8-31-2017

Effect of Observational Cadence on Orbit Determination for Synthetic Near-Earth Objects

Thomas G. Endicott
University of Massachusetts Boston

EFFECT OF OBSERVATIONAL CADENCE ON ORBIT DETERMINATION FOR SYNTHETIC NEAR-EARTH OBJECTS

A Thesis Presented

by

Thomas G. Endicott

Submitted to the Office of Graduate Studies, University of Massachusetts Boston, in partial fulfillment of the requirements for the degree of

Master of Science

August 2017

Physics Program

EFFECT OF OBSERVATIONAL CADENCE ON ORBIT DETERMINATION FOR SYNTHETIC NEAR-EARTH OBJECTS

A Thesis Presented

by

Thomas G. Endicott

Approved as to style and content by:

Jonathan Celli, Assistant Professor
Chairperson of Committee

Bala Sundaram, Professor
Member

Chandra Yelleswarapu, Assistant Professor
Member

Stephen Arnason, Program Director
Physics Program

Bala Sundaram, Chairperson
Physics Department

ABSTRACT

EFFECT OF OBSERVATIONAL CADENCE ON ORBIT DETERMINATION FOR SYNTHETIC NEAR-EARTH OBJECTS

August 2017

Thomas G. Endicott,
B.S., University of Massachusetts Boston
M.S., University of Massachusetts Boston

Directed by Assistant Professor Jonathan Celli

Near-Earth Objects (NEOs) are generally small, dark, and fast-moving. Multiple observations over time are necessary to constrain NEO orbits. Orbits based on observational data are inherently uncertain. Here we describe code written in Python and Fortran used to generate synthetic asteroids and compare calculated orbital fit based on noisy ephemeris using the a distance criteria, D-value. Observational sessions separated by more than one month produce very good orbital fits (low D-values) even at the highest noise level. Daily observational sessions show the highest D-values, as expected, since observed points on the orbital ellipse are not well separated. D-value is closely correlated to differences in the eccentricity and inclination of compared orbits.

TABLE OF CONTENTS

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER 1

INTRODUCTION

1.1 Near-Earth Objects

Near-Earth Objects (NEOs) are orbiting bodies in near-Earth space. This can include comets, asteroids, and natural and artificial space debris. Here we are primarily concerned with asteroids in this space. These asteroids typically have small diameters and low albedo, they are very small, very dark, and difficult to observe [Ia11]. Since these objects exist in near-Earth space they have the potential for collision with Earth. In addition, due to their proximity, these asteroids are prime targets for sample return missions, and resource mining operations [Elv14]. To that end, once discovered, continued observances and constraining the orbital uncertainty of these objects is a high priority. Building instruments with increased precision and extending observations of these objects for a long baseline would provide good orbital fits, however, each of these methods is resouce-expensive. The goal of this project is to determine what observational cadence provides an acceptable orbital fit by approximating observations made with existing resources.

1.2 Discovery & Challenges

Discovery of new NEOs is increasing exponentially, as more powerful telescopes and telescopic networks come online objects that were previously too dim to see are now observable. Currently there are more than sixteen thousand known NEOs, including over one thousand eight hundred potentially hazardous asteroids [Jet].

Discovery surveys such as The Catalina Sky Survey, NEO-Wise, PanStarrs, and others aim to cover as much of the night sky as possible. Using an extremely large field of view (the Catalina Sky Survey using a FOV of 5.0 deg^2 sweeps 1000 deg^2 per night [Ari]) these large scale surveys are designed to detect rapidly moving objects in the night sky, identify asteroids (both known and previously undiscovered), and trigger rapid follow-up observations in order to classify and categorize these objects and their properties. Discovery survey data is typically at too low a resolution to do more than determine a rough orbital fit for these objects. Newly discovered asteroids are disseminated to the wider asteroid community who are then able to perform more deliberate follow-up observations in order to determine physical properties of these objects including surface composition, albedo, size, and a more precise orbital solution. In the case of an imprecise orbital fit, these small, dark, fast-moving objects can be lost. With too short a discovery observation window these objects would need to be rediscovered on a subsequent approach [PHW00].

Observational bias also affects NEO discovery. Objects with low inclination and eccentricities are more likely to be detected. Additionally, more NEOs are discovered in November to December [ENS11], a result of longer nights in the northern hemisphere which is where most telescopic resources are located.

Family	Semi-major Axis, a (AU)	Perihelion, q (AU)
generic NEO		$q < 1.3$
Atira	$a < 1.0$	$q < 0.983$
Aten	$a < 1.0$	$q > 0.983$
Apollo	$a > 1.0$	$q < 1.017$
Amor	$a > 1.0$	$q > 1.017$

Table 1: Perihelion boundaries for NEO subgroups. All NEOs have semi-major axis less than 1.3 AU. Atira orbits are strictly within Earth's orbit, Apollo orbits are strictly outside Earth's orbit. Aten and Amor are Earth-crossers.

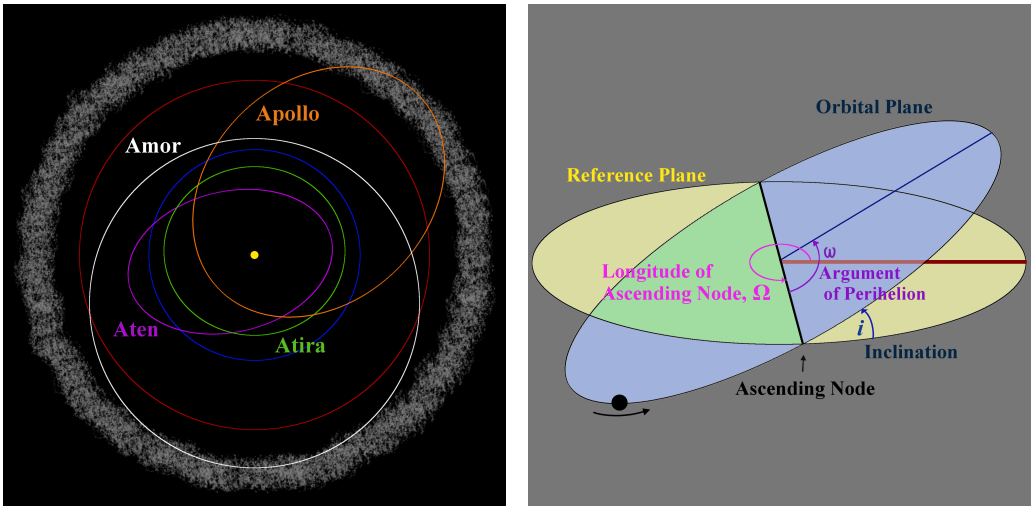
1.2.1 Classifications/Subgroups

These NEOs are categorized into four orbital subgroups; the Atens, Apollos, Atiras, and Amors. Aten asteroids have a semi-major axis less than that of Earth, and their orbits cross that of Earth. Apollo asteroids have a semi-major axis greater than that of Earth, they too have orbits that cross that of Earth (Table 1). The Atira asteroids have orbits strictly within that of Earth, and Amor asteroids strictly without (Fig. 1(a)). NEOs are not uniformly distributed among these subgroups, with Apollos and Amors making up almost 90% of the total NEO population [GNG12].

1.3 Orbit Determination

An elliptical orbit is calculated based on at least three points of observation. These observational points, the ephemeris, include datetime, right ascension, and declination. Once fit, the orbit is described by its Keplerian orbital elements: argument of perihelion (ω), longitude of ascending node (Ω), inclination (i), eccentricity (e), and semi-major

axis (a) (Fig. 1(b)). NEO orbits are affected by gravitational interaction with the planets and can change over time, this occurs on time scales of $\sim$100yr. [Tan98]. Here, we do not consider orbital comparison on a timescale that would include gravitational perturbation.



(a) NEO Orbital Subgroups (b) Angular Orbital Parameters

Figure 1: Illustration NEO orbital subgroups and diagram of angular orbital parameters.

1.4 Orbit Comparison with D-value

In order to compare two orbits we use a standard distance calculation in the orbital element space, referred to as D-value. (Eq. 1.1).

$$D^2 = k_a\left(\frac{\delta a}{a}\right)^2 + k_e(\delta e)^2 + k_i(\delta \sin i)^2 + k_\Omega(\delta\Omega)^2 + k_\omega(\delta\omega)^2 \tag{1.1}$$

$k_a = 5/4$, $k_e = k_i = 2$, $k_\Omega = k_\omega = 1\times10^{-6}$

The calculation and coefficients given above, are taken from Nesvorný [Nes06].

1.5 Orbital Uncertainty & Synthetic Asteroid Choice

It is impossible to make a comparison between the true orbit of an NEO and its calculated orbit based on observation, simply because the only information available is observational data, which is inherently uncertain. Rather than use observed asteroids, we construct synthetic asteroids with "known" orbits by specifying orbital parameters. We then simulate observations of these synthetic asteroids by adding a preset level of noise to their sky positions. Using these noisy simulated observations we recalculate an orbital fit, and compare this to the "known" orbit using the D-value. The orbital elements for our synthetic object are not drawn at random. Since the NEO population is not uniformly distributed in near-Earth space. There are selection effects due to observational bias in each of the subgroups of NEOs. The Atira subgroup, for example, does not spend a large amount of time away from the sun, and therefore the most easily observable for this group are those that observed when perpendicular to the Sun-Earth line. There are also non-uniform distribution of the angular orbital elements of the Amor subgroup, for example, due to resonances with Jupiter [JM14]. We will show, however, that the distribution of these angular elements do not correlate with D-value, and argue that a random uniform choice of angular elements is justifiable when constructing our synthetic NEOs for this analysis. In order to create a synthetic asteroid we randomly select each of the six orbital elements from a database consisting of observed NEOs, these synthetic objects are then placed into subgroups based on perihelion criteria. This choice method provides a set of synthetic objects whose orbital elements are possible, as they are sourced from observations, but otherwise randomly disctibuted in orbital space.

1.5.1 Fortran Code

The code relies on two Fortran executables written by Lowell astronomer Dr. Larry Wasserman. These are hosted on a remote server maintained by Lowell Observatory. The first executable generates ephemerides for a provided list of asteroids. The second executable is an orbital fitting routine that takes an initial set of orbital parameters as a guess for the orbital fit, and a list of ephemerides. This second executable returns Keplerian orbital elements.

1.5.2 Python Code

The code, written in Python 2.7, is an adaptation and extension of previous work by Clément Royer. The code generates a set of synthetic asteroids by randomly sampling near-Earth asteroid orbital elements from the Lowell AstOrb database [Obs], simulates observations of these asteroids, and generates a set of orbital fits. The final output of this code is a set of text files listing the original orbital elements of the synthetic asteroids, the orbital elements of the fit orbit, and the difference between them along with the D-value.

CHAPTER 2

PYTHON ALGORITHMS & MODULES

2.1 User Input

The code is designed to run from a bash shell and immediately prompts the user for a series of inputs. The required inputs, and the default values are listed in Table 2.

Variable	Default
Asteroid family	'neo'
Desired number of objects	15
Ephemeris start (YYYY, MM, DD.DDDD)	2015, 05, 12.0
Ephemeris interval (DD, HH, MM)	00, 01, 00
Ephemeris duration (DDD, HH, MM)	365, 00, 00
Date noise (sec)	0.01
Right Ascension noise (sec)	0.01
Declination noise (sec)	0.01
Number of observations per session	4
Hourly interval between observatoins (HH)	1
Number of observation sessions	5
Daily interval between sessions (DD)	15

Table 2: Required user-inputs and their default values.

2.2 Synthetic NEO Creation

The code first reads in the AstOrb flatfile and discards any asteroid that does not fall within the near-Earth range of 1.3 AU. A synthetic NEO is then created from the remaining asteroids by randomly selecting each of its six orbital elements from those available. Once created, this synthetic object is tested to confirm that it is a member of the user-specified family, and if so, is written to a file with a format identical to AstOrb. This identical formatting is required in order to generate ephemerides.

2.3 Ephemeris Generation

The newly created AstOrb-like file of synthetic objects is passed to the Fortran ephemeris calculator. Ephemerides for each asteroid is generated hourly for three hundred sixty-five days begining at start date May, 12 2015, (MJD 57154.50078). The assumed H magnitude of the synthetic objects is 0.0, and assumed slope parameter G is 0.15.

2.4 Simulated Observation & Noise Addition

Based on the user-specified observational cadence details a subset of the generated ephemeris is selected. This subset of ephemerides are the sky coordinates of the synthetic object. In order to simulate an observation of the object at these coordinates noise is added to the Julian date (in seconds), right ascension (in seconds), and the declination (in arcseconds). This noise addition is handled by the ENDICOTT_noise_loop module which adds random noise from the uniform interval ($\pm$) specified by the user.

This noise addition simulates conditions of a real telescopic observation due to atmospheric seeing conditions, and electronic noise associated with the CCD chip.

periarg(deg)	ascnode(deg)	inc(deg)	ecc(N)	SA (AU)	D-value
75.4536300	120.7310000	2.7271112	0.5259383	1.7390420	
75.5169010	120.7008757	2.7317214	0.5267244	1.7416143	
-0.0632710	0.0301243	-0.0046102	-0.0007861	-0.0025723	0.0019992

Table 3: Example of final code output.

2.5 Orbit Fitting

The noisy ephemeris are then passed to the Fortran orbital fitting code. This code also takes the original orbital elements for the synthetic object as an initial guess for the orbit. The orbital fit returns a set of orbital elements.

2.6 D-criteria calculation

Once orbital fits were determined, the "true" orbit of each synthetic asteroid was compared to the "noisy" orbital fit from the simulated observations using the D-value (Eq. 1.1). The code then takes the true orbit and the noisy orbit, computes the difference of each orbital element (or, in the case of inclination a difference of $sin(i)$), and submits these values to the D-value calculator module.

The result is output to a text file with three lines per synthetic asteroid: original orbital elements, "observed" orbital elements, and differences with D-value. An example of this output is shown below (Table 3).

2.7 Data Acquistion

For each of the orbital families, and for a generic NEO class, the code was run to simulate a large number of synthetic asteroids.

Observations were simulated at varying cadences. In order to maintain consistency across simulated observation sessions only the interval between sessions was adjusted. Each set of simulated observations consisted of five sessions of observation, with four observations per session, at one hour intervals. The number of days between sessions varied between a minimum of one day, and a maximum of 180 days.

For each of these families and cadences observations were simulated by adding noise to date, right ascension, and declination with noise levels of 0.00, $\pm$0.01, $\pm$0.10, and $\pm$1.00.

For each combination of family, cadence, and noise level a text file was output containing the original orbital elements, the calculated orbital elements, and a row of differences, including the D-value for each synthetic asteroid.

CHAPTER 3

ANALYSIS

3.1 D-value as a Function of Orbital Parameters

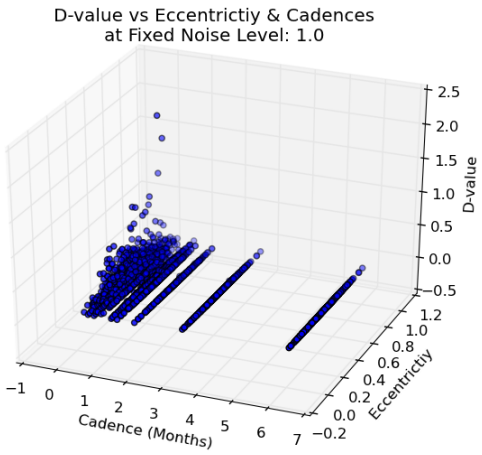
Plots of the D-values calculated at fixed noise level follows. Each orbital element is on a separate plot to illustrate D-value dependence, if any, on the value of the given element.

As an example, the eccentricity of generic NEOs at fixed noise level 1.0 is shown in Figure 2(a). Notice for orbital cadences of greater than one month the D-value is consistently zero for any value of eccentricity. This means that orbital fit is extremely good for observation sessions separated by a month or more even at the highest noise level. Figure 2(b) shows the same data for cadences less than one month.

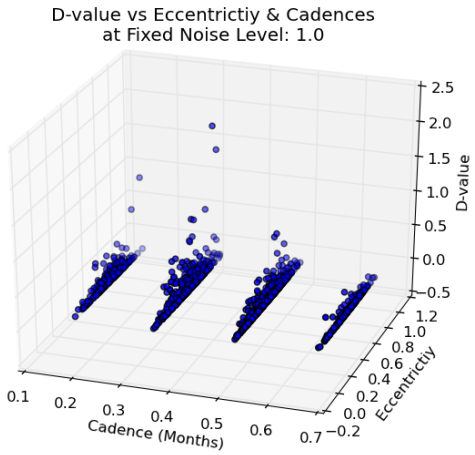
D-value is low or zero for all observing intervals greater than one month for all noise levels, orbital parameters, and asteroid subgroups. The following plots omit the cadences greater than one month. (Figs. 2 - 21).

3.2 Argument of Perihelion

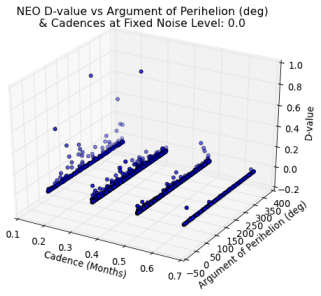
There is no correlation between argument of perihelion and D-value for any group or noise level (Fig. 2 - 5). Notably, for the Atira group at low-noise results show excellent orbital fit for all cadences (Fig. 3(e)). Also note the increased z-scale at high noise-added (Fig. 5) compared to the lower noise levels.



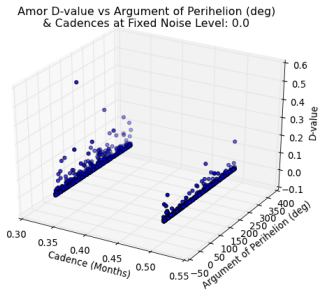
(a) Plot of D-value versus Observation Cadence (in months) and Eccentricity at maximum noise-added. Note at observational cadences greater than one monht D-value remains flat for all values of the orbital parameter, this holds true for lesser noise added.



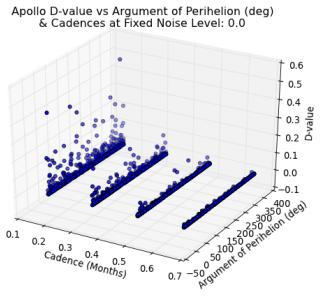
(b) Plot of D-value versus Observation Cadence (in months) and Eccentricity at maximum noise-added. For very short cadences, D-value varies greatly due to points used for orbital fit that are closely grouped in time. Maximum D-value of 2.022 occurs at 10 day observing interval.



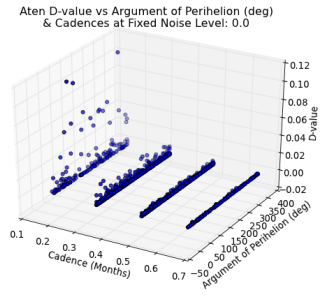
(c) NEO, ω, Zero Noise



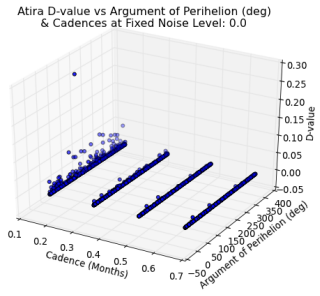
(d) Amor, ω, Zero Noise



(e) Apollo, ω, Zero Noise

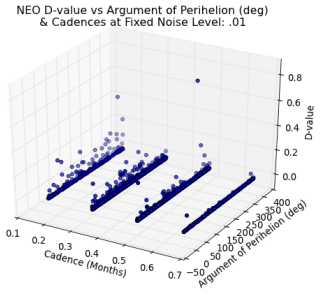


(f) Aten, ω, Zero Noise

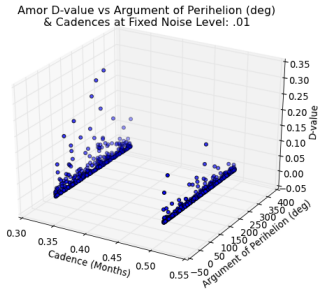


(g) Atira, ω, Zero Noise

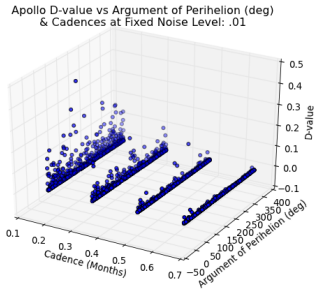
Figure 2: D-value vs. Observation Cadence and Argument of Perihelion, ω, at zero noise-added, all groups. No correlation between D-value and Argument of Perihelion at zero noise.



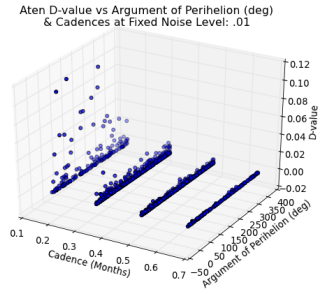
(a) NEO, ω, Low Noise



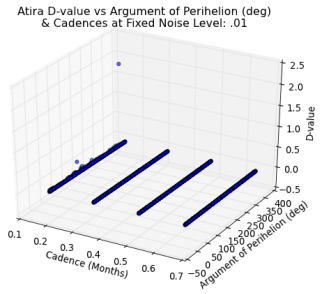
(b) Amor, ω, Low Noise



(c) Apollo, ω, Low Noise

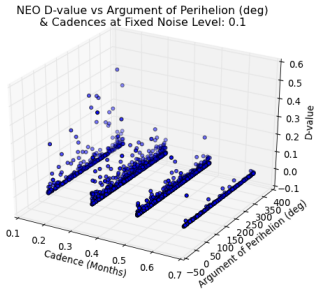


(d) Aten, ω, Low Noise

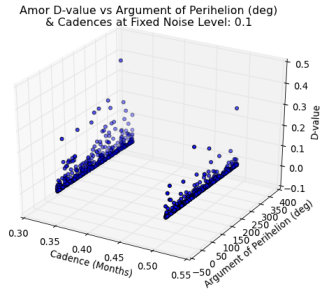


(e) Atira, ω, Low Noise

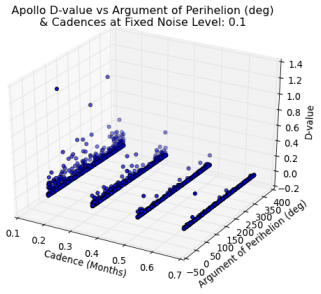
Figure 3: D-value vs. Observation Cadence and Argument of Perihelion, ω, at low noise-added, all groups. No correlation between D-value and Argument of Perihelion at the zero noise level.



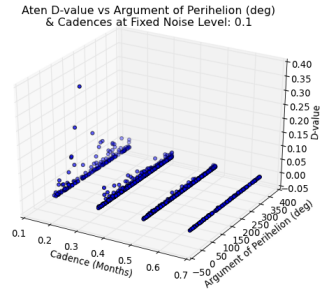
(a) NEO, ω, Medium Noise



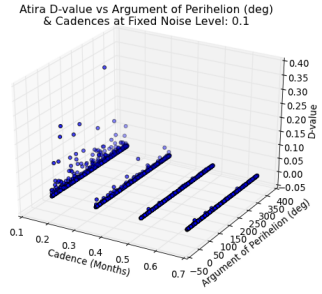
(b) Amor, ω, Medium Noise



(c) Apollo, ω, Medium Noise

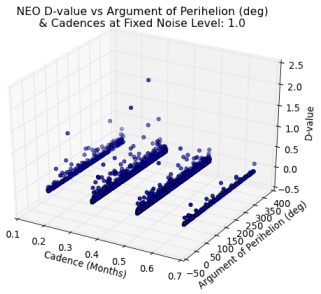


(d) Aten, ω, Medium Noise

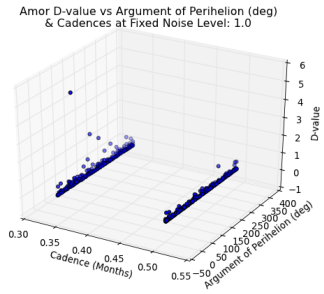


(e) Atira, ω, Medium Noise

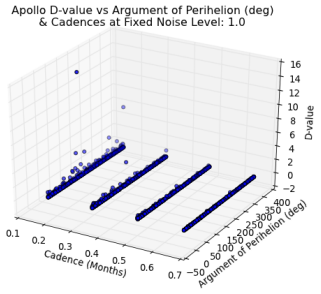
Figure 4: D-value vs. Observation Cadence and Argument of Perihelion, ω, at medium noise-added. No correlation between D-value and Argument of Perihelion at this noise level.



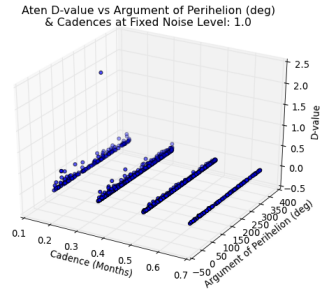
(a) NEO, ω, High Noise



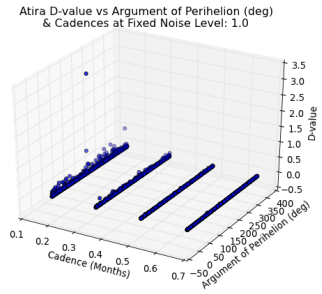
(b) Amor, ω, High Noise



(c) Apollo, ω, High Noise



(d) Aten, ω, High Noise

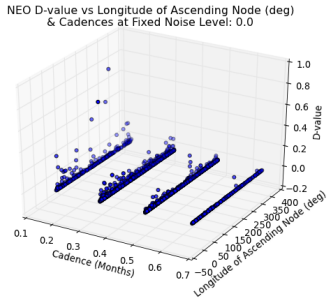


(e) Atira, ω, High Noise

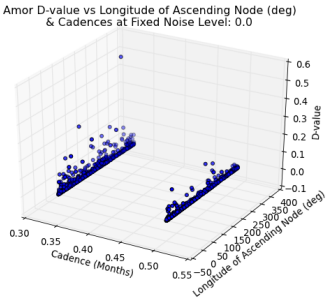
Figure 5: D-value vs. Observation Cadence and Argument of Perihelion, ω, at high noise-added. No correlation between D-value and Argument of Perihelion at the high noise level.

3.3 Longitude of Ascending Node

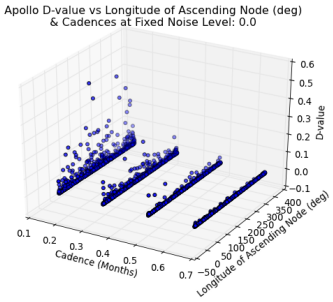
Similar to argument of perihelion, there is also no correlation between Longitude of Ascending Node and D-value for any noise level (Figs. 6 - 9). Neither argument of perihelion, nor longitude of ascending node correlate to D-value, therefore matching the non-uniform distrbution of these orbital elements in our synthetic object dataset is unnecessary.



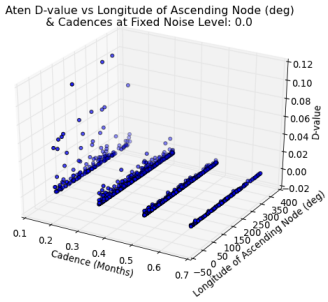
(a) NEO, Ω, Zero Noise



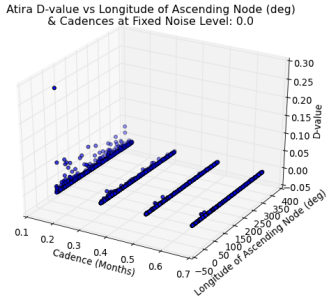
(b) Amor, Ω, Zero Noise



(c)

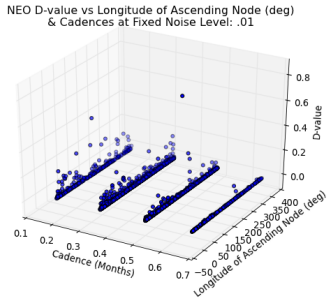


(d) Aten, Ω, Zero Noise

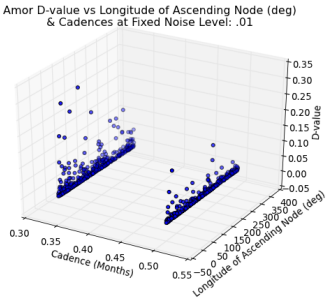


(e) Atira, Ω, Zero Noise

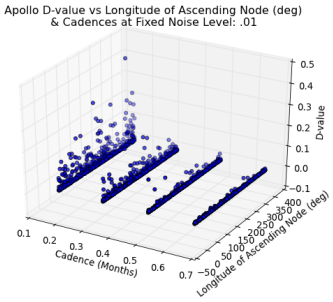
Figure 6: D-value versus Observation Cadence and Longitude of Ascending Node, Ω, at zero noise-added. No correlation between D-value and Longitude of Ascending Node at zero noise.



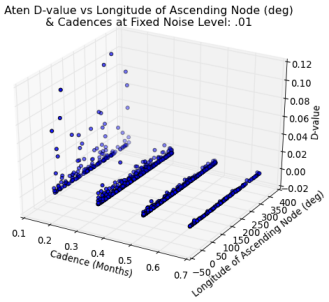
(a) NEO, Ω, Low Noise



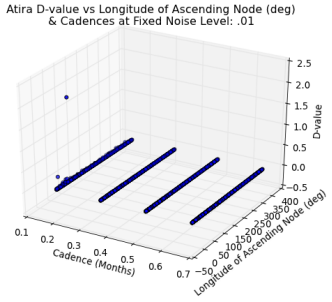
(b) Amor, Ω, Low Noise



(c) Apollo, Ω, Low Noise

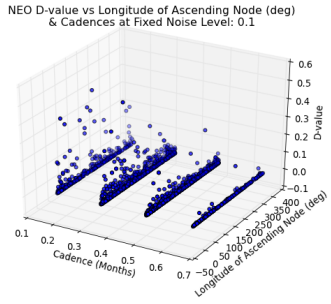


(d) Aten, Ω, Low Noise

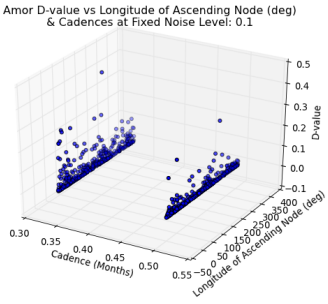


(e) Atira, Ω, Low Noise

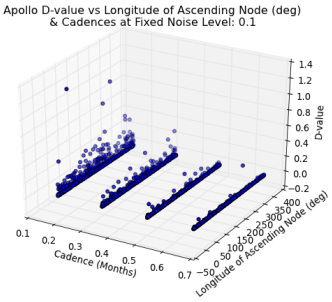
Figure 7: D-value versus Observation Cadence and Longitude of Ascending Node, Ω, at low noise-added. No correlation between D-value and Longitude of Ascending Node at low noise.



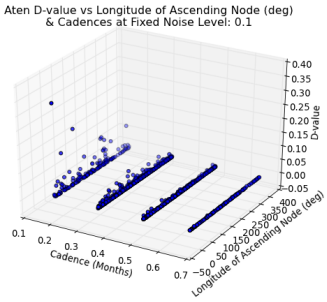
(a) NEO, Ω, Medium Noise



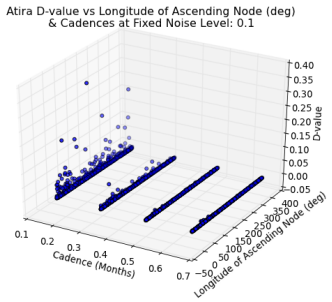
(b) Amor, Ω, Medium Noise



(c) Apollo, Medium Noise

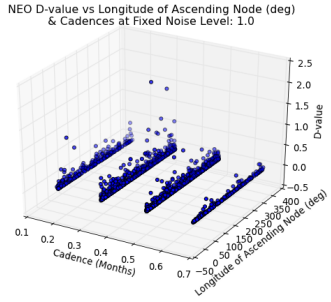


(d) Aten, Ω, Medium Noise

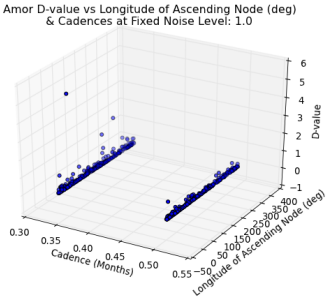


(e) Atira, Ω, Medium Noise

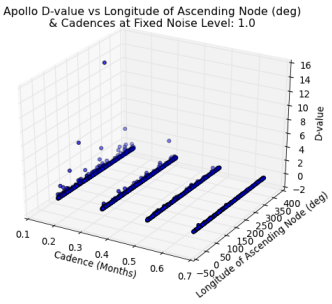
Figure 8: D-value versus Observation Cadence and Longitude of Ascending Node, Ω, at medium noise-added. No correlation between D-value and Longitude of Ascending Node at medium noise.



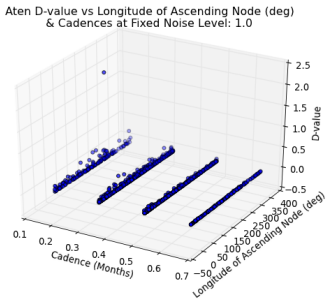
(a) NEO, Ω, High Noise



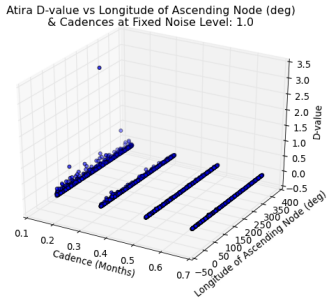
(b) Amor, Ω, High Noise



(c)



(d) Aten, Ω, High Noise

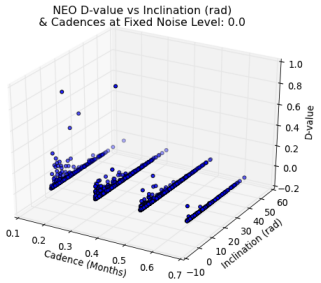


(e) Atira, Ω, High Noise

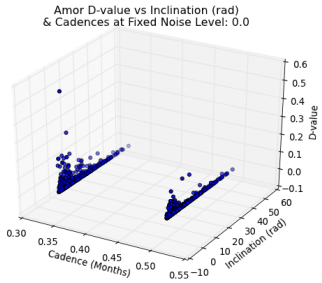
Figure 9: D-value versus Observation Cadence and Longitude of Ascending Node, Ω, at high noise-added. No correlation between D-value and Longitude of Ascending Node at high noise.

3.4 Inclination

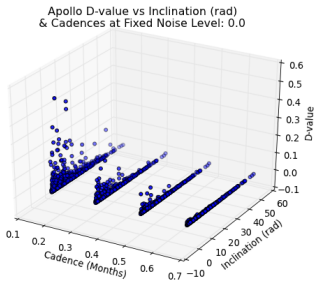
Orbital fit for low inclinations and short cadences results in large D-values, for all noise levels. but orbital fit improves with increasing inclination, even at low observing cadences. This holds true for all noise levels (Figs. 10 - 13). For objects at high inclination even observations separated by one day produce good orbital fits.



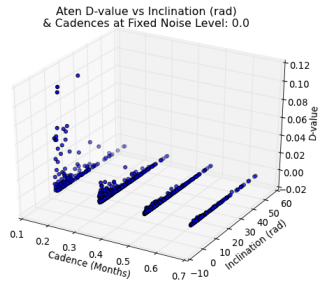
(a) NEO, *i*, Zero Noise



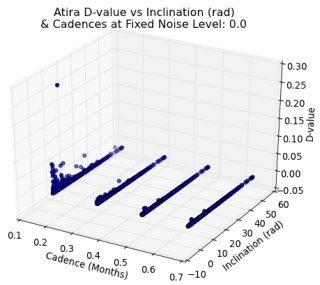
(b) Amor, *i*, Zero Noise



(c) Apollo, *i*, Zero Noise

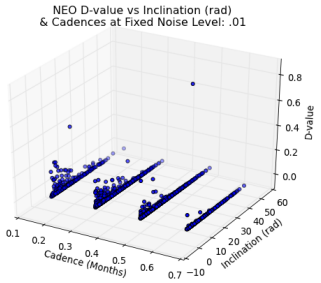


(d) Aten, *i*, Zero Noise

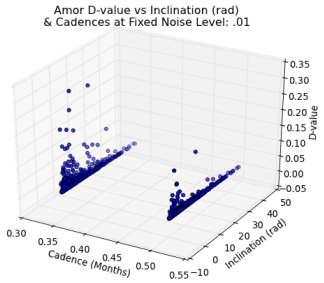


(e) Atira, *i*, Zero Noise

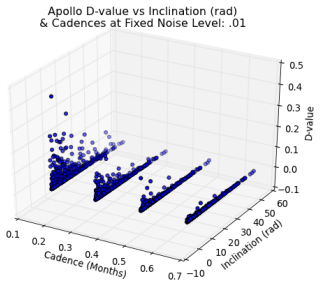
Figure 10: D-value versus Observation Cadence and Inclination, *i*, at zero noise-added. No correlation between D-value and Inclination at zero noise.



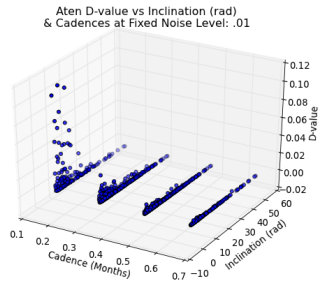
(a) NEO, *i*, Low Noise



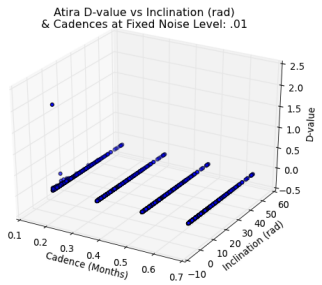
(b) Amor, *i*, Low Noise



(c) Apollo, *i*, Low Noise

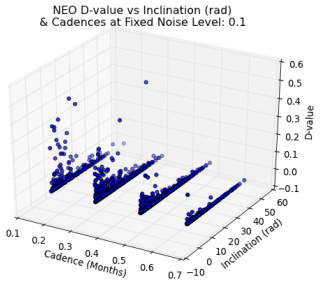


(d) Aten, *i*, Low Noise

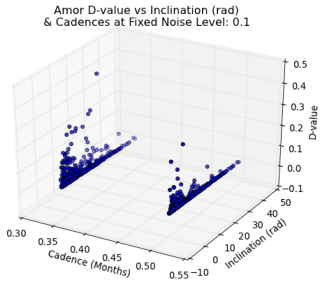


(e) Atira, *i*, Low Noise

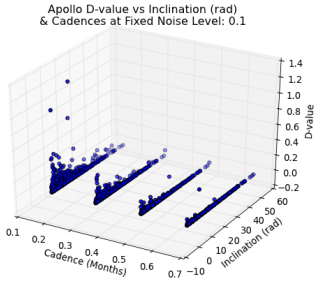
Figure 11: D-value versus Observation Cadence and Inclination, *i*, at low noise-added. No correlation between D-value and Inclination at low noise.



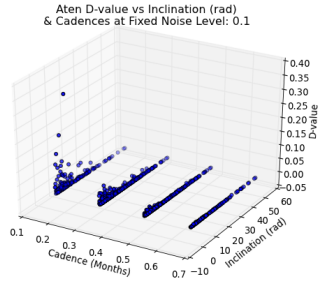
(a) NEO, *i*, Medium Noise



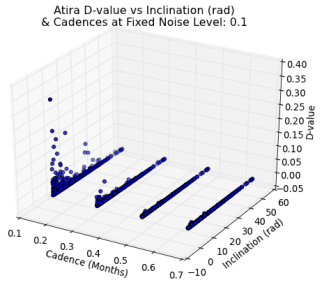
(b) Amor, *i*, Medium Noise



(c) Apollo, *i*, Medium Noise

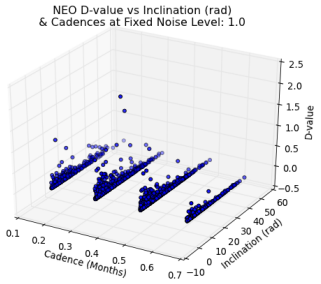


(d) Aten, *i*, Medium Noise

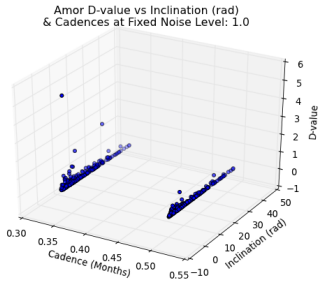


(e) Atira, *i*, Medium Noise

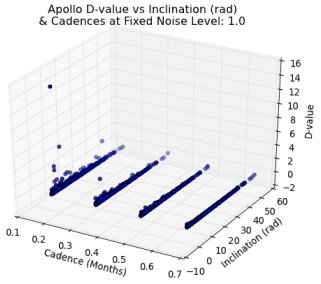
Figure 12: D-value versus Observation Cadence and Inclination, *i*, at medium noise-added. No correlation between D-value and Inclination at medium noise.



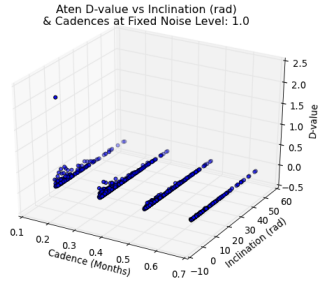
(a) NEO, i, High Noise



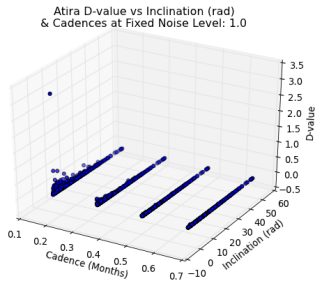
(b) Amor, i, High Noise



(c) Apollo, i, High Noise



(d) Aten, i, High Noise

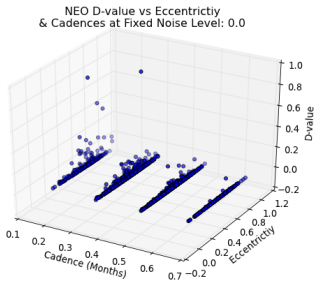


(e) Atira, i, High Noise

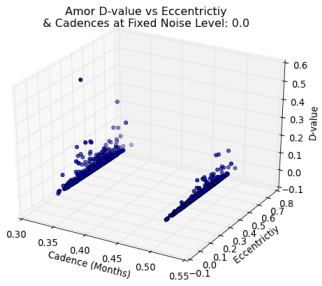
Figure 13: D-value versus Observation Cadence and Inclination, i, at high noise-added. No correlation between D-value and Inclination at high noise.

3.5 Eccentricity

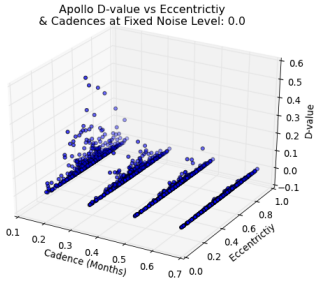
There is no consistent behaviour between eccentricity and orbital fit. The Atira subgroup shows a poor orbital fit for low eccentricities (Figs. 14(e), 16(e), 17(e)), while D-values peak in the upper half of the eccentricity range for generic NEOs (Figs. 14(a), 15(a), 16(a), 17(a)), and the Apollo subgroup (Figs. 14(c), 15(c), 16(c)).



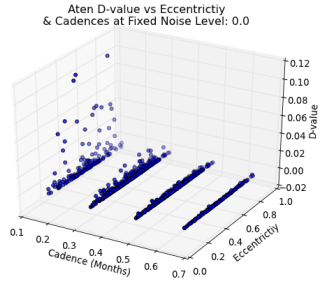
(a) NEO, *e*, Zero Noise



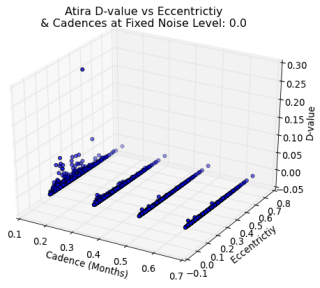
(b) Amor, *e*, Zero Noise



(c) Apollo, *e*, Zero Noise

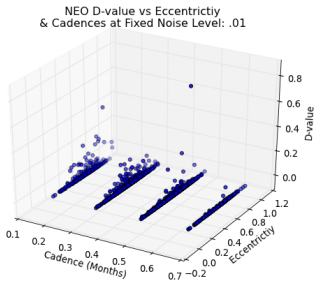


(d) Aten, *e*, Zero Noise

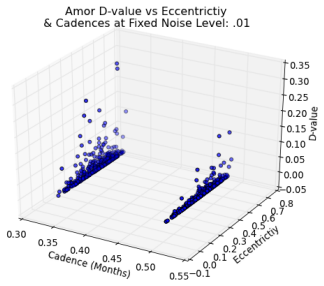


(e) Atira, *e*, Zero Noise

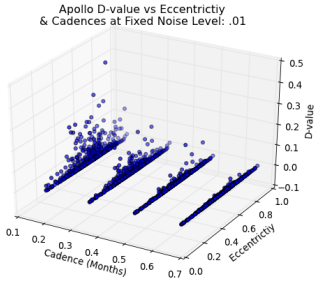
Figure 14: D-value versus Observation Cadence and Eccentricity, *e*, at zero noise-added. No correlation between D-value and Eccentricity at zero noise.



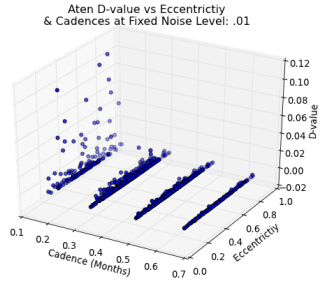
(a) NEO, *e*, Low Noise



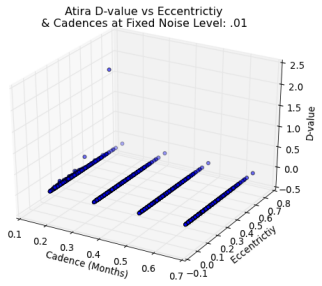
(b) Amor, *e*, Low Noise



(c) Apollo, *e*, Low Noise

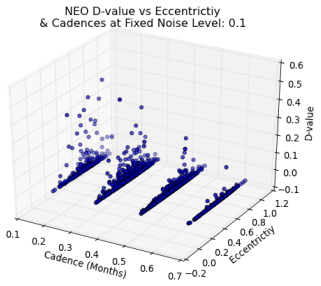


(d) Aten, *e*, Low Noise

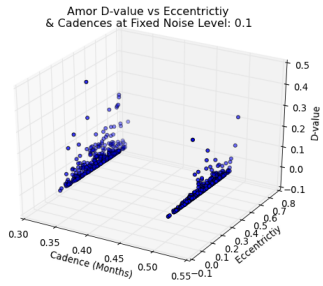


(e) Atira, *e*, Low Noise

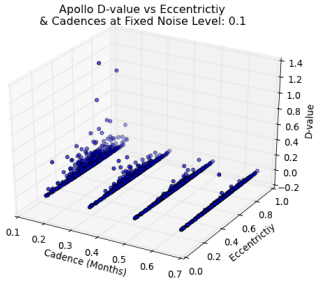
Figure 15: D-value versus Observation Cadence and Eccentricity, *e*, at low noise-added. No correlation between D-value and Eccentricity at low noise.



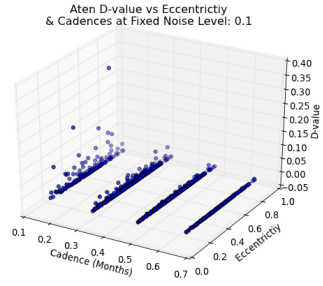
(a) NEO, *e*, Medium Noise



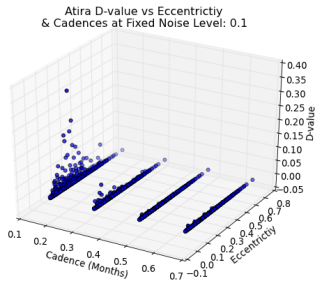
(b) Amor, *e*, Medium Noise



(c) Apollo, *e*, Medium Noise

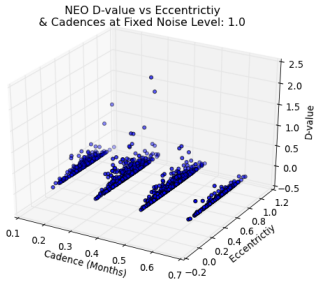


(d) Aten, *e*, Medium Noise

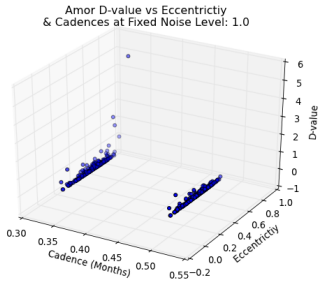


(e) Atira, *e*, Medium Noise

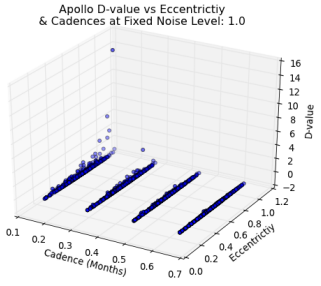
Figure 16: D-value versus Observation Cadence and Eccentricity, *e*, at medium noise-added.No correlation between D-value and Eccentricity at medium noise.



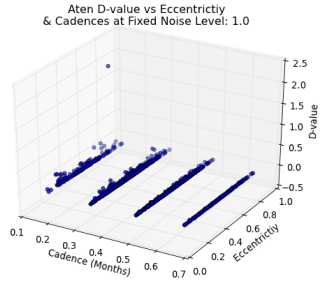
(a) NEO, *e*, High Noise



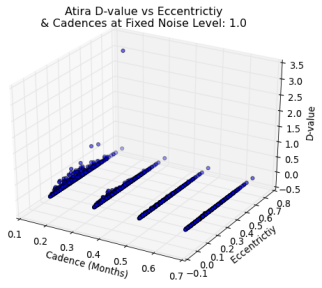
(b) Amor, *e*, High Noise



(c) Apollo, *e*, High Noise



(d) Aten, *e*, High Noise

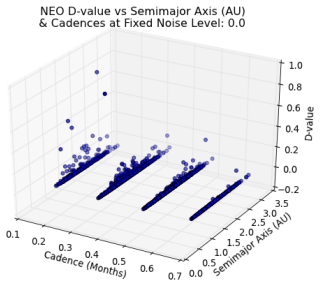


(e) Atira, *e*, High Noise

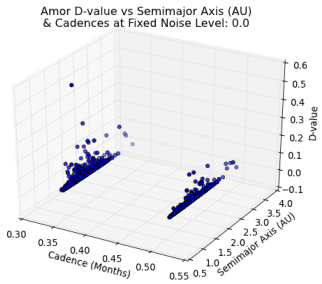
Figure 17: D-value versus Observation Cadence and Eccentricity, *e*, at high noise-added. No correlation between D-value and Eccentricity at high noise.

3.6 Semi-major Axis

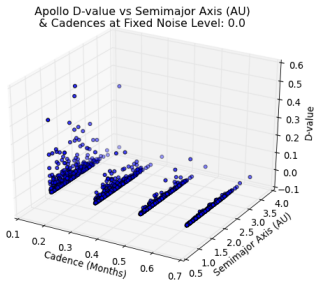
Very low values of semi-major axis correspond to good orbital fit. Large D-values occur only at the mid-to-high end of the semi-major axis range (Figs. 18(a), 19(c), 20(e), 21(a)) The Aten family shows increasing D-value as semi-major axis increases toward 1AU (and Earth's orbit), but for very low semi-major Axis values orbital fit is very good for all noise levels and observational cadences.



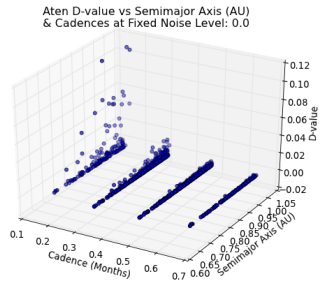
(a) NEO, a, Zero Noise



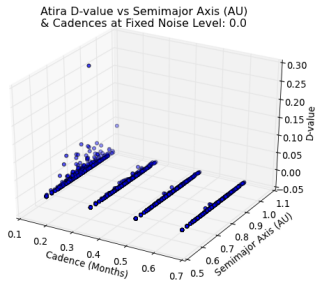
(b) Amor, a, Zero Noise



(c) Apollo, a, Zero Noise

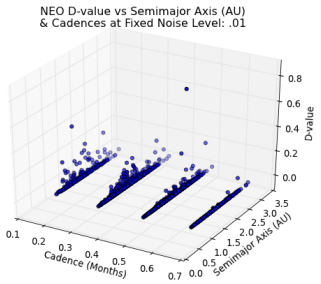


(d) Aten, a, Zero Noise

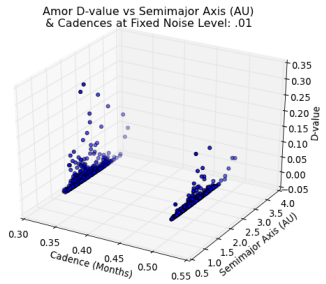


(e) Atira, a, Zero Noise

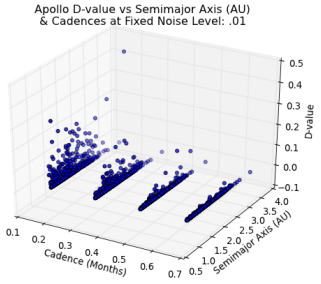
Figure 18: D-value versus Observation Cadence and Semi-major Axis, a, at zero noise-added. No correlation between D-value and Semi-major Axis at zero noise.



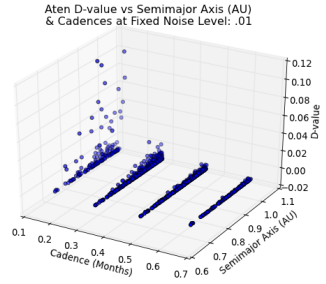
(a) NEO, *a*, Low Noise



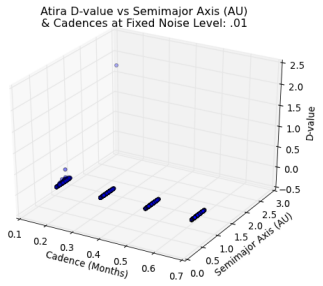
(b) Amor, *a*, Low Noise



(c) Apollo, *a*, Low Noise

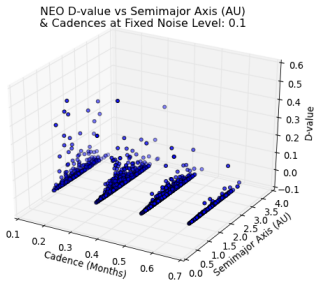


(d) Aten, *a*, Low Noise

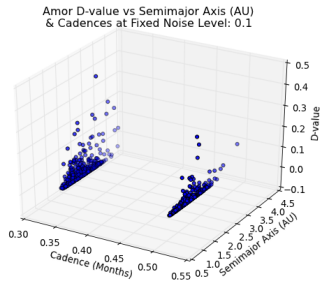


(e) Atira, *a*, Low Noise

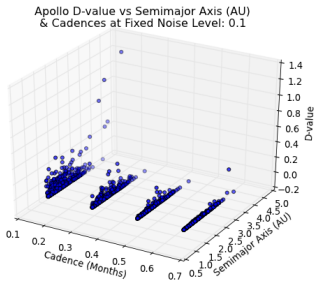
Figure 19: D-value versus Observation Cadence and Semi-major Axis, *a*, at low noise-added. No correlation between D-value and Semi-major Axis at low noise.



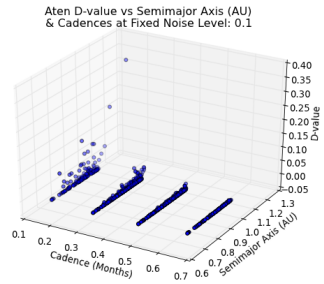
(a) NEO, a, Medium Noise



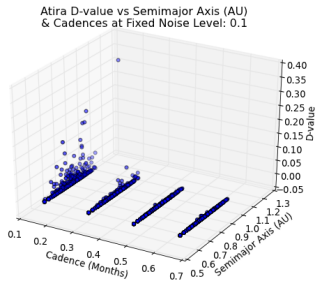
(b) Amor, a, Medium Noise



(c) Apollo, a, Medium Noise

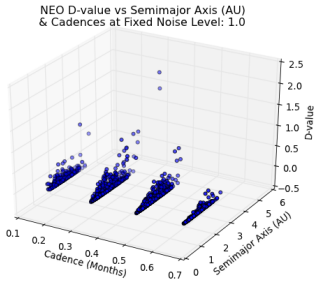


(d) Aten, a, Medium Noise

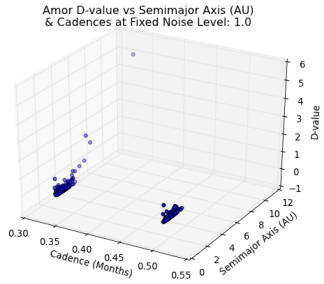


(e) Atira, a, Medium Noise

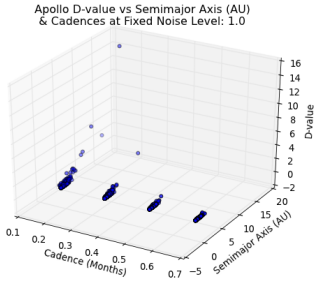
Figure 20: D-value versus Observation Cadence and Semi-major Axis, a, at medium noise-added. No correlation between D-value and Semi-major Axis at medium noise.



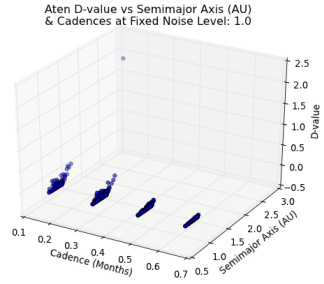
(a) NEO, a, High Noise



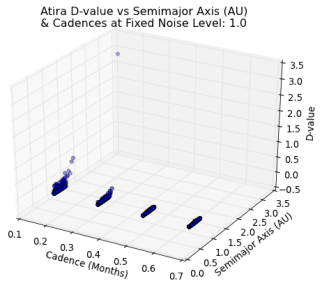
(b) Amor, a, High Noise



(c) Apollo, a, High Noise



(d) Aten, a, High Noise



(e) Atira, a, High Noise

Figure 21: D-value versus Observation Cadence and Semi-major Axis, a, at high noise-added. No correlation between D-value and Semi-major Axis at high noise.

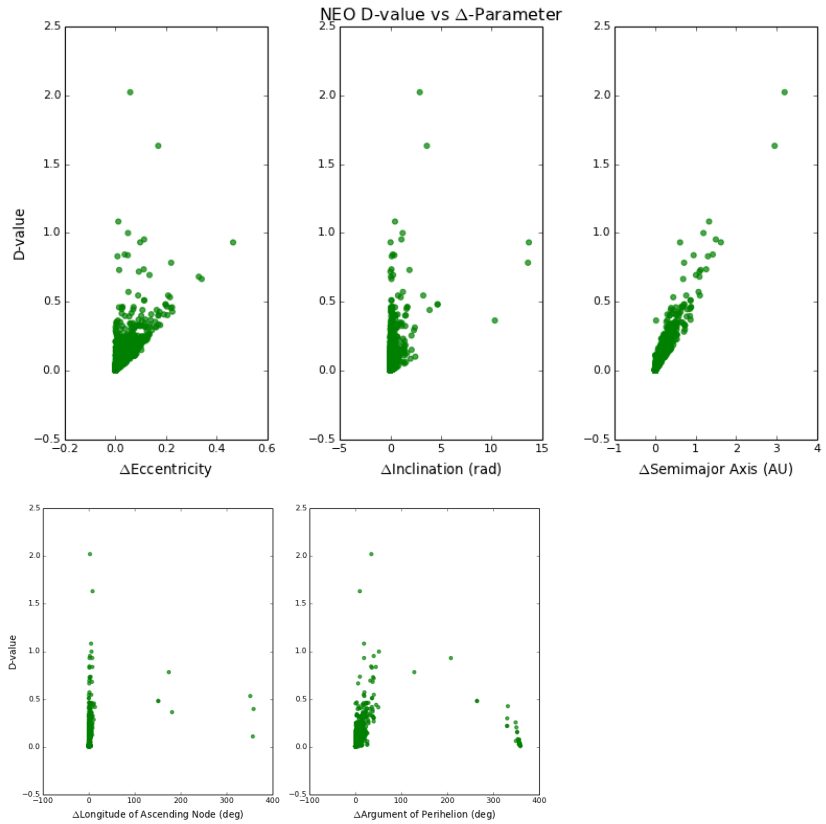


Figure 22: D-value versus Δ-Parameter for generic NEOs.

3.7 D-value as a Function of Δ-Parameter

Plots of D-value versus the difference in orbital parameter follow. Here we are interested in how a change in the value of orbital parameter between the "true" orbit and the "noisy" orbit affects orbital fit. Plots of orbital elements are arranged by NEO subgroup. We draw no conclusions regarding a difference of argument of perihelion, or longitude of ascending node with regard to D-value. However, there does appear to be a relationship between a difference in eccentricty and D-value, as well as a linear dependence between change of semi-major axis and D-value.

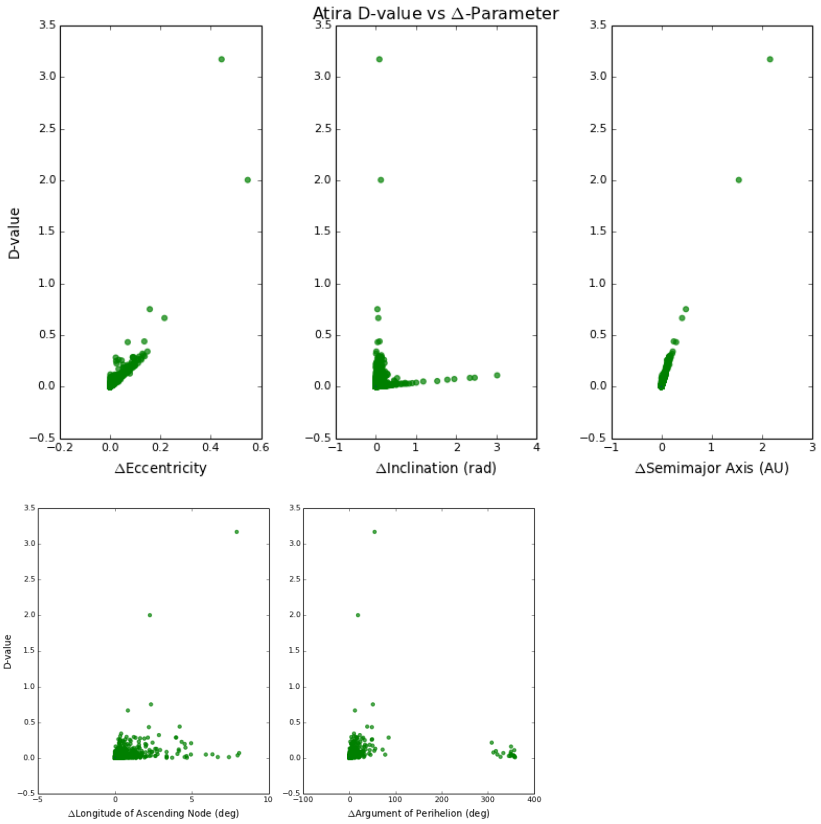


Figure 23: D-value versus Δ-Parameter for Atiras.

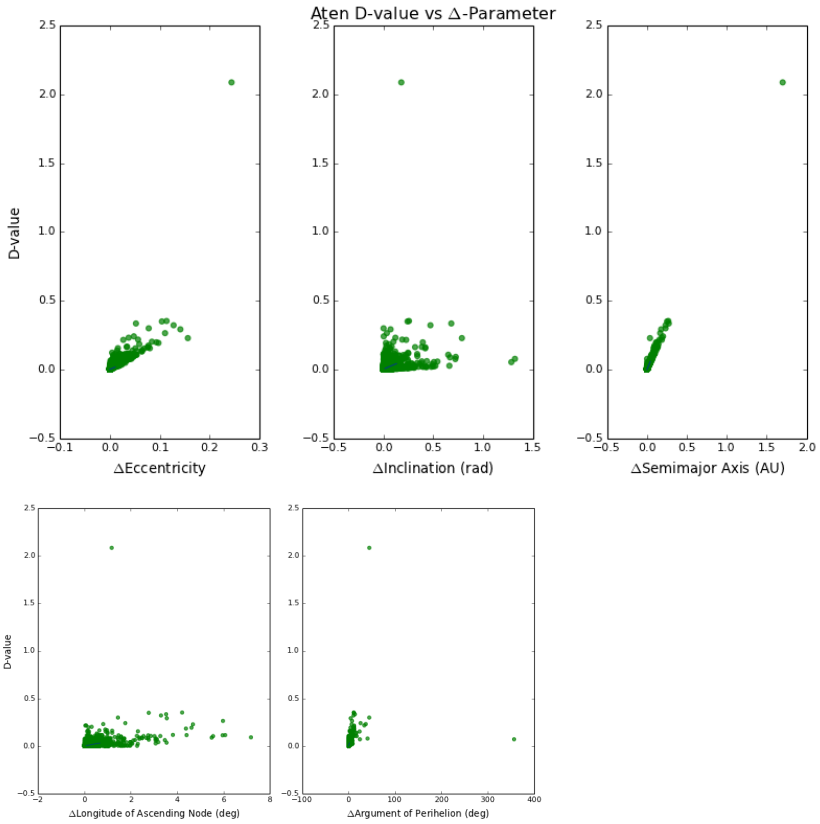


Figure 24: D-value versus Δ-Parameter for Atens.

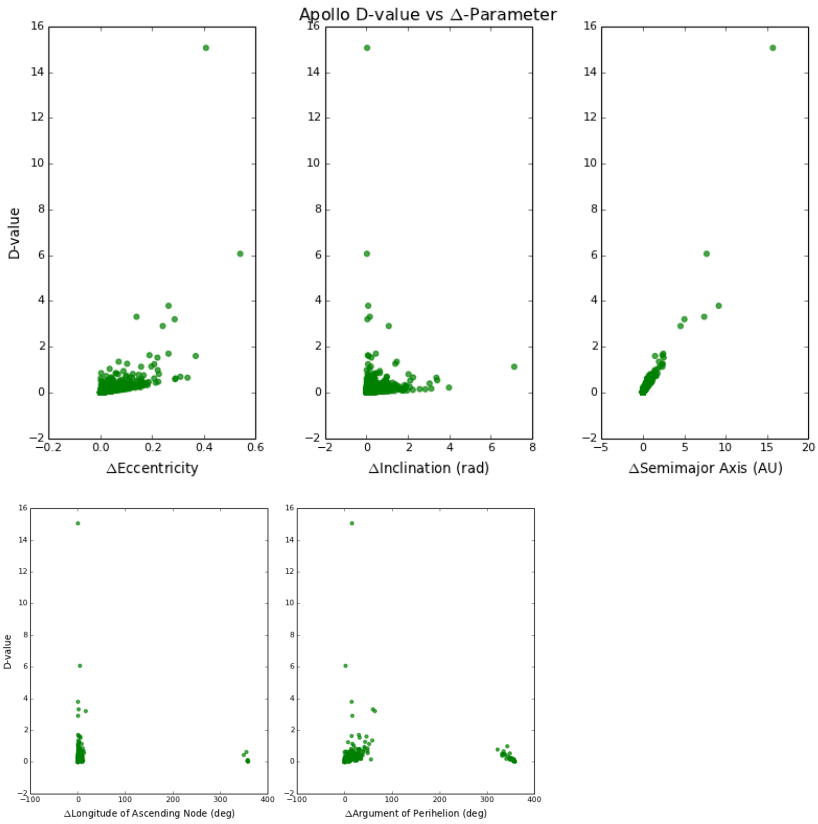


Figure 25: D-value versus Δ-Parameter for Apollos.

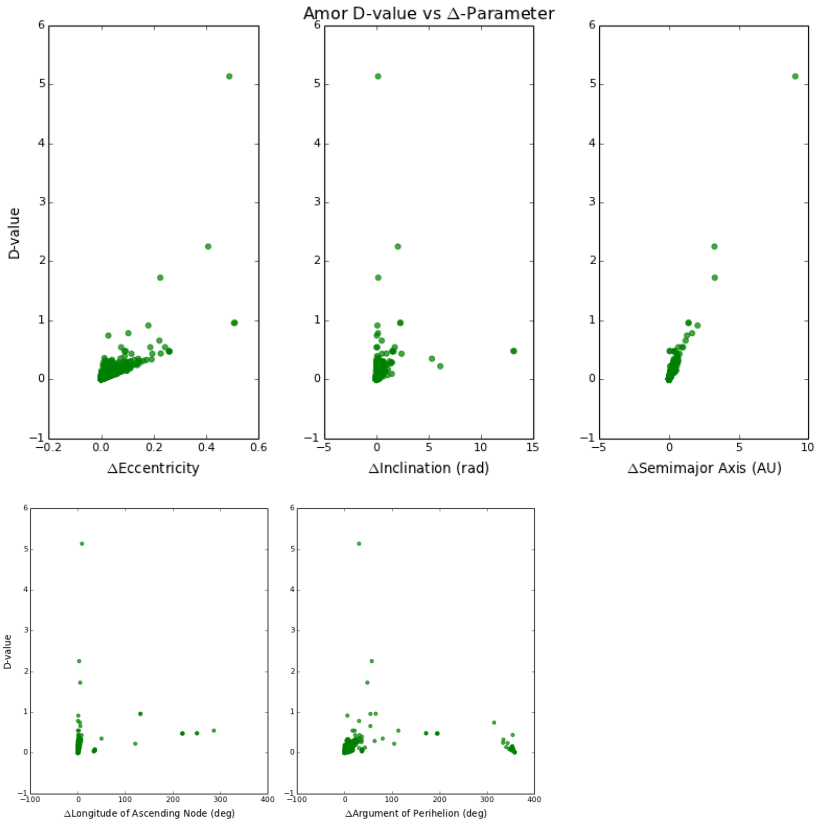


Figure 26: D-value versus Δ-Parameter for Amors.

CHAPTER 4

CONCLUSION

Observational cadences of one month or greater lead to excellent orbital fits, quantified by near-zero D-value, for asteroid subgroups without regard to the value or distribution of their orbital parameters, or the level of noise-added to the simulated observations.

For very short cadences, daily, or weekly, orbital fit is poor unless the object has inclination greater than $\sim$45 degrees, except at the highest noise level and shortest observation cadence (Fig. 13(a)).

The Earth-crossing subgroups, Atens and Apollos, show poor orbital fits at semi-major axis values close to 1AU, and improving orbital fits as semi-major axis moves inward and outward, respectively, from that distance. The improvement is more pronounced in the Aten subgroup (Fig. 19).

There is no correlation between the value of the angular orbital parameters, argument of perihelion and longitude of ascending node, and D-value. Orbital fit is not affected by the non-uniform distribution of these elements in the observed NEO population.

When comparing D-value to the difference between "true" and "noisy" orbital parameters, we see no relationship in the argument of perihelion, or longitude of ascending node. For all groups, however, difference in semi-major axis value is in direct proportion to D-value. There also appears to be a psuedo-proportionality between difference in eccentricty and D-value.

We conclude, that in order to calculate a good orbital fit for NEOs, separating observational sessions by one month or more is necessary. However, good orbital fits can still be obtained at shorter observational cadences for objects with large inclinations, or objects with semi-major axes less than 1 AU.

CHAPTER 5

FURTHER WORK

Additional goals of this project include improving the code user-interface, perhaps to develop a GUI instead of bash shell interaction. More substantively, incorporating open source, publicly available modules for ephemeris generation and for orbit fitting would allow this code to exist as a standalone executable, independent of the fortran routines hosted remotely.

The data set for this project can be expanded dramatically by allowing more freedom of choice in the intial parameters, especially with regard to the user-specified observing cadence values. Allowing the number of observations per session, and number of sessions presets to vary will introduce a larger dimensional space to explore, and this flexibility grants the ability to hew these presets more closely to the actual details of a particular telescope (ie, noise level) or observer (ie, observation pattern). As written, the code can be configured for any number of asteroids, of any type available in the AstOrb database, to generate any ephemeris interval, and simulate any specified observational noise and cadence.

Additional goals also include exploring how D-value correlates to predicted ephemeris position when comparing true orbits to noisy orbits in order to help predict recovery or loss of a particular object at some noise level.

Lower noise regimes should also be explored. The European Space Agency's Gaia Space Observatory [ES] gives much more precise astrometry ($\sim$0.001") than traditional

ground-based observing ($\sim$0.1), determining what cadences are necessary to meet a particular D-value threshold using space-based or ground-based observing would be useful.

CHAPTER 6

ACKNOWLEDGEMENTS

This research was supported in part by Dr. Nicholas Moskovitz, Assistant Astronomer, Lowell Observatory, who provided oversight and encouragement on the asteroid science, and Dr. Larry Wasserman, Astronomer, Lowell Observatory, who wrote and maintains the ephemeris generating and orbital fitting Fortran code. Additionally, the author would like to thank Clément Royer, Summer Research Assistant, Lowell Observatory, for his contribution to the Python code, and Dr. David Polishook of Weizmann Institute of Science for his commentary and support.

APPENDIX A

```
"""cadence.py

Generate synthetic NEOs, generate ephemeris, simulate observations based
     on added noise, and observational cadence, and return orbital fit
    difference (D-value)

Usage:
    python cadence.py

Notes:
    - This script requires python 2.7
    - pip dependencies can be found in requirements.txt
    - Last modified July 2, 2017
"""

# import statements
import cadence_modules as mods
import subprocess


def cadence(
        family_string, object_count, ephemeris,
        noise_list, cadence_list):
"""This function calls ephemeris calculator,
    create noisy ephemeris, grabs noisy eph for a particular
    cadence, call minidiffs for that cadence and appends
    result of minidiff and D calc to family+details.end
"""""

    # NEO_Selection creates astorb-like file for family
    mods.selectNEOs(family_string,object_count)

    astorb_like_list=[]
    # Read in synth_family_astorb.dat
    with open('synth_'+family_string+'_astorb.dat','r') as fam_astorb:
```

```
        for line in fam_astorb: astorb_like_list.append(line.rstrip('\n')
            .split('\n'))

    # call ef8, generates ef8.out
    # ef8 written and maintained by L.Wasserman
    mods.generate_eph(family_string, ephemeris)

    for x in noise_list:
    # create a noisy ephemeris file
        mods.noisy_eph(x,family_string)
        with open('noisy_'+family_string+'_'+x[3]+'_ef8.out') as nef8:
            eph_per_object = sum(1 for line in nef8)/object_count
        # simulate observations at some cadence
        for y in cadence_list:
            #once each for the synthetic objects, call minidiff and
                compare orbits
            for z in xrange(0,object_count):
                # create minidiff stuff
                mods.make_mininput()
                mods.asteroid_files(z,astorb_like_list, x, y,
                    eph_per_object,family_string)
                # minidiff written and maintained by L.Wasserman
                subprocess.call('../fortran_binaries/minidiff < mininput',
                    shell=True)
                mods.orbit_files(family,x,y)
            # compare noisy orbits and perform the D calculation. write to
                 .end files
            # .end files contain row of original orbital parameters,
            # row of noisy orbital parameters
            # and row of differences plus D-value, 3 rows per object
            mods.compare(family, x, y)



if __name__ == '__main__':

    # USER INPUT, default values included

    family = raw_input('Enter asteroid family (NEO,Aten,Atira,Amor,Apollo
        ): [neo]\n').lower() or 'neo'
    print 'You entered ', family.upper(), '\n'
```

```
objects = int(raw_input('Enter integer desired number of asteroids of
    each type: [15]\n') or 15)
print 'You entered ', objects, '\n'

# Prompt for ephemeris details
eph_date= raw_input('Please enter an ephemeris start date separated
   by commas (YYYY,MM,DD.DDDD): [2015,05,12.0]') or [2015,05,12.0]
eph_int = raw_input('Please enter an ephemeris interval separated by
   commas (DD,HH,MM): [00,01,00]') or [00,01,00]
eph_dur = raw_input('Please enter the duration for ephemeris
   calculation: (DDD,HH,MM): [365,00,00]') or [365,00,00]
eph_deets = [eph_date, eph_int, eph_dur, 'ef8.out']
print 'You entered ', eph_deets

# Prompt for Noise details
date_noise = float(raw_input('Please enter noise value for Julian
   Date: [0.01]') or 0.01)
ra_noise = float(raw_input('Please enter noise value for Right
   Ascension: [0.01]') or 0.01)
dec_noise = float(raw_input('Please enter noise value for Declination
   : [0.01]') or 0.01)
str_noise = raw_input('Please enter an identifier string for this
   noise combination') or 'noisestr'
noises = [[date_noise, ra_noise, dec_noise, str_noise]]
print 'You entered ', noises

# Prompt for cadence details
nb_obs = int(raw_input('Please enter the number of observations
   desired per session: [4]') or 4)
obs_int = int(raw_input('Please enter the hourly interval between
   these observations: [1]') or 1)
nb_days = int(raw_input('Please enter the total number of obsevation
   sessions: [5]') or 5)
days_int = int(raw_input('Please enter the interval, in days, between
    these observing sessions. \n Note: Number of sessions x Daily
   intervals must not exceed Ephemeris Duration \n Days Interval:
   [15]') or 15)
freq_str = raw_input('Please enter an identifier string for this
   cadence choice: ') or 'cadencefreq'
print 'You entered ', cadences
cadences = [[nb_obs, obs_int, nb_days, days_int, freq_str]]
# END USER INPUT
```

```
# call main process
cadence(family, objects, eph_deets, noises, cadences)

# shell command to clean up directory after successful run
remove_string = 'rm *.dat *.out *.pyc asteroid.* input mininput'
subprocess.call(remove_string,shell=True)
```

```
"""cadence_modules.py
Contains modules necessary for cadence.py

Notes:
    - This script requires python 2.7
    - Modules also require fortran executables ef8.f, and minidiff.f
    - pip dependencies can be found in requirements.txt
    - Last modified July 6, 2017
"""

# import statements
import os
import math
import random
import operator
import subprocess
import numpy as np

# function definitions

def perihelion(a,e):
"""Calculates perihelion value given a, e"""
    return a*(1-e)


def is_number(someString):
"""Tests if line in ef8.out is ephemeris string (true) or header info (
    false)"""
    try:
        float(someString)
        return True
    except ValueError:
        return False


def format_e(n):
"""Converts float numbers into a sci-not format compatible with minidiff
    """
    #C.Royer formula
    a = '%E' % n
    return a.split('E')[0].rstrip('0').rstrip('.').ljust(18,"0") + 'E' +
        a.split('E')[1]
```

```
def make_mininput():
"""Creates mininput file for minidiff call"""
    with open('mininput','w') as minidiff_input_file:
        minidiff_input_file.write('\n'.join(['asteroid.neworb','asteroid.
            ted']))


def family_check(someSA,someEcc,someFamily):
"""Tests synthetic object for family group
    based on semimajor axis and perihelion criteria"""
    ph = perihelion(someSA,someEcc)
    aph = aphelion(someSA,someEcc)
    if someSA < 1:
        if aph < 0.983: return 'atira'
        else: return 'aten'

    elif someSA > 1:
        if ph < 1.017: return 'apollo'
        if 1.017 < ph < 1.3: return 'amor'
        else: pass


def generate_synth_obj(some2dlist):
"""Generate Synthetic asteroid randomly choosing
    one data point from each column of 6xN 2D list"""
    x = [random.choice(some2dlist[:,0]),random.choice(some2dlist[:,1]),
        random.choice(some2dlist[:,2]),random.choice(some2dlist[:,3]),
        random.choice(some2dlist[:,4]),random.choice(some2dlist[:,5])]
    return x


def JD_to_Greg(someJDString):
"""Calculate Gregorian Date given some Modified Julian Date
adapted from http://aa.usno.navy.mil/faq/docs/JD_Formula.php
original formula is for JD, here we use modified JD.
note forced integer division for 'min' """
    jd = float(someJDString)
    L = jd + 68569 + 2400000.5
    N = 4 * L // 146097
    L = L - (146097 * N + 3) // 4
    I = 4000 * (L + 1) // 1461001
    L = L - 1461 * I // 4 + 31
```

```
    J = 80 * L // 2447
    day = int(L - 2447 * J // 80)
    frac_day = (L - 2447 * J // 80) % 1
    hr = int(frac_day * 24)
    min = int(frac_day * 24 % 1 * 59 // 1) #59, not 60 to correct
        unidentified rounding issue. necessary to match ef8 output
    L = J // 11
    mo = int(J + 2 - 12 * L)
    yr = int(100 * (N - 49) + I + L)
    greg_date_string = '{:4d} {:2d} {:2d} {:2d} {:2d}'.format(yr,mo,day,
        hr,min)
    return greg_date_string


def ENDICOTT_noise_loop(
        someEphemerisString,someDateNoiseFloat,
        someRANoiseFloat,someDecNoiseFloat):
"""Adds specified noise to JD, RA, and Dec, and return noisy versions""
# read in eph as tuple of fixed width strings: [JulianDate,GregorianDate
    (y,m,d,hr,min),sec,rah,ram,ras,dec_deg,dec_min,dec_sec,dm_s,dm_p,eu_s
    ,eu_pp,sAU,eAU,el,ph,eclat,galat,mag]
    [JD,Gregorian,sec,rah,ram,ras,dec_deg,dec_min,dec_sec,data] = (
        someEphemerisString[:11],someEphemerisString[11:28],
        someEphemerisString[29:33],someEphemerisString[33:36],
        someEphemerisString[36:39],someEphemerisString[39:46],
        someEphemerisString[46:50],someEphemerisString[50:53],
        someEphemerisString[53:59],someEphemerisString[60:-1])
    # [:-1] cuts final '\n' char

    # flatten DMS to deg and convert to radians, store positive/negative
        sign
    dec_deg = int(dec_deg)
    posneg = math.copysign(1,dec_deg)
    dec_noise = random.uniform(-someDecNoiseFloat,someDecNoiseFloat)
    ra_noise = random.uniform(-someRANoiseFloat,someRANoiseFloat)
    date_noise = random.uniform(-someDateNoiseFloat,someDateNoiseFloat)
    dec_sec_noisy = (float(dec_sec) + dec_noise)
    declination_noisy = math.radians(abs(dec_deg) + (int(dec_min) / 60.)
        + dec_sec_noisy / 3600.)
    ra = float(rah)+float(ram)/60.+float(ras)/3600.
    dec = math.radians(float(dec_deg) + float(dec_min)/60. + float(
        dec_sec)/3600.)
```

```
    # add noise to declination, add noise and flatten RA
    ra_noisy = float(rah) + (float(ram) / 60.) + (float(ras) / 3600.) + (
        ra_noise / (3600. * 15*math.cos(declination_noisy)))
    # unflatten dec, and format for output
    dec_deg = '{:+03.0f}'.format(posneg*(math.degrees(declination_noisy)
        % 360 // 1))
    dec_min = '{:02.0f}'.format(math.degrees(declination_noisy) % 360 % 1
         * 60 // 1)
    dec_sec = '{:05.2f}'.format(math.degrees(declination_noisy) % 360 % 1
         * 60 % 1 * 60.)

    # unflatten RA and format output
    rah = '{:02d}'.format(int(ra_noisy % 24 // 1))
    ram = '{:02d}'.format(int(ra_noisy % 24 % 1 * 60. // 1))
    ras = '{:06.3f}'.format(ra_noisy % 24 % 1 * 60. % 1 * 60.)

    # add noise to Julian Date, and recover noisy version of Gregorian
        date
    day_frac_noisy = date_noise/86400.
    JD_noisy = '{0:.5f}'.format(float(JD)+day_frac_noisy)
    Greg_noisy = JD_to_Greg(JD_noisy)
    d[0],d[1] = JD_noisy,Greg_noisy
    d[3:9] = [rah,ram,ras,dec_deg,dec_min,dec_sec]

    # format eph string to be written to noisyef8 file
    noisyEphemerisString = ' '.join(map(str,d))+'\n'

    return(noisyEphemerisString)


def clean_ef8_creator(eph_file,familyString):
"""Creates a clean copy of ef8.out for specified family"""
    with open(eph_file,'r') as ef8, open(familyString+'_clean_ef8.out','w
        ') as cleanef8:
        for line in ef8:
            test_for_ephemeris = is_number(line.split(' ')[0]) and
                is_number(line.split(' ')[1])
            if not test_for_ephemeris:
                continue
            else:
                cleanef8.write(line)
```

```
def D_calc(someOldInc,someOldSA,someDifferenceList):
"""Calculate D-value, takes original
SA, Inct, and new: nSA, nEcc, nInc,nNode, nPeriarg
order of elements is as follows:
arg of perihelion[0]; longitude of asc node[1]; inc[2]; ecc[3]; sa[4]"""
    # D-value equation coefficients
    ka = 1.25
    ke = 2
    ki = 2
    knode = 1e-6
    kperiarg = 1e-6
    # D-value formula and calculation
    d = ka*(someDifferenceList[4]/someOldSA)**2 + ke*someDifferenceList
        [3]**2 + (ki*someDifferenceList[2])**2 + knode*someDifferenceList
        [1]**2 + kperiarg*someDifferenceList[0]**2
    D = round(math.sqrt(d),7)
    return D


def compare(familyString, noise_list, cadence_list):
"""Compares orig and noisy orbital parameters and
takes their difference, and calculates D-value"""
    # open noisy and orig orbit.dat
    with open('noisy_'+familyString+'_'+noise_list[3]+'_'+cadence_list
        [4]+'_orbit.dat','r') as noisyorbit, open('synth_'+familyString+'
        _'+noise_list[3]+'_'+cadence_list[4]+'_orbit.dat','r') as
        synthorbit:
        noisy_elmnts = noisyorbit.readlines()
        synth_elmnts = synthorbit.readlines()
    noisy_lines = [x.strip('\n') for x in noisy_elmnts]
    synth_lines = [x.strip('\n') for x in synth_elmnts]

    # prep .end files with header row.
    with open(familyString+'_'+noise_list[3]+'_'+cadence_list[4]+'
        _D_values.end','a') as dval:
        if os.stat(familyString+'_'+noise_list[3]+'_'+cadence_list[4]+'
            _D_values.end').st_size == 0:
            header = ' '.join('{:16s}'.format(x) for x in ['periarg(deg)
                ','longascnode(deg)','inclination(deg)','eccentricity(N)
                ','semimajoraxis(AU)','D-criterion'])
            dval.write(header+'\n')
        noisy = []
        synth = []
```

```
difference = []
for i in xrange(0,len(noisy_lines)):
    if noisy_lines[i] == '':
        continue
    else:
        noisy.append([noisy_lines[i][60:83],noisy_lines[i
            ][83:106],noisy_lines[i][106:129],noisy_lines[i
            ][129:152],noisy_lines[i][152:175]])
        synth.append([synth_lines[i][60:83],synth_lines[i
            ][83:106],synth_lines[i][106:129],synth_lines[i
            ][129:152],synth_lines[i][152:175]])
# order of elements for D-value calc is as follows:
# arg of perihelion (deg)[0]; longitude of asc node(deg)[1]; inc
    (deg)[2]; ecc (N)[3]; semimajor axis (AU)[4]

for i in xrange(0,len(noisy)):
    # convert str to float, angles to radians, take sin(i)
    noisy[i] = map(float(noisy[i])
    noisy[i][:3] = [math.radians(z) for z in noisy[i][:3]]
    noisy[i][2] = math.sin(noisy[i][2])

    # convert str to float, angles to radians, take sin(i)
    synth[i] = map(float(synth[i])
    synth[i][:3] = [math.radians(z) for z in synth[i][:3]]
    synth[i][2] = math.sin(synth[i][2])

    # take the difference element by element
    difference.append(map(operator.sub, noisy[i],synth[i]))

    # calculate D-value and append to difference list
    difference[i].append(D_calc(synth[i][2],synth[i][4],difference
        [i]))

    #convert rads back to degrees for output
    noisy[i][:3]     = [math.degrees(z) for z in noisy[i][:3]]
    synth[i][:3]     = [math.degrees(z) for z in synth[i][:3]]
    difference[i][:3] = [math.degrees(z) for z in difference[i
        ][:3]]

    # write clean orbital elements,\n,noisy orbital elements,\n,
        differences plus D-value
    # for each asteroid in family
    synthString = ' '.join('{:16.7f}'.format(y) for y in synth[i])
```

```
            noisyString = ' '.join('{:16.7f}'.format(y) for y in noisy[i])
            diff_String = ' '.join('{:16.7f}'.format(y) for y in
                difference[i])
            dval.write('\n'.join((synthString,noisyString,diff_String))+'\
                n')
            print 'finished D calculations for ',i,'th',familyString+'_'+
                noise_list[3]+'_'+cadence_list[4],'\n\n'


def selectNEOs(familyString,no_of_objsInt):
"""Scrapes AstOrb flatfile for objects matching NEO critera
then creates a synthetic NEO by random selection of orbital parameters
    """
    # read in astorb as a list
    asteroid_list = []
    with open('../astorb.dat', 'r') as astorb: #LWasserman AstOrb
        flatfile
        #ast is a list in which each element is an asteroid and its
            orbelts
        ast = astorb.read().split('\n')
        ast.remove("") #remove final carriage return

    # evaluate asteroids from astorb and write those meeting NEO criteria
         to file
    with open('NEOs_from_astorb_ENDICOTT.txt','w') as NEW_neos_Endicott:
        for elt in ast:
            (a,e) = (float(elt[168:181]), float(elt[158:169])) #read
                semimajor axis and eccentricity from astorb list
            if perihelion(a,e) < 1.3: # calculate perihelion and add to
                NEO list if appropriate
                NEW_neos_Endicott.write(elt+'\n') #write elements to
                    NEW_neos_Endicott file

    # result is an NEO subset of astorb.
    # build a list of NEO orbital elements, then select randomly to
        create synthetic NEOs
    new_elements = [] #order matches astorb organization. [meananomaly,
        periarg, node, inc, e, a]
    for k, elt in enumerate(ast): #check perihelion criteria for NEOs, if
         match, add to 2D element list
        peri = perihelion(float(elt[168:181]), float(elt[157:167]))
        if peri < 1.3:
            try:
```

```
                elmnt_list=[float(elt[115:126]), float(elt[126:137]),
                    float(elt[137:148]), float(elt[148:158]), float(elt
                    [158:169]), float(elt[168:181])]
                new_elements.append(elmnt_list)
            except ValueError:
                print("Error on line",k)

    # convert new_element from list of lists to array of arrays to
        facilitate column-by-column random choice
    new_elements = np.array(new_elements)


    # formatting of dummy text here is for ef8 fortran compatibility,
        white space is critical, do not edit.
    dummy_text_like_astorb ='''
 0.000000000000000E+00 0.000000000000000E+00 0.000000000000000E+00
     0.000000000000000E+00
2418230.1 7.8524222429796860E+01 6.8078911704746740E+01
    8.1927089436878150E+01 1.0616577185781130E+01 7.7485210393052350E-02
    2.7664836338324990E+00
1802 511 2015 321 212.856 6387 104 8 0.53 2.30 2.29 0    0    0    0    0
      0    0    0    0    0    0    0 O332812
 7.11 1.5E-02 7.7E-05 20150507 2.1E-02 2015 729 2.5E-02 2018 2 8 0.00 2.5
     E-02 2018 2 8 2.3324E-12 7.7492E-08 4.9412E-08 5.2904E-08 5.2488E-08
  1.3520457003016260E-05 1.6988372522534680E-05 1.0477848839621680E-05
      1.8560702102107700E-06 1.6532560884719450E-08 1.1506693831363680E
      -09
 -7.7341041844849160E-01 3.8722095002188720E-03 -4.9748503444959080E-03
     -1.5390213087961620E-01 2.1599958180279470E-01 -6.0806886110876270E
     -01 1.1161225808647010E-01 1.3298922767311350E-01
  7.2947058206620400E-04 -1.7514789548836640E-01 1.8318539190005470E-04
      -2.0163675921880390E-03 -4.3927688113500150E-03 1.6374616091789350E
      -03 -1.0283396668216560E-02
0.01 DE430  1   2   3   4  10  15  31  52 511 704 4.74E-10 1.02E-10 1.35E
    -11 1.31E-10 5.29E-11 1.31E-11 1.10E-11 1.67E-11 1.48E-11 1.82E-11\n
    '''

    # build a list of size N of six orbetls for each fam, write file
        using
    N = no_of_objsInt
    H = '0'    #assumed value for reduced magnitude H
    G = '0.15' #assumed value for slope parameter G
    # for family in ('atira','aten','apollo','amor','neo'):
```

```
    counter = 0
    fam_list = []
    while counter < N:
        synth_obj = generate_synth_obj(new_elements)
        if familyString == family_check(synth_obj[-1],synth_obj[-2],
            familyString): #send a,e from each synth obj, and current
            famiy
            fam_list.append(synth_obj)
            counter += 1
        elif familyString == 'neo' and perihelion(synth_obj[-1],synth_obj
            [-2]) < 1.3:
            fam_list.append(synth_obj)
            counter +=1
        else: pass

    # write astorb-like file population
    with open('synth_'+familyString+'_astorb.dat','w') as g:
        for i,asteroid in enumerate(fam_list): # enumerates from 0, so
            adding 'i+=1' for astorb numbering
            for j,elmnt in enumerate(asteroid):
                asteroid[j] = format_e(elmnt)    # format floats to string
                    for astorb-like
            # for each synthetic object, write to file:
            # dummy number, dummy name
            # synth elmnts as str
            # dummy text block for ef8 compatibility
            g.write(str(i+1).rjust(6)+' '+familyString.ljust(6)+''.ljust
                (13)+H.ljust(6)+G.ljust(5))
            g.write(' '.join(elmnt for elmnt in asteroid))
            g.write(' 2015 1 1.0'.ljust(11)+'2015 413 T. Riker'.ljust(24)+
                dummy_text_like_astorb)

def generate_eph(familyString,ephemerislist):
    familyfile = 'synth_'+familyString+'_astorb.dat'
    # create input file for ef8 ephemeris calculator.
    with open('input', 'w') as test_input:
    test_input.write('\n'.join(['o','s','n','n','r','n',familyfile
        ,'1000','0','',ephemerislist[0],ephemerislist[1],ephemerislist
        [2],'']))
    # run ephemeris calculator for the selected family. (obligatory
        relative path warning)
    # output file is ef.out
    subprocess.call('../../../../tibro/data/bin/ef8 < input',shell=True)
```

```
def count_synthetic_objects(famiy_string):
"""Count total number of synthetic asteroids in file"
    familyfile = 'synth_'+family_string+'_astorb.dat'
    # astorb type files have 9 lines per object, do some math to get nb
        of asteroids in file
    with open(familyfile,'r') as fam_file:
        nb_ast = sum(1 for _ in fam_file)/9
    return nb_ast


def noisy_eph(noise_list, family):
"""Test line in ef8 for ephemeris then add noise"""
    with open('ef8.out','r') as ef8, open('noisy_'+family+'_'+noise_list
        [3]+'_ef8.out','w') as noisyef8:
        for line in ef8:
            test_for_ephemeris = is_number(line.split(' ')[0]) and
                is_number(line.split(' ')[1])
            if not test_for_ephemeris:
                continue
            else:
                # if line is an ephemeris, pass to noise loop and write
                    noisy version to file
                noisyef8.write(ENDICOTT_noise_loop(line,noise_list[0],
                    noise_list[1],noise_list[2]))


def asteroid_files(j,astorb_list, noise_list, cadence_list, eph_per_ast,
    family):
"""Prepare files to feed to minidiff.
asteroid.neworb contains the original orbital elements,
minidiff uses this as an initial guess for the orbital fit
asteroid.ted contains the noisy ephemeris, based on observational
    cadence"""
    with open('asteroid.neworb','w') as ast_neworb, open('synth_'+family
        +'_'+noise_list[3]+'_'+cadence_list[4]+'_orbit.dat','a') as
        synthorbit, open('asteroid.ted','w') as ast_ted:
        synthorbit.write(astorb_list[9*j][0]+'\n')
        ast_neworb.write('\n'.join(elmt_line for x in astorb_list[9*j
            :(9*(j+1))] for elmt_line in x))
        with open('noisy_'+family+'_'+noise_list[3]+'_ef8.out','r') as
            noisyef8:
```

```
            for i,line in enumerate(noisyef8):
                for nights in xrange(0,cadence_list[2]):
                    date = nights*cadence_list[3]*24
                    for observations in xrange(0,cadence_list[0]):
                        if i == j*eph_per_ast+observations*cadence_list[1]+
                            date:
                            filler = '001000N 500'       # minidiff vals
                                for Mag(blank),name,500(geocentric code)
                            obs = line.strip().split()
                            # convert hour/minute to fraction of days, to
                                get .DDD in output format as DD.DDDD
                            day_frac = str(round(((float(obs[4])+float(obs
                                [5])/60.)/24.),5)).lstrip('0')
                            ast_ted.write('{:4s} {:02d} {:02d}{:6s} {:2s}
                                {:2s} {:3s} {:3s} {:s} {:13s}'.format(obs
                                [1],int(obs[2]),int(obs[3]),day_frac,obs[7],
                                obs[8],obs[9][:-1],obs[10],obs[11],obs
                                [12][:-1])+filler+'\n')
                        else: pass
    # minidiff output is neworb.dat containing orbital elements of
        calculated orbital fit. append to output file
    def orbit_files(family_string,noise_list,cadence_list):
        with open('neworb.dat','r') as neworb, open('neworb_fail.dat','r') as
             newfail, open('noisy_'+family_string+'_'+noise_list[3]+'_'+
            cadence_list[4]+'_orbit.dat','a') as noisyorbit, open('noisy_'+
            family_string+'_'+noise_list[3]+'_'+cadence_list[4]+'_orbit_fail.
            dat','a') as noisyfail:
            noisyorbit.write(neworb.read().split('\n')[0]+'\n')
            noisyfail.write(newfail.read())
```

```
"""plot_3D_cadence.py
Script to scrape *.end files and plot 3D data

Notes:
    - This script requires python 2.7
    - pip dependencies can be found in requirements.txt
    - Last modified July 10, 2017
"""

# import statements
import glob
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# function definitions


def get_x_y_z_fromfile(someFile,someElementIndex):
"""Grab the [cadence, element, dval] from the given file cadence
comes from filename string element chosen by user from list,
index value provided by user, dval is the
last element of every third line in the file""
    x,y,z = [],[],[]
    months = int(someFile.strip().split('_')[2].replace('d',''))/30.
    with open(someFile) as f:
        next(f) # skip header row
        for i, line in enumerate(f):
            if not (i+1)%3:
                z.append(float(line.strip().split()[5]))
                x.append(months)
            if i%3 == 1:
                y.append(float(line.strip().split()[someElementIndex]))
            list_of_points = zip(x,y,z)
    return list_of_points


def cut_data(someList,someAxesLimits):
"""checks if cadence and element are in provided range,
returns x,y,z data point as tuple"""
    for i,cad in enumerate(someList[0]):
        if cad in someAxesLimits[0:1] and someList[1][i] in
            someAxesLimits[2:3]:
```

```
            return someList[i][0],someList[i][1],someList[i][2]

# orbital element dictionary
# used for plot titles and user input
element_dict = {0: 'Argument of Perihelion (deg) ', 1:'Longitude of
    Ascending Node (deg) ', 2: 'Inclination (rad) ', 3:'Eccentrictiy ',
    4:'Semimajor Axis (AU) '}

# abbrv orbelt dictionary
# used for filename
element_abbr_dict = {0: 'periarg', 1:'ascnode', 2: 'inc', 3:'ecc', 4:'SA
    '}

# scrape current directory for all .end files
# ask user to choose asteroid family from those available
list_of_filenames = glob.glob('*.end')
family = set(x.split('_')[0] for x in list_of_filenames)

# User input section
print 'Available families: ',family
family = raw_input('Please enter family to plot: ')

list_of_filenames = glob.glob(family+'*.end')
noise = set(x.split('_')[1] for x in list_of_filenames)
print 'Please enter the noise level you would like to plot from this
    list',set(noise)
noise = raw_input()

print 'Please enter the numeral corresponding to the orbital element to
    plot',element_dict
element = int(raw_input())

list_of_filenames = sorted(glob.glob(family+'_'+noise+'*.end'))

dataList = []
for g in list_of_filenames:
    dataList.append(get_x_y_z_fromfile(g,element))

points = sum(dataList,[])
print 'Cadence range is ',min(zip(*points)[0]),max(zip(*points)[0])
print 'Element range is ',min(zip(*points)[1]),max(zip(*points)[1])
print 'D-value range is ',min(zip(*points)[2]),max(zip(*points)[2])
```

```
print 'Choose cadence axis limit (default ',max(zip(*points)[0]),')'
# User can specify an axis upper-limit for cadence values
reduce = float(raw_input() or max(zip(*points)[0]))
count = sum(1 for x in points if x[0] < reduce)

colors = ['r','b','g','c','m','y',]

fig  = plt.figure()
plt.gca().set_color_cycle([i for i in colors])
ax = fig.add_subplot(1,1,1, projection = '3d')
ax.scatter(*zip(*points[:count]))
ax.set_xlabel('Cadence (Months)')
ax.set_ylabel(element_dict[element])
ax.set_zlabel('D-value')
ax.locator_params(axis='x', nticks=3)
ax.locator_params(axis='y', nticks=4)
ax.locator_params(axis='z', nticks=3)
ax.set_title('D-value vs '+element_dict[element]+'& Cadences \nat Fixed
    Noise Level: '+noise.replace('p','.'))
fig.savefig(family+'_'+noise+'_'+element_abbr_dict[element]+'_3D_single.
    png', dpi=fig.dpi)
plt.show()
```

```
"""Plot_fracdiff.py
Script to read in *.end files and plot
dval versus delta-orbelt

Notes:
    - This script requires python 2.7
    - pip dependencies can be found in requirements.txt
    - Last modified July 10, 2017
"""

# import statments
import glob
import numpy as np
import matplotlib.pyplot as plt
import pylab
import math

# function definitions

def read_results_file(someFile, someElementIndex, someList):
"""Read in results file, create list of selected delta-orbelt"""
    with open(someFile) as f:
        next(f)
        for i,line in enumerate(f):
            if not (i+1)%3:
                someList.append(abs(float(line.strip().split()[
                    someElementIndex])))


def read_results_file_D(someFile,someList):
"""Read in results file, create list of D-values"""
    with open(someFile) as f:
        next(f)
        for i,line in enumerate(f):
            if not (i+1)%3:
                someList.append(abs(float(line.strip().split()[5])))


def plotting_function(someString):
"""Plots user-specified subgroup data"""
    element_dict = {0: 'Argument of Perihelion (deg)', 1:'Longitude of
        Ascending Node (deg)', 2: 'Inclination (rad)', 3:'Eccentricity',
        4:'Semimajor Axis (AU)'}
```

```
# declare list to hold x,y variables
dvals ,periarg ,ascnode ,inc ,ecc ,sa ,fitperiargX ,fitperiargY ,
    fitascnodeX ,fitascnodeY ,fitincX, fitincY, fiteccX, fiteccY,
    fitsaX, fitsaY,coeffperiarg,coeffascnode, coeffinc, coeffecc,
    coeffsa =
    [],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]
Rperiarg,Rascnode,Rinc,Recc,Rsa = 1000,1000,1000,1000,1000

orbeltList = [periarg,ascnode,inc,ecc,sa]
fitList = [[fitperiargX,fitperiargY,Rperiarg,coeffperiarg],[
    fitascnodeX,fitascnodeY,Rascnode,coeffascnode],[fitincX,fitincY,
    Rinc,coeffinc],[fiteccX,fiteccY,Recc,coeffecc],[fitsaX,fitsaY,Rsa
    ,coeffsa]]


# build list of all .end files
results = glob.glob(someString+'*.end')
print results
# build lists of x,y data
for x in results:
    read_results_file_D(x,dvals)
    for i,elt in enumerate(orbeltList):
        read_results_file(x,i,elt)
for i,x in enumerate(orbeltList): print len(x),': [',min(x),',',max(x
    ),']', element_dict[i]

fig = plt.figure(figsize=(14, 6))

for i,orbelts in enumerate(orbeltList[3:5]):
    eq = fitList[i][3]
    axs = fig.add_subplot(1,3,2-i) #second value is no. of subplots
        created
    plt.locator_params(nbins=5,axis = 'x')
    plt.locator_params(nbins=10,axis = 'y')
    axs.plot(fitList[orbeltList.index(orbelts)][0],fitList[orbeltList
        .index(orbelts)][1],alpha = 0.5, label = eq)
    axs.scatter(orbelts,dvals,marker = 'o',color = 'g',alpha = 0.7)
    axs.set_xlabel('$\Delta$'+element_dict[orbeltList.index(orbelts)
        ])
    if i == 1:
        axs.set_ylabel('D-value')
    else:
```

```
            axs.set_ylabel('')
        for item in (axs.get_xticklabels() + axs.get_yticklabels()):
            item.set_fontsize(10)
        #axs.set_title(element_dict[i])
    fig.suptitle(someString+' $\Delta$ Orbital Parameter vs D-value')
    fig.set_tight_layout(True)
    plt.savefig(someString+'_fracdiff1.png',dpi=fig.dpi)
    plt.show()



if __name__ == '__main__':
    familyString = raw_input('Please enter family name: ').lower()
    plotting_function(familyString)
```

REFERENCE LIST

[Ari] University of Arizona. "Catalina Sky Survey." `https://catalina.lpl.arizona.edu/about/facilities`.

[Elv14] Martin Elvis. "How many ore-bearing asteroids?." *Planetary and Space Science*, **91**(1):20 – 26, 2014.

[ENS11] V. Emelyanenko, S. Naroenkov, and B. Shustov. "Distribution of the near-earth objects." *Solar System Research*, **45**(6):498 – 503, 2011.

[ES] European Space Agency (ESA). "Gaia Space Observatory." `http://sci.esa.int/gaia/`.

[GNG12] Sarah Greenstreet, Henry Ngo, and Brett Gladman. "The orbital distribution of Near-Earth Objects inside Earths orbit." *Icarus*, **217**(1):355 – 366, 2012.

[Ia11] Gautier IV and Micheli and. "PRELIMINARY RESULTS FROM NEOWISE: AN ENHANCEMENT TO THE WIDE-FIELD INFRARED SURVEY EXPLORER FOR SOLAR SYSTEM SCIENCE." *The Astrophysical Journal*, **731**(1):53, 2011.

[Jet] NASA Jet Propulsion Laboratory. "Discovery Statistics." `https://cneos.jpl.nasa.gov/stats/totals.html`. Accessed: 2017-7-31.

[JM14] Youngmin JeongAhn and Renu Malhotra. "On the non-uniform distribution of the angular elements of near-Earth objects." *Icarus*, **229**(1):236 – 246, 2014.

[Nes06] David Nesvorny. "New Candidates for Recent Asteroid Breakups." *The Astronomical Journal*, **132**(5):1950 – 1958, 2006.

[Obs] Lowell Observatory. "Lowell Observatory." `https://lowell.edu`.

[PHW00] Petr Pravec, Carl Hergenrother, Rob Whiteley, Lenka Sarounova, Peter Kusnirak, and Marek Wolf. "Fast Rotating Asteroids 1999 TY2, 1999 SF10, and 1998 WB2." *Icarus*, **147**(2):477 – 486, 2000.

[Tan98] G. Tancredi. "Chaotic dynamics of planetencountering bodies." *Celestial Mechanics and Dynamical Astronomy*, **70**(3):181 – 200, 1998.