ORBITAL DEBRIS CLOUD EVOLUTION: AN ANALYSIS OF FRAGMENTATION EVENTS IN LOW EARTH ORBIT

by

Reece Humphreys

A Thesis Submitted to the Faculty of

The Wilkes Honors College

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Biological and Physical Sciences

with a Concentration in Physics

Wilkes Honors College of

Florida Atlantic University

Jupiter, Florida

May 2021

ORBITAL DEBRIS CLOUD EVOLUTION: AN ANALYSIS OF FRAGMENTATION EVENTS IN LOW EARTH ORBIT

by

Reece Humphreys

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Yaouen Fily, and has been approved by the members of their supervisory committee. It was submitted to the faculty of the Harriet L. Wilkes Honors College and was accepted in partial fulfillment of the requirements for the degree of Bachelor of Science in Biological and Physical Sciences.

SUPERVISORY COMMITTEE:

Dr. Yaouen Fily

Dr. Shane Welker

Dean Justin Perry, Harriet L. Wilkes Honors College

Date

Acknowledgments

This thesis is dedicated to Dr. Yaouen Fily. He has provided instrumental support throughout this thesis and is the person who inspired me to concentrate in physics. The intuitions and lessons learned throughout the courses I took with him are at the essence of this thesis. Additionally, I would like to give a special thanks to Dr. Timothy Steigenga, who provided me countless opportunities and showed me there is always a path forward.

Abstract

Author:	Reece Humphreys
Title:	Orbital Debris Cloud Evolution: An analysis of fragmentation events in low Earth orbit
Institution:	Harriet L. Wilkes Honors College, Florida Atlantic University
Thesis Advisor:	Dr. Yaouen Fily
Degree:	Bachelor of Science in Biological and Physical Sciences
Concentration:	Physics
Year:	2021

The amount of orbital debris has rapidly grown due to humanity's desire to work in, explore, and utilize space. However, compared to most types of pollution that people experience daily, it is unlikely that the average person will ever encounter space pollution. Yet, it poses a significant threat to many technologies we depend on daily such as GPS and weather prediction satellites. Orbital debris are the remnants of orbital collisions, weapons tests, decommissioned satellites, and spent rocket stages that are passing over our heads faster than bullets and containing similar energy to hand grenades. This paper explores the existing models of orbital debris generation, how clouds of debris evolve with respect to time, and their ramifications.

Contents

List of Figures

List of Tables

List of Symbols

A Cross-sectional area [m^2].

H Scale height for exponential atmospheric model [km].

I_k Modified Bessel function of the first kind and order k.

J_2 Second zonal harmonic coefficient for the Earth.

L_c Fragment characteristic length [m].

M_e Reference mass for collisions [kg].

M_p Projectile mass [kg].

M_t Target mass [kg].

M Mass [kg].

Δv Relative velocity [km/s].

Ω Longitude of the ascending node [rad or deg].

Φ Flux of fragments.

$\mathcal{N}$ Normal distribution.

μ_E Earth's planetary gravitational constant [km^3/s^2].

μ Mean value.

ν True anomaly [rad or deg].

ω Argument of the periapsis [rad or deg].

ρ Atmosphere density [kg/m^3].

a Semi-major axis [km].

e Eccentricity.

h Altitude [km].

i Inclination [rad or deg].

v_c Relative impact velocity [km/s].

1 Introduction

1.1 Motivation

Due to the accelerating launch cadence in the space sector, increased accessibility and resources for small teams to create cube satellites, and satellite mega constellation constructions underway, researchers have become increasingly concerned about the implications of potential orbital collisions. These worries have been compounded by the actions taken by foreign nations with regards to anti-satellite weapons which create substantial amounts of debris. In one such instance, a 2007 Chinese anti-satellite missile test was universally condemned and received statements from government officials such as the U.K. prime minister whose spokesperson stated, "We are concerned about the impact of debris in space and we expressed that concern." These fragmentation events can be difficult to track due to the small size of some of the debris fragments that are generated, yet they can pose a great hazard to other satellites and crewed operations being conducted in space. Tens of millions of pieces of orbital debris currently exist within Low Earth Orbit (LEO) with an average size smaller than 1cm. While minuscule in size, these pieces of debris have an average impact velocity of 10 km/s which generates similar energy to that of an exploding hand grenade. It is, therefore, paramount to study the phenomena that arise within orbital debris clouds to gather methods for mitigating debris cloud formations. Without such a study, the future commercialization of space, the potential for humanity to become a multi-planetary species, and the benefits that the advanced satellites provide researchers will remain in jeopardy.

1.2 Methodology

The first component in modeling orbital debris is to implement a breakup model which makes predictions about the outcome of an orbital collision or explosion. Breakup models use experimental data to create statistical models that predict the size, mass, speed, and number of debris pieces generated in a fragmentation event. Once these characteristics are obtained, equations of motion can be implemented that account for the significant forces, such as drag, acting on each debris fragment to model how the debris position and velocity will evolve over time.

This paper implements the NASA Standard Satellite Breakup Model to simulate the orbital collisions and gather information regarding the behavior of orbital debris [10]. This was accomplished by utilizing the model to create an implementation in Python, called Orbital Debris Analysis with Python (ODAP), which has been made open source on GitHub for others wishing to build on the foundations of this research.

Following the implementation of the breakup model, the implementation of dominant orbital perturbations is given. These are forces that act on debris to change their orbits over time and include effects such as atmospheric drag and gravitational perturbations. The optimal way to represent these effects is through changes in orbital parameters, which is an alternative to expressing the current state of an object with Euclidean coordinates (x, y, z, v_x, v_y, v_z). The benefits of using orbital parameters is explored more in-depth in Chapter 3.

Once the breakup model is established, longer-term debris cloud evolution is outlined. The evolution falls into multiple distinctive phases, which are characterized by the most dominant forces. For example, the initial

phase is called the ring formation phase, and the only force that needs to be considered is gravity. However, in the band formation phase, drag needs to be considered.

Finally, an analysis showing the applications of ODAP is provided to demonstrate how we can gain insights into phenomenon such as the time for the debris to deorbit and how the spread of the debris occurs over time. This analysis juxtaposes two different satellites experiencing the same type of non-catastrophic collision to illustrate how the initial orbits of the satellites play a significant factor in the extent to which the debris will remain a hazard in orbit.

1.3 Roadmap

This current chapter serves as an introduction to orbital debris modeling. Additionally, it motivates why orbital debris is an important area of research. As such, it is a non-technical introduction to the topic that is recommended for anyone not familiar with orbital mechanics and the problem of orbital debris.

Chapter 2 introduces the NASA breakup model, which is the primary method for modeling how an explosion or collision produces debris fragments. It begins by detailing the history of the breakup model and how it functions at a high level. Following this is an in-depth explanation of how the breakup model works using statistical distributions. Finally, it provides figures that illustrate how the distributions are used in practice.

Chapter 3 explains two different methods that can be used to describe the motion of objects in orbit, Keplerian and orbital elements. It is beneficial to utilize both parameterizations as they each provide different benefits.

Orbital elements utilize the position and velocity to specify an object's motion. As such, they are the most intuitive to understand and provide a direct method for creating visualizations. On the other hand, Keplerian elements are a more abstract representation that allows for more efficient computations as they have Kepler's laws of orbital motion built into them. The chapter includes an introduction to both of these methods and definitions and explanations for readers who may not be familiar with either representation.

Chapter 4 introduces the different phases that the debris will go through as time progresses. As such, this chapter delves into the technical details and assumptions used for each phase. Additionally, it explores the methods required to perform the efficient computation of each phase. Nonetheless, the description of each phase includes a non-technical overview of the forces at play.

Chapter 5 illustrates how the breakup model and debris cloud evolution simulations can be used to analyze the characteristics of orbital debris. It performs an analysis on the cloud formation of two different types of satellites and how their initial orbits influence the debris decay time and spread of the debris over time.

2 Modeling Satellite Breakups

2.1 The NASA Standard Satellite Breakup Model

A satellite is any artificial body placed in orbit around the earth or moon or another celestial body. The definition of the term is intentionally general and, as such, can be used to reference spacecraft, remnants of rockets, and communication devices in orbit. A satellite breakup model is a mathematical model used to describe the outcome of a satellite breakup due to an explosion or collision [15]. A satellite breakup model should describe the size, area-to-mass (AM) ratio, and the ejection velocity of each fragment produced in the satellite breakup [10]. The NASA Standard Satellite Breakup Model is an industry-standard breakup model developed by NASA and is used by most major space agencies such as the European Space Agency (ESA) and the Japanese Aerospace Exploration Agency (JAXA). The breakup model's implementation was provided by Johnson et al. in 2001 [10] and was later clarified by Kristo in 2011 [12].

The NASA standard breakup model uses experimental observations performed both on Earth and in orbit to characterize the breakup using statistical distributions. The choice to use statistical distributions is a result of the stochastic nature of the breakup event, meaning it would be impossible to reproduce the same circumstances each time. For example, explosions are stochastic due to the complex chemical reactions that lead to the explosion. By using a statistical distribution and sampling from it we can more accurately represent the fragments that would be generated during a collision or explosion. In this paper, I will be focusing on how collision fragments are generated. The explosion case is handled in a similar way

albeit using slightly different parameter values.

2.2 Implementing the NASA Breakup Model

2.2.1 Characteristic Length and Number of fragments

To account for the different characteristics of each fragment of debris, the statistical distributions must be expressed as a function of some independent variable [10]. In the latest version of the NASA breakup model, this variable is called the characteristic length, denoted L_c [10]. By defining the distributions using characteristic length we ensure that the mass, area, and velocity of each fragment are not constant for all debris with the same characteristic length. This in turn guarantees that the stochastic nature of the breakups is reflected in our models. It should be noted that in prior versions of the NASA breakup model, the mass of each piece of debris was used as the independent variable [12]. However, characteristic length was found to be more directly linked to both in-orbit and terrestrial breakup data [10].

The implementation of the characteristic length distribution can be cumbersome to follow. As such, a flow chart illustrating the steps of the algorithm is provided below. Each of the steps listed in the flow chart will be explored in further detail in the rest of this subsection.

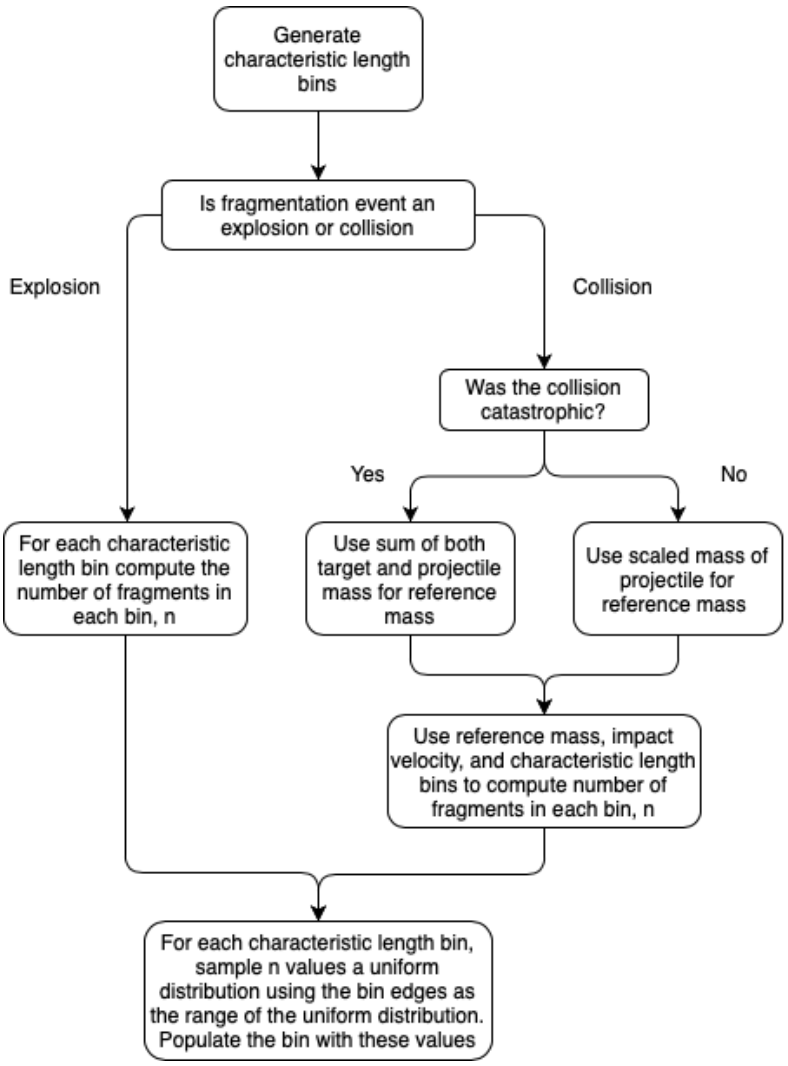


Figure 2.1: The algorithm used for producing the characteristic length distribution used in the NASA breakup model.

The creation of the characteristic lengths was not given in the original specification of the NASA breakup model by Johnson [10] but was included in the corrections by Krisko in 2001. Krisko specifies that the NASA breakup model deposits fragments of L_c from 1mm to over 1m in bins and that the number of fragments in each bin is determined by a power law that will be discussed later. However, Letizia's implementation for CiELO modified this to first create 100 bins that are equally spaced

on a logarithmic scale between 1mm and 10 cm. In this paper, we will be following Letizia's methodology for fragments smaller than 10 cm as it has the most up-to-date information about the NASA breakup model. The pieces of debris that are larger than 10cm will be handled separately using Krisko's methodology to ensure that the breakup model conserves mass, i.e., that the masses of the debris add up to the mass of the object(s) they originated from. Figure 2.2 below illustrates the process of using bins to determine the number of fragments of each size.

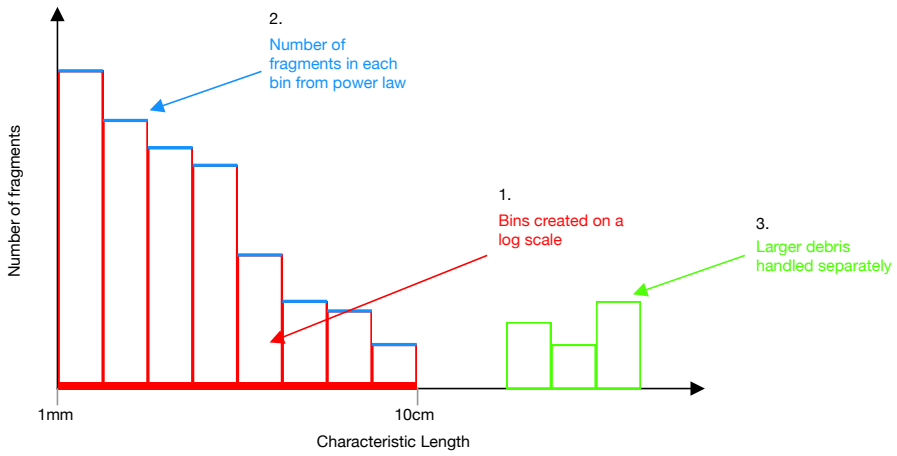


Figure 2.2: A sketch illustrating the process for determining the number of fragments in each characteristic length bin.

To determine the number of fragments in each characteristic length bin, we acknowledge that collisions and explosions will produce different types of fragments. Explosions will produce larger debris fragments with smaller velocities while collisions tend to generate a large number of small fragments with high velocities [6]. As such, the number of fragments in each bin will be determined by different power laws based on the type of breakup event.

The number of explosive fragments of size L_c is governed by the following

equation

$$N(L_c) = 6SL_c^{-1.6} \tag{1}$$

With S = 1, the relationship has been observed to be valid for rocket upper stages with masses in the range of 600-1000 kg [10]. For explosions, due to other malfunctions such as battery explosions and anti-satellite tests, a value of S between 0.1 and 1 was found to fit the experimental data better [12].

In the case of a collision, a distinction must be made between catastrophic or non catastrophic. A collision is categorized as catastrophic if it causes the complete fragmentation of both the impactor and the target [13]. This occurs when the energy per target mass exceeds $40Jg^{-1}$ [12]. The number of produced fragments for a collision is governed by

$$N(L_c[m]) = 0.1(M_e)^{0.75}L_c^{-1.71}, \tag{2}$$

where M_e is defined as follows:

$$M_e[kg] = \begin{cases} M_t[kg] + M_p[kg] & \text{Catastrophic collision} \\ M_p[kg] * (v_c[km/s]/1[km/s])^2 & \text{Non Catastrophic collision.} \end{cases} \tag{3}$$

In equation (3), M_t is the target mass, M_p is the projectile mass, and V_c is the relative impact velocity between the target and the projectile.

Figure 2.3 shows sample results from ODAP to illustrate the characteristic length distribution described above. Additionally, the relevant ODAP function can be found in the appendix section C.1.

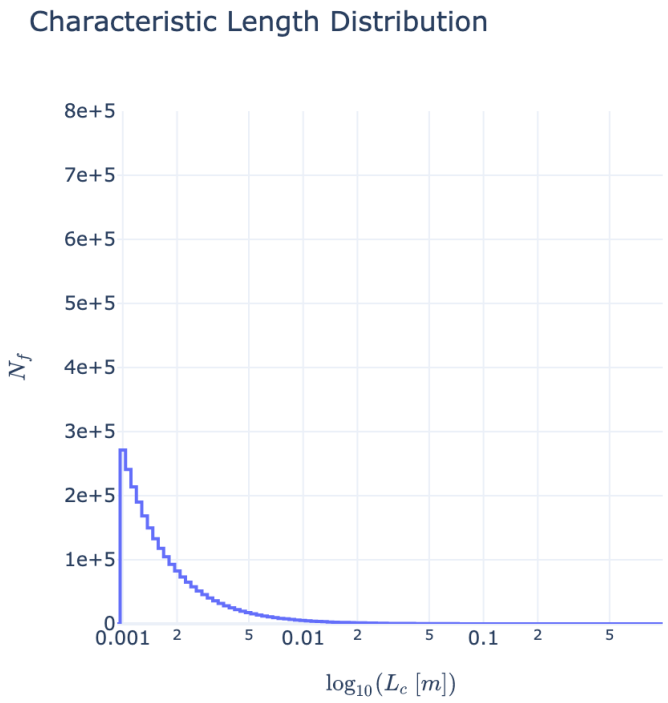


Figure 2.3: The characteristic length distribution produced by ODAP of a catastrophic collision involving a rocket body with a mass of 1000kg, a projectile of mass 10kg, and an impact velocity of 10 km/s.

2.2.2 Area to Mass Distribution

The area-to-mass ratio, A/M, for fragments is a distribution that was based on analysis of thousands of fragmentation debris and provides us with a method to determine the mass of each fragment of debris. The discrete distributions were found by using a χ^2 fit to orbital decay characteristics for 1,780 upper stage explosion fragments, and similar data was developed for spacecraft fragments [10]. Each type of debris producer —RB, SC, and SAT— will produce different size debris. As such, the distribution that determines the area to mass ratio has three variants. All three are based on a normal distribution but use different expressions for determining the mean and standard deviation of the distribution. For simplicity, the rest of this section will only provide the details of the SC distribution. However, the implementation of the other two categories is included in ODAP.

For small objects with $L_c < 8$cm, SAT, the A/M distribution is expressed as

$$D_{A/M}(\lambda_c, \chi) = \mathcal{N}(\mu_{A/M}(\lambda_c), \sigma_{A/M}(\lambda_c), \chi). \tag{4}$$

$D_{A/M}$ is the distribution function of χ as a function of λ_c, where

$$\lambda_c = \log_{10}(L_c), \tag{5}$$

$$\chi = \log_{10}(A/M) \tag{6}$$

$\mathcal{N}$ is the normal distribution function with mean $\mu_{A/M}$ and standard deviation $\sigma_{A/M}$, where

$$\mu_{A/M} = \begin{cases} -0.3, & \lambda_c \leq -1.75 \\ -0.3 - 1.4(\lambda_c + 1.75), & -1.75 < \lambda_c < -1.25 \\ -1.0, & \lambda_c \geq -1.25 \end{cases} \tag{7}$$

and (8)

$$\sigma_{A/M} = \begin{cases} 0.2, & \lambda_c \leq -3.5 \\ 0.2 + 0.1333(\lambda_c + 3.5) & \lambda_c > -3.5 \end{cases} \tag{9}$$

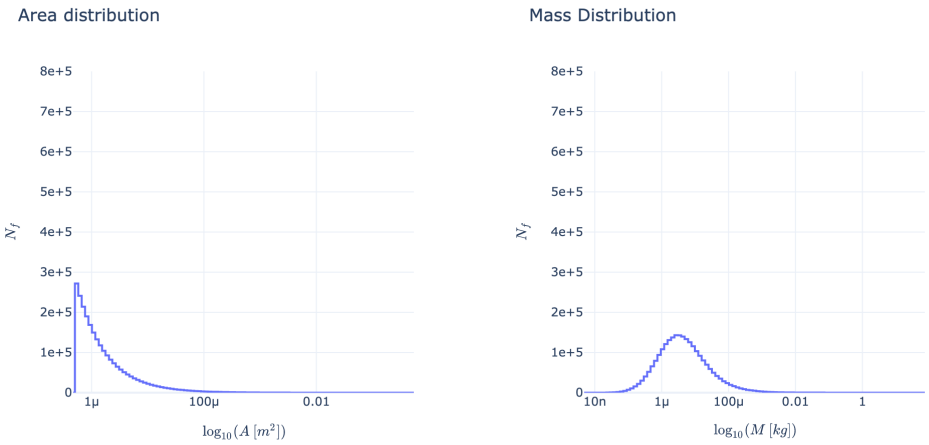
Every fragment of debris has a corresponding A/M distribution since both $\mu_{A/M}$ and $\sigma_{A/M}$ are functions of λ_c. To determine the corresponding A/M ratio for each debris, a random value is drawn from the distribution. This accounts for stochastic nature of breakup events mentioned previously.

The A/M ratio alone does not provide enough information to determine both the area and mass of a fragment. As such, the average cross-sectional

area, A, can also be obtained through a one-to-one correspondence with L_c using the following expression [10]:

$$A_x = \begin{cases} 0.540424 * L_c^2 & \text{where } L_c < 0.00167 \text{ m} \\ 0.556945 * L_c^{2.0047077} & \text{where } L_c \geq 0.00167 \text{ m} \end{cases} \tag{10}$$

Utilizing both the A/M ratio and the cross sectional area A, we can now obtain the mass M easily using $M = A_x/(A/M)$. Figure 2.4 shows a simulated A/M distribution and average cross sectional area. The code to reproduce this is found in the appendix Section C.1.



(a) (b)

Figure 2.4: The area distribution **(a)** and the mass distribution **(b)** produced by ODAP of a catastrophic collision involving a rocket body with a mass of 1000kg, a projectile of mass 10kg, and an impact velocity of 10 km/s.

2.2.3 Change in Velocity Distribution

The differential amount of velocity that each fragment will gain due to the breakup event is determined in a similar manner to the A/M ratio. The

notable differences are that the distribution is now a log-normal distribution and that an additional check is implemented to ensure that extremely high ejection velocities are not included in the case of collisions [13].

More explicitly, the velocity check is performed by sampling a value from the velocity distribution and checking if it is lower than $1.3v_c$, where v_c is the relative collision velocity. If the value fails the check then new values are drawn until the check is passed [13].

The change in velocity, Δv, is modeled by log-normal distribution that is function of the A/M ratio by

$$D_{\Delta v} = \mathcal{N}(\mu_v(\chi), \sigma_v(\chi), \xi) \tag{11}$$

where

$$\xi = \log_{10}(\Delta v), \tag{12}$$

$$\chi = \log_{10}(A/M), \tag{13}$$

$$\mu_v(\chi) = 0.2\chi + 1.85, \tag{14}$$

$$\sigma_v(\chi) = 0.4, \tag{15}$$

The differential velocity vector is constructed by sampling from a uniform distribution for a velocity unit vector and then scaling it with the values from the log-normal distribution.

It should be noted that details for efficiently handling the resampling of velocities are sparse. ODAP's implementation of the velocity distribution often takes tens of thousands of iterations to complete. One or two

fragments often have an area-to-mass ratio that causes the log-normal distribution to be highly unlikely to return a value that is lower than $1.3V_c$. To mitigate this issue, we limit the number of iterations to 100,000. If no suitable value is found before the maximum iteration limit is hit, a value of $1.3V_c$ is assigned to that piece of debris.

2.3 Validating the Implementation

Due to the niche nature of orbital debris analysis, it can be challenging to find any full implementation of the breakup model. Additionally, many of various space agencies around the world do not share the details of how they implemented the models. For this reason, the above implementations were largely modeled after the details given in the CiELO [13] which mimics the information provided in NASA literature.

To validate that ODAP's implementation of the breakup model was performed correctly, a comparison is made to existing data provided by various space agencies for a given scenario. The first scenario is an explosion event involving a rocket body that weighs 1000kg. The second scenario is a catastrophic collision event between a rocket body that weighs 1000kg and a 10kg projectile mass with an impact velocity of 10km/s. The data was provided in a presentation given by Rossi et al. [16].

As shown in the table 1, the data from the various space agencies has some significant differences, especially for the characteristic length in the $>$ 1mm range, despite implementing the same NASA breakup model. This is most likely a result of ambiguities in the original NASA breakup model specification literature, as well as differences in how various programming languages perform random sampling. The goal of ODAP is to be within

similar ranges as the rest of the data.

Model	Number of Fragments						
	Length				Mass	Area	Velocity
	>1mm	>1cm	>10cm	>1m	>1g	>1cmˆ2	>100msˆ-1
ASI	378,581	9,403	234	7	2,472	5,878	112,932
CNSA	37,865	960	32	9	254	-	11,380
DLR	1,217,054	11,724	230	0	25,844	31,124	31,124
ESA	324,886	8,159	206	6	2,093	5,024	98,717
NASA	434,928	10,731	248	8	2,525	6,416	132,032
ODAP	378,525	9,479	223	6	3613	5,850	120,481

Table 1: (NASA Breakup Model Implementation Comparison of results [16].

3 State Representation

Once the fragmentation event has been simulated, we must describe the motion of the fragments over time. To this end, we need a method of representing the fragments that will provide all of the information required for this description. An intuitive method of doing so is using **orbital elements**, which use the position and velocity in a specific coordinate system. More specifically it uses orbital vectors in an **Earth-Centered Inertial (ECI) Coordinate system**. This representation is illustrated in Figure 3.5. The orbital state vector representation is a natural result of Newton's laws of motion and the universal law of gravitation.

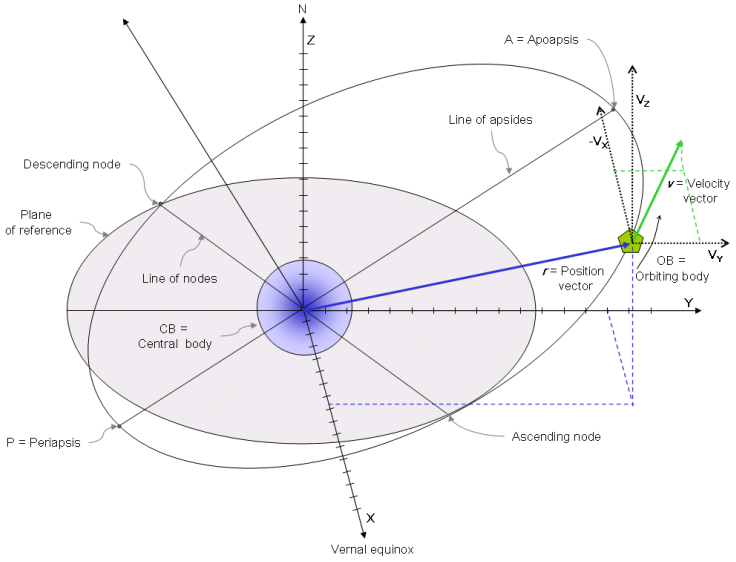


Figure 3.5: A visualization of the orbital state vector approach to representing satellites [4].

While this method of representing satellites is the most intuitive, it has inherent limitations due to it being a standard way of describing the motion of any object. As such, nothing about the way orbits behave is included in the representation. This leads to an alternate way of describing the motion

using **Keplerian elements**, which result from Kepler's laws of planetary motion. These consist of the eccentricity, e, semi-major axis, a, inclination, i, longitude of the ascending node, Ω, the argument of periapsis, ω, and the true anomaly, ν. Since this representation results from Kepler's laws, it has information about orbital mechanics built into it.

Both of these methods have advantages and disadvantages for modeling orbital debris, and as such, it is often convenient to switch back and forth between them. Keplerian elements are a more abstract representation of an object's motion but provide significant benefits for performing propagations forward in time. Orbital state vectors are often more suited to performing visualizations and give a more intuitive sense of each piece of debris's current state. Both of these representations are explored in this section, as well as their benefits and construction.

3.1 Orbital State vectors and their advantages

The orbital state vectors utilize Newton's laws of motion and Newton's universal law of gravitation to describe the motion of an object in space. Newton's universal law of gravitation states that

$$\vec{F}_{grav} = G\frac{m_1 m_2 \vec{r}}{|\ \vec{r}\ |^3}.$$

This is the dominant force that acts on orbital objects, neglecting drag. As such, application of Newton's second law yields

$$\vec{a_i} = G\sum_{i\neq j} m_j \frac{\vec{r_j} - \vec{r_i}}{|\ \vec{r_j} - \vec{r_i}\ |^3}. \tag{16}$$

Therefore, the change in an object's velocity can be expressed as the sum

of the contributions of gravitational attraction from all other surrounding bodies. Gravitational force depends on mass, meaning the gravitational attraction for a piece of debris to nearby debris is negligible compared to the gravitational attraction from a nearby planetary body. Thus, for computational efficiency, inter-particle interactions will be neglected.

The benefits of using orbital state vectors are that they provide a natural intuition for how the motion of a piece of debris will change over time and they provide the necessary information for 3D visualization. However, a large drawback of using state vectors is that all six degrees of freedom, (x, y, z, v_x, v_y, v_z), will be changing during each time step. As such, when simulating a large number of fragments orbiting a body, the memory usage will grow quite large. Additionally, the periodic nature of orbits is not captured with this parameterization. More precisely, knowing the state of a piece of debris in the far future will require propagating that debris from its initial conditions up to the desired time step. This is an undesirable consequence and one of the primary motivations for preferring the Keplerian element parameterization.

3.2 Keplerian elements and their advantages

When viewed from an inertial plane, two orbiting bodies trace distinctive trajectories, where each has a focus at the common center of mass. When switching to a non-inertial frame centered on one of the bodies, only the opposite body's trajectory is viewable. Keplerian elements are a parameterization that describes these non-inertial trajectories [2]. The reference body is called the primary body. In our case, this is the Earth, while the other body is called the secondary body. It should be noted that there is no preferred primary or secondary body.

The first step to reparameterize the motion of the orbital debris is to use the position vector, $\vec{r}$, and velocity vector, $\vec{v}$, to define the **specific angular momentum** vector as

$$\vec{h} = \vec{r} \times \vec{v}. \tag{17}$$

Using the Earth's equator as the **fundamental plane**, which is the plane that is used as a reference, we can begin constructing the parameters of the **orbital plane**—the plane created by tracing out the $\vec{r}$ vector, which contains both $\vec{r}$ and $\vec{v}$. The intersection of the orbital plane with the fundamental plane is called the **line of nodes**.

The **ascending node** is the spot where the orbiting body crosses the plane of reference/equatorial plane in a northerly direction. Similarly, the **descending node** is where it crosses the plane of reference in a southerly direction. The vector $\vec{n}$ points in the direction of the ascending node and is found by taking the cross product of the unit vector $\hat{k}$ with the angular

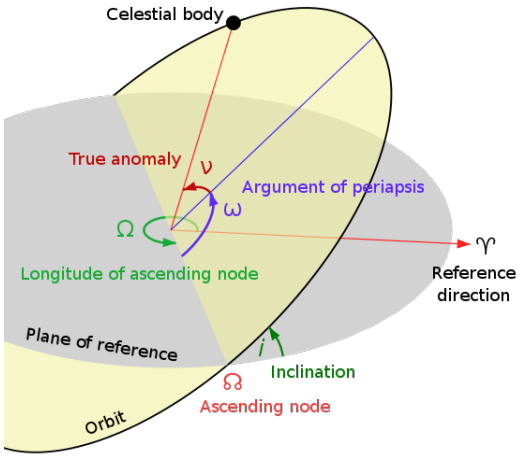


Figure 3.6: A diagram illustrating the Keplerian elements related to the orbital plane intersecting a reference plane [2]

.

momentum $\vec{h}$.

$$\vec{n} = \hat{k} \times \vec{h} \tag{18}$$

The **semi-major axis**, a, is one half of the major axis. The major axis is a line that goes through both foci of the ellipse and the center and ends at the widest point of the perimeter.

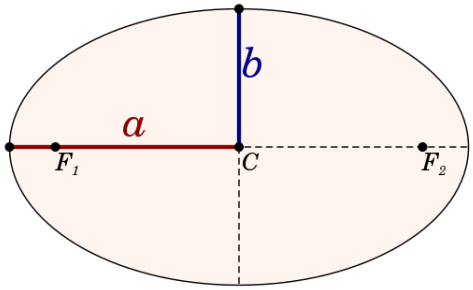


Figure 3.7: The semi-major axis, a, along with the semi-minor axis, b, and the foci of an ellipse, F_1 and F_2 [5].

Using the vis-viva equation, the semi-major axis, a, can be defined as

$$a = \frac{1}{\frac{2}{|\vec{r}|} - \frac{|\vec{v}|^2}{\mu}}, \tag{19}$$

where μ is the standard gravitational parameter of the primary body. For Earth, the value of μ is $3.986 \times 10^{14} \mathrm{m}^3/\mathrm{s}^2$.

The **orbital eccentricity**, $e = \| \vec{e} \|$, is a dimensionless parameter that indicates to what degree an orbit around another body deviates from a perfect circle. It has the value of 0 for a circular orbit, a value between 0 and 1 for elliptic orbits, 1 for parabolic escape orbits, and greater than 1 for hyperbolic orbits. For the purpose of orbital debris in LEO, the eccentricities will be in the elliptic orbit range.

The equation for eccentricity can be expressed from the the orbital state vectors as follows:

$$\vec{e} = \left(\frac{| \vec{v} |^2}{\mu} - \frac{1}{| \vec{r} |} \right) \vec{r} - \frac{(\vec{r} \cdot \vec{v})}{\mu} \vec{v} \tag{20}$$

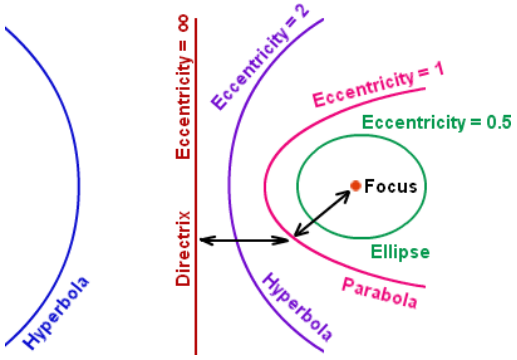


Figure 3.8: The relationship between the eccentricity and the resulting conic section [2].

The **inclination**, i, is the angle formed between the unit vector pointing in the $\hat{k}$ direction and can be calculated by

$$i = \arccos \frac{K_z}{| \vec{h} |}, \tag{21}$$

where $\vec{h}$ is the angular momentum.

The **true anomaly**, ν, is used to define the position of the body in its orbit, and is the angle between the direction of the periapsis and the current position of the body.

$$\nu[rad] = \begin{cases} \arccos \dfrac{\vec{e} \cdot \vec{r}}{\mid \vec{e} \mid\mid \vec{r} \mid} & \text{for } \vec{r} \cdot \vec{v} \geq 0 \\ 2\pi - \arccos \dfrac{\vec{e} \cdot \vec{r}}{\mid \vec{e} \mid\mid \vec{r} \mid} & \text{otherwise} \end{cases} \tag{22}$$

The **eccentric anomaly**, E, is also used to measure an object's position in an orbit. The eccentric anomaly is defined using the magnitude of the eccentricity vector, $e = \mid \vec{e} \mid$, and the true anomaly as follows:

$$E = 2 \arctan \frac{\tan \dfrac{\nu}{2}}{\sqrt{\dfrac{1+e}{1-e}}} \tag{23}$$

The **longitude of the ascending node**, Ω, is the angle between the ascending node and the unit vector $\hat{\imath}$

$$\Omega[rad] = \begin{cases} \arccos \dfrac{n_x}{\mid \vec{n} \mid} & \text{for } n_y \geq 0 \\ 2\pi - \arccos \dfrac{n_x}{\mid \vec{n} \mid} & \text{for } n_y < 0. \end{cases} \tag{24}$$

The **argument of periapsis**, ω, is the angle in the orbital plane that is between the ascending node and the periapsis and is calculated using

$$\omega[rad] = \begin{cases} \arccos \dfrac{\vec{n} \cdot \vec{e}}{\mid \vec{n} \mid\mid \vec{e} \mid} & \text{for } e_z \geq 0 \\ 2\pi - \arccos \dfrac{\vec{n} \cdot \vec{e}}{\mid \vec{n} \mid\mid \vec{e} \mid} & \text{for } e_z < 0 \end{cases} \tag{25}$$

Finally, using **Kepler's equation** we can define the **mean anomaly** as

$$M = E - e \sin E \tag{26}$$

We can now parameterize each of the debris fragments using (e, a, i, Ω, ω, M). This step is crucial to ensure that in the absence of orbital perturbations, only one of these parameters, the mean anomaly, will change with respect to time, t. This will allow for more efficient computations in the band formation phase of the simulation. Moreover, the change in mean anomaly with respect to time is analytic and expressed as

$$\frac{dM}{dt} = \sqrt{\frac{\mu}{a^3}}. \tag{27}$$

As such, the position of a piece of debris in orbit can be found at any time in the future without having to numerically approximate its change in position over time. Due to the vast amount of debris generated, this will provide major benefits for the band formation phase of the orbital debris cloud.

4 Debris Cloud Evolution

Debris cloud evolution is the collective change in the orbits and positions of fragments with respect to time. This evolution goes through a few distinctive phases, notably the ellipsoid, ring formation, toroid, and band formation, as illustrated in Figure 4.9. In addition to having different shapes, those phases differ in how long they last and the type of model used to describe them. For example, the ellipsoid and ring formations occur within a few days. As such, only the force of gravity will have a significant effect during this period. However, the toroid and band formation phases take over a year. As a result, additional forces such as drag will have more prominent effects due to the increased duration. Different methodologies need to be applied for each phase to ensure fast and accurate computations. These different phases, the forces being considered, and the methodologies used will be analyzed in-depth in the following subsections.

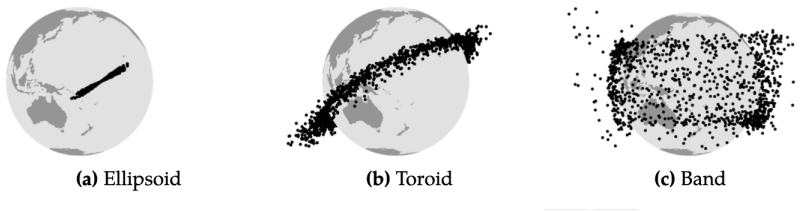


Figure 4.9: The three phases of debris cloud evolution [13].

4.1 Ellipsoid and Ring Phase

At the time of the fragmentation event, all debris fragments have the same position. However, as seen in Section 2, each fragment has a different velocity in both direction and magnitude. This causes the debris to spread

out, form an ellipsoid (Figure 4.10), and then move into a complete ring around the Earth. As the debris cloud expands, the number of other satellites it can potentially impact grows. On the other hand, the cloud also gets sparser, and the probability of any specific satellite getting hit decreases. In other words, the odds of subsequent collision spread out as well.

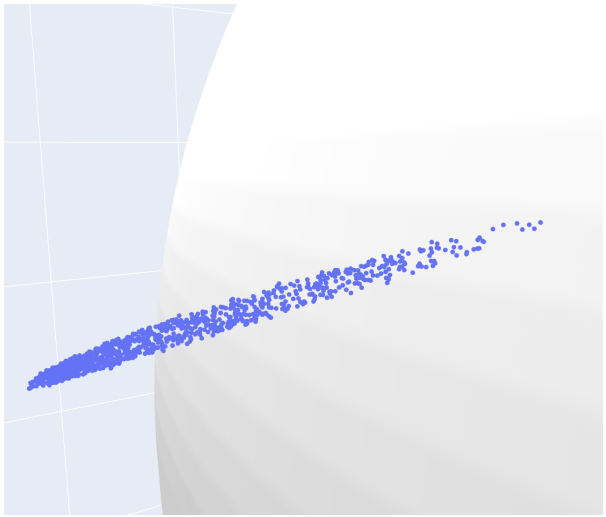


Figure 4.10: A visualization created by ODAP of the ellipsoid formation occurring.

It takes two to three days for the debris to form a uniform ring around the Earth. On this time scale, weak effects such as air drag and gravitational forces between debris are negligible. As a result, each debris fragment orbits around the Earth as if the other fragments did not exist. From a practical point of view, we can use standard results for the two-body gravitational problem [17].

While the two-body problem is well characterized, it is not computationally efficient enough to perform long duration and accurate propagation. As such, we reframe the problem as propagating Kepler orbits.

Kepler orbits are orbits in which no perturbations of inter-gravitational interactions are considered—a special case of the two-body problem. Fram-

ing the problem this way lends itself naturally to using Keplerian elements. For Kepler orbits, only the orbits' anomalies will be changing over time, which reduces the number of computations that need to be performed and aids in maintaining accuracy.

This benefit is a result of **Kepler's equation**, which is expressed in terms of the mean anomaly as

$$M = E - e\sin(E) = \sqrt{\frac{\mu}{a^3}}\,(t - t_0)\,, \tag{28}$$

where E is the eccentric anomaly, e is the eccentricity, μ is the standard gravitational parameter, a is the semi-major axis, t is some time in the future, and t_0 is the current time. Since we have an analytic expression for the mean anomaly, we do not need to use any integration methods for propagating Kepler orbits forward in time. This results in the aforementioned computational efficiencies.

The next step is to find the corresponding eccentric and true anomalies from the propagated mean anomaly. The eccentric anomaly is found from the Kepler equation; however, there is no closed-form solution given the mean anomaly. As such, numerical integration must be used to find the eccentric anomaly. The processes for performing the integration of Kepler's equation are the subject of many other research papers due to the importance it plays in orbit propagation. It is not included in this paper for conciseness, but a Python implementation of one of the methods can be found in the appendix Section C.3.

Kepler's equation has another form which enables us to convert the re-

sulting eccentric anomaly to the true anomaly. This form is given as

$$\nu = 2 \arctan \left(\sqrt{\frac{1+e}{1-e}} \tan \frac{E}{2} \right). \tag{29}$$

Since we now have expressions for how the three anomalies will change over time for Kepler orbits, we can accurately propagate the debris until the ring formation phase is completed. However, we now need a method to detect the completion of the ring formation phase, cuing the switch to performing the propagation for future phases. A visualization of end result of the ring formation phase is provided in Figure 4.11.

4.2 Transition to Toroid and Band Phase

Detecting the end of the ring formation phase is crucial in propagating the debris cloud. When transitioning from the ring phase to the toroid and band phases, additional forces such as drag must be considered. Additionally, these new phases take a much longer amount of time to form. Both of these factors result in needing to switch to a new propagation method. As

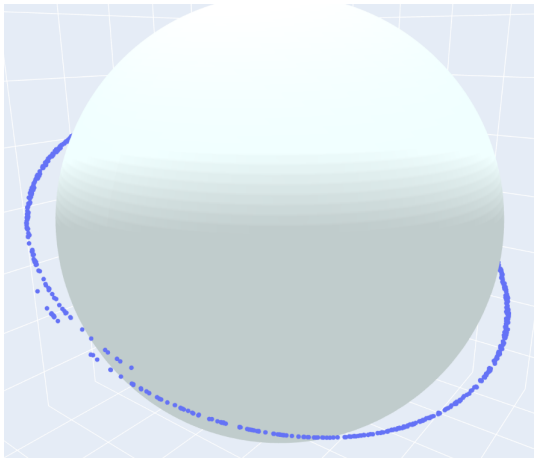
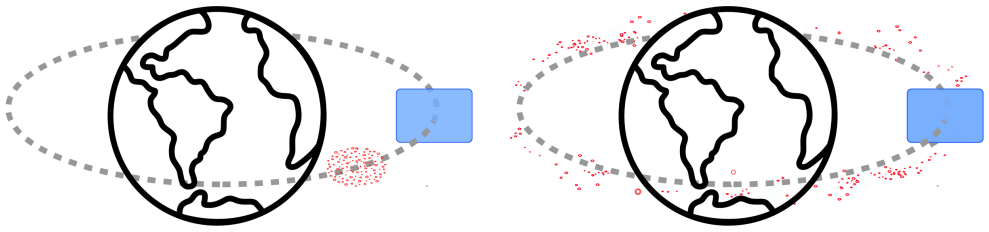


Figure 4.11: A visualization created by ODAP of the completed ring formation.

such, detecting when the ring has formed allows us to switch to this new propagation method.

To assess whether the system has formed a roughly uniform ring, we monitor the amount of debris passing through a certain region of space as a function of time. A visualization of how this process works is provided in Figure 4.12.

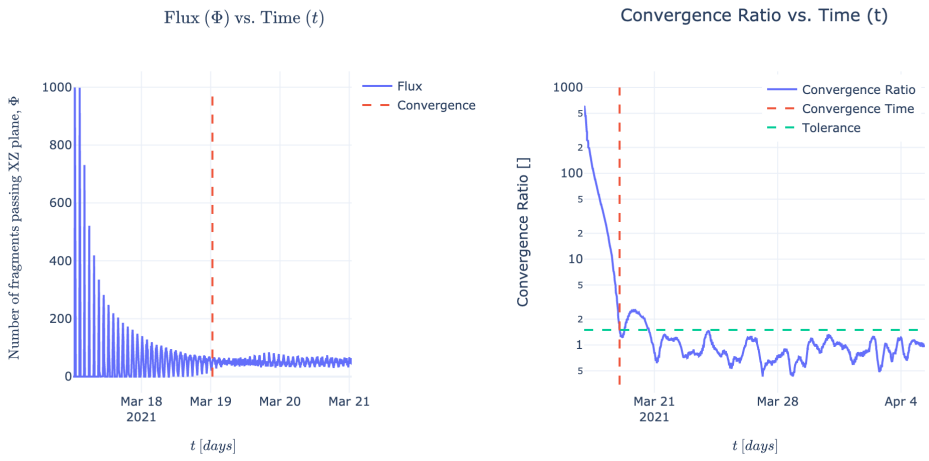


(a) (b)

Figure 4.12: Measuring the flux at the time of the ellipsoid phase **(a)** and measuring the flux near the completion of the ring phase **(b).**

More specifically, to measure the fragments' spread we will define a particle-based flux. This is accomplished by creating an xz-plane in the equatorial coordinate system and detecting when particles have switched from one side of the plane to the other. Plotting the results will show peaks when the fragments pass through, which will converge to some value as the fragments become uniformly spread out.

Plotting the flux over time shows these expected peaks and what seems to be convergence to some value. However, the data is quite noisy and we need a concrete method to determine when the fragments will be distributed uniformly. As such, we need to develop a method to test for when the data has converged. A property that is true of all uniform distributions is that the standard deviation approximately equals the mean. As such,



(a) (b)

Figure 4.13: The flux of the fragments as a function of time **(a)** and the convergence ratio of the flux as a function of time **(b).**

we can define a convergence ratio using the variance and mean of the flux. When the convergence ratio is within some defined tolerance we can test for when the flux has become uniform. Once the convergence ratio is within the defined tolerance the ring formation phase has completed. It should be noted that the ring never reaches true uniformity which results in oscillating flux. An example of how the flux and convergence ratio evolves with time is shown in Figure 4.13. Additional information about the derivation used to determine when the ring formation phase has ended can be found in Section Aof the appendix.

4.3 Toroid and Band Phase

For the evolution of the cloud to continue, we must now consider orbital perturbations that will cause the orbits to change over time. Perturbations are forces that act on each debris fragment and result in the motion changing over time. The two most dominant of these forces are drag and the J_2

perturbation. The result of drag is the debris cloud expanding inward, as the effects of the atmosphere slow down debris fragments. Similarly, the J_2 perturbation results from the gravitational field produced by the earth and causes the debris to spread out. The results of applying these perturbations are shown in Figure 4.14.

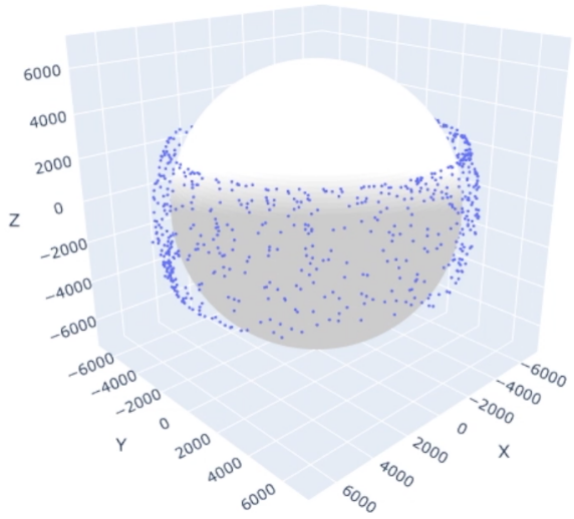


Figure 4.14: A visualization created by ODAP of the completed band formation.

4.3.1 Aerodynamic Drag

A significant perturbation that causes changes in the fragments' orbits is **atmospheric drag**. Atmospheric drag acts on a fragment due to molecules in the atmosphere colliding with its surface and is a natural consequence of the conservation of momentum. This causes a fragment to slow down, which in return lowers the orbit. This cycle continues until the object is no longer in orbit and re-enters the atmosphere. Figure 4.15 illustrates this effect.

To model the effects of drag, a suitable atmospheric model must be se-

Figure 4.15: A diagram illustrating how the force of drag slows down a satellite, causing an eventual deorbit.

lected. An atmospheric model tells us information about factors such as air pressure, air density,and wind speed at varying locations in the atmosphere. However, it would be cumbersome and computationally inefficient to utilize a model that contains all of this information. As such, atmospheric models tend to focus on a select few variables rather than creating a universal "best" model that considers all relevant factors [17].

Choosing a model that best fits a given use case comes down to the desired criterion for speed, accuracy, and applicability. For example, the DAMAGE orbital debris model assumes a rotating, oblate atmosphere with density and density scale height values taken from the 1972 COSPAR International Reference Atmosphere (CIRA) [14]. A detailed exploration of various atmospheric models is given by Gaposchkin and Coster [8], but for the purposes of this paper, we will be focusing on a model called the **exponential atmospheric model**. This is a a static model that assumes a spherically symmetric distribution of particles, where the density decays exponentially with increasing altitude [17].

In the exponential atmospheric model the air density ρ varies according

to

$$\rho = \rho_0 \ \exp\left(-\frac{h_{ellp} - h_0}{H}\right) \tag{30}$$

where ρ_0 is a reference density that is used with h_0, a reference altitude, h_{ellp} is the actual altitude above the ellipsoid, and H is a scale height.

The reference density and reference altitude are tabulated values that come from sources such as the U.S. Standard Atmosphere and CIRA. The scale height is a value used to ensure continuity throughout ρ. The combination of the U.S. Standard Model and CIRA will yield moderately accurate results for general purposes and as such will utilized in for computing the force of drag [17]. The tabulated values are included in the appendix Section B.

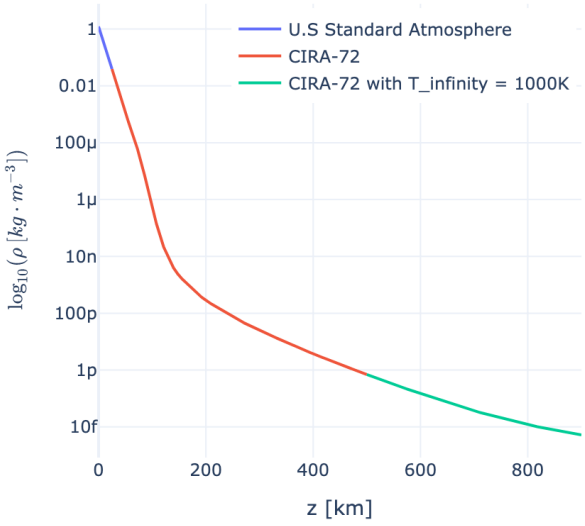


Figure 4.16: The atmospheric density of Earth as a function of altitude according to the Exponential Atmospheric Model.

The air density, ρ, as a function of altitude is shown above. Note that

the y-axis is given as a log scale due to the density of rapidly becoming thinner as altitude increases.

4.3.2 Effects of Drag on Orbital Elements

The aerodynamic drag on an object is typically expressed in the following form:

$$F_D = \frac{1}{2m}\rho v^2 \, C_D A \tag{31}$$

where ρ is the air density, v is the speed of the object relative to the fluid, C_D is an experimentally determined dimensionless number, A is the cross sectional area, and m is the mass of the satellite.

Drag is a force that will cause an acceleration that opposes the direction of motion of an object. As such, drag produces a similar effect to a retrograde thrust which enables aerobraking, a useful orbital maneuver that can be performed around planetary bodies with an atmosphere. Similarly, for orbital debris, drag is the predominant force behind changing the semi-major axis and eccentricity, gradually causing the debris to have a lower perigee.

As a result of the density increasing exponentially as altitude decreases, the effects of drag create a form of a feedback loop. A satellite experiences drag, which lowers its orbit, which in turn causes it to experience more drag. This will continue until a satellite eventually burns up in the atmosphere or reaches the ground.

The expressions for the effects of drag on orbital elements were derived by King-Hele and cover three different ranges of eccentricities [11].

$$\frac{da}{dt} = \begin{cases} -\frac{C_D A}{M}\sqrt{\mu_E a\rho}\exp\left(-\frac{a-R_h}{H}\right)[I_0 + 2eI_1 + & \\ \frac{3}{4}e^2(I_0+I_2) + \frac{e^3}{4}(3I_1+I_3)] & \text{for } 0.01 \leq e \leq 0.2 \\ -\frac{C_D A}{M}\sqrt{\mu_E a\rho}\exp\left(-\frac{a-R_h}{H}\right)[I_0 + 2eI_1] & \text{for for } 0.001 \leq e < 0.01 \\ -\frac{C_D A}{M}\sqrt{\mu_E a\rho}\exp\left(-\frac{a-R_h}{H}\right) & \text{for } e < 0.001 \end{cases} \tag{32}$$

$$\frac{de}{dt} = \begin{cases} -\frac{C_D A}{M}\sqrt{\frac{\mu_E}{a}}\rho\exp\left(-\frac{a-R_h}{H}\right) & \text{for } 0.01 \leq e \leq 0.2 \\ -\frac{C_D A}{M}\sqrt{\frac{\mu_E}{a}}\rho\exp\left(-\frac{a-R_h}{H}\right)\left[I_1 + \frac{e}{2}(I_0+I_2)\right] & \text{for } 0.001 \leq e < 0.01 \\ 0 & \text{for } e < 0.001 \end{cases} \tag{33}$$

$I_k(z)$ represents the modified Bessel function with order n which is defined as

$$I_k(z) = \frac{1}{\pi}\int_0^\pi e^{z\cos(\theta)}\cos(k\theta)d\theta \qquad k \in \mathbb{Z}, \tag{34}$$

where $z = \dfrac{ae}{H}$.

Following the precedents of many other texts, we will be assuming that the fragments have a drag coefficient of 0.7. Modifications were made by Frey et al. to find more appropriate boundary conditions and to increase the accuracy of each phase by including more terms of the series expansion [7].

The first modification to the King-Hele implementation is to introduce

two functions, k_a and k_e, that are used for describing the rate of change of a and e in all eccentricity regimes [7].

$$k_a = \delta\sqrt{\mu a}\rho(h_p)$$

$$k_e = k_a/a$$

For circular orbits, $e = 0$, the change in a and e can be solved using the following expression:

$$\frac{da}{dt} = -k_a$$

$$\frac{de}{dt} = 0$$

For low eccentric orbits, $e < e_b(a, H)$, a series expansion in e is performed and integrated using the first kind modified Bessel function as

$$\frac{da}{dt} = -k_a \exp(-z)(\boldsymbol{e}^T \boldsymbol{K}_{\boldsymbol{a}}^{\boldsymbol{l}} \boldsymbol{I} + O(e^6)) \tag{35}$$

$$\frac{de}{dt} = -k_e \exp(-z)(\boldsymbol{e}^T \boldsymbol{K}_{\boldsymbol{e}}^{\boldsymbol{l}} \boldsymbol{I} + O(e^6)), \tag{36}$$

where

$$\boldsymbol{e}^T = (1 \quad e \quad e^2 \quad e^3 \quad e^4 \quad e^5) \tag{37}$$

$$\boldsymbol{I}^T = (I_0 \quad I_1 \quad I_2 \quad I_3 \quad I_4 \quad I_5 \quad I_6) \tag{38}$$

$$\boldsymbol{K}_{\boldsymbol{a}}^{\boldsymbol{l}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ \frac{3}{4} & 0 & \frac{3}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{3}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 \\ \frac{21}{64} & 0 & \frac{28}{64} & 0 & \frac{7}{64} & 0 & 0 \\ 0 & \frac{30}{64} & 0 & \frac{15}{64} & 0 & \frac{3}{64} & 0 \end{bmatrix} \tag{39}$$

$$\boldsymbol{K}_{\boldsymbol{e}}^{\boldsymbol{l}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & -\frac{5}{8} & 0 & \frac{1}{8} & 0 & 0 & 0 \\ -\frac{5}{16} & 0 & -\frac{4}{16} & 0 & \frac{1}{16} & 0 & 0 \\ 0 & -\frac{18}{128} & 0 & -\frac{1}{128} & 0 & \frac{3}{128} & 0 \\ -\frac{18}{256} & 0 & -\frac{19}{256} & 0 & \frac{2}{256} & 0 & \frac{3}{256}. \end{bmatrix} \tag{40}$$

The vast majority of the debris falls in the low eccentricity range, so only the circular and low eccentricity expressions are implemented. The high

eccentricity range has a similar formulation provided by [7].

4.3.3 Nodal Precession

Nodal precession is the precession of the orbital plane of a satellite around the rotational axis of the central body. This is due to the non-spherical nature of the rotating central body. The non-spherical nature is a result of the centrifugal force produced by the rotation which deforms the body, causing an equatorial bulge.

As a result, the planetary body creates a non-uniform gravitational field that induces a torque on satellites.

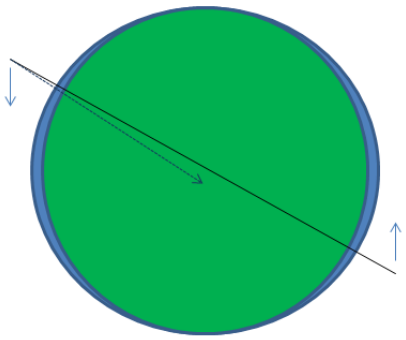


Figure 4.17: An illustration of an equatorial bulge causing nodal precession [3].

Intuitively, it would appear that the torque would reduce the inclination of the orbit. However, due to the bulge, the gravitational force is not directed towards the center of the body, but rather is offset toward the equatorial plane. As such, it causes torque-induced gyroscopic precession which causes the ascending and descending nodes to drift with time. This phenomenon is called **nodal precession**.

The effects of nodal precession on the ascending and descending nodes

is expressed by

$$\frac{d\omega}{dt} = \frac{3}{2}\, J_2 \frac{R_E^2}{p^2} \bar{n} \left(2 - \frac{5}{2}\sin^2(i)\right)$$

$$\frac{d\Omega}{dt} = -\frac{3}{2}\, J_2 \frac{R_E^2}{p^2} \bar{n} \cos(i)$$

where $p = a(1 - e^2)$ is the semi-latus rectum of the orbit, $\bar{n} = \sqrt{\frac{\mu_E}{a^3}}$ is the mean motion, R_E is the radius of the Earth, and J_2 is the Earths second dynamic form factor.

The J_2 term is a result of an infinite series equation that describes the perturbation effects of a rotating planetary body on the gravity of a planet. Each term of the series is denoted as J_n, however the J_2 term is more than 1000 times larger than the other terms [1]. This is why the J_2 effect is considered a relevant orbital perturbation for the evolution of an orbital debris cloud. For context, Earth's J_2 term has a value of 0.108×10^{-2} whereas its J_3 term has a value of 0.253×10^{-5}.

5 Results

Now that the critical components for debris cloud evolution are established, we can use ODAP to gain insights into orbital debris's evolution. The first step is choosing a suitable scenario for the fragmentation event. Once the scenario is selected, the propagation methods defined in Section 4 can be applied to study the debris throughout the various phases. This includes analyzing the estimated time for the fragments to deorbit and the fragments' spread over time.

5.1 Data Source

Selecting a realistic scenario to analyze can be difficult without prior knowledge of typical satellite orbits. As such, it is beneficial to perform analysis on existing satellites in low Earth orbit. This can be accomplished by utilizing a database of **Two-lines elements** (TLE's), which are a standardized way of describing information about a satellite's orbit.

ODAP utilizes a TLE database maintained by the website CelesTrak[1] to make all cataloged objects in orbit available for fragmentation events.

The analysis conducted in the following subsections is performed on two different satellites. The first is a satellite called OXP 1, which has an altitude of approximately 750 km and a 25-degree inclination. This is in a higher region of low Earth orbit and will allow us to analyze long-term changes on the debris cloud.

The second satellite is a Starlink satellite produced by SpaceX. This satellite is at a much lower altitude, around 580 km, and is a member of a

[1]Additional information about TLE's and CelesTrak can be found on their https://www.celestrak.com/NORAD/elements/

satellite mega constellation currently being constructed to provide global internet coverage. It was selected as part of this analysis due to the increasing number of companies expressing interest in low Earth orbit mega-constellations. Additionally, the Starlink mega-constellation has quickly grown to having over one thousand satellites in orbit, with the end goal being 30,000 [9].

5.2 Decay Time

When a fragmentation event occurs, it is relevant to study the debris fragments' expected duration in orbit. Debris in orbit for a longer duration will have a higher probability of eventually colliding with other objects in space.

We expect that fragmentation events that occur in higher altitudes will generate debris that last longer. This is a direct consequence of drag's weaker effect at high altitudes due to the decreased air density. Additionally, since drag depends on the surface area of a debris fragment, we expect to see some debris fragments that deorbit faster than others at similar altitudes.

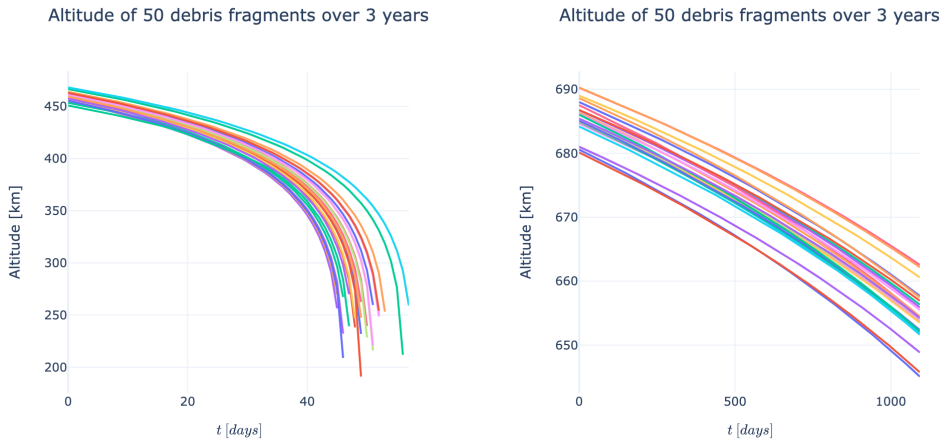
To analyze the orbital decay time using ODAP, we can use the simulated data from the band formation phase. The output of this phase is the change in the Keplerian elements over time. As such, we can use the expression for the **perigee**, which is the altitude at the lowest point of an orbit. The perigee is defined using Keplerian elements as

$$z = a(1 - e). \tag{41}$$

It should be noted that the perigee is being measured from the center of the Earth. To find the true altitude, we must subtract the radius of the Earth from the result.

Applying this process for both the Starlink satellite and OXP 1 results in the data used in Figure 5.18. The fragmentation event used in this simulation was identical for both satellites. The only changes were the starting orbits of the satellites. While only a random sample of data from both simulations is displayed in the Figure, the data revealed that 100% of the Starlink fragments were deorbited within three years while only 1.5% of the OXP 1 fragments were deorbited during this same interval.

This simulation concludes that fragmentation events occurring with mega-constellations in the lower region of LEO are of less concern than those occurring in higher orbits. While both create immense amounts of debris, the lower altitude Starlink satellites pose less of a threat to the long term health of LEO.



(a) (b)

Figure 5.18: The altitudes of 50 pieces of debris generated by a non-catastrophic collision with a relative impact velocity of 2 km/s for both the Starlink satellite **(a)**, and OXP 1 **(b).**

5.3 Spread

Another essential characteristic of orbital debris to study is the spread of the debris over time. The procedure for obtaining the necessary data to measure the spread of the fragments is similar to the decay time. However, we do not need any additional equations and can visualize the spread of the fragments directly from the results provided by the band formation phase. We can use the distribution of the right angle of the ascending node Ω over time to get a sense of how the debris are spreading apart due to the J_2 perturbation.

This component of the analysis was conducted by performing a kernel density estimation on Ω at three different times of the band formation phase, the start, middle, and end time. A kernel density estimation is a way to estimate the probability density function of a random variable. As such, we can use it to get a sense of how the distribution of Ω changes

throughout time.

Figure 5.19 shows the results of performing this type of analysis on the fragmentation event used in the above subsection. As you can see, initially, most of the debris had a right angle of ascending node that was tightly clustered around the satellite's initial value. As the debris cloud evolves, this distribution becomes increasingly uniform. Notably, since the Starlink debris gets deorbited relatively quickly, it never has an opportunity to become entirely uniform. The simulated data for OXP exhibit this same phenomenon, thus allowing the analysis performed on the Starlink satellite suffice.

In Section 5.2, we learned that the initial altitude of a breakup event dramatically impacts the time for the debris to deorbit. However, this Section revealed that it does not play a significant role in how spread out the debris becomes over time. The purpose of this analysis is to demonstrate how ODAP can be applied to gain new intuitions and understanding about the evolution of orbital debris.

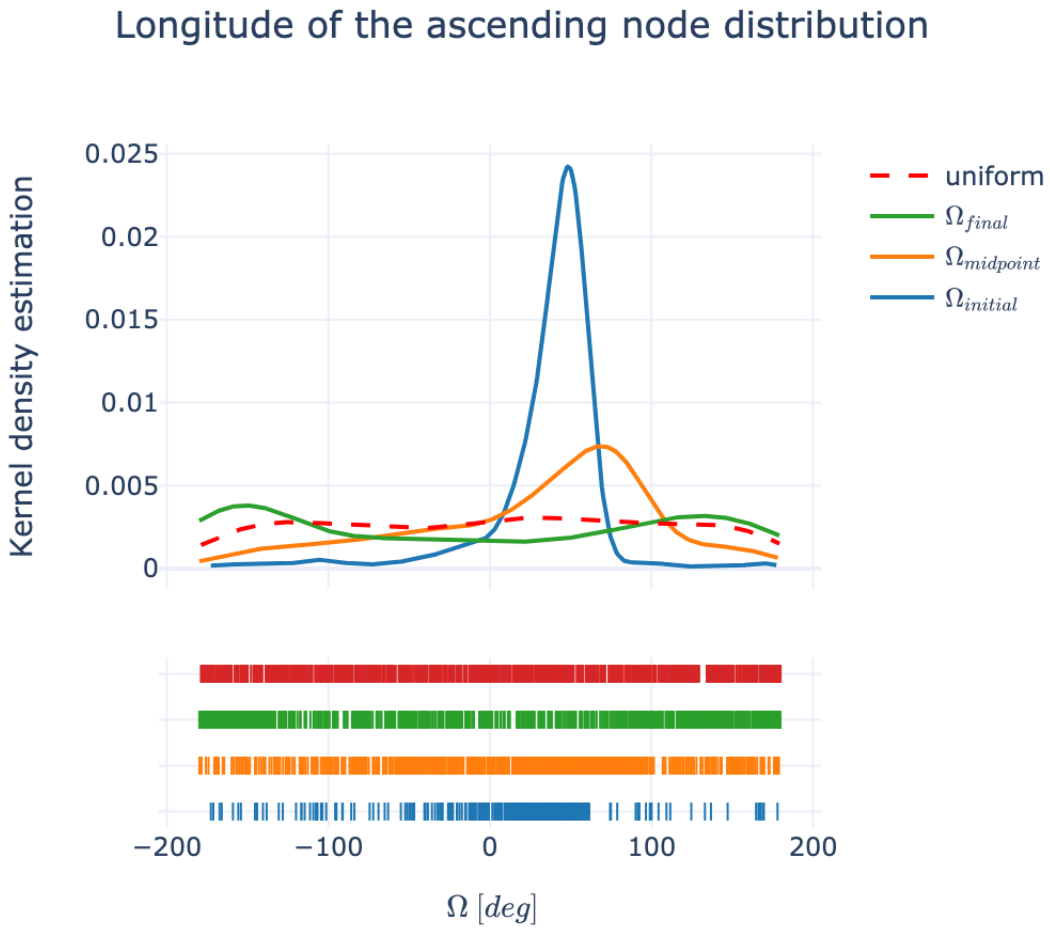


Figure 5.19: The kernel density estimation performed on the band formation phase of the Starlink satellite. The sampled values are from the beginning of the phase, the midpoint, and the end.

6 Conclusion

Orbital debris will be a persistent problem as humankind continues to explore and utilize space to advance science. As such, the ability to model and study the effects associated with it is crucial. Unfortunately, much of the existing literature is difficult to find and gives an incomplete picture of the entire process needed to model orbital debris. Additionally, most of the available code to perform the analysis is written in older programming languages that are no longer widely used. Thus, ODAP was developed to be an open-source, easily accessible tool that utilizes a modern programming language to aid in research conducted in this field. This thesis also serves as a complete introduction to the necessary information needed to understand the process of modeling orbital debris.

There are many avenues for expanding on the research conducted in this thesis. One of the most interesting is modeling the formed debris band analytically. This provides many benefits for conducting analysis but requires a significant amount of additional background information to understand. As such, it was outside the scope of this research. Additionally, ODAP will be continually refined to add new functionality, such as adding support for different atmospheric models. This would allow for a more in-depth analysis of the time it takes for debris to deorbit.

Many of these additional complexities are planned to be implemented into ODAP in the future. As such, the code base for ODAP is expected to evolve significantly over time. The appendix of this paper includes some of the most important functionality for reference, but the most up-to-date version will be found on Github.

Index

References

[1] J2 perturbation. https://ai-solutions.com/_freeflyeruniversityguide/j2_perturbation.htm#:~:text=The%20term%20J2%20comes%20from,the%20gravity%20of%20a%20planet.&text=The%20two%20main%20orbital%20elements,Argument%20of%20Perigee%20(%CF%89). (Accessed on 04/19/2021).

[2] Astrodynamics/Classical Orbit Elements - Wikibooks, open books for an open world. URL https://en.wikibooks.org/wiki/Astrodynamics/Classical_Orbit_Elements.

[3] Nodal precession, Dec 2020. URL https://en.wikipedia.org/wiki/Nodal_precession.

[4] Orbital state vectors, Sep 2020. URL https://en.wikipedia.org/wiki/Orbital_state_vectors.

[5] Semi-major and semi-minor axes, Mar 2021. URL https://en.wikipedia.org/wiki/Semi-major_and_semi-minor_axes.

[6] S. Barrows. EVOLUTION OF ARTIFICIAL SPACE DEBRIS CLOUDS.

[7] S. Frey, C. Colombo, and S. Lemmens. Extension of the King-Hele orbit contraction method for accurate, semi-analytical propagation of non-circular orbits. *Advances in Space Research*, 64 (1):1–17, July 2019. ISSN 0273-1177. doi: 10.1016/j.asr.2019.

03.016. URL https://www.sciencedirect.com/science/article/pii/S0273117719301978.

[8] E. Gaposchkin and A. Coster. Analysis of satellite drag. 1988.

[9] C. Henry. Spacex submits paperwork for 30,000 more starlink satellites, Oct 2019. URL https://spacenews.com/spacex-submits-paperwork-for-30000-more-starlink-satellites/.

[10] N. L. Johnson, P. H. Krisko, J. C. Liou, and P. D. Anz-Meador. NASA's new breakup model of evolve 4.0. *Advances in Space Research*, 28(9):1377–1384, Jan. 2001. ISSN 0273-1177. doi: 10.1016/S0273-1177(01)00423-9. URL http://www.sciencedirect.com/science/article/pii/S0273117701004239.

[11] D. King-Hele and D. Walker. The effect of air drag on satellite orbits: Advances in 1687 and 1987. *Vistas in Astronomy*, 30:269–289, 1987. ISSN 0083-6656. doi: https://doi.org/10.1016/0083-6656(87)90006-7. URL https://www.sciencedirect.com/science/article/pii/0083665687900067.

[12] P. H. Krisko. Proper Implementation of the 1998 NASA Breakup Model. *Orbital Debris Quarterly News*, 15(4):4,5, Oct. 2011. URL http://orbitaldebris.jsc.nasa.gov/.

[13] F. Letizia. *Space debris cloud evolution in Low Earth Orbit*. PhD thesis, temp, Feb. 2016.

[14] H. Lewis, A. E. White, R. Crowther, and H. Stokes. Synergy of debris mitigation and removal. *Acta Astronautica*, 81(1):62–68, December 2012. URL https://eprints.soton.ac.uk/388786/.

[15] J. C. Liou. Orbital Debris Modeling. URL https://ntrs.nasa.gov/citations/20120003286.

[16] A. Rossi. NASA Breakup Model Implementation Comparison of results. *temp*, page 29, 2021.

[17] D. A. Vallado and M. C. W. D. *Fundamentals of astrodynamics and applications*. Microcosm Press., 2013.

Appendices

Section A of this appendix covers the details of the flux derivation used to determine the end of the ring formation phase. Section B contains the tabulated values used in the exponential atmospheric mode. These values were difficult to find in other sources and are included in this appendix for convenience. Finally, Section C comprises of several relevant ODAP code samples. It should be noted that the latest version of ODAP can be found at https://github.com/ReeceHumphreys/ODAP. As such, some of the code in the appendix may be out of date when compared to the version hosted on Github. Additionally, the Github version includes a Jupyter notebook used for performing the analysis in Chapter 5 and generating figures included in this thesis.

A Flux

Let's pick a time interval T and call N the number of debris pieces that cross the blue cross-section in Figure 4.12. Let us divide the time interval into M subinterval of length T/M.

What we measure in the simulation is the number of debris crossing the cross-section during each subinterval. We are interested in predicting the statistics of those numbers when the positions of the debris along their orbits are distributed uniformly and independently from each other.

Each debris has a probability of 1/M of crossing during a specific subinterval. The probability of exactly n debris crossing during a specific subinterval obeys a binomial distribution:

$$P_n = C_N^n \left(\frac{1}{M}\right)^n \left(1 - \frac{1}{M}\right)^{N-n}.$$

Expectedly, the average number of debris crossing during a single subinterval is the number of debris crossing during the entire interval, T, divided by the number of subintervals:

$$\langle P_n \rangle = \sum_0^N n P_n = \frac{N}{M}.$$

The variance is:

$$\left\langle (P_n - \langle P_n \rangle)^2 \right\rangle = \frac{N}{M}\left(1 - \frac{N}{M}\right).$$

Therefore, if the debris ring is fully randomized and the average number of debris crossing per time step is ϕ, we should expect the actual measurement to fluctuate through time with a variance $\phi(1-\phi)$.

This, in turn, provides a criterion to assess whether the ring is fully randomized. Initially, when the debris are concentrated in a tight cluster, the variance is much higher. It then decreases gradually as the ring randomizes. When it becomes of order $\phi(1-\phi)$, we declare the ring randomized and move on to the band formation phase.

B Atmospheric Model Tabulated Values

Lower Bound (km)	Upper Bound (km)	Base Altitude (km)	Nominal Density (kg/m^3)	Scale Height (km)
0	25	0	1.225	7.249
25	30	25	3.899×10^{-2}	6.349
30	40	30	1.774×10^{-2}	6.682
40	50	40	3.972×10^{-3}	7.554
50	60	50	1.057×10^{-3}	8.382
60	70	60	3.206×10^{-4}	7.714
70	80	70	8.770×10^{-5}	6.549
80	90	80	1.905×10^{-5}	5.799
90	100	90	3.396×10^{-6}	5.382
100	110	100	5.297×10^{-7}	5.877
110	120	110	9.661×10^{-8}	7.263
120	130	120	2.438×10^{-8}	9.473
130	140	130	8.484×10^{-9}	12.636
140	150	140	3.845×10^{-9}	16.149
150	180	150	2.070×10^{-9}	22.523
180	200	180	5.464×10^{-10}	29.74
200	250	200	2.789×10^{-10}	37.105
250	300	250	7.248×10^{-11}	45.546
300	350	300	2.418×10^{-11}	53.628
350	400	350	9.518×10^{-12}	53.298
400	450	400	3.725×10^{-12}	58.515
450	500	450	1.585×10^{-12}	60.828
500	600	500	6.967×10^{-13}	63.822
600	700	600	1.454×10^{-13}	71.835
700	800	700	3.614×10^{-14}	88.667
800	900	800	1.170×10^{-14}	124.64
900	1000	900	5.245×10^{-15}	181.05

Table 2: The values used in the exponential atmospheric model [17].

C Code

C.1 Breakup Model

Listing 1: Fragmentation event implementation

```
import numpy as np
import scipy
from enum import IntEnum

debris_category = IntEnum('Category', 'rb sc soc')
from numba import njit, prange

""" ----------------- Mean ----------------- """
def make_mean_AM(debris_type):

    def RB_mean_AM(lambda_c):

        mean_am_1 = np.empty_like(lambda_c)
        mean_am_2 = np.empty_like(lambda_c)

        mean_am_1[lambda_c<=-0.5] = -0.45
        I = (lambda_c>-0.5) & (lambda_c<0)
        mean_am_1[I] = -0.45 - (0.9*(lambda_c[I] +0.5))
        mean_am_1[lambda_c>=0] = -0.9

        mean_am_2.fill(-0.9)

        return np.array([mean_am_1,mean_am_2])

    def SC_mean_AM(lambda_c):
        mean_am_1 = np.empty_like(lambda_c)
        mean_am_2 = np.empty_like(lambda_c)

        mean_am_1[lambda_c<=-1.1] = -0.6
        I = (lambda_c>-1.1) & (lambda_c<0)
        mean_am_1[I] = -0.6 - (0.318*(lambda_c[I] +1.1))
        mean_am_1[lambda_c>=0] = -0.95

```

```
        mean_am_2[lambda_c<=-0.7] = -1.2
        I = (lambda_c>-0.7) & (lambda_c<-0.1)
        mean_am_2[I] = -1.2 - (1.333*(lambda_c[I] + 0.7))
        mean_am_2[lambda_c>=-0.1] = -2.0

        return np.array([mean_am_1,mean_am_2])

    def SOC_mean_AM(lambda_c):

        mean_am_1 = np.empty_like(lambda_c)
        mean_am_2 = np.empty_like(lambda_c)

        mean_am_1[lambda_c<=-1.75] = -0.3
        I = (lambda_c>-1.75) & (lambda_c<-1.25)
        mean_am_1[I] = -0.3 - (1.4*(lambda_c[I] +1.75))
        mean_am_1[lambda_c>=-1.25] = -1.0

        mean_am_2.fill(0)
        return np.array([mean_am_1,mean_am_2])

    if debris_type == debris_category.rb:
        return RB_mean_AM
    elif debris_type == debris_category.sc:
        return SC_mean_AM
    else:
        return SOC_mean_AM

""" ----------------- Standard Deviation ----------------- """

def make_standard_dev_AM(debris_type):

    def RB_std_dev_AM(lambda_c):

        std_dev_1 = np.empty_like(lambda_c)
        std_dev_2 = np.empty_like(lambda_c)

        std_dev_1.fill(0.55)

        std_dev_2[lambda_c<=-1.0] = 0.28
        I = (lambda_c>-1.0) & (lambda_c<0.1)
```

```
74          std_dev_2[I] = 0.29 - (0.1636*(lambda_c[I] +1))
75          std_dev_2[lambda_c>=0.1] = 0.1
76
77          return np.array([std_dev_1,std_dev_1])
78
79
80      def SC_std_dev_AM(lambda_c):
81
82          std_dev_1 = np.empty_like(lambda_c)
83          std_dev_2 = np.empty_like(lambda_c)
84
85          std_dev_1[lambda_c<=-1.3] = 0.1
86          I = (lambda_c>-1.3) & (lambda_c<-0.3)
87          std_dev_1[I] = 0.1 + (0.2*(lambda_c[I] +1.3))
88          std_dev_1[lambda_c>=-0.3] = 0.3
89
90          std_dev_2[lambda_c<=-0.5] = 0.5
91          I = (lambda_c>-0.5) & (lambda_c<-0.3)
92          std_dev_2[I] = 0.5 - ((lambda_c[I] + 0.5))
93          std_dev_2[lambda_c>=-0.3] = 0.3
94
95          return np.array([std_dev_1,std_dev_1])
96
97
98      def SOC_std_dev_AM(lambda_c):
99          std_dev_1 = np.empty_like(lambda_c)
100         std_dev_2 = np.empty_like(lambda_c)
101
102         std_dev_1[lambda_c<=-3.5] = 0.2
103         I = (lambda_c>-3.5)
104         std_dev_1[I] = 0.2 + (0.1333*(lambda_c[I] +3.5))
105
106         std_dev_2.fill(0)
107
108         return np.array([std_dev_1,std_dev_1])
109
110     if debris_type == debris_category.rb:
111         return RB_std_dev_AM
112     elif debris_type == debris_category.sc:
113         return SC_std_dev_AM
```

```
    else:
        return SOC_std_dev_AM

""" ----------------- Alpha ----------------- """
def alpha_AM(lambda_c, debris_type):
    def RB_alpha_AM(lambda_c):
        alpha = 1
        if lambda_c <= -1.4:
            alpha = 1
        elif (lambda_c > -1.4 and lambda_c < 0):
            alpha = 1 - (0.3571*(lambda_c + 1.4))
        else:
            alpha = 0.5
        return alpha

    def SC_alpha_AM(lambda_c):
        alpha = 1
        if lambda_c <= -1.95:
            alpha = 0
        elif (lambda_c > -1.95 and lambda_c < 0.55):
            alpha = 0.3 + (0.4*(lambda_c + 1.2))
        else:
            alpha = 1
        return alpha

    def SOC_alpha_AM(lambda_c):
        # Is not used by SOC, for saftey returning 1
        alpha = 1
        return alpha

    if debris_type == debris_category.rb:
        return RB_alpha_AM(lambda_c)
    elif debris_type == debris_category.sc:
        return SC_alpha_AM(lambda_c)
    else:
        return SOC_alpha_AM(lambda_c)

alpha_AM = np.vectorize(alpha_AM)

""" ----------------- Distribution A/M ----------------- """
```

```
154 def distribution_AM(lambda_c, debris_type):
155 
156     N = len(lambda_c)
157     lambda_c = np.array(lambda_c)
158 
159     mean_factory = make_mean_AM(debris_type)
160     std_dev_factor = make_standard_dev_AM(debris_type)
161 
162     mean_preSwitch = np.array(mean_factory(lambda_c))
163     std_dev_preSwitch = np.array(std_dev_factor(lambda_c))
164 
165     alpha = np.array(alpha_AM(lambda_c, debris_category.rb)) # This
        takes a long time
166     switch = np.random.uniform(0,1, N)
167 
168     if debris_type == debris_category.rb or debris_type ==
        debris_category.sc:
169 
170         means = np.empty(N)
171         I,J = switch<alpha, switch>=alpha
172         means[I] = mean_preSwitch[0, I]
173         means[J] = mean_preSwitch[1, J]
174 
175         devs = np.empty(N)
176         devs[I] = std_dev_preSwitch[0, I]
177         devs[J] = std_dev_preSwitch[1, J]
178 
179         return np.random.normal(means, devs, N)
180 
181     else:
182         means = mean_preSwitch[0]
183         devs = std_dev_preSwitch[0]
184         return np.random.normal(means, devs, N)
185 
186 """ ----------------- Area ----------------- """
187 def avg_area(L_c):
188     A = np.copy(L_c)
189     I = A < 0.00167 #(m)
190     A[I] = 0.540424 * A[I]**2
191     I = A >= 0.00167 #(m)
```

```
    A[I] = 0.556945 * A[I]**2.0047077
    return A

""" ----------------- Mean ----------------- """
@njit()
def mean_deltaV(kai, explosion):
    if explosion == True:
        return (0.2 * kai) + 1.85
    else:
        return (0.9 * kai) + 2.9

""" ----------------- Standard Deviation ----------------- """
def std_dev_deltaV():
    return 0.4

""" ----------------- Distribution delta V ----------------- """
@njit()
def distriNormale(mu,sigma,x):
    p = (1/(sigma*np.sqrt(2*np.pi))*np.exp(-1/2.*((x-mu)/sigma)**2))
    return p

@njit()
def distriDeltaVExpl(nu,chi):
    mu = 0.2*chi + 1.85
    return distriNormale(mu,0.4,nu)

@njit( parallel=True )
def distribution_deltaV(chi, v_c, explosion=False):
        N = len(chi)
        result = np.empty_like(chi)
        progress = 0

        for i in prange(N):
            mean = mean_deltaV(chi[i], explosion)
            dev  = 0.4
            x = np.random.rand()
            dv =x*1.3*v_c
            dist = distriDeltaVExpl(np.log10(dv),chi[i])
            y = np.random.rand()
            while y > dist:
```

```
                x = np.random.rand()
                dv = x*1.3*v_c
                dist =  distriDeltaVExpl(np.log10(dv),chi[i])
                y = np.random.rand()
            result[i] = dist
        return result

""" ----------------- Unit vector delta V ----------------- """
def unit_vector(N):
    vectors = np.random.normal(0, 1, np.array([N, 3]))
    vectors /= np.sqrt((vectors**2).sum(axis=1))[:, None]
    return vectors

def velocity_vectors(N, target_velocity, velocities):
    unit_vectors = unit_vector(N)
    velocity_vectors = velocities[:, None] * unit_vectors
    return target_velocity + velocity_vectors

from numpy.random import uniform

""" ----------------- Num. Fragments & Char. Length-----------------
    """
def number_fragments(l_characteristic, m_target, m_projectile, v_impact
    , is_catastrophic, debris_type, explosion):
    # Defining reference mass
    if explosion == True:
        print("explosion")
        return 6*(l_characteristic)**(-1.6)
    else:
        m_ref = 0
        if is_catastrophic:
            m_ref = m_target + m_projectile
        else:
            m_ref = m_projectile * (v_impact)**2
        return 0.1 * ((m_ref)**0.75) * l_characteristic**(-1.71)


def characteristic_lengths(m_target, m_projectile, v_impact,
    is_catastrophic, debris_type, explosion):
    bins = np.geomspace(0.001, 1, 100)
```

```
269     N_fragments = number_fragments(bins, m_target, m_projectile,
        v_impact, is_catastrophic, debris_type, explosion)
270     N_per_bin = np.array(N_fragments[:-1] - N_fragments[1:]).astype(int
        )
271     L_c = np.concatenate([uniform(bins[i], bins[i+1], size=N_per_bin[i
        ]) for i in range(len(bins) - 1)])
272     return L_c
273
274 def fragmentation(m_target, m_projectile, v_impact, is_catastrophic,
        debris_type, explosion):
275     prelim_L_c = characteristic_lengths(m_target, m_projectile,
        v_impact, is_catastrophic, debris_type, explosion)
276     prelim_lambda_c = np.log10(prelim_L_c)
277     prelim_areas = avg_area(prelim_L_c)
278     prelim_AM = np.array(distribution_AM(prelim_lambda_c, debris_type))
279     prelim_masses = prelim_areas / 10**prelim_AM
280
281     if explosion == True:
282         unaccounted_mass = m_target - np.sum(prelim_masses)
283         n_large_deb = np.random.randint(2, 8) # Pick 2-8 pieces of deb
        > 1m to spread out the rest of the mass
284
285         # Create mass range, will use `n_large_deb` to split into
        sections, using 10**-4 to enure endpoints are not included
286         mass_range = np.linspace(10**-4, (unaccounted_mass - 10**-4),
        10**4)
287         ranges = np.sort(np.random.choice(mass_range, n_large_deb - 1,
        replace=False))
288         ranges = np.concatenate([[0],ranges,[unaccounted_mass]])
289
290         # Adding zeros for subtraction to have correct number of dim.
291         mass_per_deb = np.concatenate((ranges[1:],np.zeros(1))) -
        ranges
292         mass_per_deb = np.resize(mass_per_deb, mass_per_deb.size - 1)
293
294         # For L_c > 1, A/M Distribution is basically deterministic,
        therefore will just use avg value
295         assumed_AM_factory = make_mean_AM(debris_type)
296         assumed_len = np.ones(mass_per_deb.shape)
297         assumed_AM = assumed_AM_factory(assumed_len)
```

```
298
299         # Each mean has two possible values, randomly pick one of them
    for each piece of deb
300         AM_choices = np.random.choice([0,1], len(mass_per_deb), replace
    =True)
301         assumed_AM = 10**np.array([assumed_AM[AM_choices[i], i] for i
    in range(assumed_AM.shape[1])])
302
303         # mass * AM = A(L_c), therefore can reverse Area function for
    L_c
304         area = mass_per_deb * assumed_AM
305         found_L_c = np.sort((area / 0.556945)**(1/2.0047077)) #
    Inversing the Area function defined above
306         found_lambda_c = np.log10(found_L_c)
307         found_areas = avg_area(found_L_c)
308
309         found_AM = np.array(distribution_AM(found_lambda_c, debris_type
    ))
310         found_masses = found_areas / assumed_AM # Using assumed A/M
    since A/M is a distribution and could get diff values.
311
312         L_c = np.concatenate([prelim_L_c, found_L_c])
313         areas = np.concatenate([prelim_areas, found_areas])
314         masses = np.concatenate([prelim_masses, found_masses])
315         AM = np.concatenate([prelim_AM, assumed_AM])
316
317         return L_c, areas, masses, AM
318     else:
319         if is_catastrophic == True:
320             # Put the rest of the mass in many fragments in last bin
321             unaccounted_mass = (m_target + m_projectile) - np.sum(
    prelim_masses)
322             deposit_bin = (np.geomspace(0.001, 1, 100)[-1] + np.
    geomspace(0.001, 1, 100)[-2])/2
323             n_large_deb = np.random.randint(15, 50) # Pick 2-8 pieces
    of deb > 1m to spread out the rest of the mass
324
325             # Create mass range, will use `n_large_deb` to split into
    sections, using 10**-4 to enure endpoints are not included
```

```
326             mass_range = np.linspace(10**-4, (unaccounted_mass -
    10**-4), 10**4)
327             ranges = np.sort(np.random.choice(mass_range, n_large_deb -
     1, replace=False))
328             ranges = np.concatenate([[0],ranges,[unaccounted_mass]])
329 
330             # Adding zeros for subtraction to have correct number of
    dim.
331             found_masses = np.concatenate((ranges[1:],np.zeros(1))) -
    ranges
332             found_masses = np.resize(found_masses, found_masses.size -
    1)
333 
334             found_L_c = np.ones_like(found_masses) * deposit_bin
335             found_areas = avg_area(found_L_c)
336             found_AM = found_areas / found_masses
337 
338             L_c = np.concatenate([prelim_L_c, found_L_c])
339             areas = np.concatenate([prelim_areas, found_areas])
340             masses = np.concatenate([prelim_masses, found_masses])
341             AM = np.concatenate([prelim_AM, found_AM])
342 
343             return L_c, areas, masses, AM
344 
345         else:
346             # Is a non catastrophic collision, Deposit remaining mass
    in 1 large piece of deb
347             unaccounted_mass = np.array([(m_target + m_projectile) - np
    .sum(prelim_masses)])
348 
349             # For L_c > 1, A/M Distribution is basically deterministic,
     therefore will just use avg value, can get using np.inf
350             assumed_AM_factory = make_mean_AM(debris_type)
351             assumed_len = np.ones(unaccounted_mass.shape)
352             assumed_AM = assumed_AM_factory(assumed_len)
353 
354             # Each mean has two possible values, randomly pick one of
    them for each piece of deb
355             AM_choices = np.random.choice([0,1], len(unaccounted_mass),
     replace=True)
```

```
356             assumed_AM = 10**np.array([assumed_AM[AM_choices[i], i] for
        i in range(assumed_AM.shape[1])])
357
358             # mass * AM = A(L_c), therefore can reverse Area function
       for L_c
359             area = unaccounted_mass * assumed_AM
360             found_L_c = np.sort((area / 0.556945)**(1/2.0047077)) #
       Inversing the Area function defined above
361             found_lambda_c = np.log10(found_L_c)
362             found_areas = avg_area(found_L_c)
363
364             L_c = np.concatenate([prelim_L_c, found_L_c])
365             areas = np.concatenate([prelim_areas, found_areas])
366             masses = np.concatenate([prelim_masses, unaccounted_mass])
367             AM = np.concatenate([prelim_AM, assumed_AM])
368
369             return L_c, areas, masses, AM
```

C.2 Coordinate Transforms

Listing 2: State representation implementation

```
1  import numpy as np
2  from numpy import cross
3  from numba import njit as jit, prange
4  from numpy.core.umath import cos, sin, sqrt
5  from numpy.linalg import norm
6
7  """
8  Converting from Keplerian to Cartesian
9  ---------------------------------------
10 Helpful links:
11     https://downloads.rene-schwarz.com/download/M001-
       Keplerian_Orbit_Elements_to_Cartesian_State_Vectors.pdf
12     https://gitlab.eng.auburn.edu/evanhorn/orbital-mechanics/blob/
       a850737fcf4c43e295e79decf2a3a88acbbba451/Homework1/kepler.py
13
14 Notes: Code was modified from Poliastro source, elements.py
```

```
15 """
16
17 mu = 398600.4418 #km^3s^-2
18
19 @jit
20 def rotation_matrix(angle, axis):
21
22     c = cos(angle)
23     s = sin(angle)
24
25     if axis == 0:
26         return np.array([[1.0, 0.0, 0.0], [0.0, c, -s], [0.0, s, c]])
27     elif axis == 1:
28         return np.array([[c, 0.0, s], [0.0, 1.0, 0.0], [s, 0.0, c]])
29     elif axis == 2:
30         return np.array([[c, -s, 0.0], [s, c, 0.0], [0.0, 0.0, 1.0]])
31     else:
32         raise ValueError("Invalid axis: must be one of 'x', 'y' or 'z'"
       )
33
34 @jit
35 def rv_pqw(k, p, ecc, nu):
36     pqw = np.array([[cos(nu), sin(nu), 0], [-sin(nu), ecc + cos(nu),
       0]]) * np.array(
37         [[p / (1 + ecc * cos(nu))], [sqrt(k / p)]]
38     )
39     return pqw
40
41 @jit
42 def coe_rotation_matrix(inc, raan, argp):
43     """Create a rotation matrix for coe transformation"""
44     r = rotation_matrix(raan, 2)
45     r = r @ rotation_matrix(inc, 0)
46     r = r @ rotation_matrix(argp, 2)
47     return r
48
49 @jit
50 def coe2rv(k, p, ecc, inc, raan, argp, nu):
51
52     pqw = rv_pqw(k, p, ecc, nu)
```

```
    r, v = rv_pqw(k, p, ecc, nu)
    rm = coe_rotation_matrix(inc, raan, argp)
    ijk = pqw @ rm.T

    return ijk

# ks = np.array([a, e_mag, i, Omega, omega, M, nu, p_semi, T, E])
@jit(parallel=True)
def coe2rv_many_new(state, mu=mu):
    inc = np.deg2rad(state[2, :])
    raan = np.deg2rad(state[3, :])
    argp = np.deg2rad(state[4, :])
    nu = np.deg2rad(state[6, :])
    p = state[7, :]
    ecc = state[1, :]

    n = nu.shape[0]
    rr = np.zeros((n, 3), dtype=np.float64)
    vv = np.zeros((n, 3), dtype=np.float64)

    for i in prange(n):
        rr[i, :], vv[i, :] = (coe2rv(mu, p[i], ecc[i], inc[i], raan[i],
     argp[i], nu[i]))

    return rr, vv

@jit(parallel=True)
def coe2rv_many(k, p, ecc, inc, raan, argp, nu):
    inc = np.deg2rad(inc)
    raan = np.deg2rad(raan)
    argp = np.deg2rad(argp)
    nu = np.deg2rad(nu)

    n = nu.shape[0]
    rr = np.zeros((n, 3), dtype=np.float64)
    vv = np.zeros((n, 3), dtype=np.float64)

    for i in prange(n):
        rr[i, :], vv[i, :] = (coe2rv(k, p[i], ecc[i], inc[i], raan[i],
    argp[i], nu[i]))
```

```
91
92      return rr, vv
93
94  """
95  Converting from Cartesian to Keplerian
96  ---------------------------------------
97  """
98
99  def rv2coe(r, v, mu):
100     ''' Converts a position, `r`, and a velocity, `v` to the set of
        keplerian elements.'''
101     """
102     Parameters
103     ----------
104     r: array (3, n)
105        Position of the body in 3 dim. Measured using center of Earth as
         origin. (m)
106     v: array (3, n)
107        Velocity of the body in 3 dim relative to Earth. (m / s)
108
109     Returns
110     -------
111     ks: array (9, n)
112         An array containing all of the keplerian elements + extra
        useful info.
113         a: Float
114         e: Float
115         i: Float
116         Omega: Float
117         omega: Float
118         nu: Float
119         p_semi: Float
120         T: Float
121     """
122
123     def testAngle(test, angle):
124         """Checks test for sign and returns corrected angle"""
125         angle *= 180./np.pi
126         I = test < 0
127         angle[I] = 360. - angle[I]
```

```
128         return angle
129
130     r_hat = np.divide(r, norm(r, axis=1)[:, None])
131
132     # Orbital momentum vector, p
133     p = np.cross(r, v)
134
135     # Eccentricty vector, e, and magnitude, e_mag (used freq)
136     e = (np.cross(v, p) / mu) - r_hat
137     e_mag = norm(e, axis = 1)
138
139     # Longitude of the ascending node, Omega
140     Omega_hat = np.cross(np.array([0, 0, 1])[None, :], p)
141     Omega = np.arccos(Omega_hat[:,0]/norm(Omega_hat, axis=1))
142     Omega = testAngle(Omega_hat[:, 1], Omega)
143
144     # Argument of periapsis, omega
145     omega = np.arccos(np.sum(Omega_hat*e, axis=1) / (norm(Omega_hat,
        axis=1)*norm(e, axis=1)))
146     B = e[:,2] < 0
147     omega[B] = 2*np.pi - omega[B]
148     omega *= 180. / np.pi
149
150     # True Anomaly, nu
151     nu = np.arccos( np.sum(e*r, axis=1) / (norm(e, axis=1) * norm(r,
        axis=1)))
152     B = np.sum(r*v, axis=1)<0
153     nu[B] = 2*np.pi - nu[B]
154     nu *= 180. / np.pi
155
156     # Inclination, i
157     i = np.arccos(p[:, 2] / norm(p, axis=1))*180./np.pi
158
159     # Eccentric anomaly, E
160     E = 2*np.arctan(np.tan(np.deg2rad(nu)/2) / np.sqrt((1 + e_mag)/(1 -
         e_mag)))
161
162     # Mean anomaly, M
163     M = np.mod(E - e_mag * np.sin(E), 2*np.pi)
164     M *= 180./np.pi
```

```
165
166     # Semi-Major axis, a
167     R = norm(r, axis =1)
168     V = norm(v, axis =1)
169     a = 1/((2 / R) - (V*V / mu))
170
171     # Semi-parmeter, p_semi
172     p_semi = norm(p, axis=1)**2 / mu
173
174     # Orbital period
175     T = 2*np.pi * np.sqrt(a**3 / mu)
176
177     # Keplerian State + Extra info
178     ks = np.array([a, e_mag, i, Omega, omega, M, nu, p_semi, T, E])
179
180     return ks
```

C.3 Orbit Propagation

Listing 3: Perturbations and orbit propagation implementation

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from numba import njit as jit, prange
4  from numpy import pi, sin, cos, sqrt
5  from scipy import integrate
6  from scipy.special import iv
7
8  # User defined libearayr
9  import planetary_data as pd
10 import CoordTransforms as ct
11 import Aerodynamics as aero
12
13 def null_perts():
14     return {
15         'J2': False,
16         'aero': False,
17         'moon_grav': False,
```

```
        'solar_grav': False
    }

class OrbitPropagator:

    def __init__(self, states0, A, M, tspan, dt, rv=False, cb=pd.earth,
     perts=null_perts()):

        # Need to add support for initializing with radius and velocity
        if rv:
            self.states = 0

        else:
            self.states = states0

        # Setting the areas and masses
        self.A = A
        self.M = M

        # Integration information
        self.tspan = tspan
        self.dt = dt

        # Central body properties
        self.cb = cb

        # Defining perturbations being considered
        self.perts = perts

        # Defining constants for aerodynamic drag
        if self.perts['aero']:
            self.K_a = np.matrix([[1, 0, 0, 0 ,0 ,0, 0],
                   [0, 2, 0, 0, 0, 0, 0],
                   [3/4, 0, 3/4, 0, 0, 0, 0],
                   [0, 3/4, 0, 1/4, 0, 0, 0],
                   [21/64, 0, 28/64, 0, 7/64, 0, 0],
                   [0, 30/64, 0, 15/64, 0, 3/64, 0]])

            self.K_e = np.matrix([[0, 1, 0, 0, 0, 0, 0],
                   [1/2, 0 , 1/2, 0, 0, 0, 0],
```

```
                    [0, -5/8, 0, 1/8, 0, 0, 0],
                    [-5/16, 0, -4/16, 0, 1/16, 0, 0],
                    [0, -18/128, 0, -1/128, 0, 3/128, 0],
                    [-18/256, 0, -19/256, 0, 2/256, 0, 3/256]])

    def cartesian_representation(self):
    # Returns the cartesian state representation of states for vis.
    purposes
        N_t = self.states.shape[0]
        N_frag = self.states.shape[2]
        cartesian_states = np.empty(shape=(N_t, 2, N_frag, 3))

        for i in prange(self.states.shape[0]):
            cartesian_states[i, :, :] = ct.coe2rv_many_new(self.states[
    i, :, :])

        return cartesian_states


    def diffy_q(self, t, state):
        e, a, i, Omega, omega = state.reshape(5, len(self.A))
        N_f = len(self.A)

        # Central body information
        mu     = self.cb['mu']
        radius = self.cb['radius'] #[m]
        J2 = self.cb['J2']

        # Local variables
        delta_e = np.zeros_like(e)
        delta_a = np.zeros_like(a)
        delta_i = np.zeros_like(i)
        delta_Omega = np.zeros_like(Omega)
        delta_omega = np.zeros_like(omega)


        # Current orbital information
        peri = a * (1 - e) #[m]
        p    = a * (1 - e**2) #[m] (Semi parameter)
        n    = np.sqrt(mu / a**3) # (Mea motion)
```

```
95
96          ############### Drag effects ###############
97          if self.perts['aero']:
98              h_p = (peri - radius) #[m]
99              rho = aero.atmosphere_density(h_p/1e3) #[kg * m^-3]
100             H = aero.scale_height(h_p/1e3) * 1e3 #[m]
101
102             z = a*e / H
103             Cd = 0.7
104             tilt_factor =1
105             delta = Cd * (self.A[0] * tilt_factor) / self.M[0]
106
107             e_T = np.array([np.ones_like(e), e, e**2, e**3, e**4, e
    **5])
108             I_T = np.array([ iv(i, z) for i in range(7)])
109             k_a = delta * np.sqrt(mu * a) * rho
110             k_e = k_a / a
111
112             delta_e = np.zeros_like(e)
113             delta_a = np.zeros_like(a)
114
115             # CASE e < 0.001
116             delta_e = np.zeros_like(e)
117             delta_a = -k_a
118
119             # CASE e>= 0.001
120             I = e>= 0.001
121             trunc_err_a = a[I]**2 * rho[I] * np.exp(-z[I]) * iv(0, z[I
    ]) * e[I]**6
122             trunc_err_e = a[I] * rho[I] * np.exp(-z[I]) * iv(1, z[I]) *
     e[I]**6
123
124             transform_e = e_T.T.dot(self.K_e) * I_T
125             coef_e = np.array([ transform_e[i,i] for i in range(N_f)])[
    I]
126
127             transform_a = e_T.T.dot(self.K_a) * I_T
128             coef_a = np.array([ transform_a[i,i] for i in range(N_f)])[
    I]
129
```

```
            delta_e[I] = -k_e[I] * np.exp(-z[I]) * (coef_e + trunc_err_e)
            delta_a[I] = -k_a[I] * np.exp(-z[I]) * (coef_a + trunc_err_a)

            delta_e[np.isnan(delta_e)] = 0
            delta_a[np.isnan(delta_a)] = 0

            # Deorbit check
            J = h_p < 100*1e3
            delta_a[J] = 0
            delta_e[J] = 0

        ###############  J2 effects  ###############
        if self.perts['J2']:
            base = (3/2) * self.cb['J2'] * (radius**2/p**2) * n
            i = np.deg2rad(i)
            delta_omega = base * (2 - (5/2)*np.sin(i)**2)
            delta_Omega = -base * np.cos(i)
            delta_omega = np.rad2deg(delta_omega) % 360
            delta_Omega = np.rad2deg(delta_Omega) % 360

        return np.concatenate((delta_e, delta_a, delta_i, delta_Omega, delta_omega))

    # Performing a regular propagation, i.e. w/ perturbations
    def propagate_perturbations(self):

        # Initial states
        a0, e0, i0, Omega0, omega0 = self.states[-1, :5, :]
        y0  = np.concatenate((e0, a0, i0, Omega0, omega0))

        # Propagation time
        T_avg = np.mean(self.states[-1, 8, :])
        times = np.arange(self.tspan[0], self.tspan[-1], self.dt)
        output = integrate.solve_ivp(self.diffy_q, self.tspan, y0, t_eval = times)

        # Unpacking output (Need to drop first timestep as sudden introduction of drag causes discontinuities)
```

```
165         N_f = len(self.A)
166         de = output.y[0:N_f, 1:]
167         da = output.y[N_f:2*N_f, 1:]
168         di = output.y[2*N_f:3*N_f, 1:]
169         dOmega = output.y[3*N_f:4*N_f, 1:]
170         domega = output.y[4*N_f:, 1:]
171         dnu = np.random.uniform(low=0., high=360., size=domega.shape)
172         dp = da * (1 - de**2)
173 
174         # Results
175         return de, da, di, dOmega, domega, dnu, dp
176 
177     # Performing a Keplerian propagation, i.e. w/o perturbations
178     def propagate_orbit(self):
179 
180         times      = np.arange(self.tspan[0], self.tspan[-1], self.dt)
181 
182         # Mean anomaly rate of change
183         M_dt       = sqrt(self.cb['mu']/self.states[0, :]**3)
184 
185         Nd         = len(M_dt)
186         Nt         = len(times)
187 
188         # Mean anomaly over time
189         M_t        = np.deg2rad(self.states[5, :, None]) + M_dt[:, None
    ] * times[None, :]
190         M_t        = np.rad2deg(np.mod(M_t, 2*pi))
191 
192         # Eccentric anomaly over time. Note need to use E_t in rad,
    thus convert to deg after using it in
193         # x1 and x2
194         E_t        = np.empty(shape=(Nd, Nt), dtype=np.float32)
195         E_t        = M2E(self.states[1], np.deg2rad(M_t))
196 
197         x1         = sqrt(1 + self.states[1, :])[:, None] * sin(E_t /
    2)
198         x2         = sqrt(1 - self.states[1, :])[:, None] * cos(E_t /
    2)
199         E_t        = np.rad2deg(E_t)
200 
```

```
201         # True anomaly over time
202         nu_t       = (2*np.arctan2(x1, x2) % (2*pi))
203         nu_t       = np.rad2deg(nu_t).T
204 
205         n_times    = nu_t.shape[0]
206         states     = np.empty(shape = (n_times, self.states.shape[0],
        self.states.shape[1]))
207 
208         for i in prange(n_times):
209             state = self.states.copy()
210             state[6, :] = nu_t[i, :]
211             states[i] = state
212 
213         # Update internal states
214         self.states = states
215 
216 
217 
218 # Modified from OrbitalPy.utilities
219 @jit(parallel=True, fastmath=True)
220 def M2E(e_deb, M_t, tolerance=1e-14):
221 #Convert mean anomaly to eccentric anomaly.
222 #Implemented from [A Practical Method for Solving the Kepler Equation
        ][1]
223 #by Marc A. Murison from the U.S. Naval Observatory
224 #[1]: http://murison.alpheratz.net/dynamics/twobody/
        KeplerIterations_summary.pdf
225     n_deb = M_t.shape[0]
226     n_times = M_t.shape[1]
227 
228     E_t = np.empty_like(M_t)
229 
230     for i in prange(n_deb):
231         e = e_deb[i]
232         for j in prange(n_times):
233             M = M_t[i, j]
234 
235             MAX_ITERATIONS = 100
236             Mnorm = np.mod(M, 2 * pi)
```

```
            E0 = M + (-1 / 2 * e ** 3 + e + (e ** 2 + 3 / 2 * cos(M) *
    e ** 3) * cos(M)) * sin(M)
            dE = tolerance + 1
            count = 0
            while dE > tolerance:
                t1 = cos(E0)
                t2 = -1 + e * t1
                t3 = sin(E0)
                t4 = e * t3
                t5 = -E0 + t4 + Mnorm
                t6 = t5 / (1 / 2 * t5 * t4 / t2 + t2)
                E = E0 - t5 / ((1 / 2 * t3 - 1 / 6 * t1 * t6) * e * t6
    + t2)
                dE = np.abs(E - E0)
                E0 = E
                count += 1
                if count == MAX_ITERATIONS:
                    print('Did not converge, increase number of
    iterations')
            E_t[i, j] = E
    return E_t
```