Statistically Weighted Orbital Elements for Kuiper Belt Objects

Steven Maggard

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Darin Ragozzine, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Statistically Weighted Orbital Elements for Kuiper Belt Objects

Steven Maggard
Department of Physics and Astronomy, BYU
Bachelor of Science

Thousands of asteroid-like objects reside in the Kuiper Belt Region. For accurate dynamical classification, the precision of their orbits needs rigorously tested. Using an analysis pipeline we created, we generated 30 statistically-weighted orbital clones for over 2000 Kuiper Belt Objects (KBOs). These orbits are integrated backwards in time 50 Myr. We created a database from the propagated orbits, from which we calculated the proper orbital elements for each KBO. We used the method established by Ragozzine and Brown (2007) to determine each KBOs relation to the dwarf planet Haumea. Currently, we have more than tripled the number of Haumea Family Members established by Ragozzine and Brown (2007). We conclude that other collisional families can be found using similar methods applied to Haumea and the orbital database we created.

Keywords: Haumea, Collisional Family, Kuiper Belt, KBOs, TNOs

ACKNOWLEDGMENTS

Professor Chip Galloway, for instilling in me a solid understanding of calculus. Without such, I could never have delved into the realm of physics. Drs. Denise Stephens, Eric Hintz, and Joseph Moody for guiding a complete novice in the field of astronomy. My rather large family. If I could survive six sisters, few things can present much difficulty. Last but not least, Dr. Darin Ragozzine; for guiding my way as I journeyed into the world of dynamical astronomy.

Contents

List of Figures

Chapter 1

Introduction

From an early age, most individuals are familiar with the planets that orbit the Sun. Few, however, learn about the Kuiper Belt region without taking an active effort to delve into astronomy.

The Kuiper Belt is a region of our solar system that extends from Neptune's orbit to about fifty astronomical units from our Sun (AU–the average distance between the Earth and the Sun). Thousands of asteroid-like bodies–dubbed Kuiper Belt Objects (KBOs)–reside here. The categorization of Pluto as a dwarf planet arose from the discovery of KBOs with comparable size to the former ninth planet. The current list of dwarf planets in the Kuiper Belt includes: Pluto, Eris, Makemake, and Haumea.

Haumea is of special interest. This dwarf planet is shaped like a football and spins very rapidly. It completes one rotation on its axis in a mere four hours. This rapid spin originates from a collisional impact during the early stages of the solar system. This collision flung shards of Haumea's surface throughout the Kuiper Belt. These shards, along with Haumea, comprise the Haumea collisional family. In 2007, Ragozzine and Brown published a paper and identified seven KBOs as part of this collisional family.

The motivation for this thesis builds upon their efforts as well as setting a foundation to better understand the Kuiper Belt and the asteroid-like objects found in the region. We calculate the orbits for each KBO and identify how accurate each calculated orbit is. To compensate for small gravitational perturbations over the age of the Solar system, we take the calculated orbits and project, or integrate, them backwards in time. We catalog these integrations to use in dynamical analysis and demonstrate how to use the database by identifying several candidate Haumea Family members. The process of calculating orbits from observational date to storing integrated orbits comprises the BUNSHIN analysis pipeline we created for our research.

1.1 Propagating Orbital Elements

Orbits consist of several elements. With respect to this research, the semi-major axis, eccentricity, and inclination are of greatest importance. The semi-major axis is the element that describes the size of an orbit. Often, units of AU are used to describe the size of the orbits of bodies found in our solar system. For example: Neptune's orbit has a semi-major axis of 30 AU. The eccentricity is a dimensionless value that describes the shape of the orbit. The closer the eccentricity is to zero, the more circular the orbit. As the eccentricity approaches one, the orbit becomes closer to elongated oval. If the eccentricity is equal to or greater than one, we are now dealing with an unbound parabolic or hyperbolic orbit. The inclination refers to the angle between the plane of the KBO' s orbit compared with the plane of Earth's orbit.

The Minor Planet Center gathers observational data on every sub-planetary body observed in our solar system and compiles the data into a database. They also calculate the orbit for each object, but fail to specify the uncertainties of the calculations. Thus, the accuracy and reliability of the orbits are unknown. Additionally, the orbital elements (i.e. the semi-major axis, eccentricity, and orbital angles) are not constant, but vary over time. Without further refinement of the data, dynamical classification can never reach an acceptable level of precision.

Of the methods available for orbital calculation, we have chosen to use the Open ORB program.

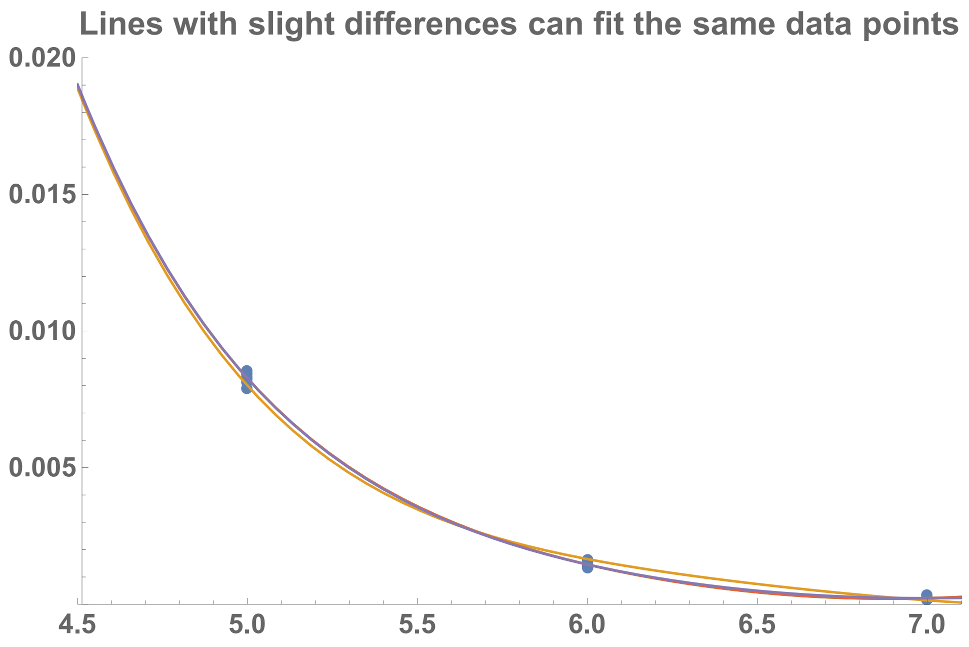

Figure 1.1 Multiple lines with slight variations can match a given set of data. Likewise, we use a set of orbital clones to find the parameters to match the observational data for every KBO. For clarity, this plot contains no important data apart from highlighting why we used orbital clones.

Mikael Granvik et al.(Granvik et al. 2009) developed the OpenORB program as a method to rigorously calculate orbits for objects in our solar system. This program generates a specified number of orbital clones, or orbits that match observational data, and calculates the statistical weight of each clone. From these statistical weights, we can determine the accuracy of any orbit used for dynamical classification. Fig 1.1 is a simplified expression of how multiple lines can fit the same set of data. Observational data are the recorded position of objects at specific points in time and each orbital clone has a calculated orbit that moves the object through the each recorded position.

With refined orbtital calculations, we can project the paths of these orbits backwards through time. REBOUND (Rein & Liu 2012) is a program developed to project, or integrate, orbits over a designated period of time. We used REBOUND to integrate the orbital clones 50 Myr to calculate the proper elements. Proper elements are the averaged, osculating value of the orbital elements over a period of at least 50 Myr. The likelihood of finding collisional families increases as more KBO's converge to a single point within the history of the solar system.

Using OpenOrb and REBOUND, we created our own database of statistically weighted proper elements for multiple KBO's that can be used for dynamical classification. We demonstrate the potential this database has by building off the efforts made by Darin Ragozzine and Mike Brown (Ragozzine & Brown 2007). In their publication, they identified seven KBOs that originated from the surface of the dwarf planet Haumea. With our efforts, we have tripled the number of known Haumea family members. Other students at BYU use our database to find other collisonal families in the Kuiper Belt.

1.2 The Haumea Collisional Family

Haumea is a dwarf planet found in the Kuiper Belt region with a shape that resembles a football. (Ragozzine & Brown 2007) This KBO is one of the fastest rotating objects in our solar system, completing one rotation in just four hours. The rapid spin and peculiar shape are due to a collisional impact during the early stages of the solar system that sent shards of Haumea's surface flying throughout the Kuiper Belt. Ragozzine and Brown identified seven Haumea Family Members (HFMs). Initially, candidate family members were classified due to strong water-ice spectra. Ragozzine and Brown proved that methods developed for classifying collisional families in the asteroid belt apply to the Kuiper Belt with minor modifications. In the decade since their publication, no new family members have been identified apart from the efforts of this thesis.

1.3 Overview

In Chapter 2, I elaborate on the methods we used to generate our database of orbital elements. OpenORB uses a sequence of orbital calculations and statistical methods to generate orbital clones.

REBOUND takes these orbital clones and orbital data for the Sun and other giant planets from NASA's Horizon database and integrates their orbits 50 Myr. Since the effect of terrestrial planets is minimal on the Kuiper Belt, they were excluded from the integrations to improve speed and efficiency of the REBOUND program. Proper elements were calculated by averaging the oscillating orbital elements. Others, such as the Asteroids Dynamic Site (AstDys) (Milani & Knezevic 2000), calculate proper elements using Fourier transforms. Our simplified method produces results within two standard deviations of theirs, which is sufficiently accurate for astronomy calculations. The analytical pipeline is applied to every KBO with data held by the Minor Planet Center.

In Chapter 3, I show how we used the database to analyze several Kuiper Belt Objects. From the database we generate, we apply the methods from Ragozzine and Brown (2007) to determine all candidate Haumea Family Members.

Chapter 2

Methods

In this section, I discuss the methods we used to generate proper elements for every KBO. Open ORB takes the observational data recorded by the Minor Planet Center as input and produces 30 statistically weighted orbital clones. REBOUND takes those clones and integrates, or projects, the orbits back 50 Myr while accounting for the gravitational influence of the largest bodies in the solar system. These steps form the BUNSHIN pipeline that we use to map out the Kuiper Belt.

2.1 Open ORB

Open ORB is a program created by Granvik et al. (2009) that calculates orbits from data collected by the Minor Planet Center (MPC) through a sequence of Bayesian methods. Each method solves the n-body problem which remains highly difficult to solve analytically. We discuss the preparations in using this program and the process of each method Open Orb employs in the following sections.

2.1.1 Adjusting Weights of Observations

Surprisingly, MPC data does not have associated uncertainties. By uncertainty, we refer to the precision of the KBO's recorded position in each observation. In other words, data taken from a primitive telescope from the late 19th century has the same established precision as data taken by our modern day telescopes.

This thesis relies heavily on the efforts of Veres et al. (2017) which uses large global studies to estimate the uncertainties as a function of observatory and time where each observation was taken. We adjusted the weights of each observation from the Minor Planet Center based on this catalog.

2.1.2 Statistical Ranging

The first method Open Orb uses to calculate orbits generates 200 possible orbits using Markov Chain Monte Carlo statistical ranging, also referred to simply as ranging. This technique takes two random data points from the observation file to generate an orbital clone. These initial orbital clones lack high degrees of accuracy, but narrow down the boundaries for the orbital elements considerably. This technique can be compared to the variational principle used in quantum mechanics. The variational principle uses known wave functions with adjustable paramaters to find an upper bound to the energy level of the ground state of an electron. Likewise, statistical ranging uses a numerical method to establish the boundaries for each orbital element.

Unlike the following processes, statistical ranging requires the observational arc to be much smaller than the full data sets. We found the most efficient observational arc to use was $\sim$1 year, but there were several cases that needed modified manually. The method to determine an acceptable subset of the observational data to use for this initial process is ongoing.

2.1.3 Least Squares Solution (LSL)

The next process uses a linear least squares method to determine the best orbit of these estimates. The 200 sample orbits generated by ranging and the full observational data set are taken as inputs by LSL to generate a best fit orbit. If all we wanted was a likely set of orbital parameters, we would stop here. But, the goal of our efforts is to establish a rigorous method to determine uncertainties in orbital propagation.

Several KBOs include data from 1950 and earlier. When initially running the LSL method to determine a best fit from the ranging generated orbits, the process would conclude all the data must be outliers. To overcome this statistical error, we split the LSL process into two steps.

First, we created an observational data file to feed into the process. We took the original observations and ignored all data retroactively added to the KBO's data set. We fed this subset of observations and the ranging orbits into the LSL process to generate a best fit orbit.

Then, we reran the LSL process and used the entire data set and the LSL orbit as inputs to generate a best fit orbit for the entire data set.

2.1.4 Covariance Sampling

The orbit selected by LSL is run through a covariance matrix process that perturbs the orbit to assess if the perturbed orbit satisfies the observational data. If the perturbed orbit still matches the data, it is selected as a sample orbit. Once this process establishes 100 orbits that vary slightly from the linear best fit, BUNSHIN moves to the last process in Open Orb.

2.1.5 Virtual Observation Markov Chain Monte Carlo (VOMCMC)

The final step uses a Virtual Observation Markov Chain Monte Carlo method that produces 30 orbital clones and determines the probability of each one. This method follows similar patterns for ranging, but with better initial positions established for the numerical calculations. In effect, VOMCMC produces a Bayesian posterior probability distribution for each parameter by assigning each clone a weight. These weights are a measure of how well a particular clone fits the data. At this point, we can integrate this orbits backwards in time.

2.2 REBOUND

Hanno Rein developed REBOUND, an open source orbital integrator. Orbital dynamics require us to acknowledge several orbital elements as we propagate an orbit in time as can be seen in Fiq 2.1. This figure illustrates a simple simulation using REBOUND that uses NASA's HORIZON data of multiple KBOs. Each KBO has an individual inclination, semi-major axis, and eccentricity. While there are the other orientation angles to consider, this thesis places emphasis on the elements conserved beyond short time scales. In the following sections, the semi-major axis is described in terms of AU, the inclination in terms of degrees from the orbital plane of Earth, and the eccentricity is a dimensionless ratio always less than one for ellipses.

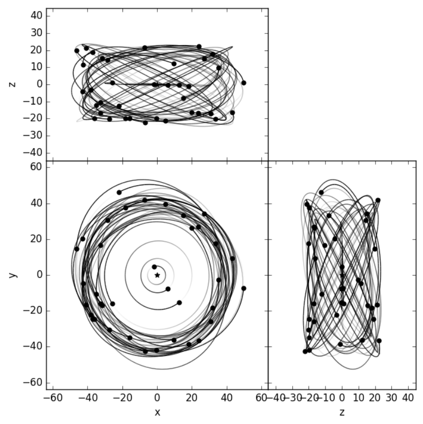

Figure 2.1 This is a simple example of how REBOUND takes the positions of objects and propagates their orbits over a specified time. These orbits are plotted in Cartesian coordinates with the Sun as the origin.

Taking the calculated orbital elements from Open Orb, we ran integrations for a simulated 50 Myr for every clone of each KBO. The proper elements are determined from averaging the values over this period. Fig 2.2 plots the proper elements of E5453, a known Haumea family member, in orbital element space. Ragozzine and Brown 2007 illustrated how the Haumea collisional family all fall within a specific region of orbital element space. Fig 2.3 take plots the results of KBO 2014 LO28. Further investigation of this object almost guarantees it originated from Haumea, but it was originally due to its orbital elements falling with specific parameters that prompted me to make it a focus of my research.

For KBOs that were likely to be collisional family members, we pushed the integrations back to the beginning of the solar system to analyze if there was indeed a point of convergence sometime in the past.

2.3 Haumea Family Members calculations

Ragozzine and Brown 2007 established the method to determine the dynamical kick of each HFM from the collisional center. Recently, Benjamin Proudfoot and Darin Ragozzine have refined this process with a more accurate collisional center. The dynamical kick corresponds to the change in orbital velocity from the primitive surface of Haumea to its present orbit.

Using this new location and the established method, we ran calculations to determine Δv for each candidate HFM.

With orbital mechanics, having multiple objects in one system impacts the orbital parameters. Large, relatively close bodies can perturb the orbits of nearby objects. For the Kuiper Belt, Neptune has the greatest sway in affecting the orbits of KBOs. We compensate for this by calculating the δv, the resonate dynamical kick that allows the eccentricity and inclination to vary while conserving the Tisserand as can be seen from the following equations,

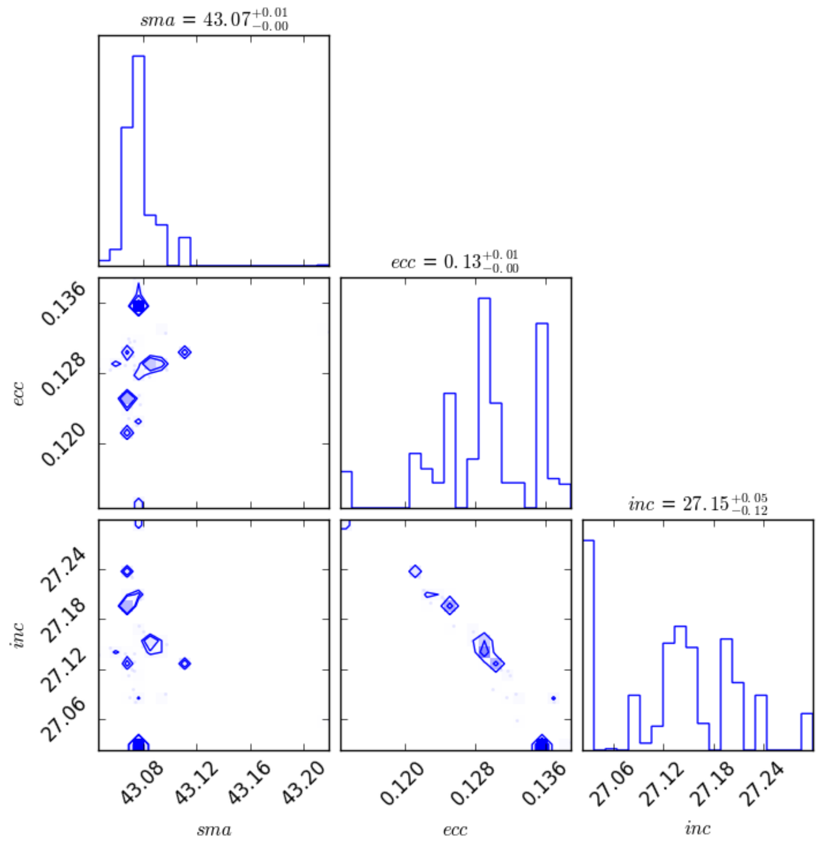

Figure 2.2 E5453 is a known Haumea family member. This plot takes the conserved proper orbital elements of the orbital clones based on observations of E5453 and plots them in orbital element space. Darker, concentrated areas indicate where more orbital clones had the same calculated proper elements. Haumea family members fall within a small, compacted region. (Ragozzine & Brown 2007)

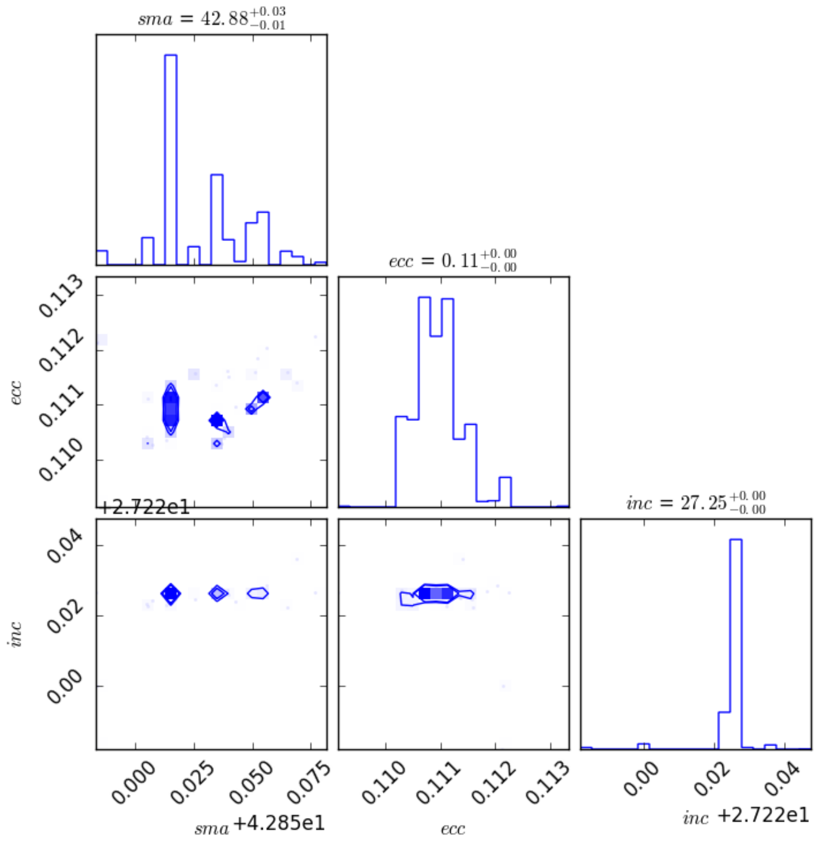

Figure 2.3 2014 LO28 is one of the new HFMs identified by our efforts. Its orbital elements fall withing expected parameters.

$$T = \frac{a_N}{a} + 2\cos(i - i_N)\sqrt{\frac{a_N}{a}(1 - e^2)}. \tag{2.1}$$

We used the calculated resonate and non-resonate dynamical kicks to determine the probability of each KBO's relation to the dwarf planet Haumea. It is left to other astronomers to research other statistical probabilities to determine the true shattered remains of Haumea's primoridial surface.

A sample of these calculations are found in the following figures. Fig. 2.4 shows the results for a known HFM and Fig. 2.5 shows the results for a newly identified HFM.

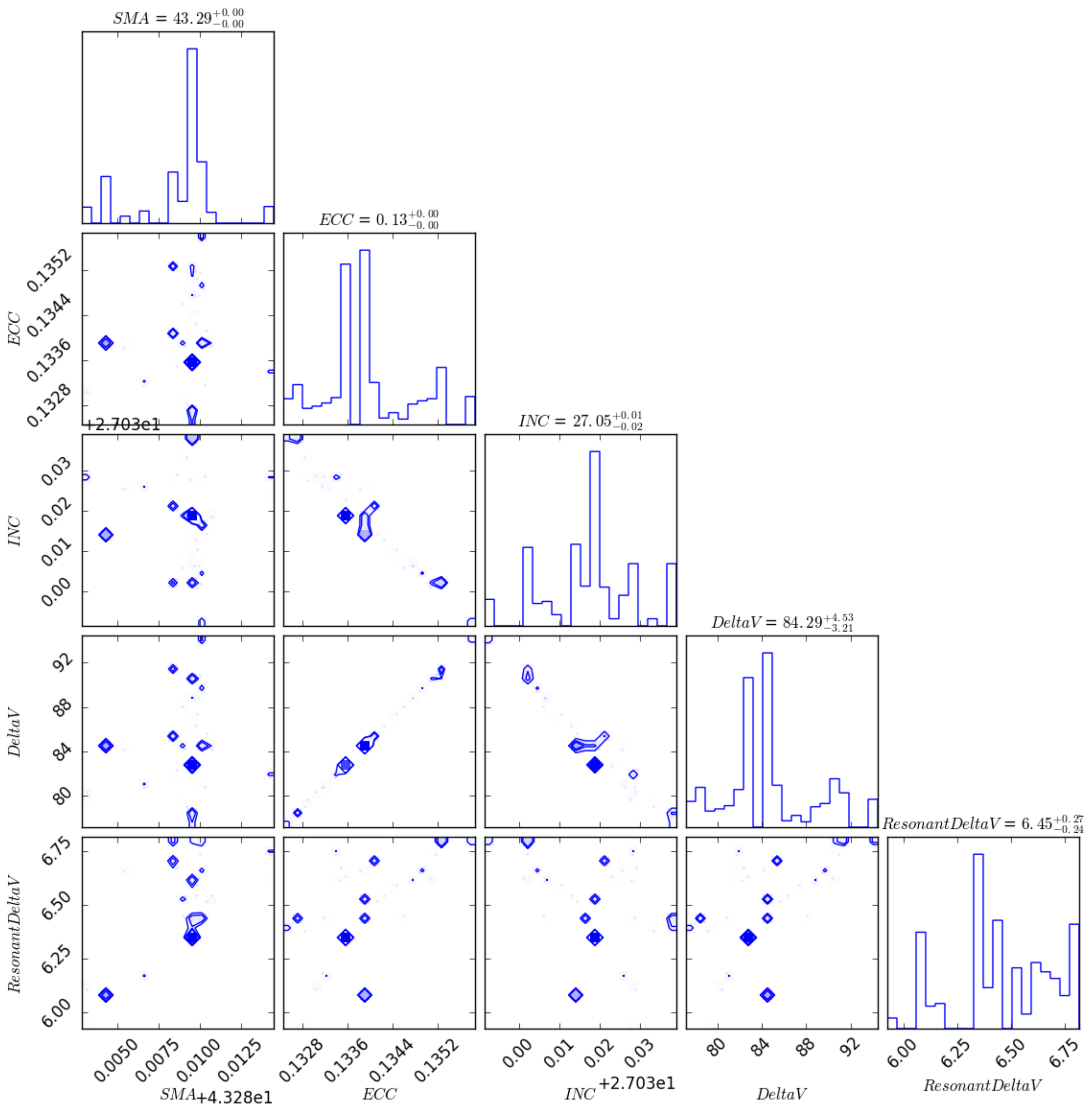

Figure 2.4 The conserved proper elements, minimum Δv, minimum δv, and histograms for each component for E5453

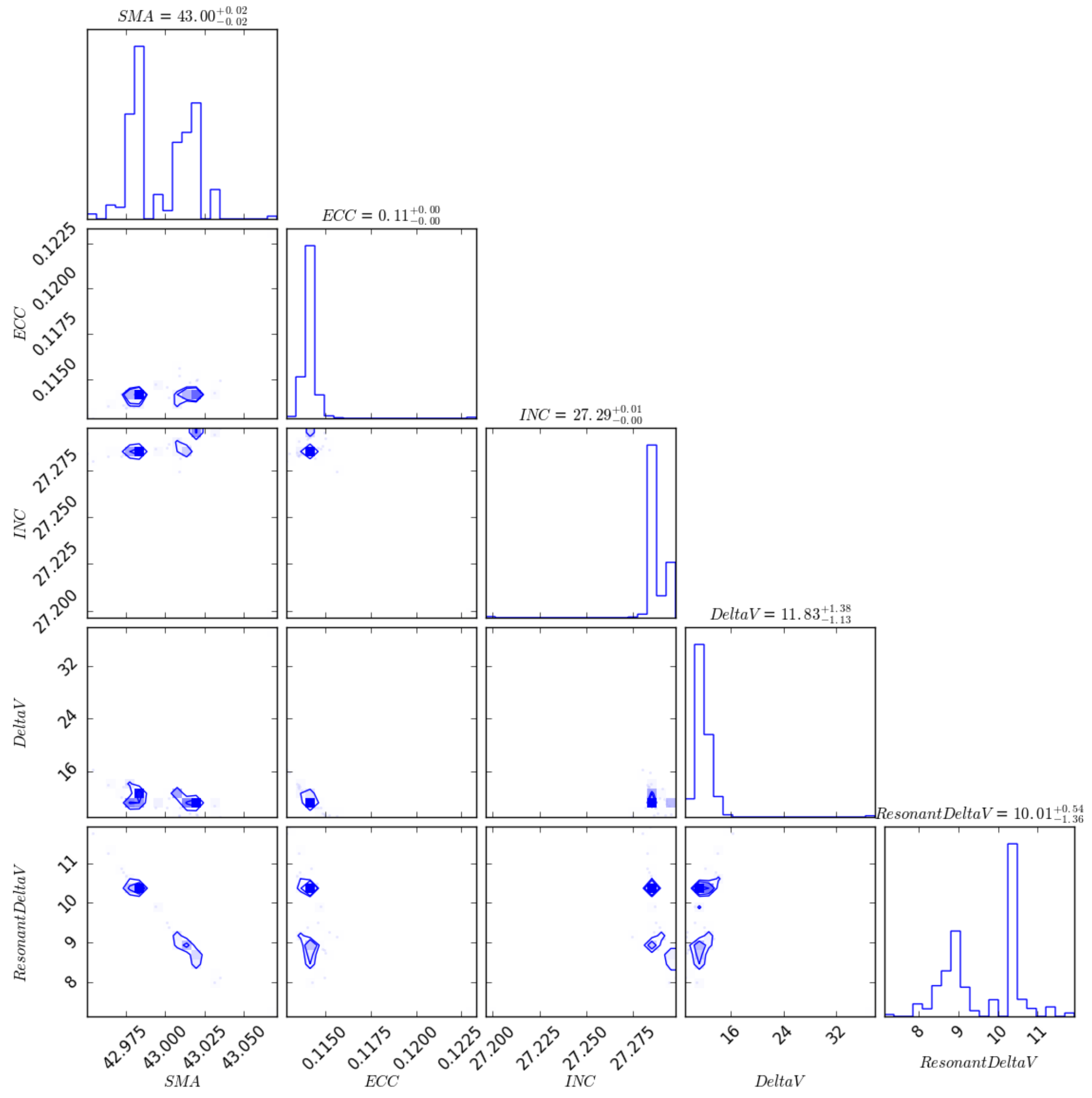

Figure 2.5 The conserved proper elements, minimum Δv, minimum δv, and histograms for each component for 2014 LO28

Chapter 3

Results and Conclusions

In Chapter 2, we described the process in which we generated a database of proper orbital elements (semi-major axis, eccentricty, and inclination) for every KBO whose observational data is archived be the Minor Planet Center. We demonstrate an example of how to use the database and what conclusions we can draw from our efforts.

3.1 Database of Orbital Data

As demonstrated with the identification of more HFMs in Sect. 2.4, the database generated by our pipeline at the end of Sect. 2.3 can be used for dynamical classification of every KBO in our solar system. For each KBO in our database, we have generated 30 statistically weighted orbital clones which were integrated over 50 Myr. The orbital elements and Cartesian coordinates for these KBOS can be extracted from any point in within the integration period thanks to the nature of the Simulation Archive bin files generated by REBOUND (Sect. 2.3).

3.2 Identifying all Haumea Family Members

We identify several KBOs that have been proven as originating from the dwarf planet Haumea. For each KBO, Table 3.1 gives the proper semi-major axis, eccentricity, and inclination after all the weighted orbital clones have been consolidated into one set of orbital values. The dynamical kick and resonant dynamical kick of each clone are likewise consolidated into one set of values.

The dynamical kick, Δv, measures the change in orbital velocity experienced by a KBO with the primordial collision of Haumea as the origin. The resonant dynamical kick, δv, likewise determines the change in orbital velocity but also takes into account the gravitational perturbations from Neptune. KBO K14HJ9Z, the thirteenth KBO in Table 3.1 illustrates the importance of accounting for Neptunian resonances. The standard dynamical kick for this object is far too large to be considered part of the Haumea Family. The story changes, however, after we adjust a few parameters due to Neptune's influence and we can readily see that this KBO must originate from the dwarf planet. By contrast, KBO K14F43P could not have originated from Haumea's surface despite a possible resonance with Neptune. The lowest resonate dynamical kick for this object is just outside the acceptable velocity dispersion for the collisional family.

3.3 Conclusions

The BUNSHIN pipeline created a database that contains orbital elements of known KBOs from today to 50 Myr in the past. With the bin files created by the REBOUND program, we can know the orbit of each KBO at any point of time recorded in our database. Using the orbital data, we calculated the proper elements for every KBO. Building on the efforts of Ragozzine and Brown (2007), we identified many more Haumea Family members beyond the original seven KBOs identified. As mentioned in Chapter 1, the MPC and AstDys have both calculated orbital elements for KBOs. The former calculates the osculating, or instantaneous, elements while the latter uses a sequence of

Table 3.1 With the BUNSHIN pipeline, the proper elements, Δv, and δv are calculated. The Δv values represent the dynamical kick the KBO would have experienced if it had originated from Haumea. The δv values use the same calculations, but with an extra parameter for the possibility of an orbital resonance with Neptune. Haumea family members cannot have a greater dynamical kick than 150 m s^{-1}. (Ragozzine & Brown 2007)

KBO Name	Semi-Major Axis (AU)	Eccentricity	Inclination(°)	Δv(m s^{-1})	δv (m s^{-1})
f6400	44.2104	0.12897	27.8994	75.9467	75.6248
U8193	43.3493	0.13427	27.1999	83.0167	18.0226
V5530	41.8081	0.13734	27.3826	145.6948	83.7463
K08Q43B	41.7579	0.09237	27.6340	117.5445	76.0852
c6723	44.3302	0.13578	28.0004	96.9874	91.1670
K10OC7O	42.2779	0.13547	26.8373	130.5444	48.5941
K10VK1K	43.1682	0.10665	27.9888	82.9455	49.1014
K11Uf2K	40.7253	0.10131	27.1585	149.4930	134.6936
K13U15Q	42.7597	0.12018	27.2276	37.7317	22.3339
l1954	43.3273	0.12530	28.7764	132.1368	131.8391
K14F43P	44.8442	0.15148	28.5321	171.2117	152.1488
K14F71T	43.5650	0.15424	28.5317	205.7718	148.7843
K14HJ9Z	43.1395	0.16607	26.6267	250.3192	19.7613
K14L28O	42.9248	0.11291	27.2592	16.6888	13.3396
K14Qi1W	44.2854	0.10624	27.9889	139.2483	65.3407
K14UM4F	45.4034	0.13028	26.6249	130.8308	122.8445
K14X40S	42.3395	0.16251	27.5320	253.5191	103.5059
K14Y50B	41.6956	0.10022	26.9927	101.7069	85.0416
K15AS1J	43.2888	0.13746	28.1587	125.3066	97.4022
K15FY5N	41.8586	0.08886	27.6065	127.0085	69.9292

Fourier transforms to calculate the proper elements. Our analysis pipeline consistently falls within acceptable limits of standard deviations when compared with the MPC and AstDys assessments. While our orbital element calculations fall in acceptable values, one of the key initial motivations for research is determining the uncertainties for the proper elements. We are currently finishing these final calculations for each KBO and will publish our results within the next few months. Our preliminary values for these calculated uncertainties tend towards 0.2 AU in semi-major axis, 0.001 in eccentricity, and 0.0005 degrees in inclination.

3.4 Directions for further work

Further work consists of identifying other collisional families in the Kuiper Belt and acquiring the spectra to completely prove the origin of candidate HFMs classified by this thesis. Any dynamicist can access the archival data at any point in time for any KBO to use in their efforts to dynamically classify the Kuiper Belt Region.

Appendix A

BUNSHIN

This code, BUNSHIN, takes its name from the Japanese term "bunshin no jutsu" which translates to "clone technique," quite fitting for a code that generates thousands of orbital clones. The code we use for our research is given below.

```
from subprocess import call
import os
import time
import math
import rebound
import pandas
import smtplib
import numpy as np
from sys import argv

kboname = argv[1] #The Linux script will feed in each KBO
```

```
time.sleep(4)
print("Beginning BUNSHIN for: "+kboname)
time.sleep(4)

#Establishing the time needed to run Open Orb for each KBO
starttime=time.time()

#setting the save path for the output
destination = 'output/'+kboname
call(['mkdir',destination])
out=os.chdir('./'+destination)

# Cleaning out old data.

call(['rm',kboname+'.ranging.orb'])
call(['rm',kboname+'.lsl.orb'])
call(['rm',kboname+'.step.lsl.orb'])
call(['rm',kboname+'.cov.sampling.orb'])
call(['rm',kboname+'.vomcmc30.orb'])
call(['rm','problematic_observation_sets.des'])

#getting observational data
mpcfile=kboname+'.mpc'
call(['cp','/fslhome/subok114/MPCdata/mpcfiles/'+mpcfile,mpcfile])
if os.path.exists(mpcfile):
    print('Observational Data Acquired')
    time.sleep(1)
else:
    print('Observational Data Not Found')
    time.sleep(1)

#Preparing the directory to run OpenOrb
call('/fslhome/subok114/python_codes/OrbitalPreparation.sh')

#converting the file into mpc3 format
mpc3file=kboname+'.mpc3'
os.system('/fslhome/subok114/oorb-master/main/oorb --task=tompc3
           --obs-in='+mpcfile+' --obs-out='+mpc3file)

####################################################################################
# corrects uncertainties in MPC3 data files
# using Veres et al. 2017 results in Tables 2, 3, 4, and 5

mpc3weightedfile = kboname+'.w.mpc3'

#getting range to write file
beta = sum(1 for line in open(mpc3file))

mpc3w = open(mpc3weightedfile,'w')
```

```
fileobject = open(mpc3file)

changelist = fileobject.read().split('\n')

#print(changelist)


newstring=[]


for index in range(beta):

   year =changelist[index][21:25]

   month = changelist[index][21:25]

#Table 2

   if changelist[index].endswith('0703') & int(year) < 2014:

       if int(year) == 2013 & int(month) >= 9:

          RAerr="0.800"

          Decerr="0.800"

       else:

          RAerr="1.000"

          Decerr="1.000"

   elif changelist[index].endswith('0703') & int(year) >= 2014:

       RAerr="0.800"

       Decerr="0.800"

   elif changelist[index].endswith('0691') & int(year) < 2003:

       RAerr="0.600"

       Decerr="0.600"

   elif changelist[index].endswith('0691') & int(year) >= 2003:

       RAerr="0.500"
```

```
        Decerr="0.500"
    elif changelist[index].endswith('0644') & int(year) < 2004:
        if int(year) == 2003 & int(month) >= 9:
            RAerr="0.400"
            Decerr="0.400"
        else:
            RAerr="0.600"
            Decerr="0.600"
    elif changelist[index].endswith('0644') & int(year) >= 2004:
        RAerr="0.400"
        Decerr="0.400"
#from table 3
    elif changelist[index].endswith('0704'):
        RAerr="1.000"
        Decerr="1.000"
    elif changelist[index].endswith('0G96'):
        RAerr="0.500"
        Decerr="0.500"
    elif changelist[index].endswith('0F51'):
        RAerr="0.200"
        Decerr="0.200"
    elif changelist[index].endswith('0G45'):
        RAerr="0.600"
        Decerr="0.600"
    elif changelist[index].endswith('0699'):
        RAerr="0.800"
        Decerr="0.800"
    elif changelist[index].endswith('0D29'):
        RAerr="0.750"
        Decerr="0.750"
    elif changelist[index].endswith('0C51'):
        RAerr="1.000"
        Decerr="1.000"
    elif changelist[index].endswith('0E12'):
        RAerr="0.750"
        Decerr="0.750"
    elif changelist[index].endswith('0608'):
        RAerr="0.600"
        Decerr="0.600"
    elif changelist[index].endswith('0J75'):
        RAerr="1.000"
        Decerr="1.000"
#from table 4
    elif changelist[index].endswith('0645'):
        RAerr="0.300"
        Decerr="0.300"
    elif changelist[index].endswith('0673'):
        RAerr="0.300"
        Decerr="0.300"
    elif changelist[index].endswith('0689'):
        RAerr="0.500"
        Decerr="0.500"
    elif changelist[index].endswith('0950'):
        RAerr="0.500"
        Decerr="0.500"
    elif changelist[index].endswith('0H01'):
        RAerr="0.400"
        Decerr="0.400"
    elif changelist[index].endswith('0J04'):
        RAerr="0.400"
        Decerr="0.400"
    elif changelist[index].endswith('LCOGT'):
        RAerr="0.400"
        Decerr="0.400"
 #for catalog PPMXL, GAIA-DR1
 #elif changelist[index].endswith('0Y28'):
 #    RAerr="0.300"
 #    Decerr="0.300"
 #for catalog USNO-B1.0, USNO-B2.0
 #   elif changelist[index].endswith('0568'):
 #       RAerr="0.500"
 #       Decerr="0.500"
 #for catalog GAIA-DR1
 #   elif changelist[index].endswith('0568'):
 #       RAerr="0.100"
 #       Decerr="0.100"
 #for catalog PPMXL
 #   elif changelist[index].endswith('0568'):
 #       RAerr="0.200"
 #       Decerr="0.200"
 #for catalog GAIA-DR1
 #   elif changelist[index].endswith('T12'):
 #       RAerr="0.100"
 #       Decerr="0.100"
 #default error
    else:
        RAerr="1.000"
        Decerr="1.000"
 #from table 5
    if int(year) < 1890:
        RAerr="10.00"
        Decerr="10.00"
    elif int(year) <= 1950:
        RAerr="5.000"
        Decerr="5.000"
    elif int(year) <= 1996:
        RAerr="2.500"
        Decerr="2.500"

    if int(changelist[index][18]) == 2:
       newstring.append(changelist[index][:41]+RAerr+"  "+Decerr+changelist[index][53:])
    else:
       newstring.append(changelist[index])

 for index in range(beta):
     mpc3w.write(newstring[index]+'\n')

 mpc3w.flush()
 mpc3w.close()
 print('Weighted mpc3 file created')
 time.sleep(1)

 #################################################################################
 #running through Open Orb
 #creating the ranging.mpc3 file

 mpcshortfile=kboname+'.ranging.mpc3'
 mpcsf=open(mpcshortfile,'w')

 fileobject = open(mpc3weightedfile)
 linelist = fileobject.read().split('\n')
 fulldata=len(linelist)-1 #because the list is 1 too long
 ranglist=[]

 #This section determines when the first observation of the object
```

```
#was made. For ranging, we'll ignore all observations
#retroactively attributed to the KBO.
for line in range(fulldata):
    if linelist[line][19] == '*':
        year = linelist[line][21:25]
        month = linelist[line][25:27]
        day=linelist[line][27:29]

#This section takes all the oservational data from when the KBO
#was discovered until the end of the year
for line in range(fulldata):
    if linelist[line][21:25] == year:
        if int(linelist[line][25:27])>=int(month):
            if int(linelist[line][27:29])>=int(day):
                ranglist.append(linelist[line])

#This section takes the observations until about a year from
#the initial discovery and adds it to the ranging file
for line in range(fulldata):
    if linelist[line][21:25] == str(int(year)+1):
        if int(linelist[line][25:27])<=int(month):
            ranglist.append(linelist[line])

#the actual writing of the ranging file
for line in range(len(ranglist)):
    mpcsf.write(ranglist[line]+'\n')

mpcsf.flush()
mpcsf.close()
##################################################################################
#Creating a step_lsl file
lslshortfile=kboname+'.lsl.mpc3'
lslsf=open(lslshortfile,'w')

lslllist=[]
for line in range(fulldata):
    if linelist[line][21:25] == year:
        if int(linelist[line][25:27])>=int(month):
            if int(linelist[line][27:29])>=int(day):
                lslllist.append(linelist[line])
for line in range(fulldata):
    if int(linelist[line][21:25]) > int(year):
        lslllist.append(linelist[line])
for line in range(len(lslllist)):
    lslsf.write(lslllist[line]+'\n')

lslsf.flush()
lslsf.close()
##################################################################################
#Running Open ORB
```

```
#if os.path.exists(mpcfile):

#ranging
alphatest = 0
while alphatest < 5:
    os.system('/fslhome/subok114/oorb-master/main/oorb --task=ranging
        --conf=oorb.conf.2bk --obs-in='
        +mpcshortfile+' --orb-out='+kboname+'.ranging.orb')
    alphatest += 1
    time.sleep(1)
    if os.stat(kboname+'.ranging.orb').st_size > 0:
        print('###################################################################')
        print('###################################################################')
        print('################   Ranging Successful  ############################')
        print('###################################################################')
        print('###################################################################')
        break
    else:
        print('###################################################################')
        print('###################################################################')
        print('################ Running Ranging Again ############################')
        print('##############      '+str(alphatest)+'      #######################')
        print('###################################################################')
        print('###################################################################')
```

```
##################################################################################

# lsl subset
if os.stat(kboname+'.ranging.orb').st_size > 0:
    os.system('/fslhome/subok114/oorb-master/main/oorb --task=lsl
        --conf=oorb.conf.m --obs-in='
        +lslshortfile+' --orb-in='+kboname+'.ranging.orb --orb-out='
        +kboname+'.step.lsl.orb')
    time.sleep(1)
# lsl
    if os.stat(kboname+'.step.lsl.orb').st_size > 0:
        os.system('/fslhome/subok114/oorb-master/main/oorb --task=lsl
            --conf=oorb.conf.m --obs-in='
            +mpc3weightedfile+' --orb-in='+kboname+'.step.lsl.orb --orb-out='
            +kboname+'.lsl.orb')
        time.sleep(1)
# covariance sampling
        if os.stat(kboname+'.lsl.orb').st_size > 0:
            os.system('/fslhome/subok114/oorb-master/main/oorb
                --task=covariance_sampling
                --conf=oorb.conf.m --obs-in='
                +mpc3weightedfile+' --orb-in='+kboname+'.lsl.orb --orb-out='
                +kboname+'.cov.sampling.orb')
            time.sleep(1)
```

```
#running vomcmc with only 30 orbital clones
            if os.stat(kboname+'.cov.sampling.orb').st_size > 0:
                os.system('/fslhome/subok114/oorb-master/main/oorb --task=vomcmc
                    --conf=oorb.conf.v30k --obs-in='
                    +mpc3weightedfile+' --orb-in='+kboname+'.cov.sampling.orb
                    --orb-out='+kboname+'.vomcmc30.orb')
                time.sleep(1)
                if os.stat(kboname+'.vomcmc30.orb').st_size > 0:
#epoch propagation for the day of the eclipse
                    os.system('/fslhome/subok114/oorb-master/main/oorb
                        --task=propagation --conf=oorb.conf.v30k
                        --epoch-mjd-tt=57986'
                        +' --orb-in='+kboname+'.vomcmc30.orb
                        --orb-out='+kboname+'.eclipse.v30.orb')
                    time.sleep(1)
#extracting orbital elements from eclipse.v30.orb
#At a later date, convert this process into a Python DataFrame
                    eclipse30file = kboname+'.eclipse.v30.orb'
                    vomcmc30file = kboname+'.vomcmc30.orb'
                    wgt30file=kboname+'.vomcmc30.wgt'
                    sma30file=kboname+'.eclipse.v30.sma'
                    ecc30file=kboname+'.eclipse.v30.ecc'
                    inc30file=kboname+'.eclipse.v30.inc'
                    Oma30file=kboname+'.eclipse.v30.Oma'
                    oma30file=kboname+'.eclipse.v30.oma'
```

```
mean30file=kboname+'.eclipse.v30.mean'

sma30 = open(sma30file,'w')
ecc30 = open(ecc30file,'w')
inc30 = open(inc30file,'w')
wgt30 = open(wgt30file,'w')
Oma30 = open(Oma30file,'w')
oma30 = open(oma30file,'w')
mean30 = open(mean30file,'w')

alpha = sum(1 for line in open(eclipse30file))
observations = alpha - 4
element30= open(eclipse30file).read().split('\n')
weight30= open(vomcmc30file).read().split('\n')

for index in range(observations):
    sma30.write(element30[index+4][18:38]+'\n')
    ecc30.write(element30[index+4][40:60]+'\n')
    inc30.write(element30[index+4][62:82]+'\n')
    Oma30.write(element30[index+4][84:104]+'\n')
    oma30.write(element30[index+4][106:126]+'\n')
    mean30.write(element30[index+4][128:148]+'\n')
    wgt30.write(weight30[index+4][223:]+'\n')

sma30.flush()
```

```
                ecc30.flush()
                inc30.flush()
                Oma30.flush()
                oma30.flush()
                mean30.flush()
                wgt30.flush()

                sma30.close()
                ecc30.close()
                inc30.close()
                Oma30.close()
                oma30.close()
                mean30.close()
                wgt30.close()

            else:
                time.sleep(10)
                print('***VOMCMC File empty***')
                OrbFail = open('/fslhome/subok114/output/00_text_files/OrbFail.txt',
                OrbFail.write(str(kboname)+'\n')
                time.sleep(10)
        else:
            time.sleep(10)
            print('***Covariance Sampling File empty***')
            OrbFail =
                open('/fslhome/subok114/output/00_text_files/OrbFail.txt','a')
                OrbFail.write(str(kboname)+'\n')
                time.sleep(10)
        else:
            time.sleep(10)
            print('***Lsl File empty***')
            OrbFail =
            open('/fslhome/subok114/output/00_text_files/OrbFail.txt','a')
            OrbFail.write(str(kboname)+'\n')
            time.sleep(10)
    else:
        time.sleep(10)
        print('***Step Lsl File empty***')
        OrbFail = open('/fslhome/subok114/output/00_text_files/OrbFail.txt','a')
        OrbFail.write(str(kboname)+'\n')
        time.sleep(10)
else:
    time.sleep(10)
    print('***Ranging File empty***')
    OrbFail = open('/fslhome/subok114/output/00_text_files/OrbFail.txt','a')
    OrbFail.write(str(kboname)+'\n')
    time.sleep(10)
################################################################################


#for clearing out the uneeded files after Open Orb is complete
```

```
call('/fslhome/subok114/python_codes/AftermathCleanup.sh')

################################################################################
endtime=time.time()
totaltime= str((endtime-starttime)/60)

if os.stat(kboname+'.eclipse.v30.orb').st_size > 0:
    print('***Analysis of '+kboname+' complete***')
    print('Time needed to process '+kboname+': '+totaltime+' minutes')
    time.sleep(1)
else:
    print('***Analysis of '+kboname+' failed***')
    time.sleep(1)
################################################################################
if os.stat(kboname+'.eclipse.v30.orb').st_size > 0:
#This next section is based off of rebound.50M.py

    numofobs = 30 #for the v30 files we're working with
    observations = str(numofobs)

    ssdata = pandas.read_table('/fslhome/subok114/ssdata.txt', delimiter=',',header=None

    smafile ='/fslhome/subok114/output/'+kboname+'/'+kboname+'.eclipse.v'
              +observations+'.sma'
    eccfile ='/fslhome/subok114/output/'+kboname+'/'+kboname+'.eclipse.v'
```

```
          +observations+'.ecc'
incfile ='/fslhome/subok114/output/'+kboname+'/'+kboname+'.eclipse.v'
          +observations+'.inc'
Omafile ='/fslhome/subok114/output/'+kboname+'/'+kboname+'.eclipse.v'
          +observations+'.Oma'
omafile ='/fslhome/subok114/output/'+kboname+'/'+kboname+'.eclipse.v'
          +observations+'.oma'
meanfile ='/fslhome/subok114/output/'+kboname+'/'+kboname+'.eclipse.v'
          +observations+'.mean'

sma = open(smafile)
ecc = open(eccfile)
inc = open(incfile)
Oma = open(Omafile)
oma = open(omafile)
mean = open(meanfile)

smalist = sma.read().split('\n')
ecclist = ecc.read().split('\n')
inclist = inc.read().split('\n')
Omalist = Oma.read().split('\n')
omalist = oma.read().split('\n')
meanlist = mean.read().split('\n')

smaarr=np.zeros((numofobs,1))
```

```
    eccarr=np.zeros((numofobs,1))
    incarr=np.zeros((numofobs,1))
    Omaarr=np.zeros((numofobs,1))
    omaarr=np.zeros((numofobs,1))
    meanarr=np.zeros((numofobs,1))

    for index in range(numofobs):
        smaarr[index]=float(smalist[index])
        eccarr[index]=float(ecclist[index])
        incarr[index]=math.radians(float(inclist[index]))
        Omaarr[index]=math.radians(float(Omalist[index]))
        omaarr[index]=math.radians(float(omalist[index]))
        meanarr[index]=math.radians(float(meanlist[index]))

#****************************************************************************
    #onto the rest of rebound
    sim=rebound.Simulation()

    sim.add(m=1.0, hash='Sun')

    sun = sim.particles[0]

    for i in range(5,9):
        sim.add(a=ssdata[0][i], e=ssdata[1][i], inc=ssdata[2][i],
                omega=ssdata[3][i], Omega=ssdata[4][i], M=ssdata[5][i],
                m=ssdata[6][i], hash=str(ssdata[7][i]), primary = sun)

    for obs in range(numofobs): #run through each orbital clone of the kbo
        sim.add(a=smaarr[obs], #semi-major axis
                e=eccarr[obs], #eccentricity
                inc=incarr[obs], #inclination
                Omega=Omaarr[obs], #argument of periapsis
                omega=omaarr[obs], #argument of ascending node
                M=meanarr[obs], #mean anomaly
                primary=sun)

#****************************************************************************

#Adjusting initial velocities to run backgrounds
    for ipart in range(len(sim.particles)):
        sim.particles[ipart].vx=-sim.particles[ipart].vx
        sim.particles[ipart].vy=-sim.particles[ipart].vy
        sim.particles[ipart].vz=-sim.particles[ipart].vz
    sim.status()

#copied from rebound.lsl.prep.py by Steven Maggard
#which is based off of intfakefam.py by Darin Ragozzine

    sim.integrator="whfast"
    safile = "rebound."+kboname+".bin"
    saint=1.0e6

    sim.initSimulationArchive(safile,interval=saint)

    sim.dt=2.0


    starttime=time.time()

    sim.integrate(50e7*2.0*np.pi)

    endtime=time.time()

    integrationtime=(endtime-starttime)/3600

    sim.status()

    print("Time needed to integrate "+kboname+": "+str(integrationtime)+" hours.")

    time.sleep(1)

#****************************************************************************

    call(['cp',safile,'/fslhome/subok114/output/00_binfiles'])

    if integrationtime < 4:

        Success = open('/fslhome/subok114/output/00_text_files/Success.txt','a')

        Success.write(str(kboname)+'\n')

        #run analysis.py

    else:

        print("Rebound Integration took over 4 hours. This KBO needs rerun.")

        ReboundFail = open('/fslhome/subok114/output/00_text_files/ReboundFail.txt','a')

        ReboundFail.write(str(kboname)+'\n')
```

Bibliography

Granvik, M., Virtanen, J., Oszkiewicz, D., & Muinonen, K. 2009, Meteoritics and Planetary Science, 44, 1853

Milani, A., & Knezevic, V. 2000, AstDyS-2

Ragozzine, D., & Brown, M. E. 2007, The Astronomical Journal, 134, 2160

Rein, H., & Liu, S.-F. 2012, A&A, 537, A128

Index