? We are performing all of our calculations in the barycentric mean ecliptic frame. This is very nearly an inertial frame of reference, so physics behaves "nicely" and everything works as expected. The only small departures from this frame being inertial are due to the orbit of the solar system around

the Milky Way and the acceleration of the Milky Way in the entire universe. If we used a different reference frame, e.g. the heliocentric frame, we would make errors due to the acceleration of this frame unless we accounted for them, which would be equivalent to the calculations shown here. In the BME frame, we have determined the positions of both the Earth and the asteroid at the instant t_1 at which photons landed on the detector of the observatory. To calculate the direction from which these photons arrived in the BME frame, we need only know where the asteroid was at the moment t_0 when photons emitted from it would have arrived at t_1. This calculation of an astrometric direction is implemented in the function `astrometric_dir`, also in `ra_dec.py`. It takes as inputs `q_body`, `v_body` and `q_obs`, and returns the astrometric direction $\mathbf{u}$ from the observer to the body accounting for light time.

It's worth pointing out here that a more fully correct description would be given by solving

$$\|\mathbf{q}_{\text{earth}}(t_1) - \mathbf{q}_{\text{ast}}(t_0)\| = c \cdot (t_1 - t_0)$$

This is the approach taken by the astronomy library `SkyField`, which solves this equation iteratively. Each iterative step matches the first order solution shown above. For the purpose of this problem, the light time between an asteroid and the Earth will typically be on the order of 20 minutes or so. (The speed of light in AU / minutes is 0.120, so 20 minutes of light travel covers 2.4 AU). This is short enough time interval that approximating the motion of the asteroid as linear is highly accurate.

We can also use this approach to approach to make a back of the envelope estimate of the errors we might make if we disregarded the light time adjustment Suppose an asteroid has $a = 3$, so by Kepler's thrid law this body would have an orbital period of $3^{3/2} \approx 5.2$ years. At a time when this asteroid is 3.0 AU from Earth, the light time would be 25 minutes, which workds out to 2.20E-4 of its orbital period. Converting this into degrees is a multiplication by 360, and into arc seconds a further multiplication by 3600. I estimate that ignoring light time could lead in this case to errors on the order of 285 arc seconds if the asteroid were moving perpendicular to its displacement to Earth. While that's close enough to aim an amateur's optical telescope, it's a catastrophic error here.

There is one more correction we need to account for: the position of the observatory on the surface of the Earth, or "topos". Our integration of the solar system gives us $\mathbf{q}_{\text{earth}}$, the position of

the Earth's center of mass in the BME as a function of time. But our observatory of course sits on the surface of the earth, not at the center. It would be too hot in the center, plus you couldn't see anything from there. Fortunately this is another well studied problem that is handled well by mature software libraries. I spent a lot of time trying to get a solution working based on the implementation in `astropy`, but I simply could not get it to work. Eventually I gave up and decided to use `SkyField` instead. The function `calc_topos` in `ra_dec.py` computes the topos adjustment at a named observatory site as of a vector of times. Currently the only observatory site required is Palomar mountain in California. The topos correction is a pair of corrections $\Delta\mathbf{q}_{\text{topos}}$ and $\Delta\mathbf{v}_{\text{topos}}$ such that

$$\mathbf{q}_{\text{obs}} = \mathbf{q}_{\text{earth}} + \Delta\mathbf{q}_{\text{topos}}$$

The spline is generated with the following code snippet (slightly edited for brevity)

```
obsgeoloc = SkyField.EarthLocation.of_site(site_name))
longitude, latitude, height = obsgeoloc.geodetic
topos = SkyField.Topos(latitude=latitude, longitude=longitude, elevatiom=elevation)
dq_topos = topos.at(obstime_sf).ecliptic_position().au.T * au
```

The function `qv2dir` combines the capabilities of `astrometric_dir` and `calc_topos`. It takes as inputs `q_body`, `v_body`, `q_earth`, `obstime_mjd` and `site_name`. It returns the astrometric direction from an observer at the named site on earth, to a body with the victor position and velocity vectors aligned with the observation times.

The functions described above are based on `numpy` arrays and run normally on the CPU. I also created a custom Keras Layer in TensorFlow that performs these calculations called `AsteroidDirection`. At initialization, an `AsteroidLayer` requires an array `ts` of MJDs, the row lengths for each observation in the batch, and the site name. The main `call` method of this layer accepts seven inputs: the orbital elements including the epoch, namely $(a, e, i, \Omega, \omega, f, t_0)$. It returns the predicted astrometric direction $\mathbf{u}$ from the named observatory to an asteroid with these orbital elements. This is the core layer that is used to predict directions to candidate orbital elements during the asteroid search.

We are now ready to review the demonstration that my calculations are consistent with JPL and SkyField. You can follow this demonstration interactively in the Jupyter notebook `03_RA_DEC.ipynb` in the `jupyter` directory of the project repository. I downloaded from Horizons a data set with with the positions and velocities of Earth and Mars. These sets contained

Sources	Difference
SKY vs. JPL	1.598
MSE vs. JPL	1.604
MSE vs. SKY	0.027

Table 2.1: *Mean differences between JPL, SkyField, and my calculations (MSE) in Arc Seconds*
My results are essentially identical to SkyField. Both SkyField and I disagree with JPL by 1.6 arc seconds.

29,317 rows spanning a 10 year period from 2010-01-01 to 2020-01-01 sampled at 3 hour intervals. I also downloaded from Horizons what they refer to as "observer" calculations including a RA/Dec. I specified an observer location at Palomar Mountain, which is a preset observatory in Horizons. The SkyField calculations use a slick end to end capability to predict where an observer would see an object. It uses a downloaded JPL ephemeris file, but is otherwise a self contained calculation that is independent from both JPL and my calculations.

I compared my calculations to two sources: JPL and SkyField, as well as comparing SkyField to JPL. The angular distance between two directions on the unit circle can be calculated with a simple formula that I will review later. For very small distances, the angular difference in radians is equal to the Cartesian difference between the two direction vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ on the unit sphere in $\mathbb{R}^3$.

Here is a table summarizing the mean difference between Horizons, SkyField and my calculations: My results are substantially identical to SkyField, to the miniscule tolerance of 0.027 arc seconds. Both SkyField and I differ a tiny bit from JPL, to the tune of 1.6 arc seconds.

I did some further tests, this time comparing my predicted directions from Earth to the first 16 asteroids with the results of applying my `radec2dir` function to the RA/Dec quoted by JPL, with the directions I predicted using the integrated orbits. In pseudocode, the test looks like this:

$$\mathbf{u}_{\text{JPL}} = \texttt{radec2dir(RA\_JPL, DEC\_JPL)}$$

$$\mathbf{u}_{\text{MSE}} = \texttt{qv2dir(q\_ast\_MSE, v\_ast\_MSE, q\_earth\_MSE, 'palomar')}$$

This test is exercising both the integration and the angle conversion. On 10 years of daily data for 16 asteroids, the root mean squared error is ***0.873 arc seconds.***

2.5 Predicting the Apparent Magnitude of an Asteoid

FILL THIS IN

2.6 Conclusion

I have presented in this chapter the calculations required to determine the astromentric direction of an asteroid observed from earth given its orbital elements. This calculation takes into account the time for light to travel from the asteroid to the Earth and the position of the observatory on the Earth's surface. I have tested these calculations against two independent sources, NASA Horizons and the SkyField astronomy library. I have demonstrated that my results are within 1.0 arc second of NASA and substantially identical to SkyField to within 0.01 arc seconds. Finally, I have demonstrated a high speed TensorFlow implementation of these calculations in the `AsteroidDirection` layer. The TensorFlow model can predict astrometric directions of a set of candidate orbital elements along with the derivatives of these predicted directions with respect to the six orbital elements.

Chapter 3

Analysis of ZTF Asteroid Detections

3.1 Introduction

Zwicky Transient Facility (ZTF) is a time-domain survey of the northern sky that had first light at Palomar Observatory in 2017. It is run by CalTech. My advisor Pavlos suggested it as a data source for this project. The ZTF dataset has two major advantages for searching for asteroids:

- ZTF gives a wide and fast survey of the key, covering over 3750 square degrees an hours to a depth of 20.5 mag
- A machine learning pipeline has been developed to classify a subset of ZTF detections that are classified as probable asteroids

The data set I analyze here consists of all ZTF detections that were classified as asteroids. Data on each detection include:

- **ObjectID** an identifier of the likely ojbect associated with this detection; multiple detections often share the same ObjectID
- **CandidateID** a unique integer identifier of each detection
- **MJD** The time of the detection as an MJD
- **RA** The right ascension of the detection
- **Dec** The declination of the detection

- **mag** The apparent magnitude of the detection

Available data also includes a number of additional fields that were not used in the analysis.

ALeRCE (Automatic Learning for the Rapid Classification of Events) is an astronomical data broker. ALeRCE provides a convenient API to access the ZTF asteroid data, which can be installed with `pip`. I used ALeRCE on this project to download the ZTF asteroid data set.

3.2 Exploratory Data Analysis of ZTF Asteroid Data

Before plowing into the search for new asteroids, I conducted an exploratory data analysis (EDA) of the ZTF asteroid dataset. This can be followed interactively in the Jupyter notebook `05_ztf_data.ipynb`. I took a download of the data running through 26-Feb-2020. The first detection is on 01-Jun2018. The dataset contains 5.69 total detections. The volume of detections increases very significantly beginning in July 2019; for practical purposes the dataset consists of 8 months of detections spanning July 2019 through February 2020.

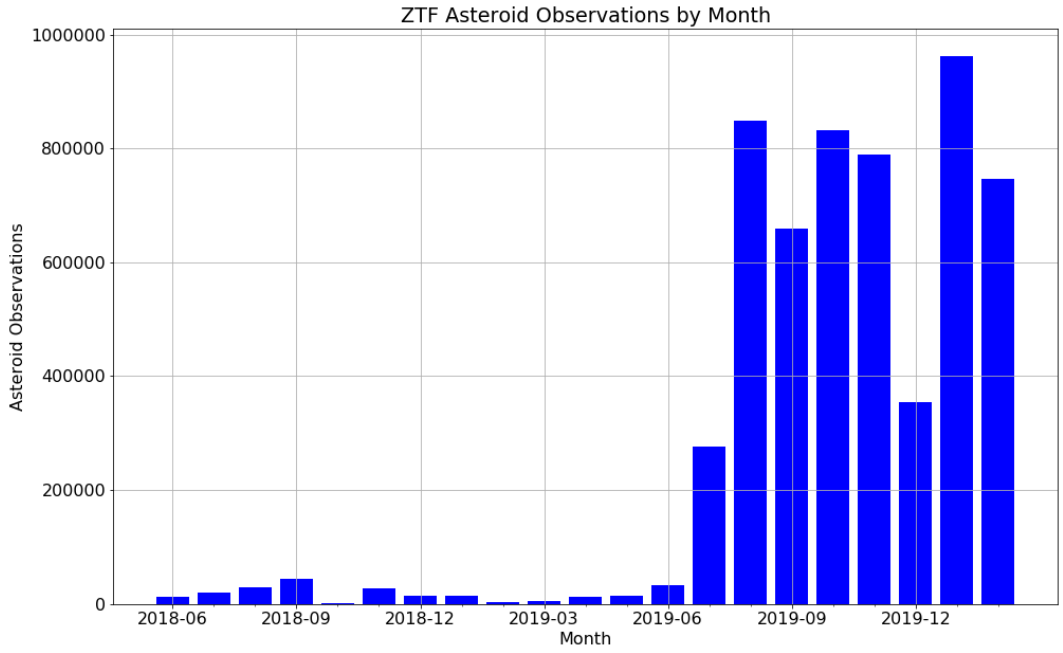

Figure 3.1: *ZTF Asteroid Detections per month*

The fields `mjd`, `ra`, `dec`, and `mag_app` are part of the original dataset. I have populated the columns `ux`, `uy` and `uz` by running `radec2dir` on the quoted RA/Dec from ZTF.

	ObjectID	CandidateID	TimeStampID	mjd	ra	dec	ux	uy	uz	mag_app	asteroid_prob
0	b'ZTF18acebhfp'	676397301515010013	14490	58430.397303	41.357345	58.879488	0.387942	0.653853	0.649598	18.946699	0.865682
1	b'ZTF18abodmwk'	596403415715010014	5831	58350.403414	30.969721	65.305308	0.358224	0.558644	0.748059	19.010401	0.855504
2	b'ZTF18abodmwk'	626428345715010011	10614	58380.428345	30.969705	65.305294	0.358224	0.558644	0.748059	18.935900	0.855504
3	b'ZTF18abodmwk'	630507595715015045	11250	58384.507593	30.969940	65.305305	0.358223	0.558645	0.748059	19.260401	0.855504
4	b'ZTF18abodmwk'	618384965715010022	9040	58372.384965	30.969643	65.305179	0.358226	0.558644	0.748058	19.220200	0.855504
...	...	...	...	...	...	...	...	...	...	...	...
5697957	b'ZTF20aareruw'	1151532523515015015	97109	58905.532523	253.007910	55.485537	-0.165587	-0.169403	0.971537	19.192400	0.608023
5697958	b'ZTF20aarerwx'	1151533002615015009	97110	58905.533009	232.886408	53.509617	-0.358833	-0.115301	0.926253	19.687099	0.559474
5697959	b'ZTF20aarerww'	1151533002115010003	97110	58905.533009	236.167899	54.618457	-0.322375	-0.116973	0.939357	19.957001	0.392662
5697960	b'ZTF20aarervr'	1151526063515015015	97098	58905.526065	286.235286	33.876902	0.232120	-0.509626	0.828494	19.049299	0.517241
5697961	b'ZTF18aajqsjs'	1151533002315015001	97110	58905.533009	237.382168	53.766297	-0.318612	-0.135924	0.938089	18.847700	0.992615

5697962 rows × 11 columns

Figure 3.2: *Preview of Pandas DataFrame of ZTF Detections*

Here is a chart showing the distribution of apparent magnitudes in the ZTF detections. It's shown on a log scale because there are so many more detections around the peak 19.5 than at the brightest (10) and and dimmest (22) magnitudes.

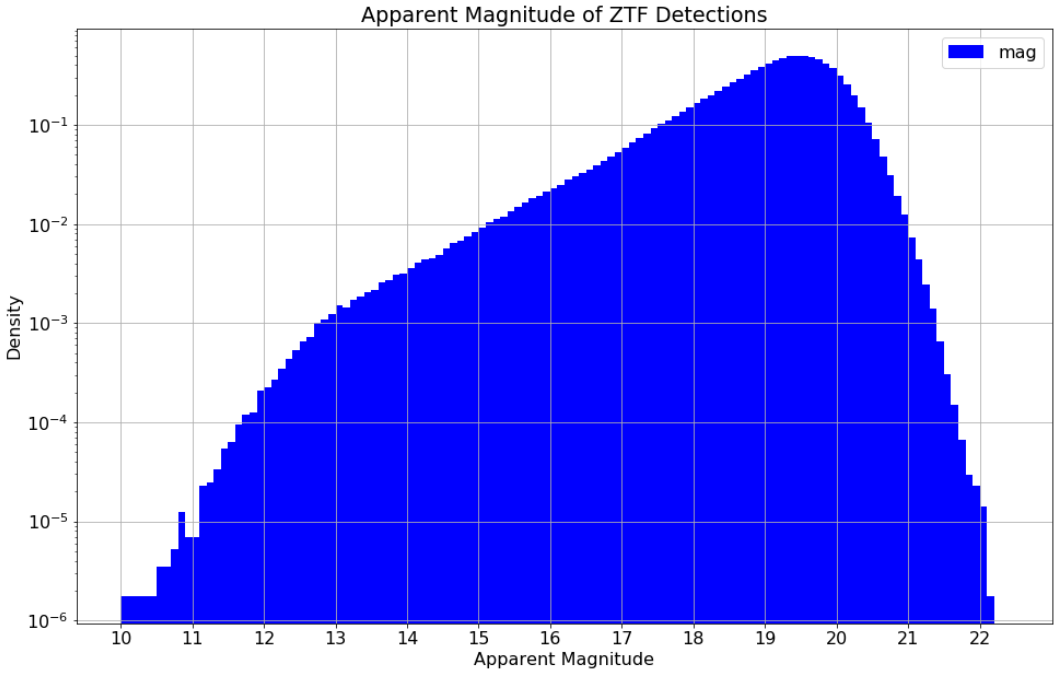

Figure 3.3: *Apparent Magnitude of ZTF asteroid detections.*

3.3 The Angular Distance Bewteen two Directions $\vec{u}_1$ and $\vec{u}_2$

A recurring task in this thesis to compute the angular distance bewteen two directions on the unit sphere. If $\mathbf{u}_1$ and $\mathbf{u}_2$ are on the unit sphere, we can compute their Cartesian distance s in the usual way,

$$s = \|\mathbf{u}_2 - \mathbf{u}_1\|$$

s will be in the interval $[0, 2]$. We would also like to know the angular distance θ of the shortest path on the surface of the sphere (geodesic) connecting these two points. On Earth, this would be analogous to the length of the great circle route by airplane between two cities. Here is a simple picture demonstrating the derivation of the formula relating s and θ. The two directions

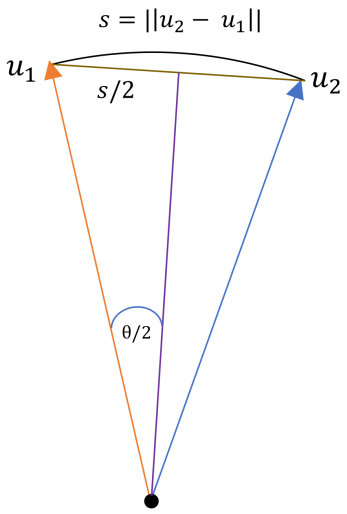

Figure 3.4: *The angular distance between two directions on the unit sphere.*

are shown as arrows pointing up. The distance between them s is bisected by a line segment from the center of the sphere. This forms a right triangle with hypotenuse 1 and side length $s/2$ opposite angle $\theta/2$. We thus obtain the formulas

$$\sin(\theta/2) = s/2$$
$$\theta = 2\arcsin(s/2)$$
$$s = 2\sin\theta(\theta/2)$$

This function is implemented in `astro_utils` as `deg2dist` and `dist2deg` to convert between degrees and Cartesian distance in either direction.

3.4 Finding the Nearest Asteroid to Each ZTF Observation

We now have in principle all the tools required to find which asteroid was closest in angular distance to each ZTF observation. To recap the key steps, each ZTF observation is converted from a RA/Dec to a direction in the BME. The position of Earth and each of the 733,489 catalogued asteroids are integrated as of the objervation time, and the direction bewteen them is calculated. There are a few problems with the brute force approach implicitly suggested above. We have 5.7E6 detections and 7.3E5 catalogued asteroids for a total of 4.16E12 (4.16 billion) interactions. Even if we work in single precision with 4 bytes per float, we need 12 bytes for a 3D direction difference translating to about 50 GB to load the matrix in memory. The bigger problem is that a brute force integration of all the asteroids at the MJDs of the all 5.7 million observations would be brutally slow.

Fortunately there are a few simple tricks we can use that together make this problem tractable. The ZTF detections come from a series of images taken through the same telescope, so they come in blocks made at the same time. The 5.7 million rows share "only" 97,111 distinct MJDs. We also don't need to re-integrate the asteroid orbits. We've already done a high precision numerical integration at a 1 day frequency and saved the results into `numpy` arrays in blocks of 1,000 asteroids at a time. Our entire data span only 635 days. Loading one block of 1,000 asteroids therefore has only 3.81 million numbers (635 days x 1,000 asteroids x 6 numbers per integration point).

The code to load the basic ZTF data from ALeRCE is included in the module `ztf_data.py`. The main function used by consumers is `load_ztf_det_all`, which loads a cached copy of all the available detections from the local disk. The module `ztf_nearest_ast.py` contains a Python program that peforms the calculation described above. The block size is a parameter that can be controlled from the commandline; I ended up leaving it at 1,000. Checking back from my handwritten notes when I ran the job, it took approximately 25 hours to complete on a powerful server with 40 Intel CPU cores. The job ended up being memory bound, maxxing out all 256 GB of available RAM on the server. The initial job described above writes only computes the nearest

asteroid in a block of 1,000 asteroids. A second reduction operation is then carried out to find the nearest asteroid overall. To limit memory usage, this was done in two steps. First, I took 16 chunks of 1,000 at a time to find the nearest asteroid in a block of 16,000 to each detection. Then I combined all the blocks of 16,000 (about 46) to generate one file with the nearest asteroid.

The work of splining the asteroid directions is done in the module `asteroid_dataframe.py`. It includes functions to load the asteroid data from disk; spline the positions and velocities to the requested dates; and compute the astrometric directions to these splined positions and velocities. The module `ztf_ast` does the work of comparing a block of ZTF observations to a block of splined asteroid directions and finding the nearest one. Once these calculations have been done once, you don't need to worry about them unless you are adding a new block of ZTF data. Consumers can load the assembled DataFrame including the nearest asteroid number and distance with a single call to `load_ztf_nearest_ast` which is defined in `asteroid_dataframe`. For the motivated reader who would like a detailed an interactive review of all these calculations, please see the Jupyter notebook `04_asteroid_dataframe.ipynb`. It includes tests comparing my splined outputs of daily data to Horizons data downloaded at a 3 hour interval.

	mjd	ra	dec	nearest_ast_num	nearest_ast_dist	ast_ra	ast_dec
0	58430.397303	41.357345	58.879488	1208789	0.005029	41.396388	58.592038
1	58350.403414	30.969721	65.305308	1227812	0.024428	33.729101	64.536183
2	58380.428345	30.969705	65.305294	1169677	0.015510	29.207596	64.817653
3	58384.507593	30.969940	65.305305	1251951	0.012386	30.227911	65.945543
4	58372.384965	30.969643	65.305179	1246591	0.025343	34.169666	64.771024
...	...	...	...	...	...	...	...
5459014	58905.532523	253.007910	55.485537	1102168	0.036944	253.707834	53.408139
5459015	58905.533009	232.886408	53.509617	1028157	0.084402	224.967815	54.919912
5459016	58905.533009	236.167899	54.618457	539940	0.052254	240.693936	56.155104
5459017	58905.526065	286.235286	33.876902	1246304	0.014054	285.998189	34.657915
5459018	58905.533009	237.382168	53.766297	539940	0.053269	240.693936	56.155104

5697962 rows × 7 columns

Figure 3.5: *Preview of Pandas DataFrame of ZTF Detections Including Nearest Asteroid*

One natural question is how the brightness of the nearest asteroid to a detection varies between hits and misses. The chart below plots two histograms of the absolute magnitude parameter H for the nearest asteroid to each ZTF detection. Hits are shown in blue, misses in reds. The distribution of both charts is similar, but there is a noticeable tilt in favor brighter asteroids being hits and dimmer asteroids being misses. This is exactly what we would expect; the nearest asteroids when the distance is over the hit threshold are close to a random sample of the asteroids. The blue series though is not a random sample, it's a sample conditional on the detection matching a real asteroid. The brightest asteroids are more likely to be successfully detected.

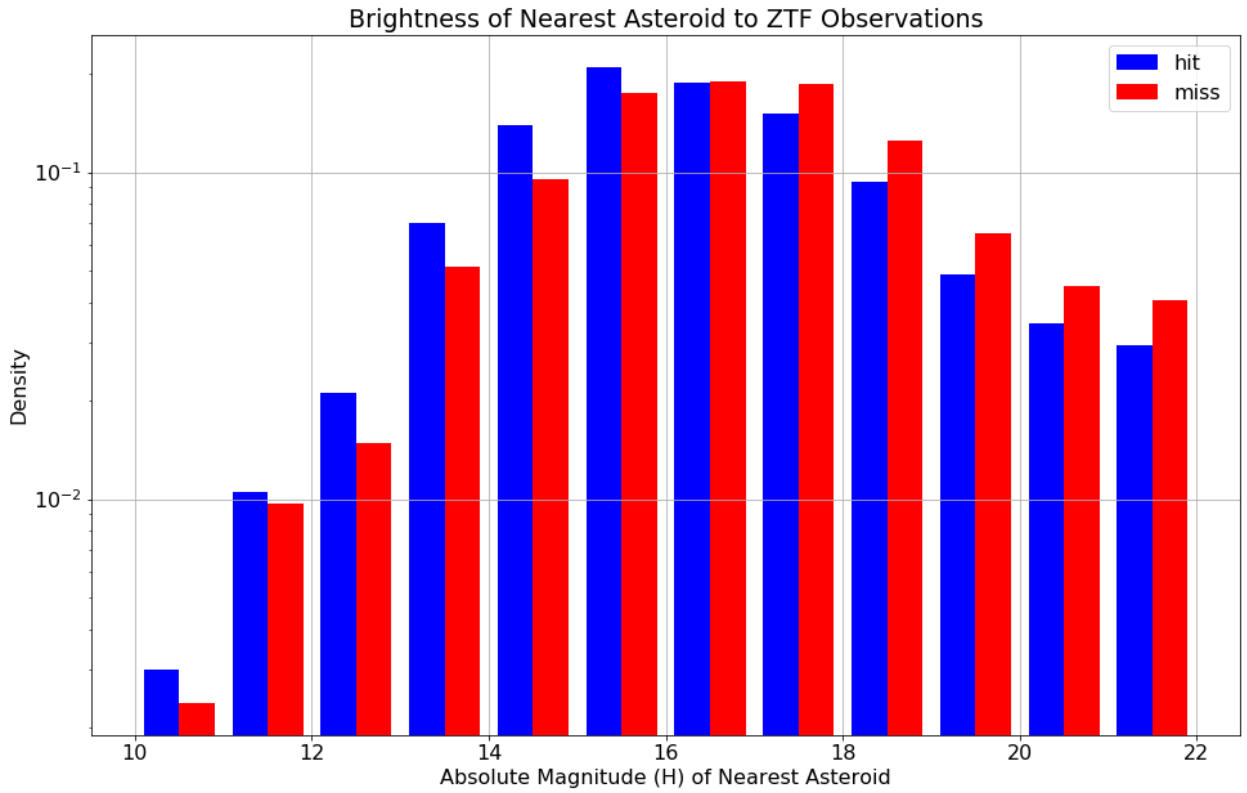

Figure 3.6: *Magnitude H of Nearest Asteroid to Each Detection*
Hits are shown in blue, misses in red; a hit is a detection within 2.0 arc seconds of its expected direction. The hits are slightly but noticeably tilted in favor of brighter asteroids.

I would like to take a step back and review what has been presented thus far. Over 4 billion interactions between a ZTF asteroid detection and the predicted position of a known asteroid in the sky have been generated. These have been filtered to associate each ZTF detection with the known asteroid it is nearest to. This is a powerful enrichment of the original ZTF dataset, and

might open the door to some additional work in the future. For example, it could be used to create a bulk data set linking the original image files to the asteroids they belong to. This could in turn be used to refine the machine learning pipeline used to classify detections and guess when they belong to the same object.

3.5 Analyzing the Distribution of Distance to the Nearest Asteroid

In this section I explore the statistical distribution of the Cartesian distance between observations and the nearest asteroid. I compare the observed distribution of this distance to the theoretical distribution we would obtain if either our observed or predicted directions were distributed uniformly at random on the sphere.

Suppose without loss of generality that the observed direction is at the north pole, i.e. $\mathbf{u}_{\text{obs}} = (0, 0, 1)$. Suppose that the direction we predict is (x, y, z). Observe that the problem is symmetric about the z axis, so we can rotate the problem into the plane containing the center, the north pole, and our guess. Let $r^2 = x^2 + y^2$ be the squared distance from the z axis to our guess. This configuration is shown in the diagram above. We can can relate the Cartesian distance s to the height z of our guess with the following simple observations.
(x, y, z) lies on the surface of a sphere, so $x^2 + y^2 + z^2 = 1$.
$r^2 = x^2 + y^2$ by definition, so $r^2 = 1 - z^2$.
In our diagram, we can see that s is the hypotenuse of a right triangle whose other two side have length r and $1 - z$. Applying the Pythagorean Theorem, we find $s^2 = (1 - z)^2 + r^s$.
This simplifies to

$$s^2 = 2(1 - z)$$
$$z = 1 - \frac{s^2}{2}$$

It turns out that this is a very useful parameterization because there is an elegant parameterization of the differential solid angle $d\Omega$ in terms of dz. When surface integrals of a sphere are taught in introductory multivariate calculus courses, students are most likely to be exposed only to the surface element in spherical coordinates (r, θ, ϕ) which is $d\Omega = \sin\theta d\theta d\phi$. See, e.g. Wikipedia - integration in spherical coordinates. But the parameterization in terms of the height z along the z

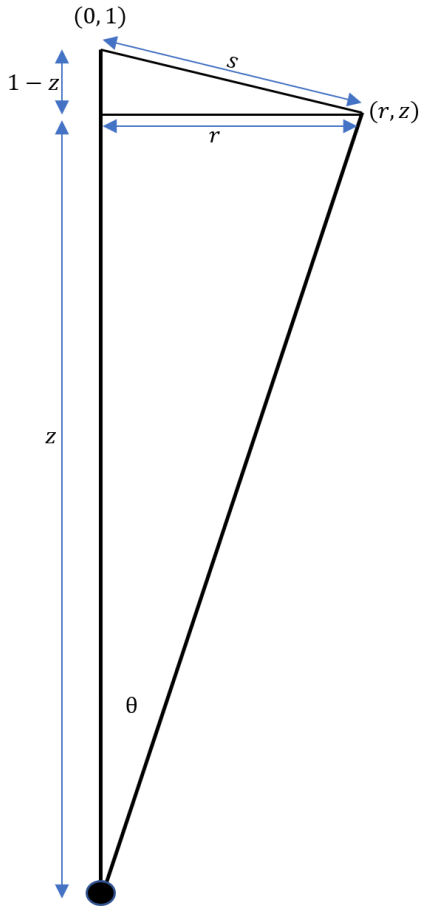

Figure 3.7: *Distance between an observation at the north pole and an arbitrary point.*
The observed direction $\mathbf{u}_{\text{obs}}$ *is assumed w.l.o.g. to be at the north pole* $(0, 0, 1)$.
The predicted direction (x, y, z) *is rotated in the xy plane to to* $(r, 0, z)$
The Cartesian distance to from the observation to (r, z) *is s, the hypotenuse of a right triangle with sides* $1 - z$ *and* r.

axis is especially clean and convenient on this problem.

Observe that $z = \cos\theta$ so $dz = |-\sin\theta|d\theta = \sin\theta d\theta$. We take absolute values because this is an application of the change of variables formula and measures are always positive.

Substituting this expression for the surface element, we obtain

$$d\Omega = dz \cdot d\phi$$

I call this result the "orange slicing theorem." It tells you that if you slice an orange into horizontal slices, the amount of rind on each slice will be equal to 2π times the height of the slice.

As a quick sanity check of this result, let's see if we can recover that the surface area of the unit sphere is 4π:

$$A = \int_{z=-1}^{1} \int_{\phi=0}^{2\pi} d\phi dz = 2\pi \int_{z=-1}^{1} dz = 4\pi$$

We can now write the probability density function (PDF) for any function that can be expressed in terms of z. Think of Z as a random variable now. The above result shows that when Z is uniformly distributed on the unit sphere,

$$Z \sim \text{Unif}(-1, 1)$$

Previously we showed that $z = 1 - s^2/2$. This is a very useful result; it tells us that if we treat the squared distance as a random variable S^2, then it is uniformly distributed on $[0, 2]$. Even more usefully, the conditional distribution of S^2, conditioned on $S^2 \leq \tau^2$, is also uniform:

$$S^2 | S^2 \leq \tau^2 \sim \text{Unif}(0, \tau^2)$$

If we apply a threshold distance τ and only consider predicted directions that are within Cartesian distance τ of an observation, then the condtional distribution of the relative distance over the threshold squared is uniform on $[0, 1]$. In mathematical notation instead of words,
Let $\mathbf{u}_{\text{pred}}$ be a random variable distributed uniformly on the unit sphere.
Let $S = \|\mathbf{u}_{\text{pred}} - \mathbf{u}_{\text{obs}}\|$ be the Cartesian distance between $\mathbf{u}_{\text{pred}}$ and $\mathbf{u}_{\text{obs}}$.
Let τ be a threshold distance in [0, 2].
Let $V = S^2/\tau^2$ be the relative squared distance of on observation vs. the threshold.
Then the conditional distribution of V, conditional on $S < \tau$ (equivalently $V < 1$) is

This describes the conditional distribution of distances we would see if we guessed one random direction in the sky. But in this experiment, we are picking 733,489 directions in the sky, one for each of the catalogued asteroids. Then we are taking the minimum of these distances. Can we still write down the conditional distribution of our nearest guess if they were indepedently and identically distributed (i.i.d.) at random as above?

Yes - Statistics 110 to the rescue! Theorem 8.6.4: PDF of Order Statistics [4] states that if $X_1, \ldots X_n$ are i.i.d. continuous random variables with PDF f and CDF F, then the PDF of the jth

order statistic (the jth smallest item $X_{(j)}$) is

$$f_{X_j})(x) = n \cdot \binom{n-1}{j-1} f(x) \cdot F(x)^{j-1} \cdot (1 - F(x))^{n-j}$$

In the special case that the X_j are uniforms, this simplifies further (Example 8.6.5: Order statistics of Uniforms) [4]:

Let $U_1, \ldots U_n$ be i.i.d. Unif$(0, 1)$. Then the distribution of U_j is the Beta distribution,

$$U_{(j)} \sim \text{Beta}(j, n - j + 1)$$

The minimum is the order statistic of $j = 1$, so

$$U_{(1)} \sim \text{Beta}(1, n)$$

Let us now apply this theoretical result to compare our distribution of distances to what we would have obtained if we were randomly throwing darts into the sky so to speak. The module `ztf_data_viz` includes these calculations as well as the generation of charts. `cdf_nearest_dist` computes the theoretical CDF using the approach described above. In the charts below, I will demonstrate that 2.0 arc seconds is a good threshold for classifying detections as "hits" against known asteroids. For now, please treat it as a parameter I have chosen to demonstrate that these calculations work and are getting astronomically more hits (no pun intended) than would arise from random chance.

Out of 5.69 million detections, 3.75 million (65.71%) are within 2.0 arc seconds of the nearest catalogued asteroid. The theoretical beta distribution says that if the predicted directions were distributed uniformly at random, we would expect only 98 hits or 0.0017% of the total be this close. The immediate conclusion is that this whole set of calculations is working with a tolerance no worse than 2.0 arc seconds.

We can get a better intuition for what's happening by visualizing the histogram of distance to the nearest asteroid. I initially plotted these against the percentile of the theoretical distribution; these plots would be a flat line with density 1 if the predicted directions were uniformly at random. I found however that a simple plot of frequency vs. distance in arc seconds is easier to interpret once you've done the statistical analysis that the results are not due to chance. Here is a plot showing hits inside a threshold of 2.0 arc seconds:

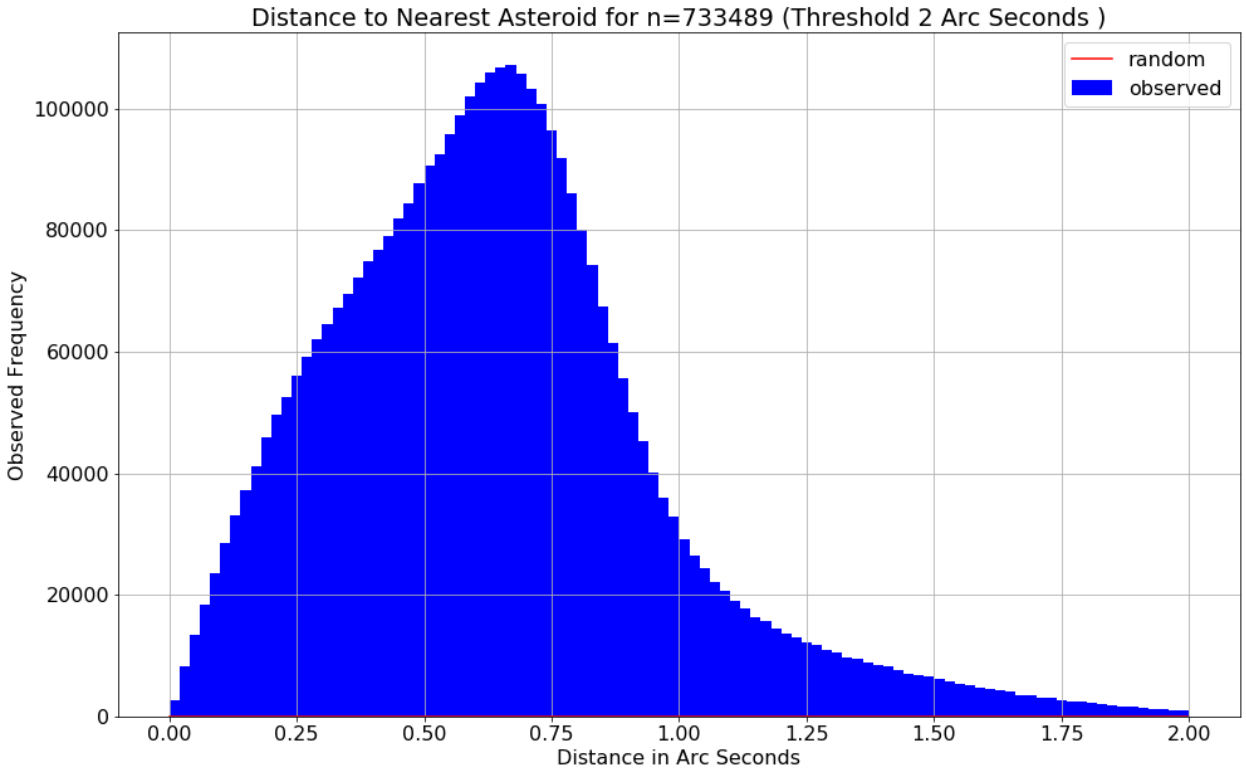

Figure 3.8: *Histogram of angular distance from each ZTF detection to the nearest asteroid in arc seconds. These account for 65.57% of the total data set.*
The number of expected detections due to random chance is shown in red; it is visually indistinguishable from zero on the chart.

Here are two additional plot to visualize the distribution of distances between ZTF detections and the nearest known asteroid. The first shows the absolute density in hits per square degree. The second show the relative density of hits per square degree over what would have been expected from the minimum of 733,489 guesses distributed uniformly at random on the sphere. These charts seemed to have a shape that could be roughly approximated in a one parameter distribution as exponential. They led me to propose a mixture model and log likelihood optimization objective function that I will describe in the next section

3.6 Conclusion

I have presented in this chapter an analysis of the asteroid detections in the ZTF astronomical data set. I have demonstrated a calculation of the nearest known asteroid to each ZTF detection

and the angular distance from that asteroid to the detection. I have also developed the statistical distribution of distances that would be observed if the predicted directions were distributed uniformly at random on the sphere. I have shown that of 5.7 million total ZTF detections, 65.7% of them (almost two thirds) are within 2.0 arc seconds of the predicted direction of the nearest known asteroid. By comparing this number of hits to the number we would expect with a random baseline, I have provided overwheliming evidence that this entire apparatus of data and calculations is accurate to a tolerance on the order of 2.0 arc seconds or better. Finally, I have provided motivation that the approximate shape of the distribution of distances decays with an exponential tail.

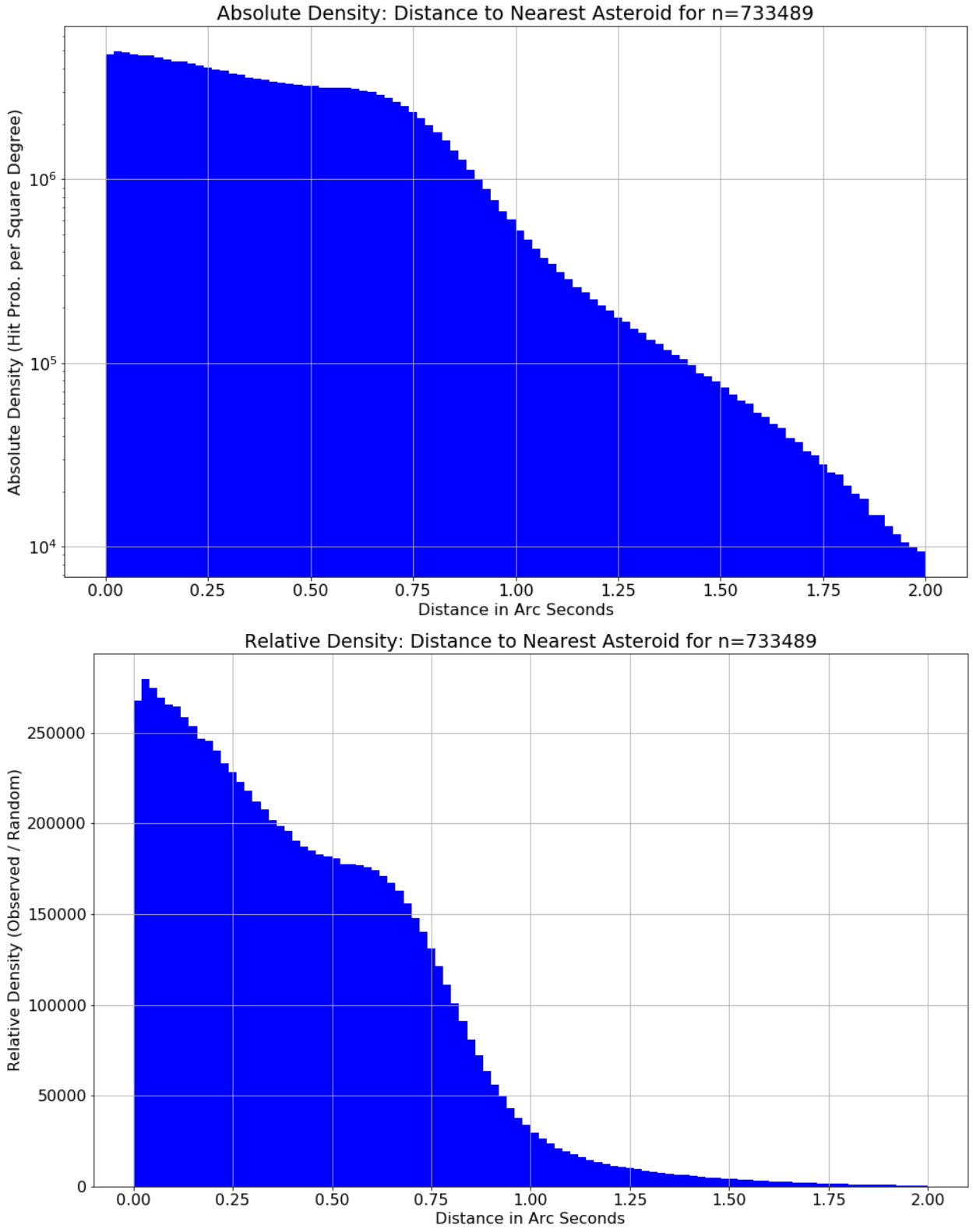

Figure 3.9: *Density of hits for distance between ZTF detections and the nearest known asteroid.*
The first chart shows the absolute density in hits per square degree.
The second chart shows the relative density of this over the baseline if the guesses were distributed randomly.

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

[4] Joseph Blitzstein, Jessica Hwang
Introduction to Probability
CRC Press. 2019 (Second Edition)