Kepler's Sieve:
Learning Asteroid Orbits from Telescopic Observations

A dissertation presented

by

Michael S. Emanuel

to

The Institute for Applied Computational Science

in partial fulfillment of the requirements

for the degree of

Master of Science

in the subject of

Data Science

Harvard University

Cambridge, Massachusetts

May 2020

Dissertation Advisors:
Professor Pavlos Protopapas
Professor Christopher H. Rycroft

Author:
Michael S. Emanuel

Kepler's Sieve:
Learning Asteroid Orbits from Telescopic Observations

Abstract

A novel method is presented to learn the orbits of asteroids from a large data set of telescopic observations. The problem is formulated as a search over the six dimensional space of Keplerian orbital elements. Candidate orbital elements are initialized 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 9 new asteroid candidates with at least 8 hits within 10 arc seconds on ZTF asteroid detections.

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

Contents

Abstract . . . iii
Acknowledgments . . . ix

Introduction **1**

1 Integrating the Solar System **8**
1.1 Introduction . . . 8
1.2 The REBOUND Library for Gravitational Integration . . . 9
1.3 A Brief Review of the Keplerian Orbital Elements . . . 11
1.4 Numerical Integration of the Planets in REBOUND . . . 14
1.5 Efficient Integration of All Known Asteroids . . . 18
1.6 Integration of the Kepler Two Body Problem in TensorFlow . . . 26
1.7 Conclusion . . . 34

2 Predicting Directions from Positions **36**
2.1 Introduction . . . 36
2.2 A Brief Review of Right Ascension (RA) and Declination (DEC) . . . 37
2.3 Mapping Between RA/Dec and Direction $\vec{u}$ in the Ecliptic Frame . . . 40
2.4 Computing a Direction $\vec{u}$ from Position $\vec{q}$ and Velocity $\vec{v}$ . . . 42
2.5 Predicting the Apparent Magnitude of an Asteroid . . . 47
2.6 Conclusion . . . 47

3 Analysis of ZTF Asteroid Detections **48**
3.1 Introduction . . . 48
3.2 Exploratory Data Analysis of ZTF Asteroid Data . . . 49
3.3 The Angular Distance Bewteen two Directions $\vec{u}_1$ and $\vec{u}_2$ . . . 51
3.4 Finding the Nearest Asteroid to Each ZTF Detection . . . 52
3.5 Analyzing the Distribution of the Distance to the Nearest Asteroid . . . 55
3.6 Conclusion . . . 60

4 Asteroid Search Using Orbital Elements **63**
4.1 Introduction . . . 63
4.2 Generating Candidate Orbital Elements . . . 65

4.3 Assembling ZTF Detections Near Candidate Elements . . . 70
4.4 Filtering the Best Random Elements . . . 73
4.5 Formulating the Log Likelihood Objective Function . . . 74
4.5.1 Adding the Magnitude to the Joint Likelihood Function . . . 77
4.6 Performing the Asteroid Search . . . 78
4.6.1 Controlling Parameters on a Uniform Scale . . . 78
4.6.2 Gradient Clipping by Norm . . . 79
4.6.3 Scoring Trajectories: Log Likelihood and Hits by Candidate Element . . . 80
4.6.4 Training Each Candidate Element Independently . . . 81
4.6.5 Training in "Mixture" and "Joint" Modes . . . 83
4.6.6 Modifying the Objective Function to Encourage Convergence . . . 83
4.6.7 Organizing Training: Batches, Epochs, Episodes, and Sieving Rounds . . . 85
4.6.8 Performing the Asteroid Search: Summary . . . 87

5 Asteroid Search Results **88**
5.1 Comparing Candidate Elements to the Nearest Asteroid . . . 88
5.2 Recovering the Unperturbed Elements of Known Asteroids . . . 93
5.3 Recovering the Perturbed Elements of Known Asteroids . . . 96
5.3.1 Small Perturbation . . . 96
5.3.2 Large Perturbation . . . 100
5.4 Searching for Known Asteroids with Random Initializations . . . 103
5.5 Presenting 9 New Asteroid Candidates . . . 105
5.6 Conclusion . . . 108
5.7 Future Work . . . 109
5.7.1 Intelligent Initialization of Candidate Elements . . . 109
5.7.2 Incorporating Additional Data . . . 110
5.7.3 Incorporating Magnitude . . . 110
5.7.4 Rebuilding the Known Asteroid Catalogue . . . 110

References **112**

List of Tables

1.1 Root Mean Square Error in Integration of Planets vs. Horizons 16

2.1 Mean differences between JPL, SkyField, and my calculations of astrometric direction 46

List of Figures

1.1 Definition of the traditional Keplerian orbital elements . . . 12
1.2 Three Orbital Anomalies: Eccentric, Mean and True . . . 14
1.3 Position and Angle Error of 10 Test Asteroids Initialized from Horizons . . . 22
1.4 Orbital Elements of 733,489 asteroids downloaded from Horizons . . . 23
1.5 Position and Angle Error of 25 Test Asteroids . . . 24

2.1 The Topographic Coordinate System . . . 37
2.2 The Celestial Sphere . . . 38
2.3 The International Celestial Reference Frame (ICRF) . . . 40
2.4 The Heliocentric Ecliptic Frame . . . 42
2.5 Computing the Apparent Direction from Earth to an Asteroid . . . 43

3.1 ZTF Asteroid Detections per month . . . 49
3.2 Preview of Pandas DataFrame of ZTF Detections . . . 50
3.3 Apparent Magnitude of ZTF asteroid detections . . . 50
3.4 The Angular Distance Between Two Directions on the Unit Sphere . . . 51
3.5 Preview of Pandas DataFrame of ZTF Detections Including Nearest Asteroid . . . 53
3.6 Magnitude H of Nearest Asteroid to Each Detection . . . 54
3.7 Distance Between an Observation at the North Pole and an Arbitrary Point . . . 56
3.8 Histogram of Angular Distance from Each ZTF Detection to the Nearest Asteroid . . . 59
3.9 Density of Hits for Distance Between ZTF Detections and the Nearest Known Asteroid 61
3.10 Cumulative Frequency of Asteroids by Number of Close Observations < 2.0 arc seconds . . . 62

4.1 PDF and CDF for $\log(a)$, Log of the Semi-Major Axis. . . . 66
4.2 PDF for Eccentricity e and $\sin(i)$ (Sine of the Inclination). . . . 67
4.3 PDF for Longitude of Ascending Node Ω and True Anomaly f . . . 68
4.4 PDF for Argument of Perihelion ω and Mean Anomaly M . . . 69
4.5 ZTF Detections Within a 1.0 Degree Threshold of a Batch of 64 Orbital Elements . . . 71
4.6 Histogram of $v = (s/\tau)^2$ for Three Sets of Candidate Orbital Elements . . . 72

5.1 Transformations of a and e to Standardized Variables a_z and e_z . . . 90
5.2 Transformations of $\sin(i)$ and $\sin(\Omega)$ to Standardized Variables. . . . 92

5.3 Training progress on 64 unperturbed orbital elements 95
5.4 The resolution R decreases monotonically when training the unperturbed elements 95
5.5 Two Metrics Comparing the Recovered Orbital Elements to True Elements 98
5.6 Training Progress on 64 Orbital Elements Initialized with Small Perturbations . . . 99
5.7 [Training Progress on 64 Orbital Elements Initialized with Large Perturbations . . . 102
5.8 125 Orbital Elements Fitted to Observations of Known Asteroids 104
5.9 Orbital elements of 9 candidate asteroids. 105
5.10 9 Orbital Elements Fitted to Observations of Unknown Asteroids 106
5.11 ZTF Hits Associated with 4 of the New Asteroid Candidate Elements 108

Acknowledgments

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

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

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

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

To my two children, Victor and Renée

Introduction

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

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

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

[1] Wikipedia - Oumuamua

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

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

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

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

[3] Wikipedia - Square Degrees in the Sky

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

This is the principal motivation for searching in the space of orbital elements. While it's a large 6 dimensional space, its size is fixed in relation to the amount of data we have. To be more precise, the number of candidate orbital elements will scale with the number of **asteroids** K we are trying to detect rather than the much larger number of **detections** N in our data set. The cost of the search algorithm presented below scales linearly in the observation density ρ for each candidate element analyzed. The cost of the entire algorithm is therefore on the order of $N \cdot K$, with no explosion as you consider tracklets larger than 2. The second major reason for searching in the space of orbital elements is that it permits the search algorithm to string together observations made far apart in time. This is a capability that eludes searches based on tracklets.

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

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

the predicted and observed direction is computed. A threshold (2.0 degrees) is applied, and all ZTF observations falling within this threshold are cached in memory of the search class.

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

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

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

parameters. The gradients will encourage the model to adjust the candidate orbital elements so that some of the observations, the ones it sees as highly probable hits, will be very close to what is predicted by the candidate elements. The observations modeled as highly probable misses will hardly contribute to the gradients of the candidate elements.

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

As much as possible, I have sought to validate individual components of these calculations in isolation. My numerical integration of the planets is validated against results from NASA JPL (Jet Propulsion Library) using the superb Horizons system. [4] I separately validated the numerical integration of the first 16 asteroids against positions and velocities obtained from Horizons.

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

The end to end calculation of a direction from orbital elements was verified indirectly as follows. I integrated the trajectories of all the known asteroids using a collection of orbital elements downloaded from JPL. I then computed the nearest asteroid number to each ZTF asteroid, and the distance between the predicted direction and observed direction. I reviewed the statistical distribution of these distances. I observed that out of approximately 5.69 million

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

observations from the ZTF dataset, 3.75 million (65.71%) fall within 2.0 arc seconds of the predicted directions of known asteroids. I took this as overwhelming evidence that these calculations were accurate.

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

I tested the capabilities of the search process with an increasingly demanding set of search tasks. The first three search tasks involved recovering the elements of known asteroids. I took a batch of 64 asteroids that appeared most frequently in the ZTF data set. These asteroids were represented between 148 and 194 times in the data, where hits here are counted at a threshold of 2.0 arc seconds as before. Here is a summary of the tests I ran:

- Initialize search with correct orbital elements, but resolution $R = 0.5°$ and threshold $\tau = 2.0°$. All 64 elements were recovered to a resolution of 3.0 arc seconds and 4.6E-6 AU, matching on 162 hits.

- Initialize search with small perturbation applied to orbital elements; a by 1.0%, e by 0.25%, i by 0.05°, remaining angles f, Ω and ω by 0.25°. 42 of 64 elements were recovered to 18 arc seconds and 2.6E-4 AU, matching on 118 hits

- Initialize search with large perturbation applied to orbital elements; a by 5.0%, e by 1.0%, i by 0.25°, remaining angles f, Ω and ω by 1.0°. 12 of 64 elements were recoverd to 32 arc seconds and 4.5E-4 AU, matching on 98 hits. In some cases, a different (but correct) set of orbital elements was obtained; the perturbation was so large the search found a different asteroid.

[5] Discussion with Pavlos Protopapas

- Initialize a search with **randomly initialized** orbital elements. Search against the subset of ZTF observations within 2.0 arc seconds of a known asteroid. This search converged on one set of orbital elements matching a real asteroid on the first batch of 64 random candidates.

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

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

The main thrust of this work, however, is not on refining the existing asteroid catalogue, it is finding new asteroids. The final search I ran was against the subset of ZTF observations that did not match any of the known asteroids. Random initializations for orbital elements were tried. Most of these initializations fail to converge on elements with enough hits to match real asteroids in the data, but a small number do successfully converge. I have identified 9 candidate elements for potentially new asteroids with 8 or more hits. I have verified that none of the orbital elements modeled for these asteroids are obvious matches in the known asteroid catalogue, though some are arguably close. I have also done an ad-hoc review of the ZTF records to ensure that they are plausibly belonging to the same object. I believe that at least some of these candidate elements may represent new and unkown asteroids, and plan to submit them to the Minor Planet Center for possible classification.

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

Chapter 1

Integrating the Solar System

1.1 Introduction

The calculation of planetary orbits is arguably the canonical problem in mathematical physics. Isaac Newton invented differential calculus 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 the combined planetary mass, [1] 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] Jupiter has a mass of 9.55E-4 of the Sun, and the Sun accounts for 99.8% of the Solar System mass; see Wikipedia - Solar System objects

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 at github.com/hannorein/rebound. 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 thesis, 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. [2]

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.

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

Rein and Spiegel call them substeps. The formula is rearranged in terms of new coefficients g_k 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 Jacobi coordinates in the N-body gravitational problem: 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

elements as of MJD 58600.0, corresponding to 27-Apr-2019 on the Gregorian calendar.

Figure 1.1 illustrates the six orbital elements.

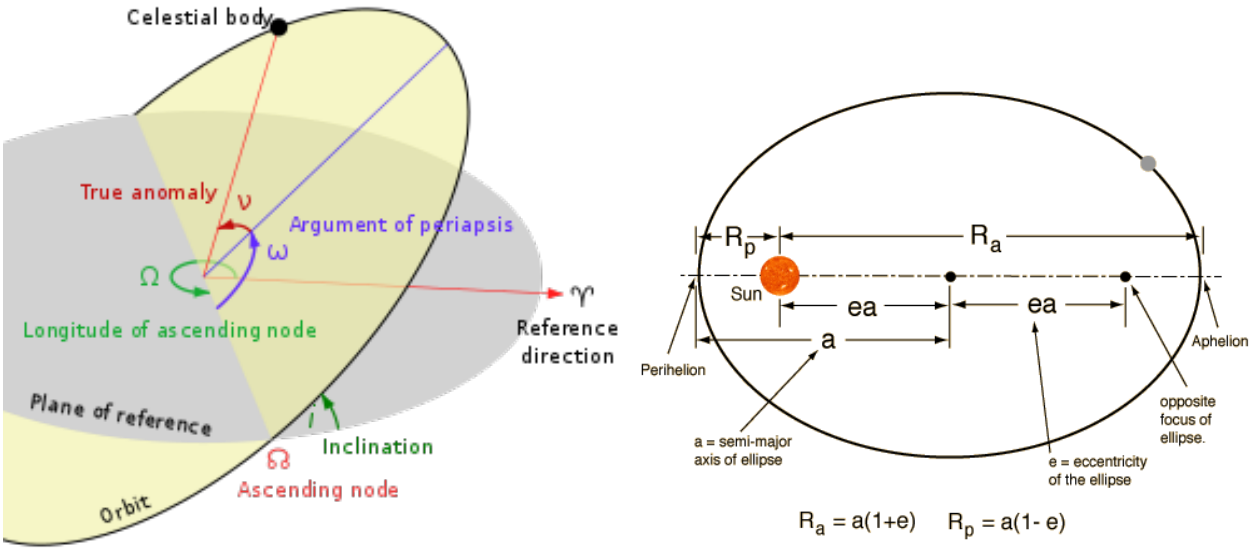

Figure 1.1: *Definition of the traditional Keplerian orbital elements, courtesy of Wikipedia and planetary.org.*

The names and conventional symbols for Keplerian elements are:

- a, the semi-major axis; a distance
- e, the eccentricity; dimensionless
- i, the inclination; an angle
- Ω, the longitude of the ascending node; an angle
- ω, the argument of perihelion; an angle
- f, the true anomaly; an angle
- M, the mean anomaly; an angle
- t_0, the epoch as a Modified Julian Date

The parameters a and e define the size and shape of the orbital ellipse. The three angles i, Ω, and ω define its orientation in a three dimensional reference frame. The true anomaly f or mean anomaly M define where the orbiting body is in its orbital ellipse as of the epoch.

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 many computations, both theoretical and numerical, because in the case of the two body problem, five of the six orbital elements remain constant. Perhaps equally useful, in an N-body system with a centrally dominant mass including our Solar System, the first five elements change very slowly, while the mean anomaly changes approximately linearly in time. These two properties make orbital elements ideal for splining interpolated positions and velocities. This approach is used in the generation of empherides from numerical integrations of the Solar System.

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 as of 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 advantages. 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 eccentric anomaly E is yet another angle describing a body in orbit. Figure **??** depicts the three orbital anomalies. The mean anomaly and eccentric anomalies are related by the famous

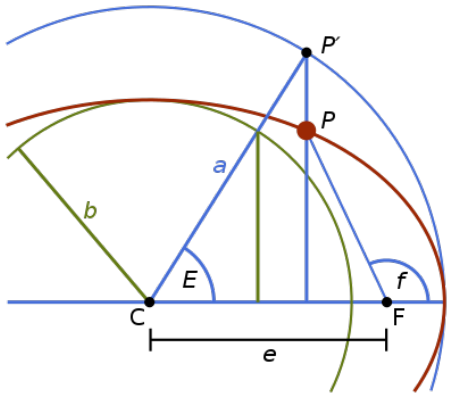

Figure 1.2: *Three Orbital Anomalies: Eccentric, Mean and True*

Kepler's Equation, shown below along with the transformation between E and f

$$M = E - e\sin(E) \qquad \text{Kepler's Equation}$$

$$\tan\left(\frac{f}{2}\right) = \sqrt{\frac{1+e}{1-e}} \cdot \tan\left(\frac{E}{2}\right) \qquad \text{true to eccentric}$$

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. Data for the initial conditions can obtained from Horizons, a service provided by the Jet Propulsion Laboratory (JPL) at NASA that is free to the public. 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 must pay attention to units and the frame of

reference. You also 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 API. 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 [3]

- **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, Tethys, 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):

[3] Masses of Solar System objects were obtained from Wikipedia - Solar System objects.

Object Collection	Position Error AU	Angle Error ArcSec
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

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 IAU 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 2040-12-31 and a time step of 16 days. I tested the integration by comparing the predicted positions of the 8 planets to the position quoted by Horizons at a series of test dates. The test dates are at 1 year intervals over the full 40 year span that is simulated. The best results were obtained by integrating smallest collection: Earth, Moon, and the barycenters of the other 7 planets. I was a bit surprised at this result and expected to do slightly better as the collection of objects became larger. Position errors are reported in AUs, with the root mean square (RMS) error over the 40 annual dates. I 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. [4]

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

[4]The arc second is an angular unit dating back to ancient Babylonian astronomers. It is part of the sexagesimal system that includes the more familiar degree. One arc second is defined as 1/3600 of one degree or equivalently as $\pi/648000$ radians.

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. [5] The angular precision on the order of ~ 0.8 arc seconds is also excellent for such a long time span and well within the tolerance of this application.

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

The second test of the integration extends the bodies under consideration from the planets to include ten test asteroids. I selected as the test asteroids the first 10 IAU numbered asteroids: Ceres, Pallas, Juno, Vesta, Iris, Hygiea, Egeria, Eunomia, Psyche, Fortuna. This test does not yet exercise the part of the code that 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.

Figure 1.3 summarizes 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
```

[5] 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`.

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

1.5 Efficient Integration of All Known Asteroids

In the previous section I have described how to integrate the Sun and Planets where the initial conditions are obtained from Horizons using the API included in `REBOUND`. While this is a reliable and simple procedure that is ideal if you have only a few bodies to integrate, there is too much overhead in querying the Horizons API for it to be feasible for integrating all known asteroids. (If you spent 1 second on each of the 733,489asteroids 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.) Horizons includes two data files available at 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 functions 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 about 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 [6]
- **G** is the G parameter (sensitivity of brightness to phase angle)
- **Ref** is the name of a JPL reference integration

Here are the asteroid orbital elements as a Pandas DataFrame:

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 `asteroid_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

[6] BAA H-G Magnitude System

center of mass of all the massive bodies in the simulation that have been entered prior. This leads to a subtle error on the order of 1E-3 that still wreaks havoc at the precisions required here. Here are the most important lines of code to add each asteroid to the simulation:

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

The code to integrate all the known asteroids is in `asteroid_integrate.py`. At the risk of stating the obvious, when integrating asteroids, they are treated as massless "test particles." That is, they move under the influence of gravity from the massive particles in the simulation (here, the Sun, 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 forward and backward 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 `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 direct cubic spline interpolation of positions and velocities. [7] 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

[7] Splining orbital elements is more accurate and would allow for smaller splines, but I did not want to reinvent the wheel of building an ephemeris library.

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.

This test 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.

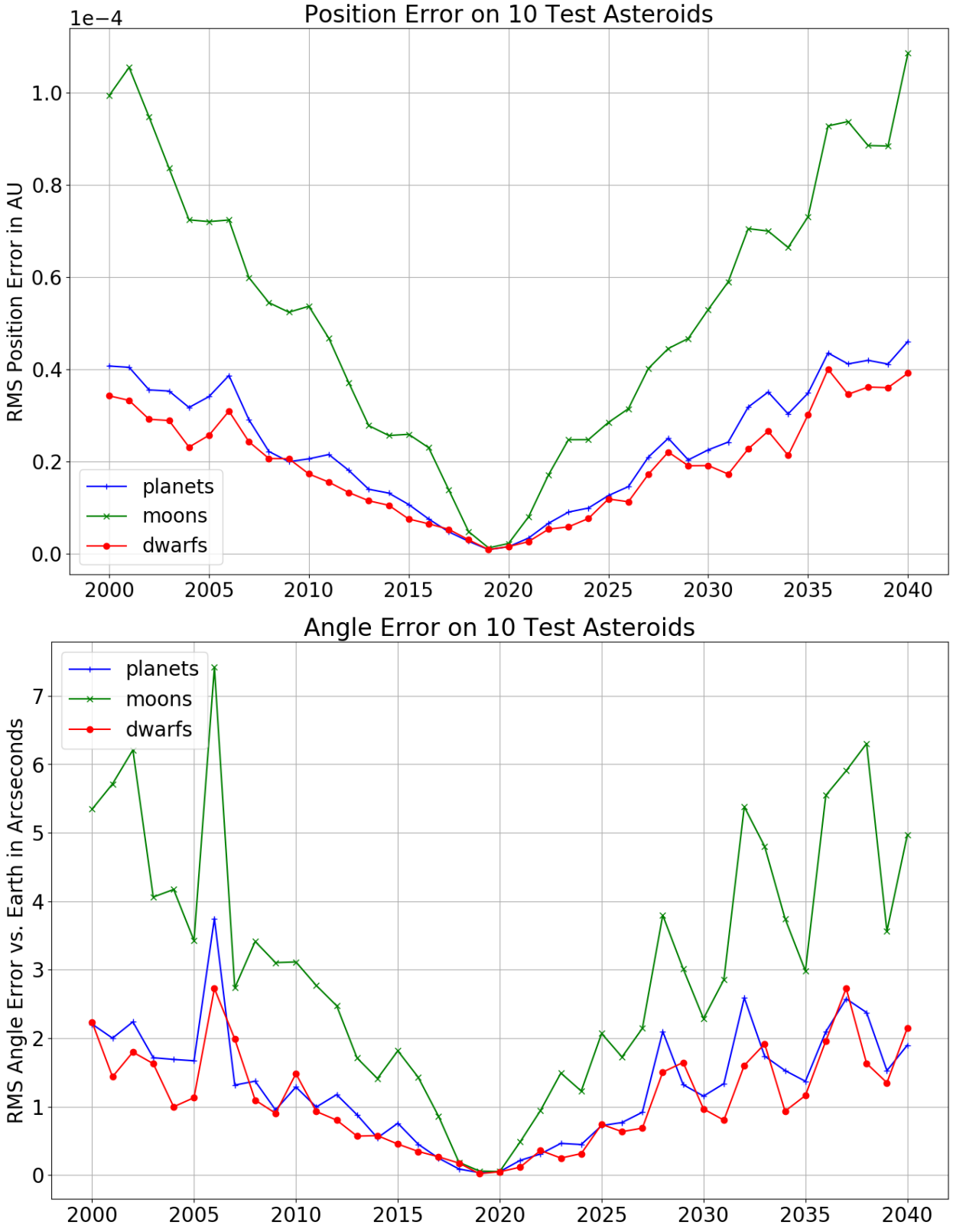

Figure 1.3: *Position and Angle Error of 10 Test Asteroids.*
My integration is compared to positions extracted from Horizons at 40 dates from from 2000 to 2040.
Initial conditions of the planets and asteroids are taken from the Horizons API as of MJD 58600 (2019-04-27).

```
# Load all the asteroid elements
ast_elt = load_ast_elt()
```

```
ast_elt
```

Num	Num	Name	epoch	a	e	inc	Omega	omega	M	H	G	Ref	f
1	1	Ceres	58600.0	2.769165	0.076009	0.184901	1.401596	1.284522	1.350398	3.34	0.12	JPL 46	1.501306
2	2	Pallas	58600.0	2.772466	0.230337	0.608007	3.020817	5.411373	1.041946	4.13	0.11	JPL 35	1.490912
3	3	Juno	58600.0	2.669150	0.256942	0.226699	2.964490	4.330836	0.609557	5.33	0.32	JPL 108	0.996719
4	4	Vesta	58600.0	2.361418	0.088721	0.124647	1.811840	2.630709	1.673106	3.20	0.32	JPL 34	-4.436417
5	5	Astraea	58600.0	2.574249	0.191095	0.093672	2.470978	6.260280	4.928221	6.85	0.15	JPL 108	-1.738676
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1255499	1255499	2019 QG	58600.0	0.822197	0.237862	0.220677	5.066979	3.770460	0.503214	21.55	0.15	JPL 1	0.807024
1255501	1255501	2019 QL	58600.0	2.722045	0.530676	0.113833	4.741919	2.351059	5.297173	19.21	0.15	JPL 1	-2.082964
1255502	1255502	2019 QQ	58600.0	1.053137	0.389091	0.172121	5.648270	2.028352	3.266522	25.31	0.15	JPL 1	-3.081905
1255513	1255513	6331 P-L	58600.0	2.334803	0.282830	0.141058	6.200287	0.091869	2.609695	18.50	0.15	JPL 8	2.827595
1255514	1255514	6344 P-L	58600.0	2.812944	0.664688	0.081955	3.199363	4.094863	2.738525	20.40	0.15	JPL 17	3.032066

733489 rows × 19 columns

Figure 1.4: *Orbital Elements of 733,489 asteroids downloaded from Horizons. I have calculated the true anomaly f in* `REBOUND`.

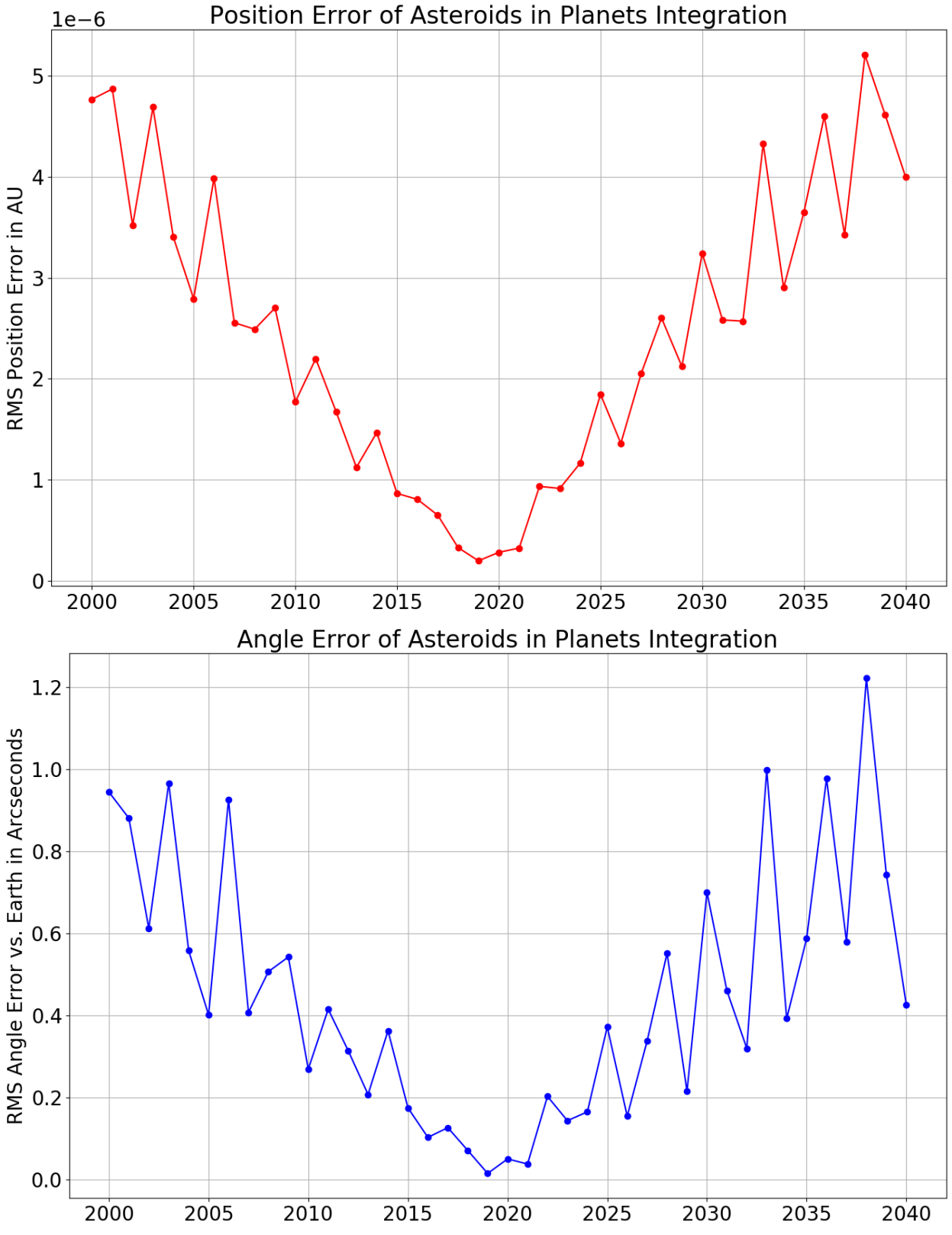

Figure 1.5: *Position and Angle Error of 25 Test Asteroids*
My integration is compared to positions extracted from Horizons at 40 dates from from 2000 to 2040.
Initial conditions of the planets are taken from the Horizons API as of MJD 58600 (2019-04-27).
Initial conditions of the asteroids are taken from orbital elements quoted in the Horizons small body elements file download as of MJD 58600.

I was 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. [8] 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

[8] The film Hidden Figures paints an evokative portrait of this era from the perspective of three black women whose contributions to the U.S. space program were only belatedly recognized.

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 Wikipedia - 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 infinitesimally 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 $\omega^2 r$. [9] 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

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

[9] 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.

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 $\omega = h \cdot r^{-2}$. 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 differential 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 $u = r^{-1}$. [10] 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. A more traditional description of an ellipse in polar coordinates [11] would be

$$r = \frac{a\cdot(1 - e^2)}{1 - e\cdot\cos(\theta - \theta_0)}$$

It can be verified that the solution above is equivalent to this one. 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

[10] One piece of physics intuition to motivate this change of variables is that the potential energy scales as r^{-1}.

[11] see e.g. Wikipedia - ellipse

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 that's not bad at all when you write it down using 3x3 rotation matrices with pencil and paper. Coding it efficiently on a computer to avoid duplicate calculations requires a bit of care. The

calculation of the inverse operation, from Cartesian coordinates to elements, is 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 with $e = 0$. 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 less familiar 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

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

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 standard (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 crude 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 `recalibrate`. 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 $d\mathbf{q}$ and $d\mathbf{v}$, 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.

Even before calibration, the position model is remarkably accurate. A test batch of 64 asteroids had a root mean square position adjustment $d\mathbf{q}$ of 9.85E-5 AU. The RMS velocity adjustment $d\mathbf{v}$ was 9.30E-7 AU / day. (This calculation is performed in the Jupter notebook `10_asteroid_model.ipynb`.

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, and this includes computing directions. 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 accurate 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 compute the direction $\mathbf{u}$ from which an observer at the Palomar observatory 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.

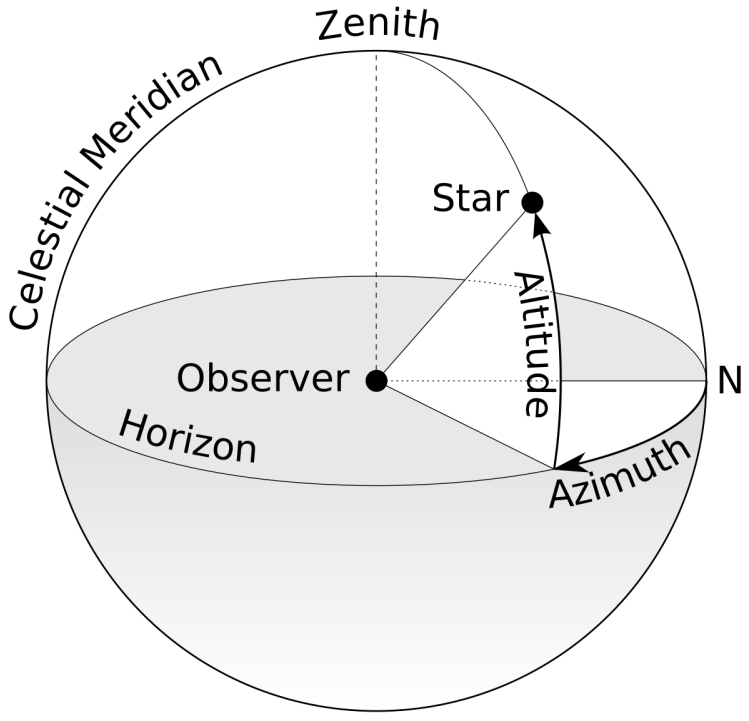

Figure 2.1: *The Topographic Coordinate System, courtesy of Wikipedia*

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:

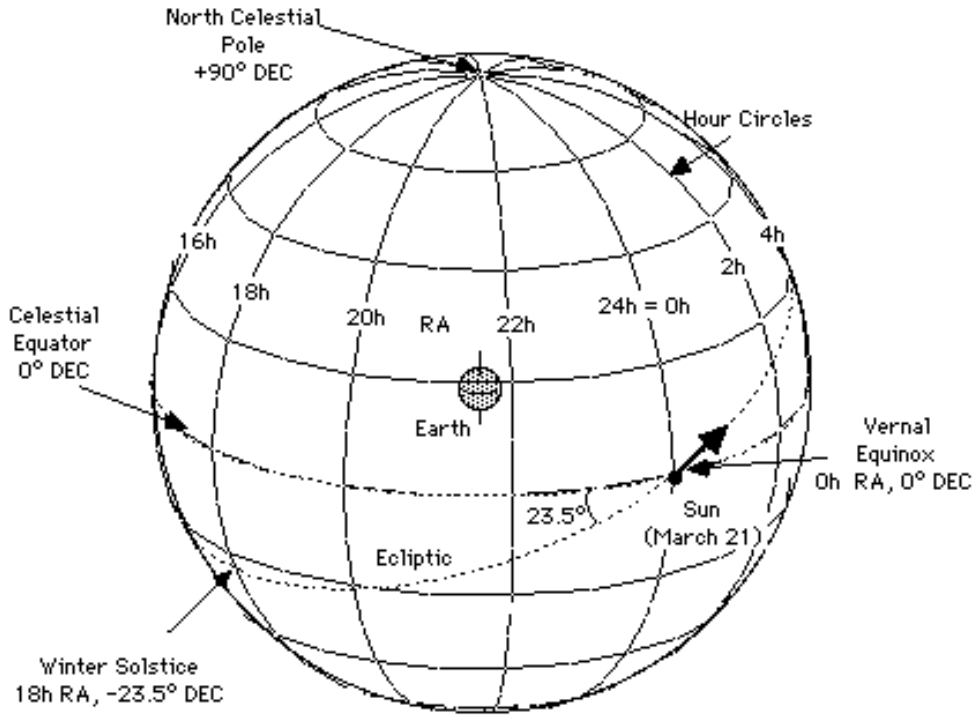

Figure 2.2: *The Celestial Sphere, courtesy of cool cosmos.*

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 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. Indeed, 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.

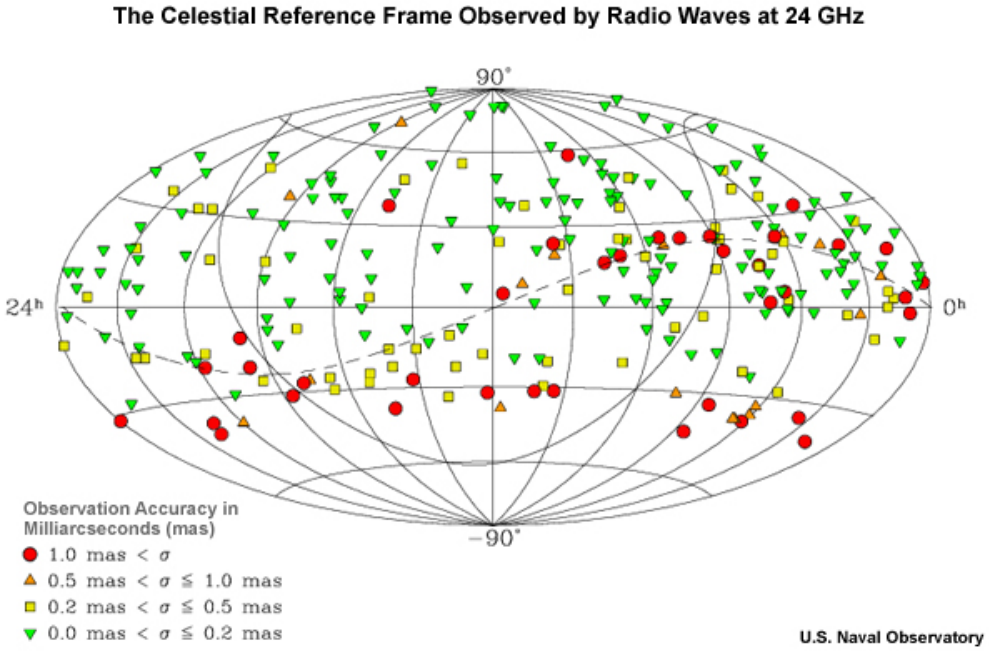

Figure 2.3: *The International Celestial Reference Frame (ICRF), courtesy of U.S. Naval Observatory.*

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 `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

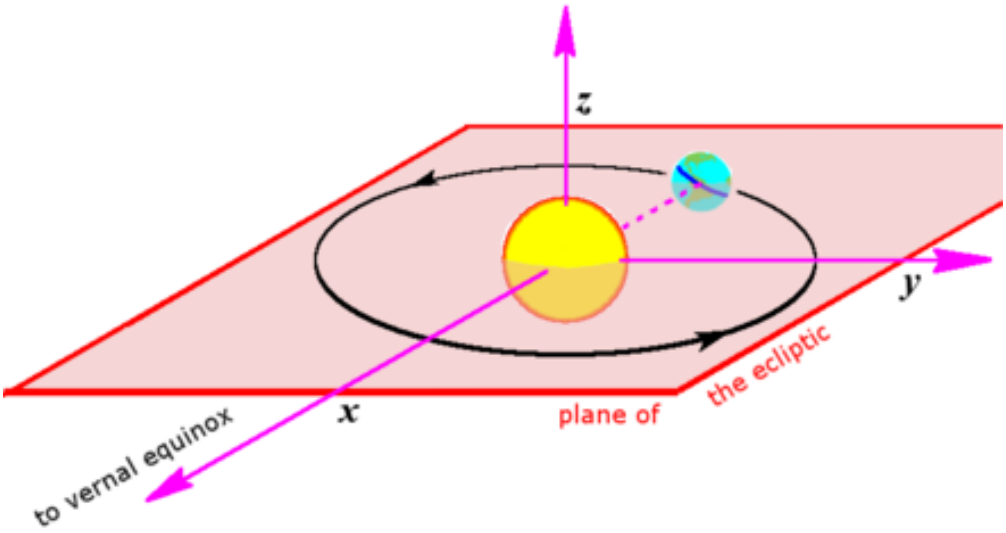

Figure 2.4: *The Heliocentric Ecliptic Frame, courtesy of Wikipedia*
The Barycentric Ecliptic Frame is analogous, but the origin is the Solar System barycenter rather than the Sun.

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

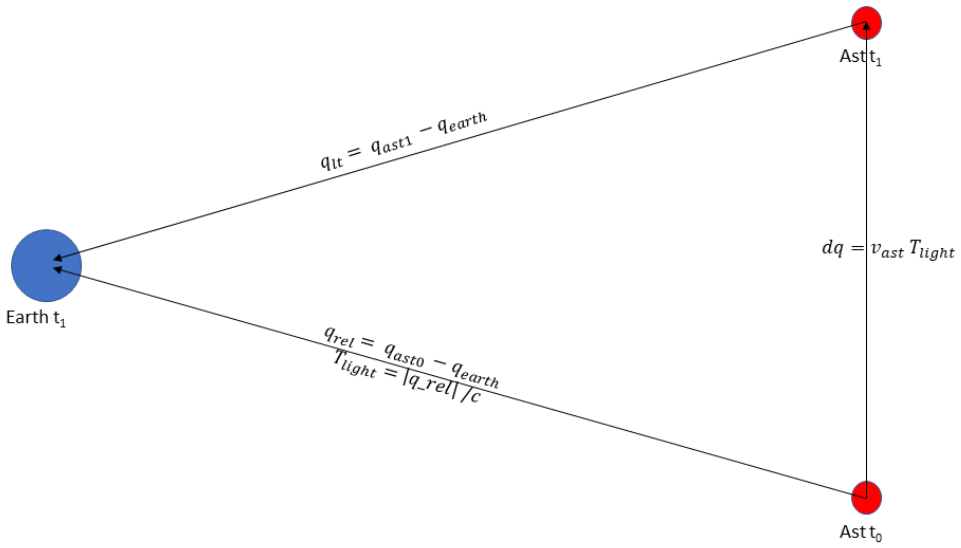

Figure 2.5: *Computing the Apparent Direction from Earth to an Asteroid, Accounting for the Speed of Light*

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 20-40 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 an 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 works 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 in this context.

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, elevation=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 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 `AsteroidDirection` layer 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) of astrometric direction 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 $\mathbb{R}^3$.

Table 2.1 summarizes 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 Asteroid

The H-G model of asteroid magnitude fits the apparent magnitude of an asteroid based on its position in the Solar System and two parameters. H is a measure of its absolute brightness, and G is how sensitive its aparent magnitude will be as a function of the "phase angle" between the Sun, Earth, and the asteroid. The principal term in the estimate is that the apparent magnitude of an object is proportional to its absolute magnitude (a function of its surface area and albedo); the distance from the asteroid to Sun; and the distance from the asteroid to the Earth. Light striking the asteroid from the Sun scales as r^{-2} and light reflecting from the asteroid to the Earth scales as Δ^{-2}. There is also a correction term involving the phase angle α.

I developed a prototype calculation and implemented this in TensorFlow as well. Unfortunately, when I incorporated the predicted magnitude into the log likelihood function used in the optimization process as explained below, the results got worse. I was forced to disregard the predicted magnitude durring the fitting process. A first pass of the code is already present in the `AsteroidModel` class, though, and I have identified this as an item of future work.

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, Horizons (NASA JPL) and the SkyField astronomy library. I have demonstrated that my results are within 1.0 arc second of JPL 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

The 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 sky, covering over 3750 square degrees an hour to a depth of 20.5 mag
- A machine learning pipeline has been developed to classify a subset of ZTF detections 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 object 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 million 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.

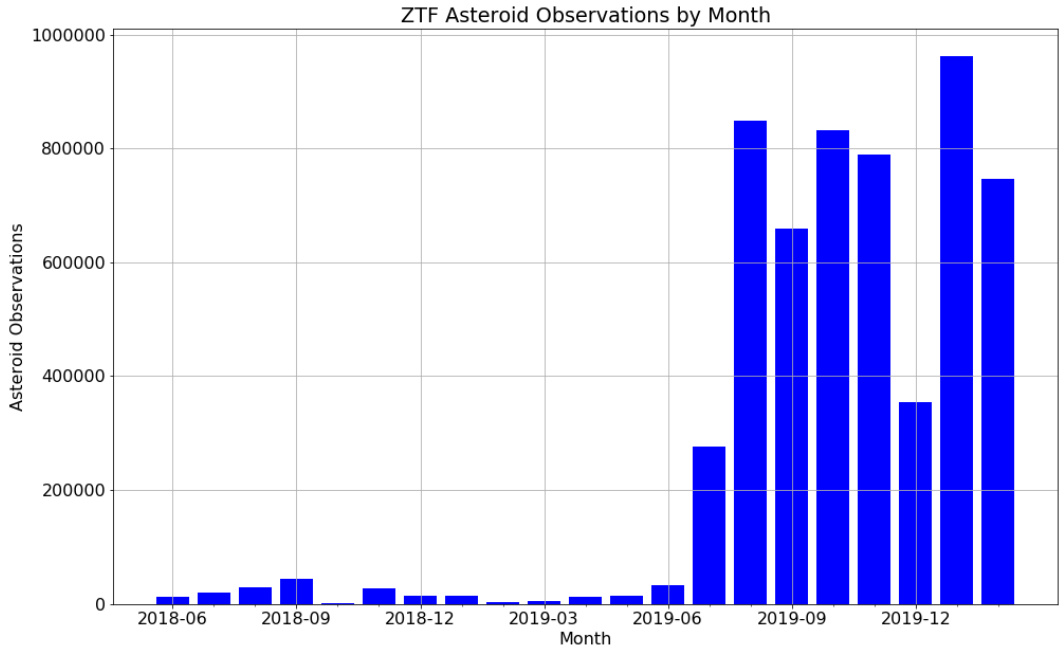

Figure 3.1: *ZTF Asteroid Detections per month*

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*

Figure 3.3 shows 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.

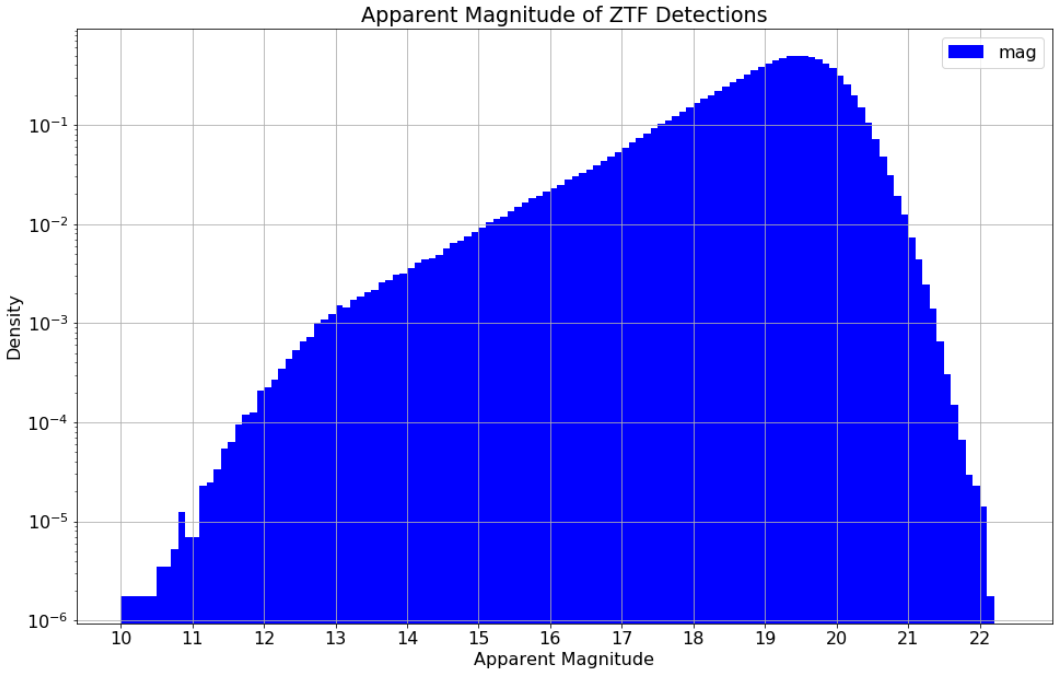

Figure 3.3: *Apparent Magnitude of ZTF asteroid detections.*

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. Figure 3.4 illustrates the derivation of the formula relating s and θ. The two directions are shown as arrows

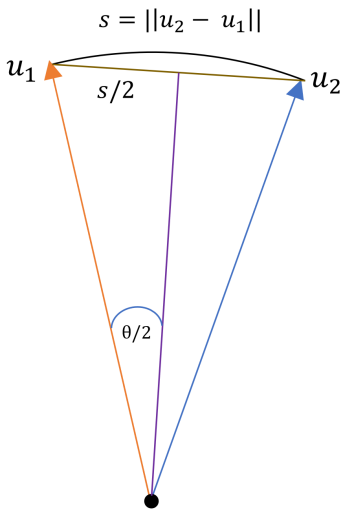

Figure 3.4: *The Angular Distance Between Two Directions on the Unit Sphere.*

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

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

This relationship 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 Detection

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 observation 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 performs 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 data frame 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 3 hour intervals.

	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. Dimmer detections are more likely to be either misclassifications or unkown objects.

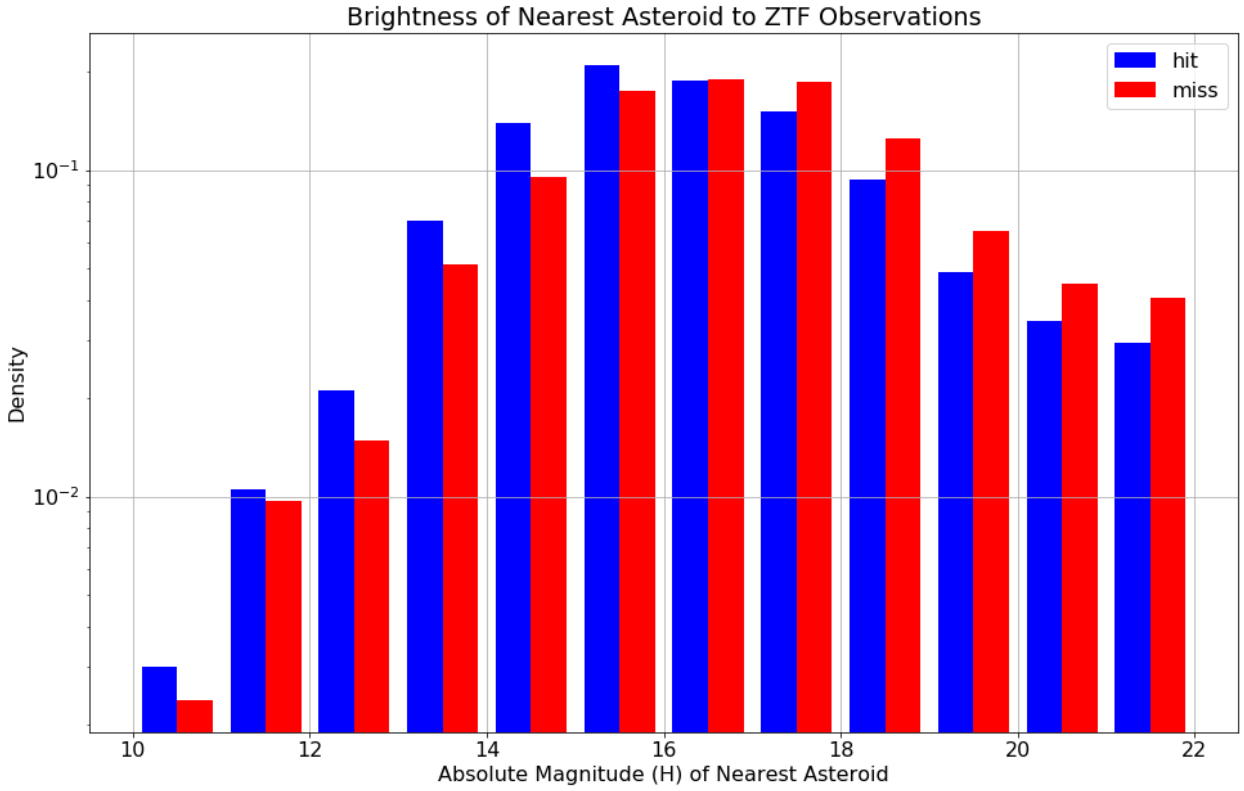

Figure 3.6: *Magnitude H of Nearest Asteroid to Each Detection*
Hits are shown in blue, misses in red; a hit is a detection within 2.0 arc seconds of its expected direction. The hits are slightly but noticeably tilted in favor of brighter asteroids.

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 the 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 Figure 3.7. 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 useful parameterization because there is an elegant representation of the differential solid angle $d\Omega$ in terms of dz. [1] When surface integrals of a sphere are taught in

[1] The notation $d\Omega$ refers to the differential of surface area on the sphere and has nothing to do with the longitude of the ascending node orbital element. This is yet another clash of Greek letters.

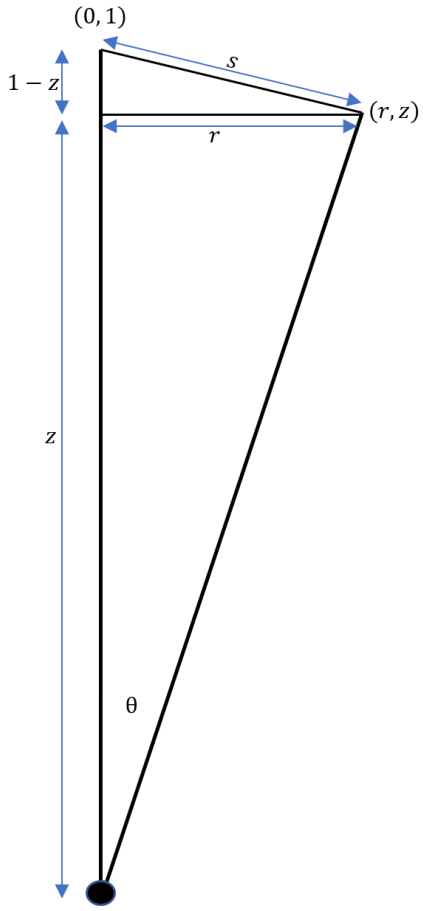

Figure 3.7: *Distance Between an Observation at the North Pole and an Arbitrary Point.*
The observed direction $\mathbf{u}_{\text{obs}}$ *is assumed w.l.o.g. to be at the north pole* $(0, 0, 1)$.
The predicted direction (x, y, z) *is rotated in the* xy *plane to to* $(r, 0, z)$
The Cartesian distance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

introductory multivariate calculus courses, students are most likely to be exposed only to the surface element in spherical coordinates (r, θ, ϕ) which is $d\Omega = \sin\theta \cdot d\theta \cdot d\phi$. See, e.g. Wikipedia - integration in spherical coordinates. But the parameterization in terms of the height z along the z axis is especially clean and convenient on this problem.

Observe that $z = \cos\theta$ so $dz = |-\sin\theta|d\theta = \sin\cdot\theta d\theta$. We take absolute values because this is an application of the change of variables formula and measures are always positive. $\sin(\theta) \geq 0$ here because $0 \leq \theta \leq \pi$.

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:

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

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

$$V \sim \text{Unif}(0, 1)$$

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

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

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. Figure 3.8 shows a histogram of hits inside a threshold of 2.0 arc seconds: Figure 3.9 shows two additional plot to

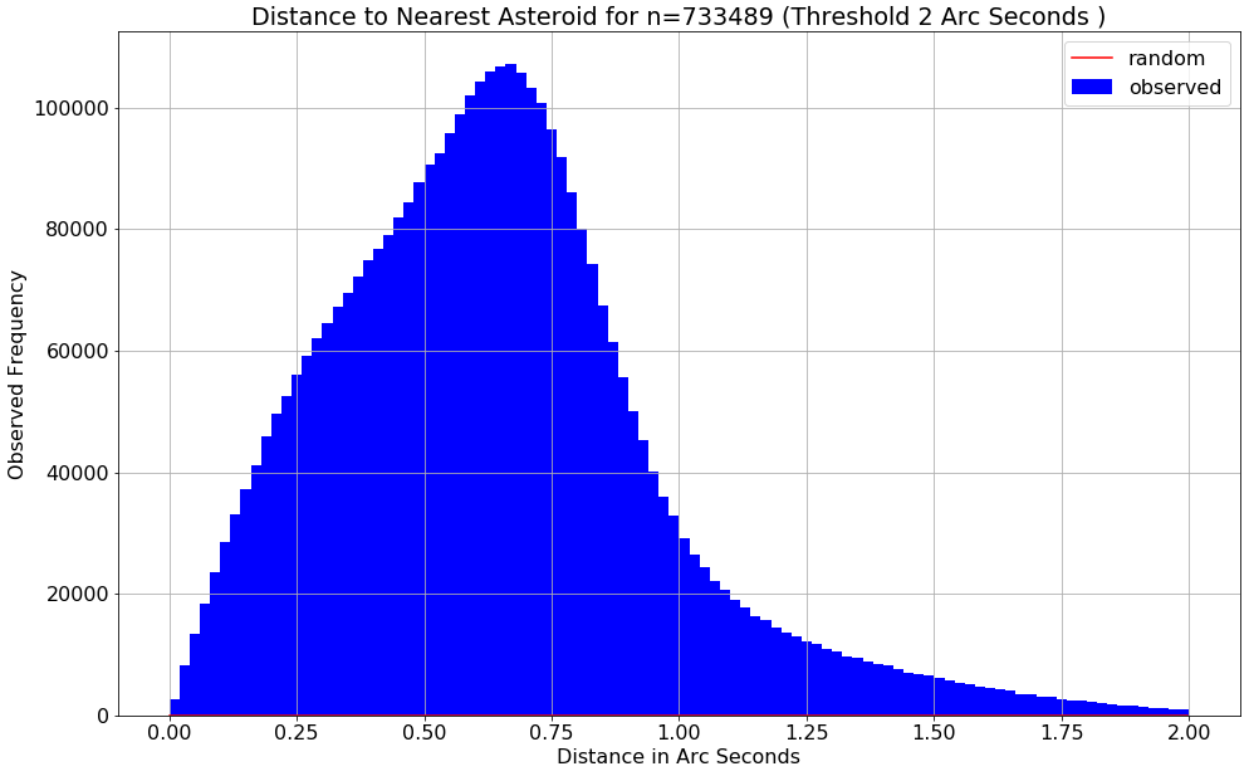

Figure 3.8: *Histogram of Angular Distance from Each ZTF Detection to the Nearest Asteroid in arc seconds. These account for 65.57% of the total data set.*
The number of expected detections due to random chance is shown in red; it is visually indistinguishable from zero on the chart.

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 shows 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

How many asteroids in the catalogue have enough detections in the ZTF dataset that we would have a sporting chance to recover their orbital elements from these observations alone? Suppose for the sake of discussion that the number is $n = 20$. There are 63,746 asteroids with 20 or more hits at 2.0 arc seconds. If we required only $n = 10$ hits to recover the orbital elements, we could do it for 100,508 asteroids.

We can visualize this in Figure 3.10 by plotting the cumulative frequency of close observations on the y axis vs. hit count on the x axis:

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. I have provided motivation that the approximate shape of the distribution of distances decays with an exponential tail. Finally, I have shown the fraction of the asteroid catalogue we might be able to recover as a function of the number of observations required to fit it. If we require $n = 10$ detections to verify or update the orbital elements of a known asteroid, then we can apply this procedure on 100,508 asteroids representing 13.70% of the known asteroid catalogue.

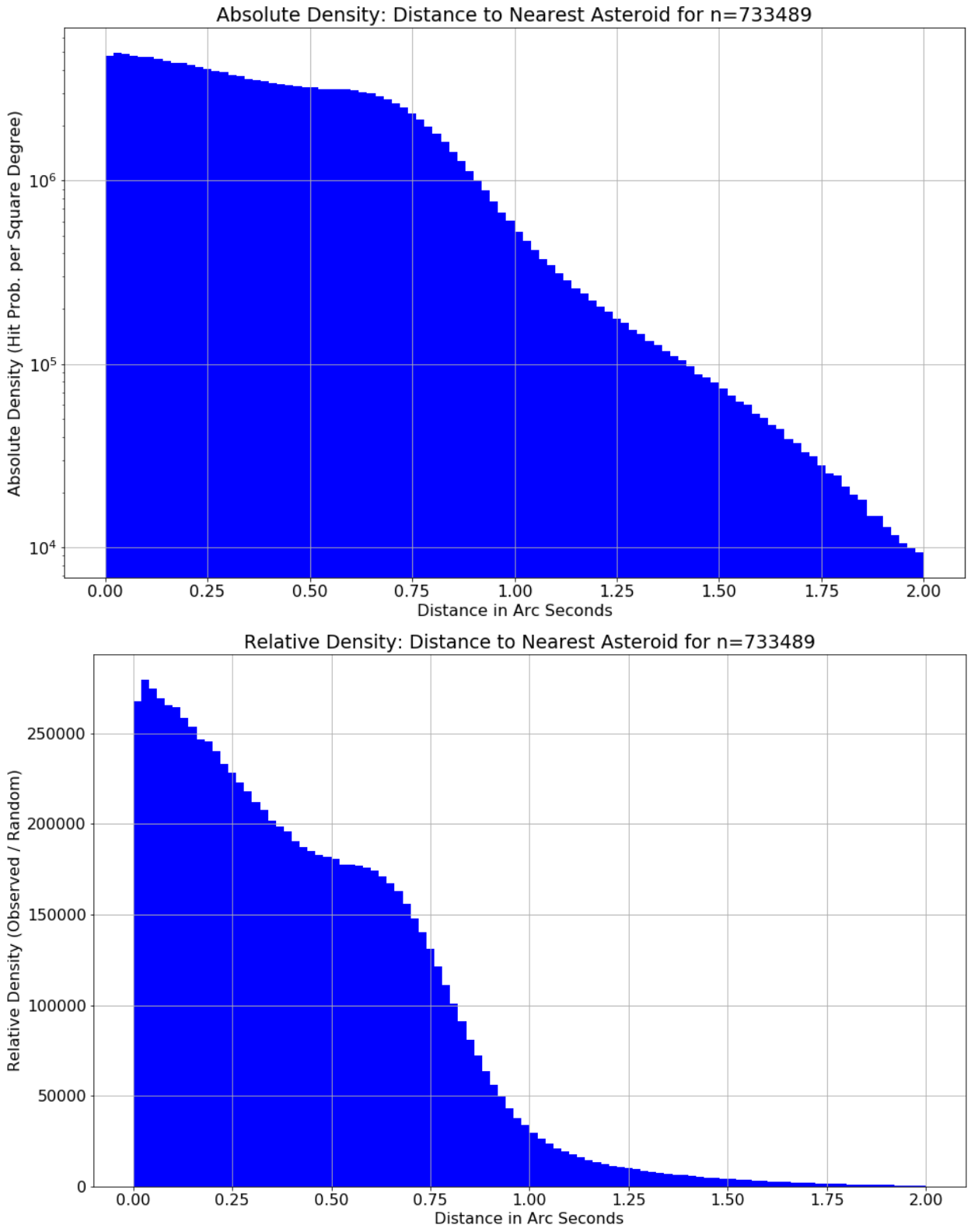

Figure 3.9: *Density of Hits for Distance Between ZTF Detections and the Nearest Known Asteroid.*
The first chart shows the absolute density in hits per square degree.
The second chart shows the relative density of this over the baseline if the guesses were distributed randomly.

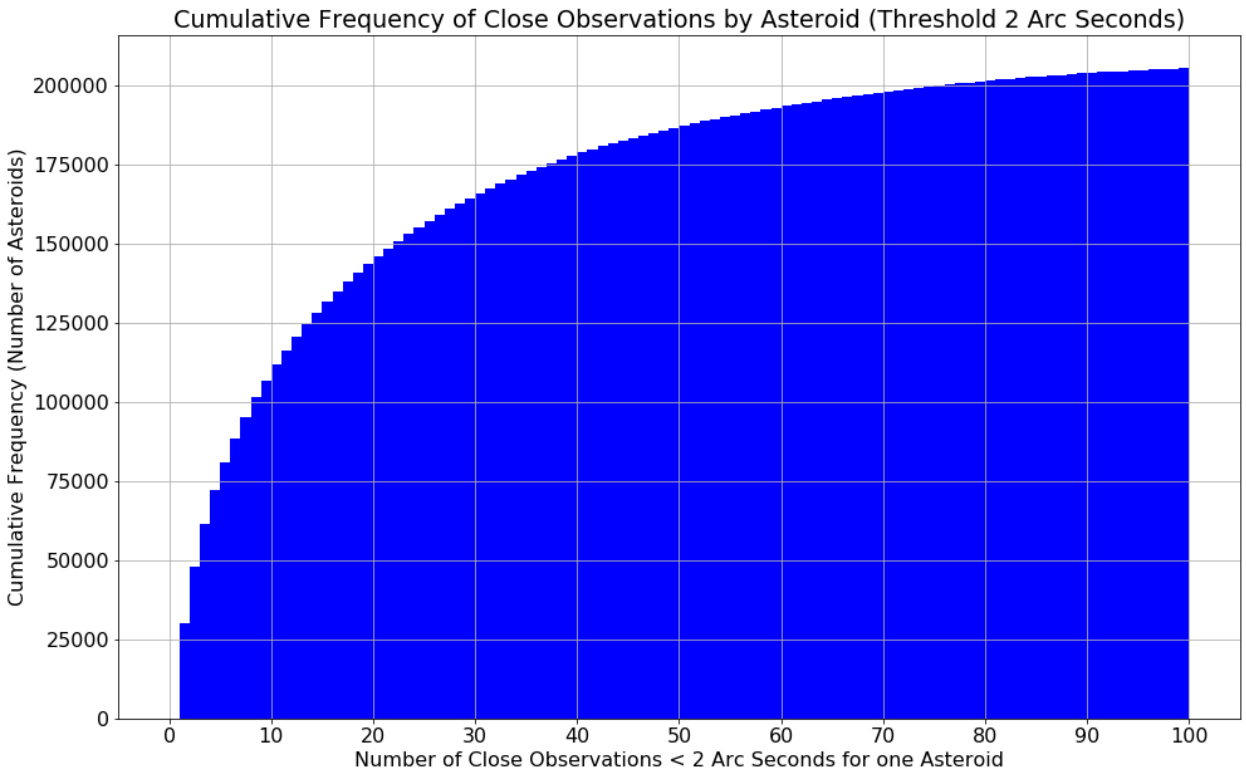

Figure 3.10: *Cumulative Frequency of Asteroids by Number of Close Observations < 2.0 arc seconds. There are 63,746 asteroids with 20 or more close matches and 100,508 asteroids with 10 or more close matches.*

Chapter 4

Asteroid Search Using Orbital Elements

4.1 Introduction

In the previous chapters we have laid the groundwork for the main event: searching for new asteroids in the ZTF dataset. Here is an outline of the search process, which will be elaborated in greater detail in the sections below. The search is initialized with a set of candidate orbital elements that is generated randomly based on the orbital elements of known asteroids. The orbits are integrated over the unique times present in the ZTF data, and the subset of ZTF detections within a threshold (2 degrees) of each candidate element is assembled.

A custom Keras model class called `AsteroidSearch` performs a search using gradient descent. This search optimizes an objective function that is closely related to the joint log likelihood of the orbital elements as well a set of parameters describing a mixture model. The mixture model describes the probability distribution of the squared distance over the threshold as a mixture of hits and misses. Hits are modeled as following an exponential distribution, and misses are modeled as being distributed uniformly. A set schedule of adaptive training is run. This training schedule has alternating periods of training just the mixture parameters at a high learning rate and jointly training the mixture parameters and orbital elements.

At the conclusion of the training process, we tabulate "hits" which are here defined as ZTF detections that are within 10 arc seconds of the predicted direction. All the fitted orbital elements are saved along with summary statistics of how well they were fit including the mixture parameters. The most important indicator is the number of hits. Candidate orbital elements with at least 5 hits

are deemed noteworthy and candidates with 8 or more hits are deemed to have been provisionally fit. The search program also saves the ZTF detections associated with each fitted orbital element.

I demonstrate the effectiveness of this method in a series of increasingly difficult tests. The easier tests involve recovering the orbital elements of known asteroids that have many hits in the ZTF dataset. The most difficult task is to identify the orbital elements of new asteroids by searching the subset of ZTF detections that don't match the known asteroid catalogue. In particular, the five tasks presented are

- recover the elements of known asteroids starting with the exact elements, but uninformed mixture parameters
- recover the elements of known asteroids starting with lightly perturbed elements
- recover the elements of known asteroids starting with heavily perturbed elements
- "rediscover" the elements of known asteroids starting with randomly initialized elements
- discover the elements of unkown asteroids starting with randomly initialized elements

The search process presented passes the first three test with varying degrees of success, recovering 64, 42 and 12 elements, respectively, out of the 64 candidates. The search for known asteroids from random initializations has 1 success on the first batch of 64 and is eventually run on a large scale, where it plausibly recovers elements of 125 known asteroids. The search for previously unknown asteroids yields 9 candidate orbital elements that I claim might plausibly belong to real but uncatalogued asteroids.

I tested the quality of the results by comparing the fitted orbital elements to the known orbital elements on two metrics. The most important indicator is to compare the orbits on a set of representative dates and compute the mean squared difference in the position in AU. A secondary metric is to compare the orbital elements themselves. This is done with a metric that standardizes each element and assigns it an importance score. Both of these metrics show excellent agreement of the recovered orbital elements with the existing elements in the asteroid catalogue.

4.2 Generating Candidate Orbital Elements

The search is initialized with a batch of candidate orbital elements. The batch size is a programming detail; I selected $n = 64$. The choice of initial orbital elements is critically important to the search. Unlike with other problems, where in theory there is often one globally correct answer that might or might not be reachable depending on the initialization, the number of local maxima in the objective function here will be at least the number of real asteroids adequately represented in the data. Based on the last chapter, that means there are over 100,000 local maxima in the objective function.

In this work, I use a simple strategy of random initializations. Improving on this initiailization strategy is the most important item of future work. I had originally planned to upgrade this to a more intelligent initialization but unfortunately ran out of time. Random initialization would be nearly hopeless if we had no information about the probability distribution of orbital elements. But because we have access to large asteroid catalogue, it is feasible to generate plausible candidate elements.

The random initialization strategy breaks the six orbital elements into two categories: empirical and uniform. The elements a, e, i and Ω are sampled from the empirical distribution. To be more precise, four random indices j_a, j_e, j_i and j_Ω between 1 and 733,489 are selected, and the initialization is done by setting e.g. a_j equal to the semi-major axis of the known asteroid with number j_a. The two orbital elements M and ω are initialized uniformly at random on the interval $[0, 2\pi)$. We know from Kepler's second law (equal time in equal area) that the mean anomaly M is linear in time, so we have a solid theoretical argument for sampling it uniformly. Once M is determined, it is converted to f using `REBOUND`. I will show empirically that the argument of perhelion ω appears to be distributed very close to uniformly as well.

Figures 4.1, 4.2, and 4.3 plot PDFs for selected mathematical transformations of orbital elements.

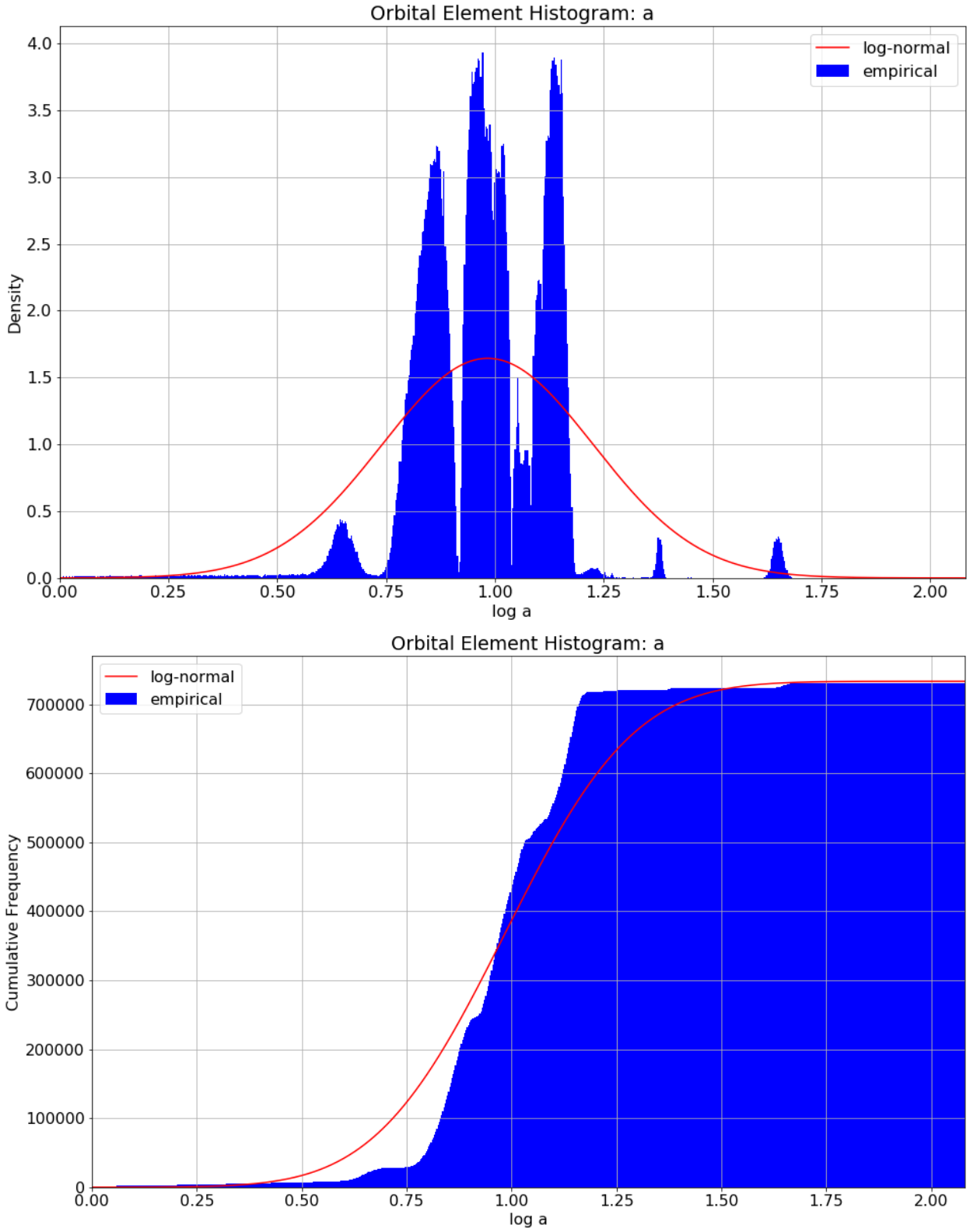

Figure 4.1: *PDF and CDF for* $\log(a)$*, Log of the Semi-Major Axis.*
We can clearly see the famous Kirkwood gaps in the PDF.
The CDF shows that on a macroscopic scale, a log-normal model isn't bad.
$\log(a)$ *is sampled empirically from the CDF.*

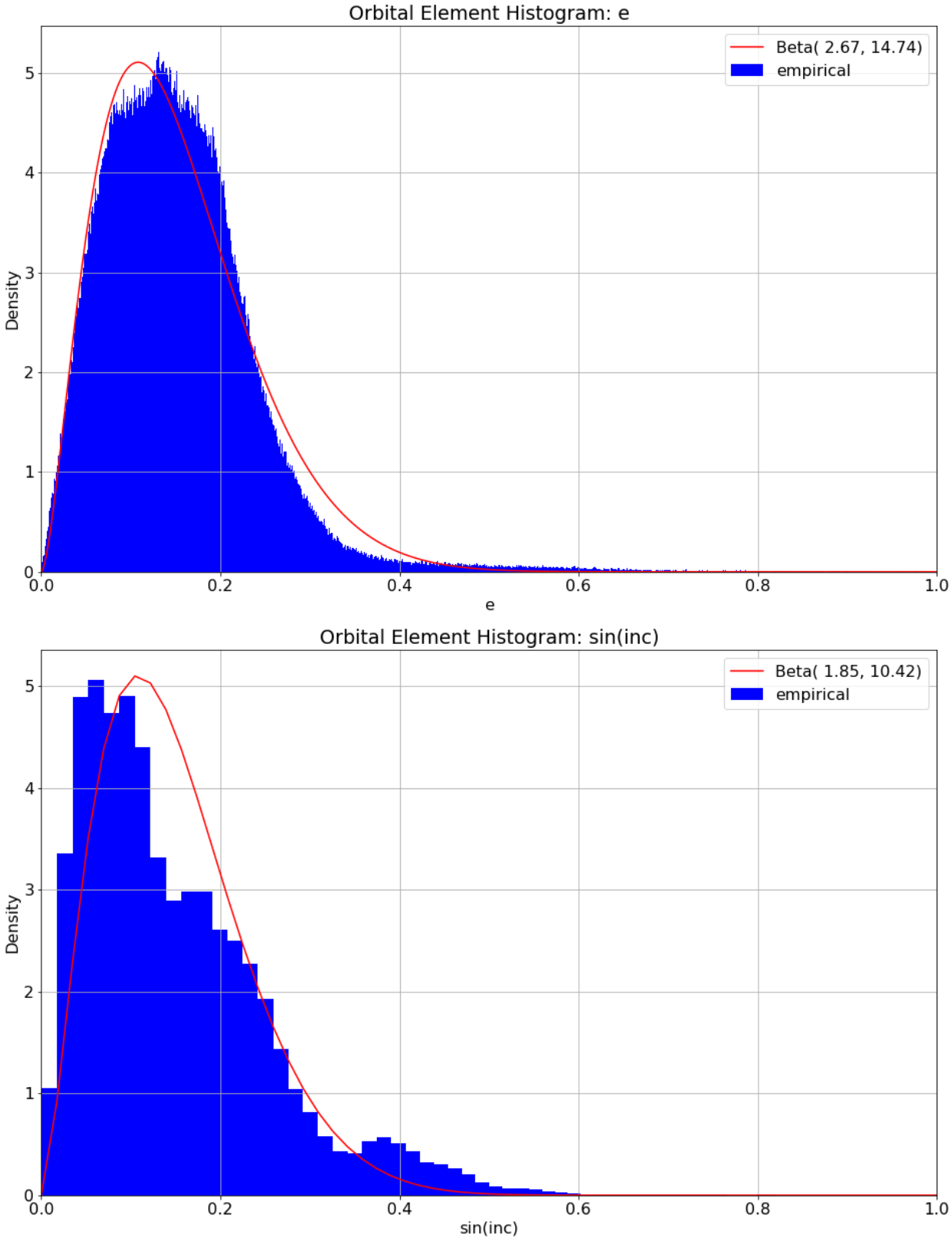

Figure 4.2: *PDF for Eccentricity e and* $\sin(i)$ *(Sine of the Inclination).*
Both e and $\sin(i)$ *are bounded in* $[0, 1]$ *and can be decently approximated by a Beta distribution.*
Both e and i are sampled empirically from the CDF; Beta sampling could have also worked well.

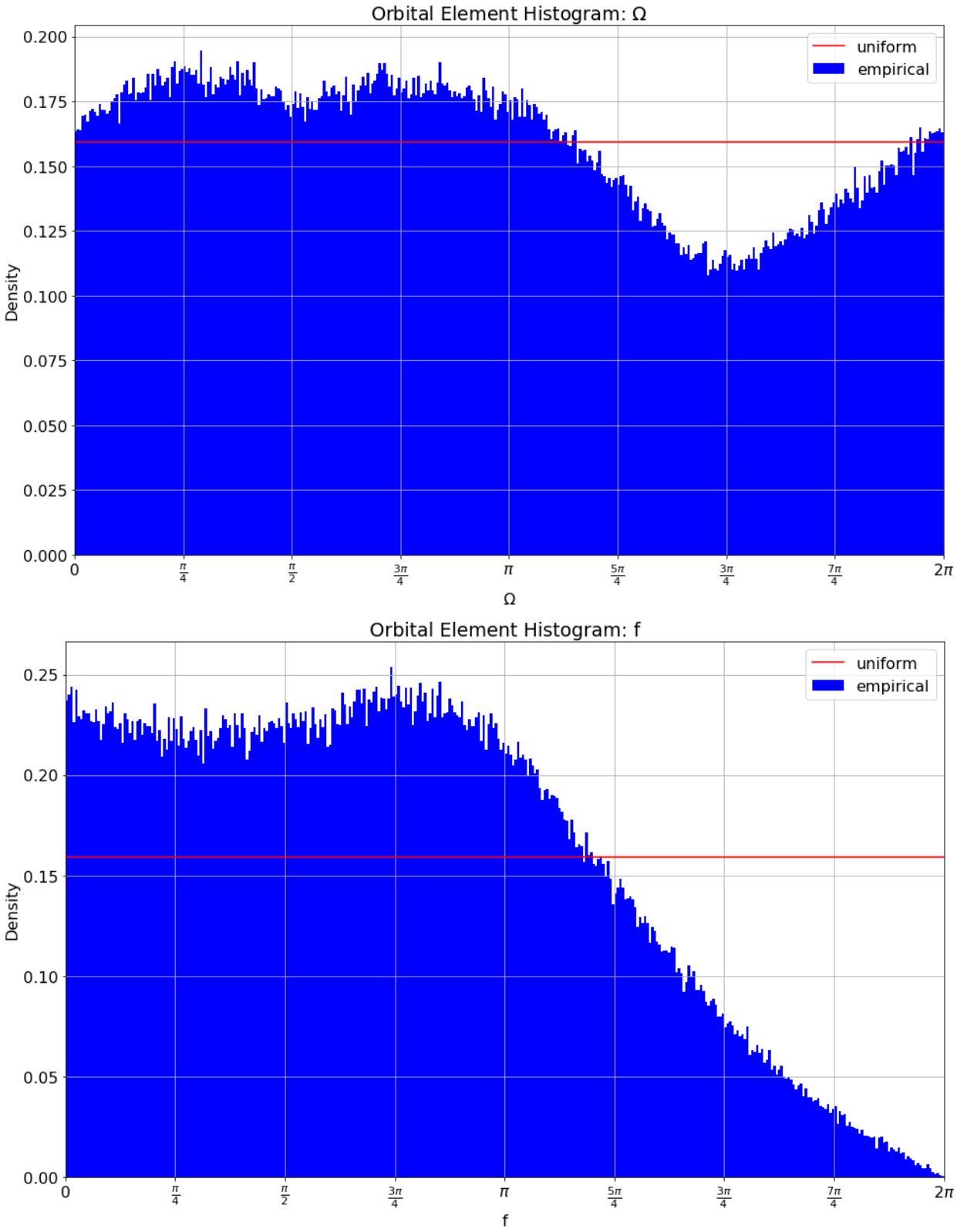

Figure 4.3: *PDF for Longitude of Ascending Node Ω and True Anomaly f.*
The PDF for Ω is somewhat close to uniform, but with a noticeable departure.
The PDF for f has an odd shape that I would have been hard pressed to predict ahead of time.
Ω is sampled empirically from the CDF; f is computed by sampling M uniformly.

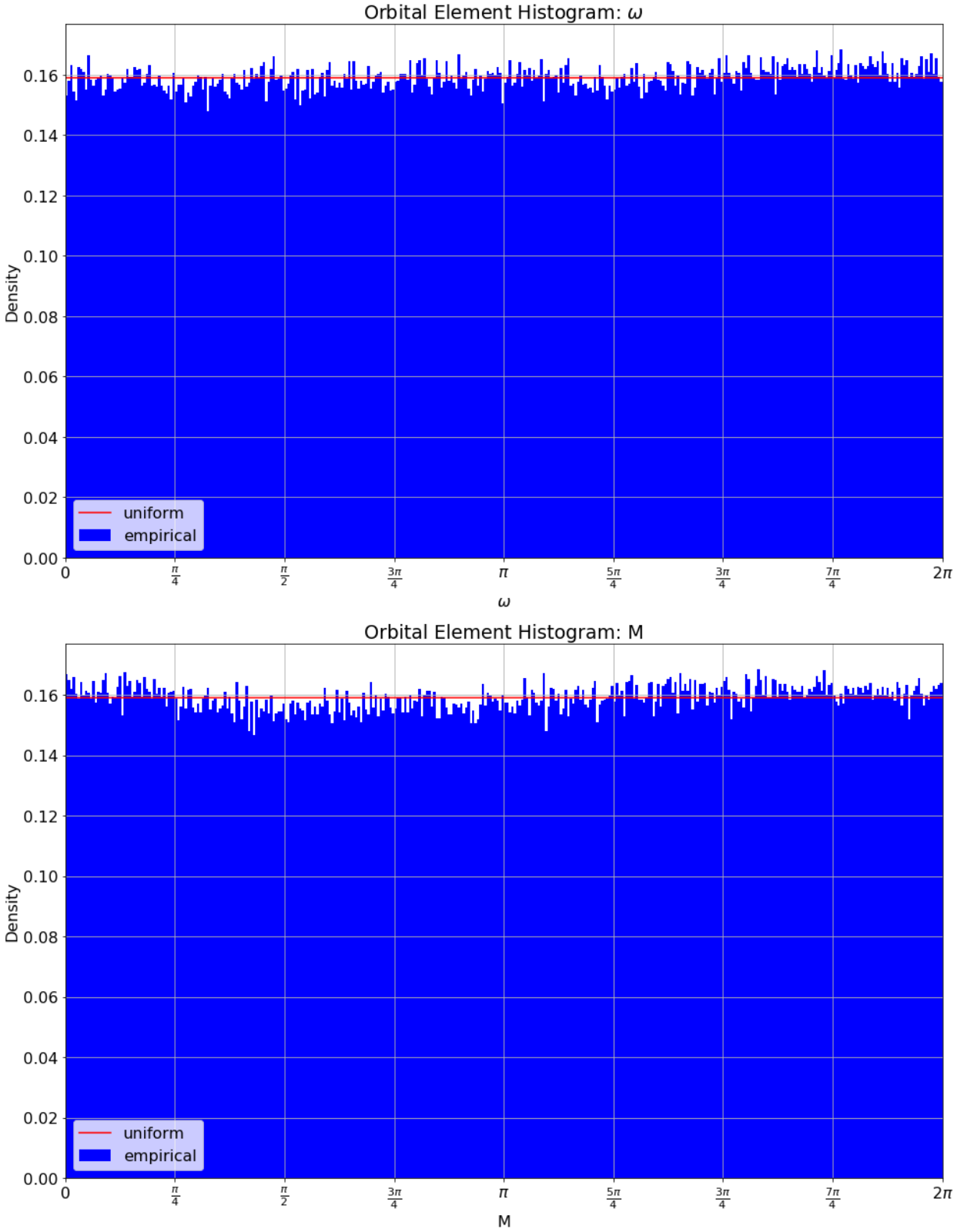

Figure 4.4: *PDF for Argument of Perihelion ω and Mean Anomaly M.*
As promised, these are empirically very close the uniform distribution we would expect.
Both of these elements are sampled uniformly at random.

If a continuous rather than discrete sampling strategy were desired, e and $\sin(i)$ could be well approximated by a fitted Beta distribution as shown in the preceding charts. Drawing $\log(a)$ from a distribution could be a bit messy. To my eye, the best solution there would be a mixture of normals with perhaps 6 to 10 components. I see little argument in favor of drawing a or ω other than empirically. Random elements are generated in the module `candidate_elements.py` with the function `random_elts`. A random seed is used for reproducible results.

4.3 Assembling ZTF Detections Near Candidate Elements

Once we've generated a set of candidate orbital elements, the next step in the computation is to find all the ZTF detections that lie within a given threshold of the elements. We've already introduced the important ideas that go into this computation in earlier sections. The only difference is that instead of calculating the direction of a known asteroid whose orbit was integrated and saved to disk, we integrate the orbit of the desired elements on the fly. Then we proceed to calculate the predicted direction from the Palomar observatory and filter down to only those within the threshold (I used 2.0 degrees in the large scale search.)

The module `ztf_element` includes a function `load_ztf_batch` that takes as arguments dataframes `elts` and `ztf` of candidate orbital elements and ZTF observations to cross reference against. It also takes a threshold in degrees. It returns a data frame of ZTF elements that is keyed by `(element_id, ztf_id)` where `element_id` is an identifier for one candidate element (intended to be unique across different batches) and `ztf_id` is the identifier assigned to each ZTF detection.

The work of integrating the candidate elements on a daily schedule is carried out by `calc_ast_data` in module `asteroid_dataframe`. The work of splining the daily integrated asteroid positions and velocities at the distinct observation times is done in `make_ztf_near_elt`. Because this computation is fairly expensive (it takes about 25 seconds to integrate a batch of 64 candidate elements), a hash of the inputs is taken and the results are saved to disk using the hashed ID. If a subsequent call for the ZTF elements is made with the same elements, it is loaded from the cache on disk.

Those readers who would like an interactive demonstration can find one in the Jupyter

notebook `06_ztf_element.ipynb`. Figure 4.5 shows a preview of the output dataframe `ztf_elt`: In Chapter 3, we showed that the quantity $v = (s/\tau)^2$ is would be distributed

```
# Load unperturbed element batch
ztf_elt_ast = load_ztf_batch(elts=elts_ast, thresh_deg=1.0, near_ast=False)
```

```
# Review
ztf_elt_ast[cols]
```

	element_id	ztf_id	mjd	ra	dec	ux	uy	uz	mag_app	elt_ux	elt_uy	elt_uz	s_sec	v	is_hit
0	733	53851	58348.197581	266.229165	-13.513802	-0.063945	-0.983101	0.171530	16.755600	-0.057300	-0.982042	0.179751	2191.408734	0.370552	False
1	733	73604	58348.197581	265.761024	-13.509148	-0.071871	-0.982578	0.171389	16.035999	-0.057300	-0.982042	0.179751	3467.151428	0.927559	False
2	733	82343	58389.193252	270.331454	-11.244934	0.005674	-0.977422	0.211222	17.196199	0.000919	-0.977996	0.208622	1124.103915	0.097503	False
3	733	257221	58685.471227	29.693832	42.180412	0.643725	0.603886	0.470042	19.289200	0.639004	0.610779	0.467571	1797.091521	0.249197	False
4	733	327000	58691.465972	33.104905	44.059131	0.601970	0.636719	0.481893	17.725201	0.606278	0.637608	0.475272	1639.539679	0.207419	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
90206	324582	5650588	58904.176701	44.164238	29.650540	0.623416	0.752309	0.213037	18.084700	0.627640	0.750696	0.206212	1688.638104	0.220027	False
90207	324582	5650589	58904.176250	44.164062	29.650536	0.623417	0.752307	0.213038	18.165199	0.627641	0.750695	0.206213	1688.601889	0.220018	False
90208	324582	5650665	58904.176250	44.368640	28.490480	0.628284	0.753618	0.193182	19.025200	0.627641	0.750695	0.206213	2757.856412	0.586871	False
90209	324582	5650697	58904.176250	43.296207	29.505908	0.633424	0.743491	0.214467	19.852800	0.627641	0.750695	0.206213	2555.278205	0.503822	False
90210	324582	5650705	58904.176250	44.621045	29.303550	0.620689	0.756675	0.205398	19.647400	0.627641	0.750695	0.206213	1898.912116	0.278236	False

90211 rows × 15 columns

Figure 4.5: *ZTF Detections Within a 1.0 Degree Threshold of a Batch of 64 Orbital Elements.*

$\sim$ Unif$[0,1]$ if predicted distributions were distributed uniformly at random. The function `plot_v` in module `element_eda` generates such a plot. I generated a list of the 64 asteroids that have the most hits in the ZTF dataset (ranging from 148 to 194). Then I generated ZTF dataframes for three collections of orbital elements:

- unperturbed orbital elements belonging to these 64 asteroids
- perturbed orbital elements of these 64 asteroids
- random orbital elements

As a test of the theory and to build intuition, Figure 4.6 plots the distribution of v against a threshold of 1.0 degree for these three sets of candidate elements. The results are exactly as predicted. The random distribution is approximately uniform as expected. The unperturbed distribution is a mixture of uniform and a spike in the first bucket. The perturbed distribution is in between, with the hits leaking out over the first few buckets out to $v \approx 0.07$ (about 250 arc seconds).

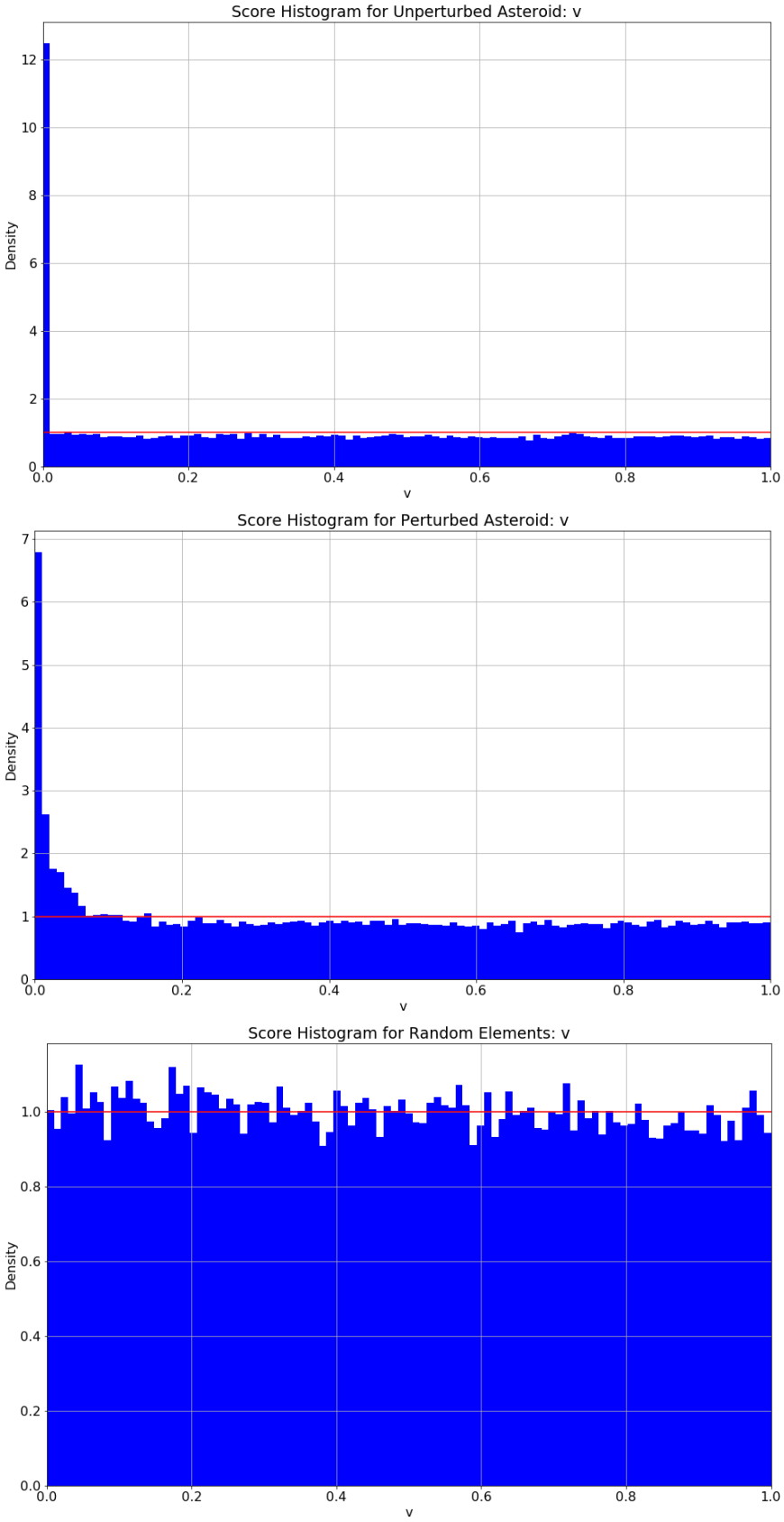

Figure 4.6: *Histogram of $v = (s/\tau)^2$ for Three Sets of Candidate Orbital Elements.*

4.4 Filtering the Best Random Elements

One idea is to perform a preliminary screening of the candidate orbital elements before investing a large amount of computational resources into running an asteroid search on them. In the next section we will show how to generate the ZTF detections within a threshold τ of the candidate elements. We've already seen that the random variable $V = (S/\tau)^2$ is distributed $V \sim \text{Unif}(0,1)$. We can assess candidate elements by taking the sample mean of $\log(v)$; we want to explore elements that have a disproportionate share of hits where v is small. Here is a quick demonstration that for $V \sim \text{Unif}(0,1)$, $\log(V)$ has expectation -1 and variance 1.

$$\begin{aligned}
\text{E}[V] &= \int_{v=0}^{\infty} \log(v) dv = v \log v - v]_0^1 = (1 \cdot \log 1 - 1) - (0 - 0) = -1 \\
\text{Var}[V] &= \text{E}[V^2] - \text{E}[V]^2 = \int_{v=0}^{1} \log(v)^2 dv - (-1)^2 \\
&= v \cdot (\log v)^2 - 2 \log v + 2) \big]_0^1 = 2 - 1 = 1
\end{aligned}$$

If a set of candidate elements has n detections within threshold τ with relative squared distances of $v_1, \ldots v_n$, their sample mean $\bar{v}$ will have expectation -1 and variance n, so I contruct a t-score for candidate elements

$$T = \frac{-(\bar{v} + 1)}{n}$$

This score would be distributed $T \sim \text{N}(0,1)$ (standard normal) if the guessed positions were uniformly random. It provides a computationally efficient way to screen candidate orbital elements.

This screening is performed in the module `random_elements`. The function `calc_best_random_elts` generates a large batch of random elements (the default size is 1024). It then builds the ZTF observations close to them and extracts the t-score as described above. The input batch size is used to select that many of the candidates that have the best score. The whole process of building the ZTF data frames, searching for the best elements, and saving the best elements and assembled ZTF data frames to disk is carried out by a Python program that can be run from the command line as

```
(kepler) $ python random_elements.py -seed0 0 -seed1 1024 -stride 4
> -batch_size_init 1024 -batch_size 64 -known_ast
```

The example call above runs the program on 256 batches of random elements, with random

seeds $[0, 4, \ldots, 1020]$. The stride argument is to facilitate parallel processing. The two batch size arguments request that 1024 initial elements be winnowed down to 64 with the highest t-scores. The flag `-known_ast` at the end asks that only the subset of ZTF detections within 2.0 arc seconds of a known asteroid be used to generate the ZTF dataframe and score the initial elements. I call this searching against known asteroids. If `known_ast` is not passed, the behavior is the opposite; only the ZTF detections at least 2.0 arc seconds (i.e. ones that don't closely match) are considered. I ran this program to generate 4096 candidate elements for each of the known and unknown asteroids. Altogether it took quite a while to run, over one day of total computer time. The vast bulk of that time is spent building the ZTF dataframe of detections near the elements.

4.5 Formulating the Log Likelihood Objective Function

The actual asteroid search is an optimization performed in TensorFlow using gradient descent. Perhaps the most important choice is that of the objective function. Qualitatively we know that we want an objective function that will be large when we are very close (within a handful of arc seconds) to some of the detections. We don't have a preference about the distance to the other detections. While it might seem tempting to write down an objective that rewards being close to everything, that's not at all what we want. Such an objective function would encourage us to find some kind of "average orbital element" for all the asteroid detections in this collection. But we want to find the elements of just one real asteroid.

A principled way to formulate an objective function is with probability. As a reminder, S is the Cartesian distance between $\mathbf{u}_{\text{pred}}$ and $\mathbf{u}_{\text{obs}}$, and τ is the threshold Cartesian distance so only observations with $S < \tau$ are considered. $V = S/\tau$ is in the interval $[0, 1]$. Introduce the following probability mixture model for the random variable V. Some unknown fraction h (for hits) of the observations are associated with one real asteroid, whose elements we are converging on. Conditional on an observation being in this category (a hit), the distribution of V is exponential with paramter λ. Conditional on an observation being a miss, V is distributed uniformly on $[0, 1]$.

In the formalism of conditional probability,

$$V|\text{Hit} \sim \text{Expo}(\lambda)$$

$$V|\text{Miss} \sim \text{Unif}(0,1)$$

We can relate the parameter λ to a resolution parameter R by observing that $v = (s/\tau)^2$ and

$$f(v) \propto e^{-\lambda v} = e^{-\lambda s^2/\tau^2}$$

This looks just like a normal distribution in the Cartesian distance s, a plausible and intuitive result! Let us identify the standard deviation parameter σ of this normal distribution with the resolution R, i.e. think of the PDF $f(s)$ as being normal with PDF $f(x) \propto e^{-s^2/2R^2}$.
Equating the exponent in both expressions, we get the relationship

$$\lambda = \frac{\tau^2}{2R^2}$$

It is convenient to use λ for calculations, both mathematical and in the code. For understanding what is going on, I find it more intuitive to use the resolution, since it's on the same scale as the threshold τ. The PDF of an exponential distribution is given by [4]

$$f(v;\lambda) = \lambda e^{-\lambda v}$$

In this case, we need to modify this PDF to account for the fact that $v \in [0,1]$ while the support the exponential distribution is $[0,\infty)$. What we want instead is the truncated exponential distribution, which is normalized to have probability 1 on the interval $[0,1]$, namely

$$f(v|\text{Hit},\lambda) = \frac{\lambda v}{1-e^{-\lambda}}$$

Of course, the PDF of the uniform distribution is just 1, so

$$f(v|\text{Miss}) = 1$$

Now we can write the PDF of the mixture model using the Law of Total Probability:

$$\begin{aligned} f(v|h,\lambda) &= f(v|\text{Hit},\lambda) \cdot P(\text{Hit}) + f(v|\text{Miss}) \cdot P(\text{Miss}) \\ &= h \cdot \frac{\lambda v}{1-e^{-\lambda}} + (1-h) \end{aligned}$$

The optimization objective function will be the log likelihood of the PDF:

$$\mathcal{L}(\mathbf{v},h,\lambda) = \sum_{j=1}^{n} \log\left(h \cdot \frac{\lambda v_j}{1-e^{-\lambda}} + 1 - h\right)$$

Please note that I've omitted the parameter τ from these expressions to lighten the notation. During the training of the model, the τ parameter is also updated. The three mixture parameters that are manipulated during training are

- N_h: the number of hits for this candidate element
- R: the resolution of this candidate element as a Cartesian distance
- τ: the threshold of this candidate element as a Cartesian distance

The hit rate h is computed from N_h by dividing by the number of rows that are within the threshold distance. The dimensionless error term v is computed by taking $v = (s/\tau)^2$. The exponential decay parameter λ is calculated as $\lambda = \tau^2/2R^2$.

In general, a likelihood function is only defined up to a multiplicative factor and a log likelihood up to an additive constant. In a theoretical analysis of maximum likelihood, the constant is typically irrelevant because one is differentiating the likelihood function anyway. In this problem, I want to set the constant term so that a log likelihood of zero equates to having no information, i.e. an uniformative baseline. This is particularly easy to do here: if we set $h = 0$, the terms involving the truncated exponential distribution all drop out and the log likelihood becomes a sum of $\log(h) = 0$. In general, we can zero out the log likelihood function by evaluating it at a set of uninformative baseline values, and subtracting this quantity $\mathcal{L}_0$ from the current optimized $\mathcal{L}$.

There is an important intuition about the role of the mixture parameters that I would like to explain. The major challenge in tuning the resolution R is for the gradients to encourage the model to adjust the orbital elements to get closer to detections that are likely to be hits, without

getting deked [1] by close-ish detections that belong to other asteroids. If the resolution parameter R is too small (sharp focus) before convergence, the model will be too far away to pick up any gradient to the hits. It will achieve a negative log likelihood because $\log(1-h) < 0$ and it won't make it up from the putative hits. If R is too high (fuzzy), the model will end up compromising and trying to fit a cloud of detections belonging to different asteroids. The whole game of getting the model to converge is to find the sweet spot of h and R where the model gradually tightens its focus, like letting your foot off the clutch when you put a car with a manual transmission [2] into gear.

In previous iterations, I attempted to write down objective functions to balance these objectives by hand and failed miserably. It was only when I used probability theory with a mixture model that plausibly describes the underlying facts that I had any success.

The likelihood function above applies to only one of the candidate orbital elements. In the actual optimization of a batch of 64 elements, we need a single scalar valued objective function. Because all of the elements in a batch are being optimized independently and have no interaction with each other, we can simply take their sum. There is an important refinement to this idea though that I will discuss in the next section.

4.5.1 Adding the Magnitude to the Joint Likelihood Function

The likelihood function written above depends only on the probability distribution of the relative distance $v = (s/\tau)^2$. A more sophisticated model will also predict the magnitude. We can write down a simple probability density for the magnitude as well by assuming that it is normally distributed. We can treat H and σ_H as learnable parameters, analagous to a learned "orbital element" for the the absolute magnitude H and a learned "resolution" for apparent magnitude. A strong argument can be made that the magnitude errors are independent of the direction errors, so the joint PDF $f(v,h) = f_v(v) \cdot f_h(h)$. The PDF $f_h(m)$ will be $\varphi((m-H)/\sigma_H)/\sigma_h$ where $\varphi(z)$ is the standard normal PDF.

This time, we will need to specify the alternate, uninformed baseline, because it isn't just 1 as

[1] deke: an ice hockey technique whereby a player draws an opposing player out of position. It is a Canadian contraction of the workd "decoyed'.'

[2] Otherwise known as a *real* car

is the case for the uniform distribution on $[0, 1]$. This can be done by taking the sample mean and variance of all the observations within the current threshold τ; this calculation yields the term $\mathcal{L}_0$ discussed above.

I tried all of this, and got it working somewhat, enough that the model could function. But I found that my predictions of the magnitude were too finicky, enough so that the model performed better when I set the objective function as the simple log likelihood depending only on the angular distance. The code is present in `AsteroidSearch` and `TrajectoryScore` to incorporate the magnitude into the likelihood in the future if the prediction can be tightened up enough to work.

4.6 Performing the Asteroid Search

We have by now covered the main theoretical ideas that go into the asteroid search. We've seen how to generate a set of candidate orbital elements and a collection of ZTF observations within a threshold of these elements. And we've identified a log likelihood function that rewards orbital elements for getting close to detections likely to be real while learning mixture parameters to describe the provenance of observations under consideration and how closely they have been fit. In this section I will explain some of the most important details that were required to get this model to actually learn orbital elements from data.

The workhorse class that searches for asteroids is called `AsteroidSearchModel`. It's a Keras custom model defined in the module `asteroid_search_model.py`. An AsteroidSearchModel is initialized with a candidate elements and the ZTF observations near these elements. It constructs two layers of type `CandidateElements` and `MixtureParameters` that maintain, respectively, the candidate orbital elements and mixture parameters. These layers are defined in the module `asteroid_search_layers.py`. The candidate orbital elements are the familiar seven Keplerian orbital elements. The six "live" ones $(a, e, i, \Omega, \omega, f)$ are trainable variables, the epoch is locked at its initial value. The mixture parameters are N_h, R and τ, all of which are trainable.

4.6.1 Controlling Parameters on a Uniform Scale

The first important idea in training the model is that all variables are controlled internally with TensorFlow variables that are scaled with a comparable range, almost always $[0, 1]$. For example,

the orbital element a in the candidate elements layer is controlled by a `tf.Variable` named `a_` that is constrained to lie in the region $[0, 1]$. (If a gradient update tries to push it less than 0 or more than 1, it is clipped back in the allowed range.) The value of a is computed on demand by $a = a_{\min} \cdot \exp(\texttt{a\_} \cdot \texttt{log\_a\_range})$ where `log_a_range` = $\log(a_{\max}/a_{\min})$.
$a_{\min}$ and $a_{\max}$ are set by policy to 0.5 and 32.0, respectively. Other orbital elements are likewise controlled via mathematical transforms into the range $[0, 1]$. The eccentricity is left as is, though it is limited to at most $63/64 = 0.984375$ to avoid numerical instabilities that occur as e approaches 1. The inclination is controlled via $\sin(i)$, which is constrained to lie in the interval $\pm 1 - 2^{-8}$. The other angle variables Ω, ω and f are unconstrained, but multiplied by 2π so the control variable $f\_$ for instance can cover its entire allowed range of values by moving 1.0.

The number of hits `num_hits` is allowed in a range of 6 to 1024 and controlled by its log. I set the minimum to 6 because any smaller than that, there is no point to searching at all. The resolution is controlled on a log scale as well and allowed in a range of 1.0 to 3600 arc seconds. The threshold is controlled on a log scale and allowed in a range of 10.0 arc seconds up to the original threshold used to assemble the data, which is 7200 arc seconds in my runs.

What is the point of these machinations? It might be obscure to readers with a background in astronomy or applied math, but machine learning practitioners should be less surprised. Scaling variables to have a common size is a basic technique that significantly aids gradient descent in practice.

4.6.2 Gradient Clipping by Norm

A second important technique for the optimization is gradient clipping. The objective function is optimized using the de facto default in TensorFlow, Adam (adaptive moments). Gradient clipping is not turned on by default, but I found it to be vital for this problem to work. The reason it's so important is that the optimization function (and its gradients) vary over a tremendous scale in this problem. At the start of training, there is little to no information; the likelihood function is near zero; and the gradients are relatively small. As the model reaches convergence if it is lucky enough, it can achieve significantly large likelihoods and huge gradients. All gradients are clipped by norm to a maximum norm of 1. A good intuition for gradient clipping by norm is that the

direction of the gradient is not changed, but the magnitude is capped. My sense from training the model extensively is that the gradient is almost always "saturated" (i.e. the original gradient has a norm larger than 1, which is reduced to 1 by the gradient clipping.)

The combination of having all the control variables in a range $[0, 1]$ and gradients clipped at a norm of 1 gives us a useful intuition about how quickly the model can update its parameters. I perform training in "joint mode" (where both the orbital elements and mixture parameters are updated) with a learning rate of 2^{-16}. This is a factor of 64 smaller than the Adam default of 0.001, which I found to be far too high for this problem. Pretend for a moment that there were only 1 parameter. Assuming the gradient is saturated, on each data sample encountered, it will either increase or decrease by the learning rate. So after encountering $2^{16} = 65,536$ samples it would be able to move from one extreme of its values to another. Of course in practice with multiple parameters, a single parameter will almost never have a partial gradient equal to 1.0, but it's a good intuition for the "speed limit" of how fast any one parameter can change during training.

4.6.3 Scoring Trajectories: Log Likelihood and Hits by Candidate Element

Let's now sketch out the flow of information from the candidate elements and mixture parameters all the way to the objective function. When the model is initialized, it also constructs an `AsteroidDirection` layer. We've discussed this before–it's the layer that computes a Kepler orbit from the current candidate orbital elements, including a calibration adjustment that is periodically updated by numerically integrating the current candidate elements. The important thing to remember is that the asteroid direction layer is predicting directions based on candidate orbital element tensors that are the output of the candidate_elements layer.

Now a new layer comes into play: the `TrajectoryScore` layer. This layer is also defined in `asteroid_search_layers`. When the model is initialized, this layer saves the directions of all the observations in the ZTF data frame as Keras backend constants. This was a deliberate design choice that is somewhat unorthodox. Keras models are largely designed around the assumption that during training you will feed in batches with equal numbers of input and output samples. This problem has a quite different flavor. There are no "outputs" we can line up against a batch of 64 candidate orbital elements. And the only "inputs" are the initial candidate orbital elements,

which are trainable variables. Eventually I needed to pass a dummy input that had `batch_size` ones in it to placate Keras.

We are just computing an objective function and trying to maximize it, using TensorFlow as a big computational back end with support for GPU computation, automatic differentiation, and gradient descent optimization. By putting all the observations into Keras constants, we write them to the GPU once when the model is initialized, and then there is no need to copy any data between CPU and GPU memory during training. Eventually I can imagine training this model on such a huge data set that it might be necessary to batch the observation data. But with modern GPUs having memory capacities on the order of 10 GB, I think this approach should scale very well and offers significant performance benefits.

The trajectory score layer is passed a tensor with the predicted directions $\mathbf{u}_{\text{pred}}$ as well as the current mixture parameters. It computes the Cartesian difference s between the predicted and observed directions, then applies the current filter τ to assemble a new tensor v of the relative distance squared in $[0, 1]$. All of these tensors should be thought of as having a shape starting with the batch size; s for instance is one long tensor that represents the distances for all 64 orbital elements, concatenated together. The trajectory score layer then goes through the calculation of the probability and log likelihood under the mixture model described above. It returns a tensor of log likelihoods, one log likelihood for each of the 64 candidate orbital elements. It also counts the number of hits, which are defined here as observations that are within 10.0 arc seconds of their predicted directions.

4.6.4 Training Each Candidate Element Independently

During my early efforts to train this model, I struggled to find a learning rate that worked. I found that different elements converged at different times and had very different gradients. In my intuition, I want to pretend that the gradient has only six terms for the candidate elements and three terms for the mixture parameters. When the gradient is clipped to a size of 1 across a row, it's helpfully giving us a direction that we should adjust the elements and mixture parameters for that candidate element. But what TensorFlow is really doing is squashing the whole gradient, all 64 candidate elements worth, to have a norm of 1. When one element has a large gradient, it will

dominate at the expense of the others.

In writing this out now, I realize that what I probably should have done was to write a custom gradient clipping class that works on one candidate element at a time. What I did instead was to reason that I wanted the ability to effectively tune the learning rate on all 64 of the candidate elements independently. Of course a model of this kind has only one scalar objective function and one learning rate. But we can achieve the same effect by weighting the contribution of each candidate element. If $\mathcal{L}_i$ is the log likelihood of the ith candidate element, and w_i is the weight on the ith candidate element, then the weighted objective function is

$$\mathcal{L} = \sum_{i=1}^{n} w_i \mathcal{L}_i$$

The weights are initialized at $w_i = 1$. If we cut the weight w_2 to 0.5, then all the gradients due to candidate element 2 will also be cut in half.

But how do we decide when to adjust the weights? One problem I ran into repeatedly was the model would make quite good progress, then it would get to a region where the learning rate was too high and in just one epoch it would fall apart. I tried to alleviate this using the built in early stopping, but this doesn't work exactly the way I want it to. And even if it did, it only knows about the single, scalar valued loss function; it has no notion that out of 64 elements, 56 improved on the last epoch but 8 got worse and so should be rolled back. I ended up writing my own custom code to do exactly this. At the end of a series of epochs of training, which I refer to as one "episode" of adaptive training, I check which elements have regressed and have worse log likelihoods than before. Any element that has regressed has its candidate elements and mixture parameters rolled back to their prior (best) values. Those elements also have their weights adjusted by a factor of 0.5, i.e. they are cut in half. I call this procedure "adaptive training" because the learning rate is adaptive. I found that this simple idea significantly improved the performance of the training. Without it, I was forced to use glacially slow learning rates to avoid overshooting and collapse (I even tried 2^{-20} in one bleak moment of desperation).

4.6.5 Training in "Mixture" and "Joint" Modes

The first test I tried was to give the model a set of candidate orbital elements that exactly matched the 64 asteroids with the most observations in the data set (around 160 each on average). I figured that this problem would be a piece of cake for the model. It would pick up close to zero gradients on the orbital elements, and a huge gradient by tightening the resolution, and focus in tighter until it converged, right? Wrong! The problem is that when the resolution and threshold distances are too large, the model is attaching a lot of weight to observations that are far away. The early iterations polluted the good orbital elements and it never managed to converge.

I hit on the idea of freezing the candidate orbital elements and only training the mixture parameters. Of course, this feels a bit like cheating. If you know the elements are right and just want to learn the mixture parameters, of course you're going to do better if you freeze the elements and only train the mixture parameters. At first, this was only a testing technique. Later on, though, I realized that it helps the model to converge even when it's not cheating. My intuition is that it's "safer" to train the model at a higher learning rate when you adjust the mixture parameters than when you adjust the candidate elements as well.

The model as it now stands alternates rounds of training in two modes: "mixture" and "joint" modes. In mixture mode, only the mixture parameters N_h, R and τ are trainable. The learning rate is higher by a factor of 16, 2^{-12} in mixture mode vs. 2^{-16} in joint mode. The most important difference, though, is that the objective function is adjusted. When I started out, I had only two trainable mixture parameters, N_h and R. I noticed that the model wasn't converging all the way, and realized that I wanted it to reduce τ along with R as it trained. This is quite intuitive.

4.6.6 Modifying the Objective Function to Encourage Convergence

You might start out with observations within 2.0 degrees and a resolution of 0.5 degrees. But if you've trained to the point of a resolution of 0.1 degrees, there's no reason to drag around observations that are 20x further away. They're just noise, and they're hampering your ability to fine tune. I noticed at this point that after I made τ trainable, the model never wanted to reduce it. In hindsight this made sense: the likelihood always looks better when you consider it against a larger threshold. If your parameters are within 50 arc seconds on 100 observations vs. a threshold

of 2.0 degrees (7200 arc seconds), you're going to pat yourself on the back and say "that's pretty good, it's not too likely I could have achieved that by chance alone." If you shrink the resolution from 2.0 to 1.0, your v is going to quadruple and your log likelihood will plummet.

This suggests that while log likelihood is an excellent objective function for separating "more likely" from "less likely", it's not exactly what we want here. Even if the log likelihood has a very strong local maximum at a fully converged solution, this experience was telling me that there was no smooth path of increasing gradients to get there. Instead, I wanted an optimization function that would encourage the model to converge. I modified the objective function to take the more general form

$$J = \sum_{i=1}^{n} w_i \frac{\mathcal{L}_i}{R_i^{\alpha} \cdot \tau_i^{\beta}}$$

This is the same as before, but now there are powers of the resolution R_i and threshold τ_i in the denominator. While the original objective function only sought to find the most likely configuration, this objective function is willing to trade a reduction in likelihood for a more concentrated (converged) prediction. It's effectively adjusting the scoring for the "degree of difficulty" like Olympic diving and gymnastics. It's harder to generate orbital elements and mixture parameters with a resolution of 10 arcseconds against a threshold of 40 arcseconds than to do it with the initial settings of 0.5 and 2.0 degrees respectively.

I would also like to quickly state a theoretical motivation for this formulation, which originally motivated it. Going back to the idea that we have random variables $V_1, \ldots V_n \overset{i.i.d.}{\sim} \text{Unif}(0,1)$, we saw that the quantity $T = n^{-3/2} \cdot \sum_{i=1}^{n} \sim \mathcal{N}(0,1)$ had the same distribution regardless of n. (We divide the sum by n to get the sample mean and by $\sqrt{n}$ to give it variance 1). Of course, the number of detections n within a threshold τ is a discrete number, so it will have a derivative of zero except at points where it is not defined. But in the baseline case where we have a uniform density of detections on the sphere, the expected number of hits $\mathrm{E}[n] \propto \tau^2$. This provides a strong intuition that the log likelihood over τ^3 is a meaningful quantity and that dividing by a power of τ is justified. In the actual event, I experimented with different configurations and settled on $\alpha = 1$ and $\beta = 1$. I wanted to guide the resolution and threshold to smaller values at the same rate. In practice it didn't make a large difference, but these values seemed to work well.

4.6.7 Organizing Training: Batches, Epochs, Episodes, and Sieving Rounds

Training is organized hierarchically into batches, epochs, episodes, and sieving rounds. The training batch and epoch are already defined terms in machine learning. One batch consists of the 64 candidate orbital elements and all of the ZTF observations they are scored against. The number of batches in an epoch is a parameter I set to 64. Keras reports training on screen by the epoch, and the early stopping will kick in to terminate training early if the loss function gets worse. The size of 64 was set by experimentation. Too low and there is excessive overhead from logging progess and so on. Too high and you lose too much progress from bad training before you cut it short.

One episode is the term I use for a period of adaptive training. It is implemented with the methods `search_adaptive` and `train_one_episode` in the `AsteroidSearch` class. At the end of an episode, I save the weights and three quantities of interest for each candidate element: log likelihood, loss function, and number of hits at 10 arc seconds. If any of these figures of merit have gotten worse during the training round, I deem the training on those elements to have been a failure, revert the weights for those elements back, and cut the weight on these elements by half. This is equivalent to selectively reducing the learning rate on these elements by half. I also re-run the numerical calibration of the position model against the `REBOUND` integrated orbit at the current orbital elements at the end of each episode.

Weights and training progress are saved in the method `save_weights`. A separate method called `save_train_hist` saves data frames as .h5 files with all information required to serialize the model to disk. The built in methods for serializing Keras models were not working on this custom model, and in any case would have necessitated saving a huge amount of redundant data with all the observations as serialized Keras constans. These custom methods save only the data that changes (e.g. the candidate elements and mixture paramters) plus some calculations used for monitoring progress. The process of checking elements to see if they have made progress and reverting them if they regressed is done in the method `update_weights`.

I set the length of an episode to at most 4 epochs; an episode ends early if the Keras early stopping callback detects that the overall loss has gotten worse. I define the effective learning rate to be the average of the element weights w_i multiplied by the global learning rate. A training

episode is terminated early if the effective learning rate drops below a threshold, which defaults to 1/64 of the original learning rate.

A sieving round is the highest level of training. It covers at most a set number of samples. I set the length of a sieving round to 512 batches for mixture and 2048 batches for joint mode. A sieving round can also end early if the effective learning rate drops below a threshold. This extra hierarchical level is added to support adjusting a few settings that change only rarely. These include manually tuning the maximum permitted resolution and threshold, and resetting the weights on all elements to 1 if desired. The reason one might choose to reset the weights to 1 is that the adaptive training only reduces the learning rate when training overshoots. Periodically tuning the w_i back to 1 gives them a chance to rebound if the elements have reached a better spot where the faster learning rate is now feasible. The sieving round also provides a convenient interface for alternating between joint and mixture mode. Finally, the sieving round allows customization of the exponents α and β applied to the resolution and threshold in the denominator of the objective function. One sieving round is implemented by the method `sieve_round`.

The method `sieve` implements a preset schedule of sieving rounds. For maximum maximum thresholds of 7200, 5400, 3600 and 2400 arc seconds, it runs first training in mixture mode for 512 batches at a learning rate of 2^{-12}, then training in joint mode for 2048 batches at a learning rate of 2^{-16}. Observe that one sieving round in mixture mode is encountering each data point $2^9 \cdot 2^6 = 2^{15}$ times. Multiplying this by a learning rate of 2^{-12}, a parameter could theoretically cover its full range 8.0 times in one round. Similarly, a sieving round in joint mode encounters each data point $2^{11} \cdot 2^6 = 2^{17}$ times. At a learning rate of 2^{-16} an orbital element could cover its full range of values 2.0 times in a single round. Remember, though, that there are 64 elements in the batch, and this learning rate is shared by all of them. If we use the rule of thumb that the norm of the gradient will scale with the square root of of the batch size, we should divide all of these quantities by $\sqrt{64} = 8$. This tells us that a mixture parameter on one typical candidate element, moving at maximum speed, might cover 1.0 times its full range of motion in a sieving round in mixture mode. An orbital element on a typical element in the batch might cover 0.25 times its range of values in a sieving round in joint mode. At the end of this initial phase, a final fine tuning training phase is run with larger powers of τ and R in the denominator. This is intended to push the model to achieve full convergence on elements where it has discovered

enough hits in the first phase.

4.6.8 Performing the Asteroid Search: Summary

There are many other details that go into the computer code that carries out the asteroid search. `asteroid_search_model.py` has over 2400 lines of code, with an additional 770 in `asteroid_model` and 930 in `asteroid_search_layers`. In the preceding sections, I have endeavored to present the highlight of the search process, with an emphasis on ideas and policy choices. In the next section, we will apply these methods to an increasingly difficult series of asteroid search tasks and review the results.

Chapter 5

Asteroid Search Results

5.1 Comparing Candidate Elements to the Nearest Asteroid

Before we review the results of our asteroid search experiments, it will be helpful to have in hand a notion of how closely two sets of orbital elements match. In particular, we will test below whether or not we successfully recovered orbital elements when we started with them as an initial guess. Answering this question requires that we have a useful metric on the space of orbital elements.

I spent a fair amount of time trying to develop such a metric. While orbital elements are convenient for intuition and calculating orbits, there isn't an obvious distance metric we can put on them that makes a lot of sense. Eventually I decided that the canonical way to compare two orbits by comparing the predicted vector of positions on a set of representative dates. Logically this is hard to argue with, but it is somewhat computationally expensive compared to a computation that can be run directly on the elements.

The method `nearest_ast` searches the asteroid catalogue for the known asteroid whose orbit is closest to that predicted by the candidate elements. It delegates its work to the function `nearest_ast_elt_cart`, which is defined in `nearest_asteroid.py`. This function creates a set of 240 sample time points over 20 years spanning 2010 to 2030 sampled monthly. The resulting table of positions for the asteroid catalog is fairly large, with a size of $[733490, 240, 3]$ (5.28E8 elements and about 2.11 GB using 32 bit floats). Computing the nearest asteroid against 64 candidate elements by brute force in TensorFlow would necessitate creating a tensor with 3.38E10 elements or 135 GB of memory–two orders of magnitude too large for a high quality consumer

grade GPU with ~ 10 GB of memory. The nearest asteroid method is therefore forced to iterate through the elements one at a time, taking the norm of the difference against the table. In one important optimization, the tensor of known asteroid positions in loaded into memory once as a TensorFlow constant to avoid recomputing it every time the function is called.

I also sought to develop a sensible metric of the distance between a pair of arbitrary orbital elements. This is implemented in the same module with the function `elt_q_norm` and `nearest_ast_elt_cov`. The idea is to transform the elements to a Cartesian representation where they have a well behaved covariance matrix. In particular, the goal is to find a deterministic transform of the elements that is distributed approximately as a multivariate normal. Then the Mahalanobis distance is a natural metric on the transformed elements. This process can be reviewed in the Jupyter notebook `11_nearest_asteroid.ipynb`. I initially tried working with the full empirical distribution to convert every orbital element to a percentile and then to a normally distributed z score, but the results didn't make any sense, so I dropped that approach.

Instead, I switched to a simpler approach and standardized variables so they would have mean 0 and variance 1. I attempt to make them close to normal if possible, but without overfitting against the empirical distribution. I standardized the log of the semimajor axis a and directly standardized the eccentricity e. (Even though this admits mappings from z to eccentirities outside $[0, 1]$, the mapping is only used in the direction from a reported eccentricity e to a transformed e_z that is approximately normal). The quantity $\sin(i)$ was also also standardized. Here is a summary of the mathematical transformations to create standardized (and hopefully approximately normal) variables from a, e and i:

$$
\begin{aligned}
a_z &= \frac{\log(a) - \mathrm{E}[\log(a)]}{\sqrt{\mathrm{Var}[\log(a)]}} \\
e_z &= \frac{e - \mathrm{E}[e]}{\sqrt{\mathrm{Var}[e]}} \\
i_z &= \frac{\sin(i) - \mathrm{E}[\sin(i)]}{\sqrt{\mathrm{Var}[\sin(i)]}}
\end{aligned}
$$

The expectation and variance here are estimated using the sample mean and sample variance respectively.

Figure 5.1 shows visualizations comparing the hypothetical and empirical distributions of a and e:

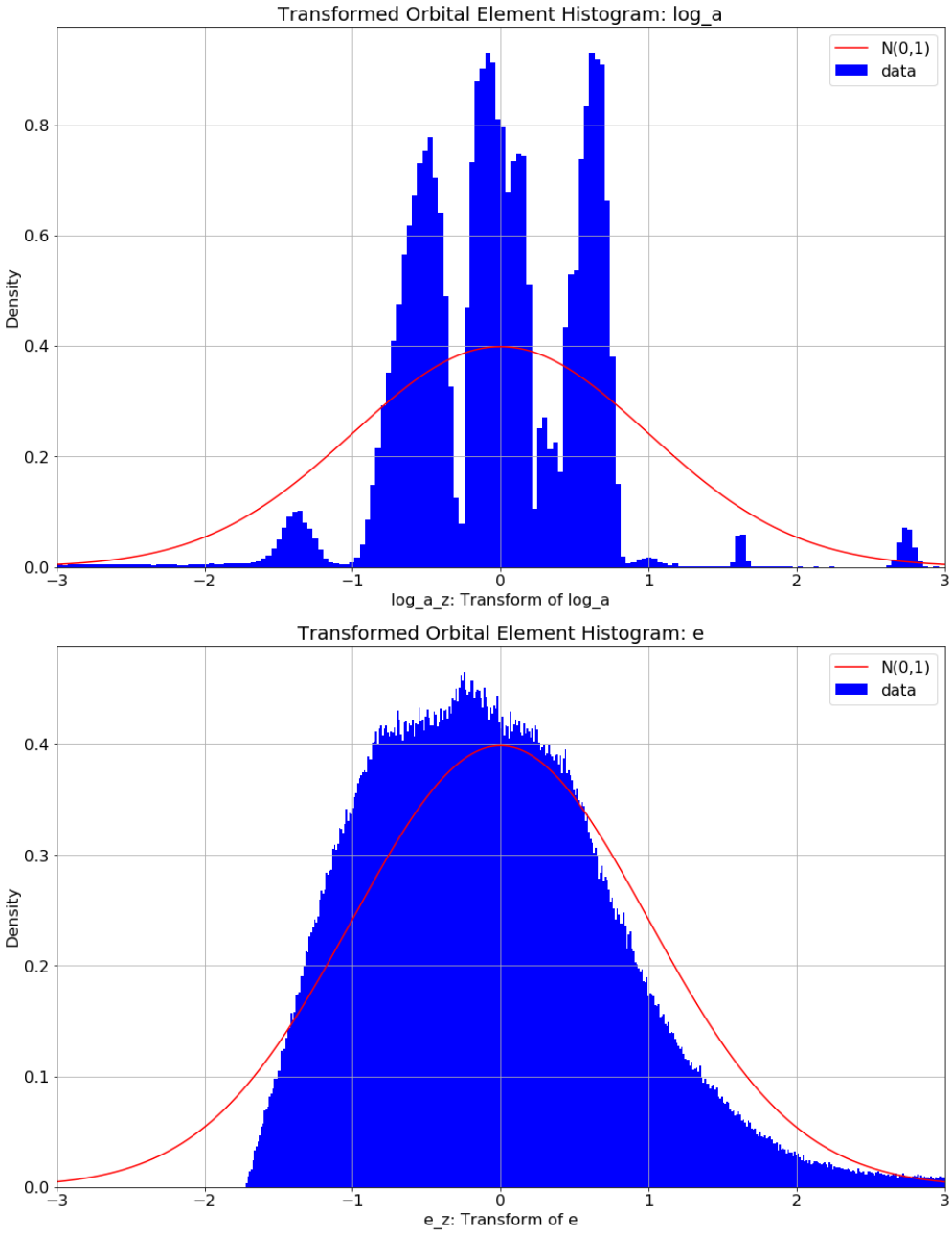

Figure 5.1: *Transformations of a and e to Standardized (and "Approximately" Normal?) Variables a_z and e_z.*

We can inject i into Cartesian space with only its sine because it is constrained to $[-\pi/2, \pi/2]$. But the other three angles are unconstrained. They are handled identically. I will take Ω as an example. I transform Ω into *two* variables, named `cos_Omega_z` and `sin_Omega_z`. These are not transformed empirically, but using a theoretical distribution. Let x be the sine or cosine of one of Ω, ω or f. x is mapped to a variable z that is distributed appproximately normal by applying the tranformation

$$u = \frac{1/2 + \arcsin(x)}{\pi}$$
$$z = \Phi^{-1}(u)$$

where Φ is normal CDF. The simple idea is that if the angle x distirubuted uniformly on the circle $[0, 2\pi]$, then u as defined here is uniform on $[0, 1]$ and z is standard normal.

Figure **??** shows plots comparing the hypothetical and empirical distributions of $\sin(i)$ and $\sin(\Omega)$. Better results for e and i could be obtained by using the Beta distributions noted above, but for this purpose the simple standardization is adequate. The plot shown for the tranform of $\sin(\Omega)$ shows that it is very close to the theoretical distribution. I generated analogous plots for the sin and cos of Ω, ω and f which are in the Jupyter notebook. They are qualitatively similar to this one and all show excellent fits.

I have now given a recipe by which six orbital elements can be injected into $\mathbb{R}^9$. Let X be the $N \times 9$ matrix of transformed elements (N = 733,489 is the number of asteroids). The orbital elements are only very lightly correlated with each other, and so are the X_j except for the tightly correlated pairs with the sin and cos of the same angle. Next, using the Spectral Theorem, I find a 9×9 matrix β such that the covariance matrix of $X\beta$ (which also has shape $N \times 9$) is the 9×9 identity matrix. The only wrinkle is that I assign importance weights to the 9 columns before building X and computing β. The importance weights are:

- 1.0 for a_z and e_z
- 0.5 for i_z
- 0.1 for the sin and cos of Ω, ω and f

These are admittedly qualitiative judgments on my part. I initially only compensated for the

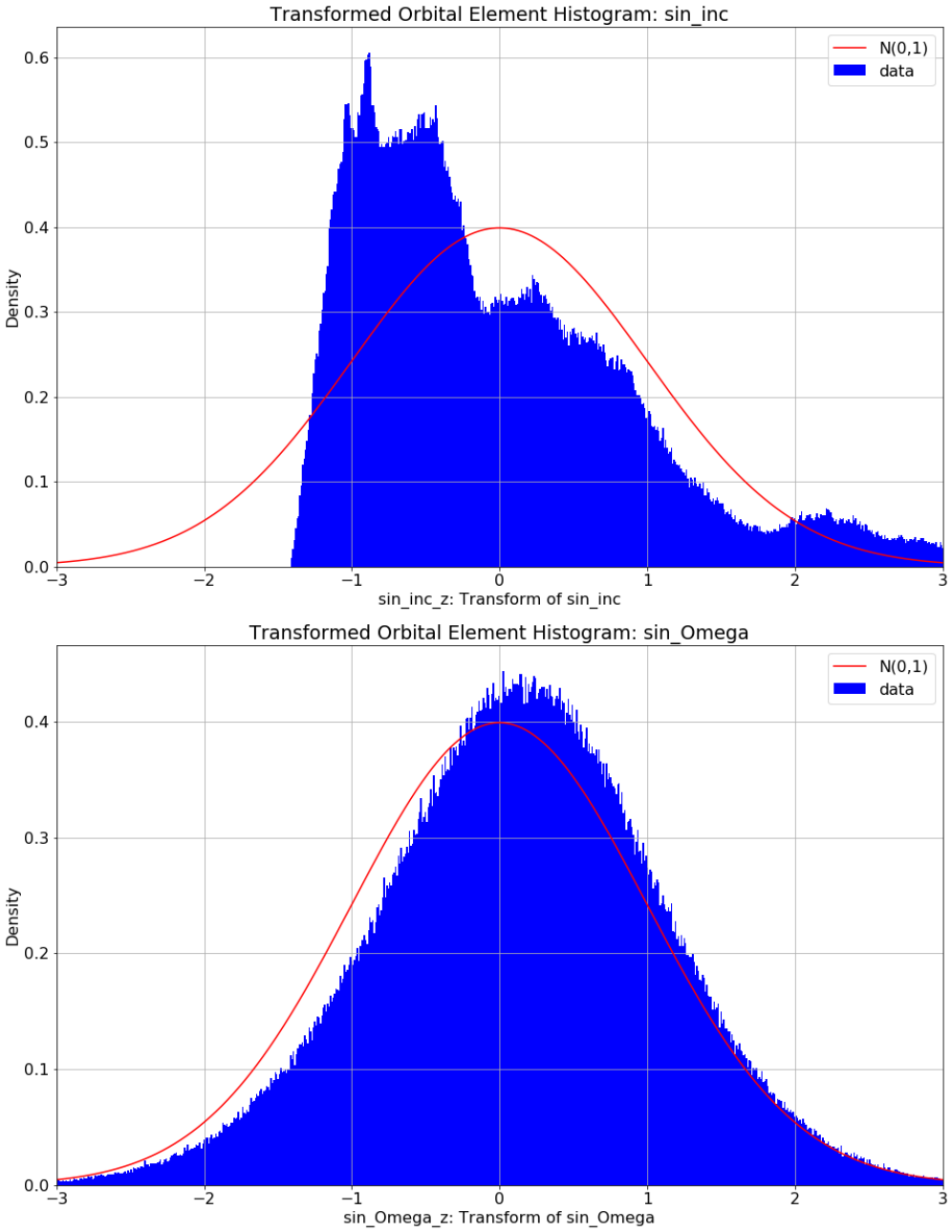

Figure 5.2: *Transformations of* $\sin(i)$ *and* $\sin(\Omega)$ *to Standardized (and approximately normal) Variables.*

double counting of Ω, ω and f, but I noticed that relatively small differences in e.g. ω on a near circular orbit that hardly effected the shape of an orbit were having a disproportionately large influence on the covariance score.

The covariance metric between two sets of orbital elements is ϵ_1 and ϵ_2 is defined by

$$\|\epsilon_2 - \epsilon_1\|_{\text{cov}} = \|\epsilon_2\beta - \epsilon_1\beta\|$$

The importance weights are rescaled so the diagonal of the covariance matrix sums to 1 and a random pair of elements would have distance 1. In practice I've observed that many pairs have norms far above 1. These calculations are also in `nearest_asteroid.py` and done by the functions `elts_to_X_cov`, `calc_beta`, `elt_q_norm` and `nearest_ast_elt_cov`. Now that we know what it means for two orbital elements to be "close," we are ready for our first test: recovering unperturbed elements.

5.2 Recovering the Unperturbed Elements of Known Asteroids

The first and easiest proof of concept for the search process is to see if the mixture parameters will converge correctly when the search is initialized with correct orbital elements for asteroids that are well represented in the data, but with "neutral" or uninformative mixture parameters. I liken this test to a kid learning to swing a bat by trying to ball sitting on a tee. This test is demonstrated in the Jupyter notebook `14_asteroid_search_unperturbed.ipynb`.

It's worthwhile to follow through the steps to assemble the data to get familiar with how everything fits together. These two lines of codes load the ZTF data observations associated with the nearest asteroid, and count hits by asteroid number:

```
ast_elt = load_ztf_nearest_ast()
ast_num, hit_count = calc_hit_freq(ztf=atf_ast, thresh_Sec=2.0)
```

The next few lines sort the asteroids in descending order by number of hits, and assemble a data frame of the orbital elements belonging to the 64 "best" asteroids The function `asteroid_elts` in `candidate_elements` provides a batch of candidate orbital elements that exactly match known asteroids; it assigns an `element_id` matching the original asteroid number to make it easy to check later if the fitted elements match the original. The function `load_ztf_batch` assembles the batch of ZTF observations within a threshold, here 2.0 degrees, of these candidate elements

```
elts_ast = asteroid_elts(ast_nums=ast_num_best[0:64])
ztf_elt = load_ztf_batch(elts=elts_ast, thresh_deg=2.0)
```

Reviewing the `ztf_elt` on screen we can see that there are 322,914 rows which include 10,333 hits: an average of 161.5 hits per candidate element and 3.2% of the total rows of data. A call to `score_by_elt` computes the t-score described earlier based on the mean and standard deviation of $\log(v)$. This shows a mean t-score of +45.0, which is off the charts good. It's interesting to see that a set of observations with 3.2% hits and 96.8% noise achieves such a good score. This also puts into context the challenge of the search problem: we have an average of 5,045 detections within 2.0 degrees of each set of candidate elements, of which 162 are hits and the remaining 4,883 are random detections belonging to other asteroids. If we want a search process to detect asteroids with as few as 8 hits in the data, we will need a process selective enough to pick out just 0.16% of the observations.

To initiate the search, we also need to choose our initial mixture parameters. We set `num_hits` to 10 and the resolution to 0.5 degrees

```
elts_add_mixture_params(
        elts=elts_ast, num_hits=10, R_deg=0.5, thresh_deg=2.0)
```

Now that we have the candidate elements and the ZTF data frame, we are ready to instantiate the asteroid search model:

```
model = AsteroidSearchModel(
        elts=elts_ast, ztf_elt=ztf_elt,
        site_name='palomar', thresh_deg=2.0)
```

Before we start training the model, we can get a plain text report or a visualization of the starting point. I will omit these here. The report shows that at the start of training, all 64 elements are "good" with 5 or more hits, and the overall mean log likelihood is 3.13 and the mean number of hits is 162.7. Here is an excerpt from the plain text model report at the end of training:

```
model.report()
Good elements (hits >= 5):  64.00

         \  log_like :  hits  :    R_sec : thresh_sec
Mean Good:  1096.95  : 162.55 :     3.01 :   165.95
Trained for 7808 batches over 122 epochs and 49 episodes (elapsed time 312 seconds).
```

Figures 5.3 and 5.4 illustrate the training progress. We can see that the model is behaving as hoped. It is gradually ratcheting the resolution parameter and scoring a high log likelihood as it does so. It does this without getting deked and polluting the originally correct orbital elements. These are very happy diagnostics. The log likelihood averages over 1,000 and the hits average 160.

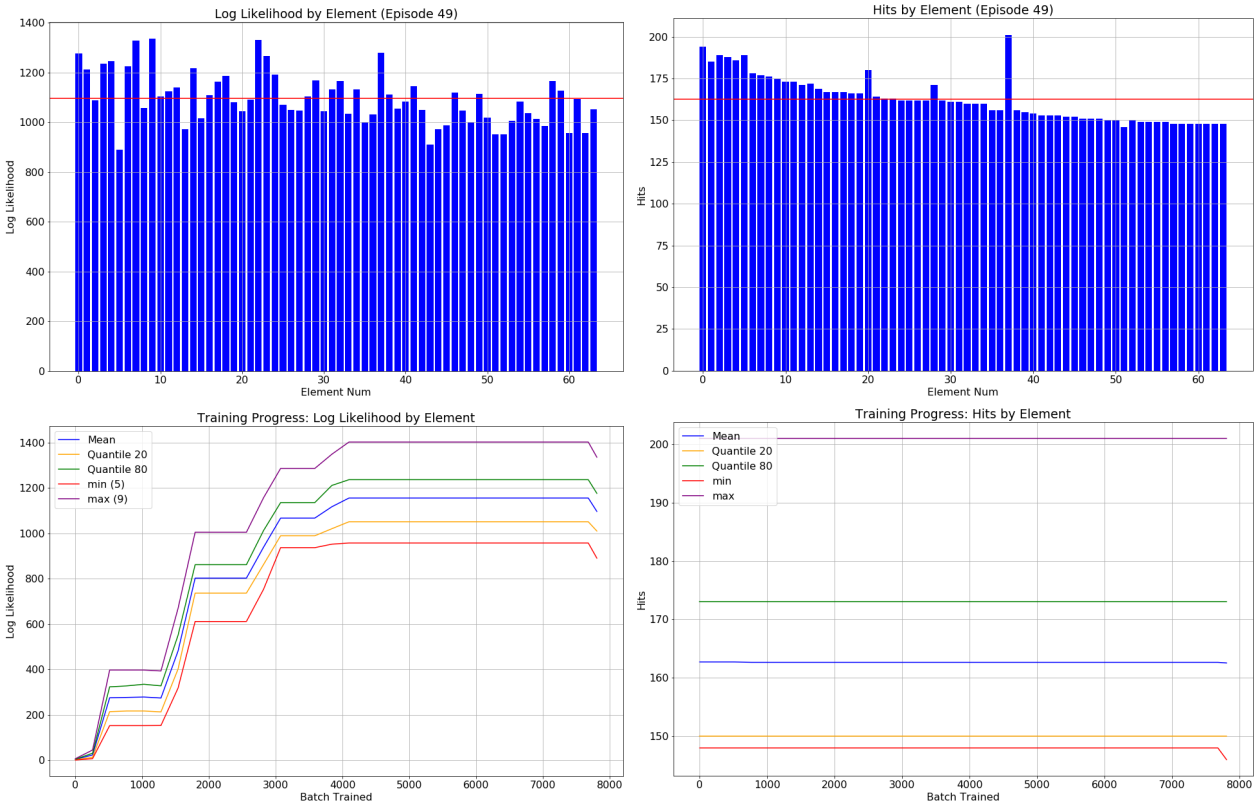

Figure 5.3: *Training progress on 64 unperturbed orbital elements.*

Figure 5.4: *The resolution R decreases monotonically when training the unperturbed elements.*

An earlier iteration of the training protocol that did not roll back training when hits were lost in an episode did considerably less well.

The diagnostics presented above work equally well for any set of candidate orbital elements, whether or not they are ostensibly associated with a known asteroid. In this case, we can further validate the results by comparing our fitted elements to the nearest asteroid using the two metrics described in the previous section. The simplest test is how many of 64 recovered elements have as their nearest asteroid the same asteroid used to initialize the elements. The answer is 64: the fitting process "tried" to converge back to the right asteroid every time. A more substantive question is how close did it come. Here are the summary statistics of two metrics:

- Mean distance in AU for 240 test points: 4.57E-8 (median over 64 elements)
- Covariance Norm of elements: 5.70E-6 (geometric mean)

This is an excellent level of agreement. The covariance norm is a good summary statistic, but may be hard to relate to astronomy. The mean absolute error in the recovered a is 4.3E-5 and in the recovered e is 1.0E-5. Figure 5.5 illustrates the distance in AU and covariance norm to the nearest (original) asteroid for all 64 candidate elements. Thanks to the one way ratchet that prevents it from losing hits once they acquired, the model can hit a tee ball out of the infield. In the next section we will see how it fares against a moving target.

5.3 Recovering the Perturbed Elements of Known Asteroids

5.3.1 Small Perturbation

The next experiment is similar to the previous one. This time we will apply a small perturbation to the orbital elements in our initial guess. If the last experiment was like hitting a tee ball, this one may be likened to hitting a ball gently pitched by your little league coach in batting practice. The elements are perturbed using the function `perturb_elts` in `candidate_elements.py`. The perturbation adds normally distributed random noise with the specified standard deviation to $\log(a)$, $\log(e)$, and the four angles i, Ω, ω and f. The small perturbation shifts $\log(a)$ by 0.01, $\log(e)$ by 0.0025, i by 0.05 degrees, and the the other angles by 0.25 degrees. A random seed is used for reproducible results. The code to do this is

```
elts_pert= perturb_elts(elts_ast, sigma_a=0.01, sigma_e=0.0025,
        sigma_inc_deg=0.05, sigma_f_deg=0.25,
        sigma_Omega_deg=0.25, sigma_omega_deg=0.25,
        random_seed=42)
```

Last time the pre-training summary statistic based on $\log(v)$ showed a very positive t score. This time the t score has dropped to $+3.71$, and the model has zero hits before it begins training. Even this small perturbation is enough that the model is going to have to work quite a bit to recover the elements. Here is the text report after sieving:

```
Good elements (hits >= 5):  42.00
         \  log_like :  hits  :    R_sec : thresh_sec
Mean Good:   798.24  : 117.50 :    36.06 :   779.79
Mean Bad :    42.97  :   0.64 :   266.21 :  2374.53
GeoMean  :   218.79  :  20.01 :    42.97 :   934.96
Trained for 15552 batches over 243 epochs and 105 episodes (elapsed time 751 seconds
```

Here are summary statistics for the run on the small perturbation of real asteroid elements:

- Successfully converged for 42 out of 64 candidate elements (65.6%)
- Mean hits on converged elements: 117.50
- Resolution on converged elements: 18.2 arc seconds
- Distance in AU to nearest asteroid: 2.58E-4
- Covariance Norm to nearest asteroid: 1.22E-2

These results are not as strong as on the unperturbed elements, but the method is still clearly working. It's coverged on almost two thirds of the candidate orbital elements. The converged elements are fit well, averaging 117 hits at 18 arc seconds. The distance to the nearest asteroid is 2.58E-4 AU, which is still very close and an excellent description of the orbit. The nearest asteroid to the recovered elements matches the original asteroid on 48 out the 64 elements.

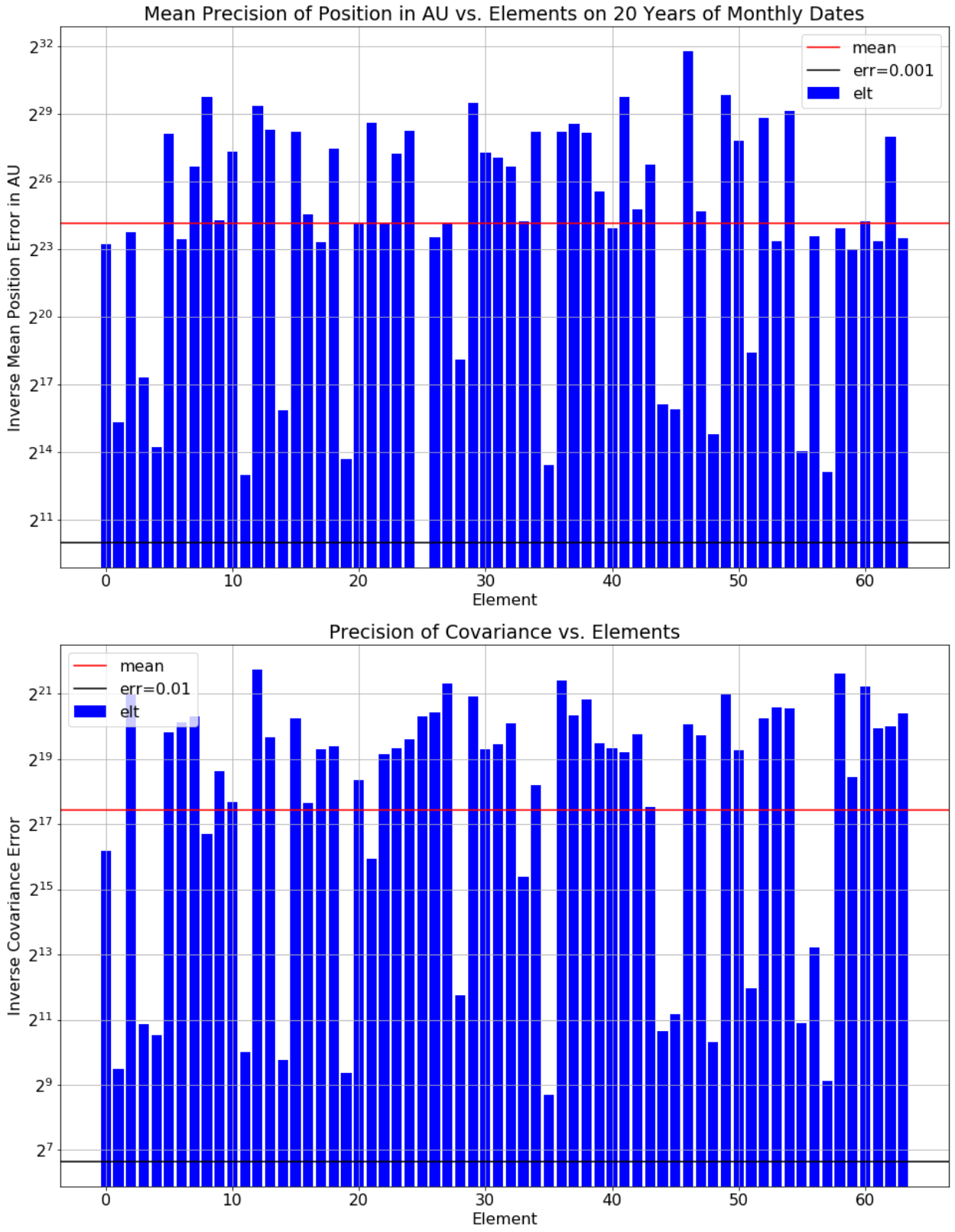

Figure 5.5: *Two Metrics Comparing the Recovered Orbital Elements to the True Elements of the Asteroid in Question. Both charts are plotted on a log scale with preicision (reciprocal of the error) on the y axis. The geometric mean error is shown in red: 6.61E-6 AU and 2.00E-3 on the covariance norm.*

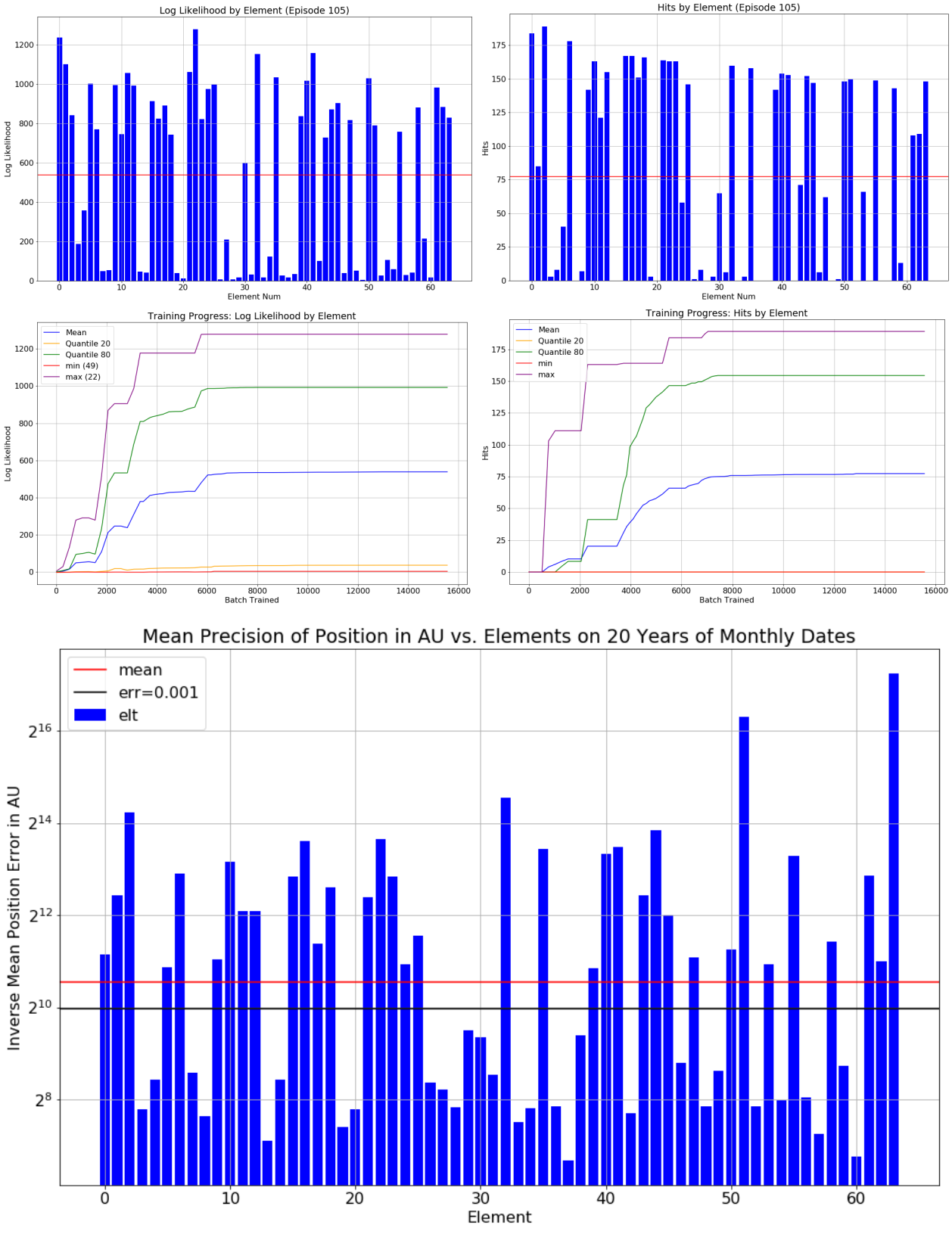

Figure 5.6: *Training Progress on 64 Orbital Elements Initialized with Small Perturbations from Real Asteroids. 42 of the 64 candidate elements converge, averaging 118 hits each.*

5.3.2 Large Perturbation

In our third test, we will again start with perturbed orbital elements. But this time, we will apply a larger perturbation, about five times larger. This is a much harder task. To continue with the baseball analogy, it might be likened to facing a high school pitcher. The perturbation size this time is 0.05 on $\log(a)$, 0.01 on $\log(e)$, 0.25 degrees on i, and 1.0 degree on the other three angles. While this might not sound like much at first, they are large perturbations. In fact, they are so large they led me down a painful rabbit hole. I repeatedly failed to recover the orbital elements of the original asteroids before I realized that the perturbations were large enough that in many cases, the nearest asteroid to the perturbed elements was no longer the original asteroid! The results started to make much more sense when I compared each fitted element to the nearest real asteroid, regardless of whether this matched the original source of the elements before perturbation.

Here is the text report after sieving:

```
Good elements (hits >= 5):  12.00
         \  log_like :  hits  :    R_sec : thresh_sec
Mean Good:   748.07  :  98.17 :    61.36 :  1178.94
Mean Bad :    30.35  :   0.58 :   261.02 :  2296.18
Trained for 13120 batches over 205 epochs and 72 episodes (elapsed time 533 seconds).
```

Here are summary statistics for the run on the large perturbation of real asteroid elements:

- Successfully converged for 12 out of 64 candidate elements (18.8%)
- Mean hits on converged elements: 98.2
- Resolution on converged elements: 32.4 arc seconds
- Distance in AU to nearest asteroid: 4.45E-4
- Covariance Norm to nearest asteroid: 3.32E-2

This time we've only converged on 12 of the 64 orbital elements. But the encouraging news is that when we have converged, the fit is still adequate. The average hits are 98 and the resolution is 32.0 arc seconds. The distance to the nearest asteroid is somewhat larger to the batch initialized with small perturbations, at 4.5E-4. This is telling us something important: when the model starts from a good enough guess that it has a path in search space to a local maximum, it will converge to

an adequate solution. Starting from a poor initialization will reduce the probability of successful convergence, but it doesn't dilute the quality of the results.

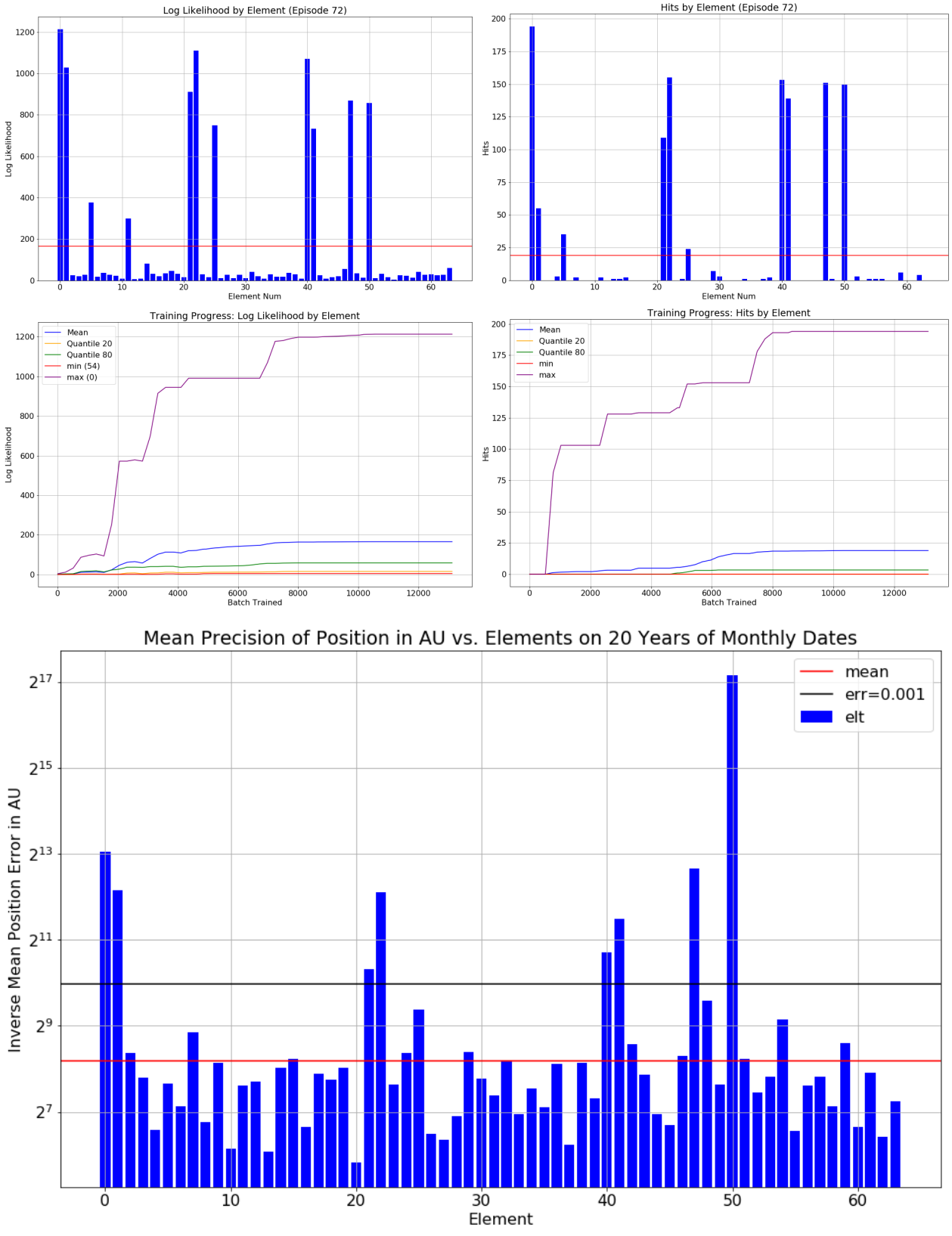

Figure 5.7: *]*

Training Progress on 64 Orbital Elements Initialized with Large Perturbations from Real Asteroids. 12 of the 64 elements converge, averaging 98 hits.

5.4 Searching for Known Asteroids with Random Initializations

Our final test case before search for new asteroids is to attempt to recover asteroids in the known catalog, but without peeking at the answers. I will now transition from small tests on a single batch 64 candidate elements to analyisis of the results of a large scale computational job. We've entered the major leagues. [1] The program `asteroid_search.py` can be run from the command line. It searches against one of two subsets of the ZTF data set. When run in "known asteroids" mode, the ZTF observations are filtered to include only the 3.75 million rows that are within 2.0 arc seconds of a known asteroid. When run in "unknown asteroids" mode, it searches in the complement, the 1.95 million ZTF detections that are at least 2.0 arc seconds from a known asteroid. The other arguments include the range of random seeds `seed0` to `seed1` and a stride to support parallelization.

This program was run on approximately 4096 random seeds against the known asteroids over the better part of a week. Once a ZTF observation was associated with a set of candidate elements, it was subtracted from the data set so it would not be included in subsequent fits. I filtered the results to those with at least 8 hits and a resolution of at most 20 arc seconds. The results are reviewed in the Jupyter notebook `19_search_known.ipynb`. Since this was a search against known orbital elements, we can gauge the quality by measuring the distance of the recovered orbits to the nearest known asteroid. Here are the summary statistics for the resulting fitted elements:

- 125 fitted orbital elements were found
- they had 19.20 hits on average
- the geometric mean resolution was 9.3 arc seconds
- the geometric mean distance to the nearest asteroid was 2.66E-3 AU
- the geometric mean covariance norm to the nearest asteroid as 0.73

[1] Sadly, trying to fit asteroids with random initializations is a bit like facing Roger Clemens, hoping to get hit by the pitch so you can get on base cheaply.

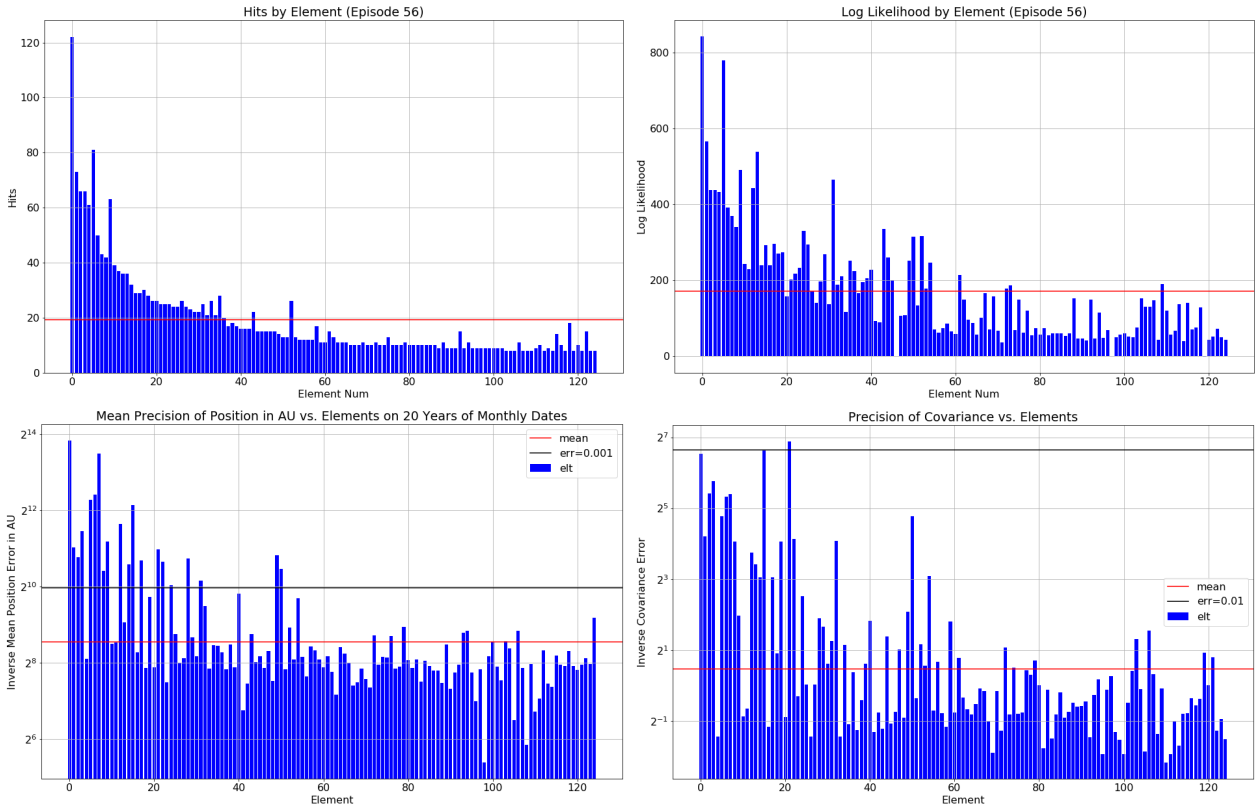

Figure 5.8: *125 Orbital Elements Fitted to Observations of Known Asteroids.*
The first row shows internal estimates of training quality: number of hits and log likelihood.
The second row shows distance to the nearest known asteroid in mean distance of orbit and covariance metric.

Figure 5.8 shows training results for the fitted elements. The first row has the internal metrics number of hits and log likelihood. The second row has the comparison to the known asteroids. I thought these results were respectable but not great. I will need to significantly improve the initializations to get a higher yield. The plots with the precision to the nearest asteroid are uneven. Some of these elements have clearly achieved excellent agreement with known asteroids, with positions accurate to 1E-3 AU or better and covariance norms better than 0.1 (this is very close in orbital element space). While there is room for improvement, I view this as a successful proof of concept that this technique can recover correct orbits of asteroids in the catalogue without peeking at the correct orbital elements.

5.5 Presenting 9 New Asteroid Candidates

The search for new undiscovered asteroids is very similar to the search discussed in the previous section. The only difference is that this time we limit the search to the subset of 1.95 million ZTF observations more than 2.0 arc seconds away from any known asteroid. A search was carried out starting from 4,096 random seeds, taking about 5 days running on 4 GPUs in parallel. This search identified 9 candidate orbital elements that achieved at least 8 hits within 10 arc seconds and a converged resolution less than 20 arc seconds. These are the same criteria used to select the ostensibly high quality recovered orbital elements of known asteroids presented in the last section.

element_id	a	e	inc	Omega	omega	f	epoch	num_hits	R_sec	thresh_sec	num_rows_close	log_like
178421	3.160327	0.089064	0.153620	2.668766	4.773995	-0.463213	58600.0	10.996569	3.773662	350.229950	15.0	68.716019
3308	3.026962	0.119945	0.129409	3.903006	4.525979	4.819063	58600.0	9.996422	3.821569	351.971649	14.0	63.716438
44117	2.935863	0.187419	0.124516	3.166528	1.230836	-3.122138	58600.0	9.994445	5.225097	357.466431	16.0	61.298466
170789	2.735335	0.152867	0.403704	6.038029	3.016815	-3.443415	58600.0	9.996418	7.083936	347.111053	13.0	60.860794
113970	2.897024	0.068932	0.209250	5.663728	3.868474	4.450756	58600.0	7.999145	3.958026	348.778046	9.0	60.584415
45801	2.754677	0.047293	0.118126	3.139070	5.767782	-1.949476	58600.0	8.996888	4.321019	326.387482	14.0	56.090981
50775	2.374712	0.100280	0.165483	4.280114	5.955170	2.995417	58600.0	9.948814	9.705722	809.804260	29.0	55.641151
96507	2.820351	0.068964	0.080233	2.222034	0.960931	-2.630966	58600.0	8.821539	3.905472	262.245087	12.0	52.416988
191915	2.315446	0.192885	0.057156	2.130249	2.865086	-4.122024	58600.0	7.982198	9.668445	388.469055	21.0	38.837540

Figure 5.9: *Orbital elements of 9 candidate asteroids.*

Figure 5.9 shows the candidate orbital elements. Here are the summary statistics for the new asteroid candidates:

- 9 fitted orbital elements were found
- they had 9.11 hits on average
- the geometric mean resolution was 5.3 arc seconds
- the geometric mean distance to the nearest asteroid was 4.10E-3 AU
- the geometric mean covariance norm to the nearest asteroid as 1.10

Figure 5.10 shows the fit quality and distance to the nearest asteroid, as with the known asteroid search. These results are encouraging but not quite as definitive as I had hoped. The

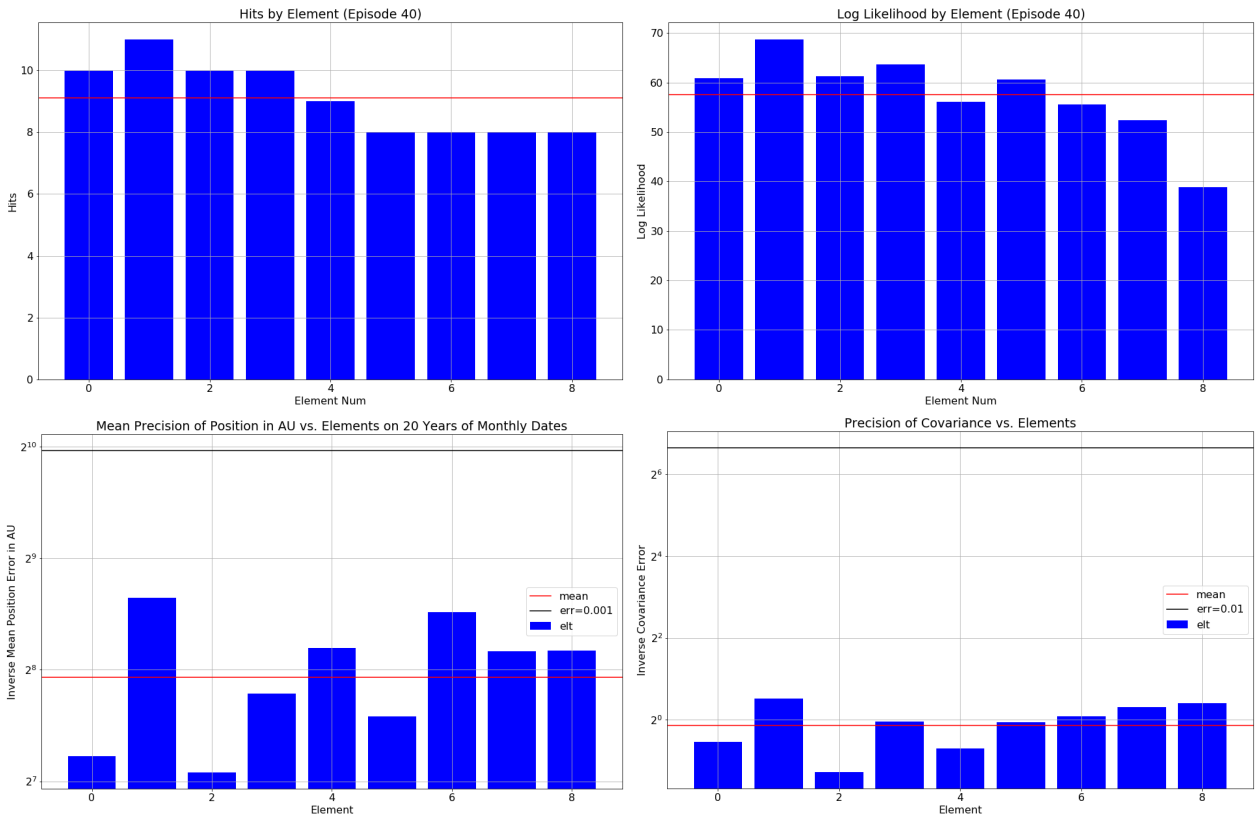

Figure 5.10: *9 Orbital Elements Fitted to Observations of Unknown Asteroids.*
The first row shows internal estimates of training quality: number of hits and log likelihood.
The second row shows distance to the nearest known asteroid in mean distance of orbit and covariance metric.

convergence with 9.1 hits to a resolution of 5.3 arc seconds strikes me as excellent. A RA/Dec observation has 2 degrees of freedom, so a collection of 6 free parameters (the orbital elements) is matching 18 degrees of freedom of data at a composite tolerance of about 5 arc seconds. Log likelihood scores in the range of 40-60 lead me to believe to me that this process has successfully identified collections of observations that are highly likely to belong to the same or perhaps related objects, and that it has fitted orbital elements to the observations to a good tolerance.

The news is not quite as a good when I look at the distance to the nearest asteroid in the catalogue. I had hoped to present compelling evidence that the recovered elements trained against the known asteroid subset of the ZTF detections were much closer to the nearest known asteroid than these elements. While it is true that the these elements are further away by a factor of 1.54 in mean distance, and 1.51 on the covariance norm, that is not a big margin. I have to admit here the possibility that while these elements correspond to a real object, they might have converged to

elements close to those of a known asteroid. When I set up the training to exclude detections near to known asteroids, I thought it was unlikely that it would converge to a known asteroid, and I still believe that to be true. There is a possibility that these candidates might be better interpreted as proposed revisions to the orbital elements of nearby asteroids than as new, undiscovered objects.

Figure 5.11 shows the ZTF hits the search process has associated with each set of candidate elements. We can gain understanding by paying close attention to the ObjectID column. These are provisional assignments by ZTF indicating that multiple detections are likely to belong to the same object. Take four example candidate elements in turn.

Element `178421` is stringing together four detections of object `ZTF18aboluox` and 4 detections of object `ZTF18acewaex`. They were seen 433 days apart. The magnitudes are very different, so this is probably a spurious connection between two different objects with compatible orbits.

Element `3308` is piecing together 6 detections of `ZTFV18abtpdzg` and 2 detections of `ZTF18abspkzw`. These detections were made 193 days apart. The magnitudes of these are compatible, making it far more plausible that this is a legitimate connection between two sets of detections that may not have been made previously.

Element 191915 is again stringing together 6 + 2 detections of different objects (`ZTF18abtxgd` and `ZTF19abtsqmn`) made 469 days apart. The magnitudes are all compatible. This too seems likely to be a legimate and novel connection between different sets of detections.

Element `170789` is an example of a fit made by stringing together just one set of detections made at similar times. All of the detections are of the same ZTF object `ZTF17aaqwwg` made on the same date, indeed made in a time span of only 55 minutes. The model has independently matched the conclusion of the ZTF classifier that all 8 detections belonged to the same object. While I would like to marshall further evidence that these candidate elements belong to new objects, I find the detailed analysis of the claimed series of ZTF hits to be persuasive and encouraging. This search process is clearlying identifying plausible connections between related detections by brute force analysis of their location in the sky.

element_id	ObjectID	mjd	ra	dec	mag_app	s_sec
178421	b'ZTF18aboluox'	58430.166620	346.704046	-10.675142	15.787200	1.260372
178421	b'ZTF18aboluox'	58430.170313	346.704079	-10.675160	15.800000	0.982009
178421	b'ZTF18aboluox'	58430.166620	346.704001	-10.675028	15.587700	1.699850
178421	b'ZTF18aboluox'	58430.170313	346.704073	-10.675099	15.654600	1.175734
178421	b'ZTF18acewaex'	58863.138472	67.229666	17.295098	19.328100	6.219038
178421	b'ZTF18acewaex'	58863.138472	67.229789	17.295055	19.343500	5.789846
178421	b'ZTF18acewaex'	58863.152465	67.229835	17.295083	19.283501	2.389730
178421	b'ZTF18acewaex'	58863.152465	67.229783	17.295077	19.263100	2.392794

element_id	ObjectID	mjd	ra	dec	mag_app	s_sec
191915	b'ZTF18abtxtgd'	58430.170313	341.181109	-12.234676	19.693501	0.036218
191915	b'ZTF18abtxtgd'	58430.166620	341.181092	-12.234595	19.260099	2.621655
191915	b'ZTF19abtsqmn'	58899.139884	94.436593	22.583914	18.914301	6.521414
191915	b'ZTF19abtsqmn'	58899.140336	94.436579	22.583982	18.900700	6.623162
191915	b'ZTF19abtsqmn'	58899.192569	94.436720	22.583899	19.819700	0.946245
191915	b'ZTF19abtsqmn'	58899.222442	94.436672	22.583855	19.873501	4.829147
191915	b'ZTF19abtsqmn'	58899.220162	94.436545	22.583886	20.234699	4.129297
191915	b'ZTF19abtsqmn'	58899.220613	94.436587	22.583902	20.670900	4.263810

element_id	ObjectID	mjd	ra	dec	mag_app	s_sec
3308	b'ZTF18abtpdzg'	58670.439884	354.565806	-8.964600	17.605200	3.286346
3308	b'ZTF18abtpdzg'	58670.440336	354.565888	-8.964426	17.601101	3.888707
3308	b'ZTF18abtpdzg'	58670.462604	354.565787	-8.964429	16.858500	0.697156
3308	b'ZTF18abtpdzg'	58670.463056	354.565909	-8.964536	17.128700	1.329765
3308	b'ZTF18abtpdzg'	58670.462604	354.565824	-8.964445	17.092899	0.837276
3308	b'ZTF18abtpdzg'	58670.463056	354.565809	-8.964412	16.658600	0.769468
3308	b'ZTF18abspkzw'	58863.110058	346.642277	1.047679	16.541800	8.282505
3308	b'ZTF18abspkzw'	58863.109606	346.642261	1.047839	16.946899	7.604068

element_id	ObjectID	mjd	ra	dec	mag_app	s_sec
170789	b'ZTF17aaaqwwg'	58903.113588	84.725827	15.284655	19.437901	7.917067
170789	b'ZTF17aaaqwwg'	58903.116806	84.725856	15.284678	19.699699	7.043546
170789	b'ZTF17aaaqwwg'	58903.129097	84.725897	15.284685	20.146000	3.737848
170789	b'ZTF17aaaqwwg'	58903.126840	84.725899	15.284678	19.689501	4.333303
170789	b'ZTF17aaaqwwg'	58903.128194	84.725824	15.284668	19.916201	3.767049
170789	b'ZTF17aaaqwwg'	58903.149248	84.726023	15.284693	19.143600	3.802050
170789	b'ZTF17aaaqwwg'	58903.149699	84.725870	15.284728	19.707399	4.209481
170789	b'ZTF17aaaqwwg'	58903.151053	84.725889	15.284665	19.904301	4.672150

Figure 5.11: *ZTF Hits Associated with 4 of the New Asteroid Candidate Elements.*

5.6 Conclusion

In this thesis I have presented a novel technique for learning orbital elements of asteroids from telescopic data: searching in the space of orbital elements. This approach has significant theoretical advantages over methods that rely on stringing tracklets together, which suffer from a combinatorial explosion. I have demonstrated a working prototype and shown that it can converge well to correct orbital elements from a suitable initialization. This prototype also illustrates how to perform large scale and effient astrometric computations on GPUs using TensorFlow, which is a second potentially novel contribution. I have further demonstrated that even naive random initializations are sufficient to reidentify up to 125 known asteroids, albeit to varying degrees of quality. Finally, I have proposed candidate orbital elements for up to 9 previously unknown asteroids. By analyzing the details of the ZTF detections associated with each candidate element, I have provided anecdotal evidence that the search is identifying detections that plausibly belong to the same object.

5.7 Future Work

5.7.1 Intelligent Initialization of Candidate Elements

When I first conceived this project, I intended to devote substantial time to formulating intelligent initializations of the candidate orbital elements. The random orbital elements were intended as a quick and dirty placeholder to get the system up and running. Due to the looming deadline I have had to defer that to future work. The results presented for perturbed orbital elements were very encouraging, showing excellent convergence. If we can identify a set of candidate orbital elements that are close to real ones, this process should efficiently grab all the other detections in the data set that are consistent and combine them.

The ZTF data set includes an ObjectID column that represents a preliminary assessment of which detections are related. A straightforward method for initializing candidate orbital elements is to take these classifications at face value. For each ObjectID, take the collection of k detections and use a traditional technique (e.g. least squares fitting) to find orbital elements consistent with them. In fact, the existing `AsteroidSearch` class provides most of the code required to do this. To adapt it to this problem, we would ignore the mixture parameters and minimize the least squares loss rather than maximizing the log likelihood.

Here is a more ambitious idea for intelligent initialization. By looking at the subset of ZTF detections within 2.0 arc seconds of a known asteroid, we can generate a large collection of training data of pairs of ZTF detections that we are highly confident belong to the same object. We can randomly sample other pairs of ZTF detections, and use the two data sets to train a binary classifier to predict the probability that a pair of ZTF detections belong to the same object. This classifier could be used to come up with tracklets that might not have the same ObjectID. An additional classifier might also be trained to try to extend a tracklet with a third detection. Each detection has 2 degrees of freedom, and an orbital element has 6. So a single tracklet allows us to sample a 2D space of candidate element consistent with it, while three or more detections should in theory give us a uniquely determined initialization point.

5.7.2 Incorporating Additional Data

The method presented doesn't use anything special about the ZTF data set besides for the filtering of likely asteroid detections. While the ZTF data set is excellent, it only dates back effectively to the middle of 2019. An obvious step to improve these results would be to bring in a second major data source. I would probably start with Pan-STARRS. Ideally I would like to identify a subset of detections that have already been classified as likely to be near Earth objects. Failing that, we could start with all Pan-STARRS data; remove bogus detections with a real-bogus classifier; and subtract out detections of stars and galaxies by using the catalogue of their known positions in the sky. The code would need to be changed in some places, but the change would not be major.

5.7.3 Incorporating Magnitude

I developed code in the `AsteroidSearcch` model to predict the magnitude of a detection using the H-G model. I also had a version of the log likelihood that incorporated a joint probability density on both direction and magnitude. The theoretical advantages of an estimation framework that incorporates magnitude are substantial. The model will be much less likely to make spurious connections of detections with compatible orbits that can't be the same object because they have different brightnesses. An added benefit is that the fitting procedure would give estimates for H and G. Unfortunately, I was not able to get the fitting process to converge well when I included the magnitude in the joint probability. As time drew short, I was forced to remove it from the model presented here and punt it to future work. I am highly optimistic that given enough time, this can be made to work and will sharpen the results.

5.7.4 Rebuilding the Known Asteroid Catalogue

If the three improvements above are made, I believe there is a good chance this model might be able to rebuild a large portion of the known asteroid catalogue. We saw that just seven months of ZTF data included over 100,000 asteroids with 10 or more hits, encompassing 13.6% of the known catalogue. The random initialization is not remotely up to the task of recovering all these elements, but with intelligent initialization alone we would be in the game. I think it would be a powerful proof of concept if we could run a program that would crunch through tens or hundreds

of millions of detections and generate an output with large overlap against the known asteroids. At a minimum, it would validate a single automated procedure that could also detect new objects. It might also allow us to gradually improve the quality of the data in the known asteroid catalog.

The ultimate goal of this body of work could be to build a single, fully automated computational pipeline for asteroid classification. Telescopic detections from multiple surveys would go in, optionally alongside snapshots of the known asteroid catalogue. The output would include an association of detections to known objects; revisions to the catalog where necessary; and proposed additions to the catalogue when supported by new detections.

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)