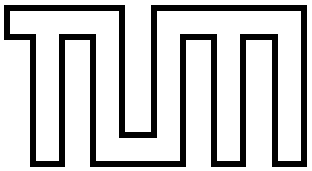


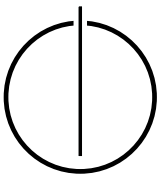
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

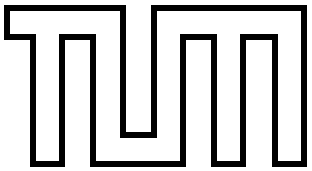
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Realistic Atmospheric Reentry Modelling and Refinement Extending ESA's Flight Dynamics Library

Jonas Schuhmacher





SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

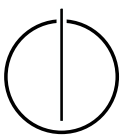
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Realistic Atmospheric Reentry Modelling and Refinement Extending ESA's Flight Dynamics Library

Realistische Atmosphärische Wiedereintrittsmodellierung und Verfeinerung zur Erweiterung der Flugdynamikbibliothek der ESA

Author:	Jonas Schuhmacher
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Fabio Gratl, M. Sc. and Dr.-Ing. Jan Siminski
Date:	15.02.2024



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.02.2024 Jonas Schuhmacher

Acknowledgements

Multiple people have made this thesis possible with their professional and motivational support. I thank my friends for cooling my sometimes melting brain during summer in relaxed Biergarten atmosphere. Moreover, I thank my parents for backing me during my study program and my mother for proofreading this thesis. Further, I thank my friends and colleagues, who I met during my internship in ESA's Space Debris Office, for hosting me and giving valuable advice on professional, culinary, and basically every topic at all. Also, many thanks for the incredible adventures in Darmstadt and its surroundings, including unforgettable Schnitzel(croissants) and hikes. Special thanks to Pablo, who wasn't directly involved in this thesis but indirectly due to monthly meetings. Without him and Fabio, I probably wouldn't have come so far. Foremost, I greatly thank my two advisors, Jan and Fabio, who both made this thesis possible. They provided me with the best thinkable feedback day and night (Trust me, superlatives are justified here). They suffered from my continuous 90%-finished syndrome. And they gave the right hints to guide this thesis in a successful direction.

To finish with a slightly modified quote from James Carville: It's the people, stupid!

Abstract

The number of objects in a geocentric orbit is constantly increasing, and so is the number of objects reentering. While tracking and collision avoidance are crucial in guaranteeing space safety, another significant task is correctly predicting satellites' reentries and overall decay to anticipate potential impacts and intersections with spacecraft trajectories - this knowledge about the state of the reentries strongly correlated with the state of the atmosphere. The European Space Agency (ESA) developed the flight dynamics library `godot` to optimize and propagate orbits. In this work, we redesign `godot`'s atmospheric core component using state-of-the-art software engineering in modern C++ while interfacing a dynamic plugin that provides the user with an array of new and legacy atmospheric models, such as Jacchia-Bowman 2008 (JB2008), MSISv2, or DTM2020. In the second step, we build upon `godot` by implementing an atmospheric optimization framework. The framework includes an extensive toolchain for the purpose of atmosphere and reentry optimization consolidating publicly available data sources like Space-Track, Discos, and the HASDM (High Accuracy Satellite Drag Model) SET Database to perform fully automatic reentry prediction for arbitrary satellites by optimizing their ballistic coefficient and the diurnal density coefficients of JB2008. The optimization framework allows a high degree of freedom in assembling and combining arbitrary optimization building blocks with one another. With a modified version of Picone et al.'s TLE density derivation algorithm to optimize ballistic coefficients, we can achieve median reentry prediction errors as low as $\sim 9\%$ in 2019 for 25 satellites, comparable to the current state-of-the-art errors with only publicly available resources and within runtimes of minutes by exploiting parallelization. The solution will be utilized by the ESA's Space Debris Office as a second toolchain for assessing reentries.

Zusammenfassung

Die Zahl der Objekte in einer geozentrischen Umlaufbahn nimmt ständig zu, ebenso wie die Zahl der Objekte, die wieder eintreten. Während Tracking und Kollisionsvermeidung für die Gewährleistung der Sicherheit im Weltraum von entscheidender Bedeutung sind, besteht eine weitere wichtige Aufgabe in der korrekten Vorhersage des Wiedereintritts von Satelliten und des Bahnabfalls. Letzteres dient dazu um potenzielle Kollisionen und Überschneidungen mit den Flugbahnen von Raumfahrzeugen zu antizipieren. Dieses Wissen korreliert stark mit dem Zustand der Atmosphäre. Die Europäische Weltraumorganisation (ESA) hat die Flugdynamik-Bibliothek `godot` entwickelt, um Trajektorien zu optimieren und zu propagieren. In dieser Arbeit gestalten wir `godot`'s atmosphärische Kernkomponente von Grund auf neu unter der Verwendung von State-of-the-Art Software-Engineering Best Practices in C++. Das neu geschaffene Plugin stellt dem Benutzer eine Reihe von neuen und alten atmosphärischen Modellen zur Verfügung, wie z.B. Jacchia-Bowman 2008 (JB2008), MSISv2 oder DTM2020. Im zweiten Schritt bauen wir auf `godot` auf, indem wir ein Framework zur atmosphärischen Optimierung implementieren. Das Framework umfasst eine umfangreiche Toolchain für die Optimierung der Atmosphäre und des Wiedereintritts, die öffentlich verfügbare Datenquellen wie Space-Track, Discos und die HASDM (High Accuracy Satellite Drag Model) SET Datenbank zusammenführt, um eine vollautomatische Wiedereintrittsvorhersage für beliebige Satelliten durch Optimierung ihres ballistischen Koeffizienten und der täglichen Dichtekoeffizienten von JB2008 durchzuführen. Das Optimierungsframework erlaubt einen hohen Freiheitsgrad an Konfigurierbarkeit durch die beliebige Kombination von Optimierungsbausteinen untereinander. Mit einer modifizierten Version des TLE-Dichte-Ableitungsalgorithmus von Picone et al. zur Optimierung der ballistischen Koeffizienten können wir im Jahr 2019 für 25 Satelliten mediane Wiedereintrittsvorhersagefehler von nur $\sim 9\%$ erreichen, welche vergleichbar sind mit aktuellen State-of-the-Art-Fehlern. Wir nutzen dabei nur öffentlich verfügbare Ressourcen und erreichen die Ergebnisse innerhalb einer Laufzeit von Minuten durch Ausnutzung von Parallelisierung. Die Lösung wird vom Space Debris Office der ESA als zweite Toolchain für die Bewertung von Wiedereintritten genutzt werden.

Contents

Part I.

Introduction and Background

1. Introduction

"And remember, we have nothing to fear, but the sky falling on our heads!" is a famous quote from the books of *Asterix and Obelix* by Albert Uderzo & René Goscinny. The sky usually does not fall to Earth's ground. In contrast, satellites and debris in orbit around Earth usually tend to come back at some point.

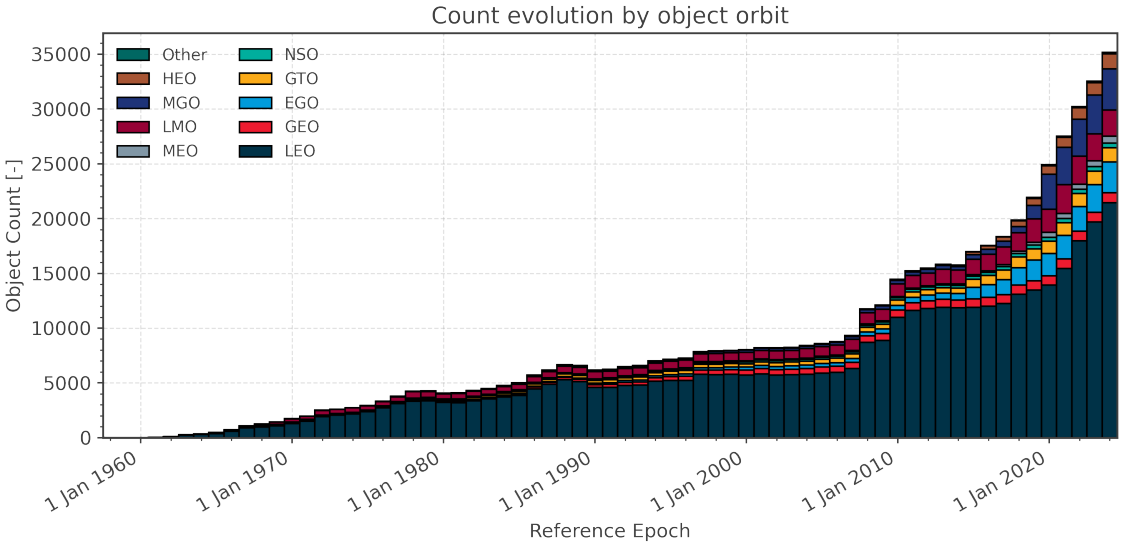


Figure 1.1.: Chart displaying the amount of objects per orbit type [1]

Figure 1.1 shows the number of objects around Earth per year and categorized by the orbit. The amount of tracked objects has increased visibly exponentially with most objects situated in Low Earth Orbit (LEO). Given Figure 1.1, it is not surprising that the amount of reentries has also increased rapidly shown by Figure 1.2

Tracking the reentry data of these satellites is usually tightly coupled with overall orbital lifetime assessment. Objects less massive than $500\,kg$ usually need to be tracked on their progressive way to reentry due to the potential catastrophic intersection potential with satellites. Objects in larger size regimes are even more crucial to observe due to the potential of surviving reentry in parts and impacting the ground - potentially harming infrastructure or even humans. [2]

The main driver for reentry in the upper atmosphere, consisting of the Thermosphere ranging from $\sim 120\,km$ to $\sim 600\,km$[1] and Exosphere, is the atmospheric drag. It slowly decelerates the spacecraft, causing its orbit to decay while the spacecraft travel through Thermosphere and lower Exosphere. Figure 1.1 shows that the majority of objects is in

[1] `https://www.noaa.gov/jetstream/atmosphere/layers-of-atmosphere`, last accessed: 11.02.2024

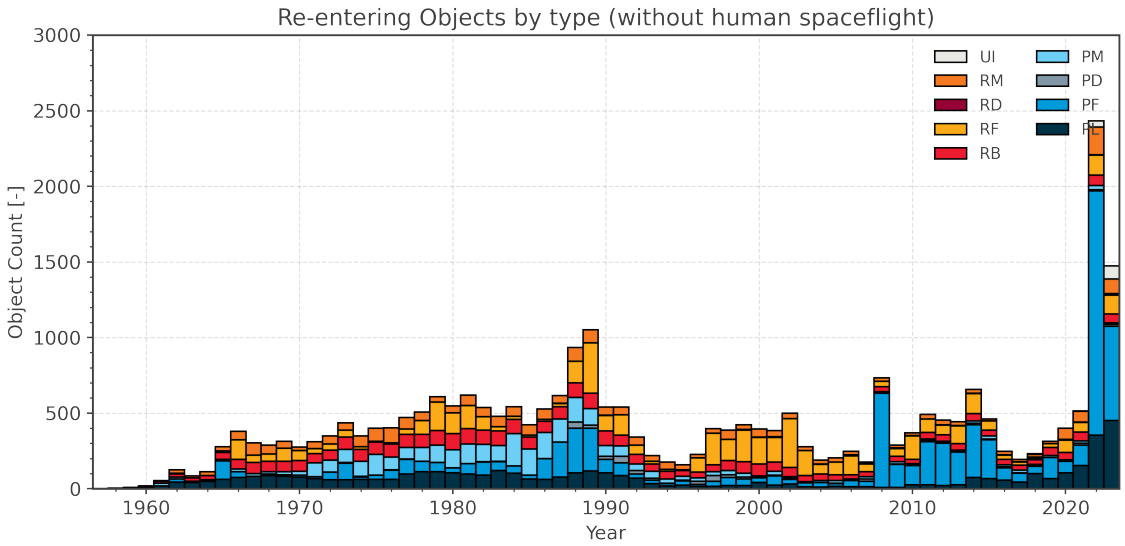


Figure 1.2.: Chart displaying the amount of objects reentering per year and their respective classification (P* = Payload Related, R* = Rocket Body Related, *F or *D = debris) [1]

LEO. A spacecraft is in LEO if its altitude is $\in [0; 2000]\, km$ [1]. Hence, most spacecraft around Earth are affected by the decelerating force of atmospheric drag given Figure 1.1. For active missions, this usually implies station-keeping measurements with their propulsive systems. If the mission ends, the current policy is to deorbit within 25 years [1].

This work is not about the lifetime assessment in the range of years. It is about reentry prediction when the reentry is foreseeable within months, with a significant focus on modeling atmospheric drag. This work is divided into three parts. Part I explores the background of atmospheric models and the art of reentry prediction. The main focus lies on semi-empirical models capable of predicting the atmospheric density at a given position and time. We investigate three prominent density model families: the DTM, Jacchia, and MSIS models. Further, several approaches to optimize the density output are examined. Part II is about the integration of twelve density models from these families and three atmospheric wind models into the flight dynamics library of the European Space Agency (ESA): `godot`. All of these models are interfaced into `godot` in C++17 with a uniform interface facilitating the use and abstracting of tedious set-up processes like collecting and accumulating solar and geomagnetic activity data. A comparison of all models compared to the High Accuracy Satellite Drag Model (HASDM) density values is presented. HASDM is sometimes considered the gold standard of atmospheric density, as its coefficients are updated on a daily basis. The model's details are unavailable to the scientific community due to its military roots, but its density output has been released for the years 2000 - 2019 [3]. Part III will assemble the previously implemented atmosphere models with a modern framework for reentry prediction in Python. The framework can take an arbitrary amount of input satellites and automatically consolidate publicly available data. We present three different approaches to optimize the ballistic coefficient and the diurnal density coefficients of the Jacchia-Bowman 2008 (JB2008)

density model. These approaches include the shooting method [4] and density derivation from TLE data. Moreover, the quality of the implementation is the focus of this work and not only an accessory. Hence, this work's major contributions are the extension of ESA's flight dynamics library and a universal framework for atmospheric reentry optimization to be used as a second reference of reentry predictions by ESA's Space Debris Office.

2. Theoretical Background

This chapter discusses the foundation of modeling and representing satellite reentries. It starts by examining how one can model the atmosphere as a whole for the purpose of satellite reentry and provides a detailed look at some exemplary models in this area in Section 2.1. Next, the chapter takes in Section 2.2 a deeper look at how one can model the satellite itself in orbit around a target body. Finally, Section 2.4 will assemble the theory and define how a satellite reentry can be modeled using the previously gained insights.

2.1. Atmospheric Modeling

By approximation, the atmosphere of the Earth follows the ideal gas law, as given by Equation 2.1 with pressure p, density ρ, temperature T, the universal gas constant R, and M as the molecular weight of the gas:

$$\frac{p}{\rho} = \frac{R}{M}T \tag{2.1}$$

The atmosphere is an hydrostatic equilibrium. This phenomena can be described as a balance between the pressure that pushes gas out into space and gravity that pulls it towards the Earth's surface. The underlying Equation 2.2 establishes a relationship between the decrease in pressure p with height h, the density ρ, and the gravitational acceleration g.

$$\frac{dp}{dz} = -\rho(z)g(h) \tag{2.2}$$

Both Equations 2.1–2.2 in relation to each other yields the formulation given by Equation 2.3.

$$\frac{d\rho}{\rho} + \frac{dT}{T} = \frac{M}{R} \cdot \frac{g(h) \cdot dh}{T(h)} \tag{2.3}$$

The problem lies with the factors M and $T(h)$. The atmosphere is considered homogeneous up to an altitude of approximately $120\,km$, where the temperature and air composition remain constant. This region is referred to as the homosphere and can be approximated using the barometric formula in Equation 2.4.

$$\rho = \rho_0 \cdot \exp\left(-\frac{h}{H}\right) \text{ with } \rho_0 = 1.752\,\frac{kg}{m^3} \text{ and } H = 6.7\,km \tag{2.4}$$

However, the thermosphere and exosphere (i.e., the atmospheric layers) above $\sim 120\,km$ have a non-uniform temperature and molecule distribution and are therefore referred to as the heterosphere. Figure 2.1 shows the variation of temperature and density given the

day-night cycle, as well as high and low solar activity. It is computed using the NRLMSIS-00 (MSIS00) empirical density model. These layers require more complex methods, which are the subject of this section. [5, 6]

The section continues listing the atmosphere models incorporated into `godot` throughout the thesis while classifying them, describing the origins and phenomena behind them, and finally distinguishing the three models' families. The abbreviations introduced for the individual atmosphere models are consistent with the implementations which are presented later.

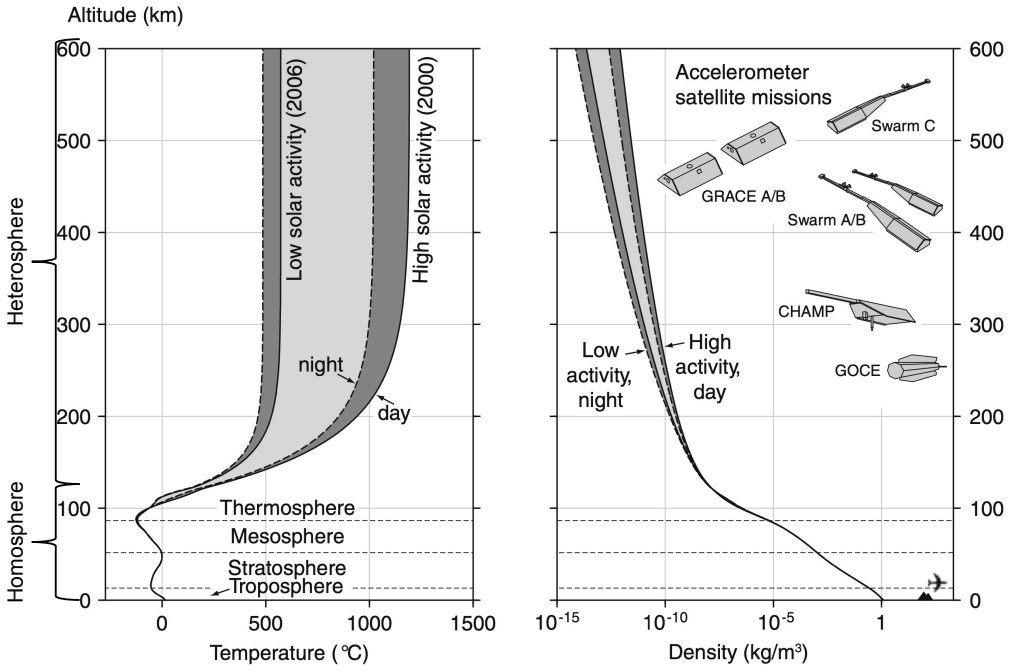


Figure 2.1.: Altitude profiles of atmospheric temperature (left) and density (right), according to the MSIS00 model, evaluated for 18:00 on July 15 2000 and 2006, over Delft, The Netherlands; adapted from [5]

2.1.1. Classification of Atmospheric Models

The heterosphere is challenging to model. This section introduces the available approaches, while the subsequent Subsection 2.1.2 introduces the effects of interest to be modeled. Overall, the primary distinction is commonly made between physical and empirical models. The former are based on the elementary physical equations governing the thermosphere, while the latter are parameterized mathematical formulations that model observational data. Nevertheless, both models usually also interoperate elements from the opposing category. Physical models require the empirical parameters to specify boundary conditions. Empirical models often include physical constraints to improve their predictions. [7]

In brief, there also exists a variety of fusion models that combine physical and empirical models using a weighted combination of both. Fusion models are reported to improve overall accuracy in some cases, like solar minima, but an optimal solution for all cases does not yet

exist [8, 9].

"These [physical] models include so-called Thermosphere Global Circulation Models (TGCMs)" [5] and local high-resolution airflow models. They simulate the temperature, density, molecular composition, and horizontal winds regarding diverse energy inputs. The models' complexity and scope bound the obtainable accuracy, often accompanied by runtime constraints. The latter also restrain their use cases. As these models are usually computationally intensive to run due to the numerical methods (like finite differences) being involved, they are suitable for studying the internals of the atmosphere rather than in the context of routine applications in the context of orbit and reentry determination [5, 10]. Next to runtime constraints, these models also lack the capabilities for short-term forecasting of atmospheric density because the thermosphere is mostly affected by external drivers like solar and geomagnetic activity - thus the boundary conditions - and has a relatively short internal memory [7]. Hence, their precision is also limited by the forecast of these drivers, which holds for empirical models as well. Overall, both model categories produce density forecasts of similar accuracy [7, 9].

The empirical models form the foundation of this work as they provide predictive capabilities and are cheap to-evaluate from a computational point of view. These models are created using observational data and typically consist of simple functions that attempt to fit significant atmospheric variations in density, temperature, and sometimes single species densities (such as molecular oxygen or hydrogen density). The observational data utilized in the fitting process comes from multiple sources. Figure 2.1 shows several satellite missions equipped with accelerometers. Given the drag acceleration affecting the spacecraft, they allow the reverse engineering of atmospheric density. This technique usually has inevitable errors due to measurement device calibration and missing data, the latter making it difficult to accurately distinguish the aerodynamic force of drag from, e.g., radiation pressure or atmospheric winds. One issue is presented in Subsection 2.4.1. The optimization of the density is conditioned by the knowledge of other parameters involved in aerodynamic drag, which are often not necessarily known beforehand and an optimization problem by themselves. Another issue is given by Chapter 7, here we show for the year 2019 that the accuracy of models that were constructed beforehand degrades. The density models are usually biased toward the data utilized for fitting, but 2019 has not been part of the fitting interval for some of the examined models. The current density sets often originate from the accelerometer data of dedicated missions like *CHAMP*, *GRACE*, *GOCE*, *Swarm A, B, C*. Of course, other sources than accelerometers are possible as well. The radar tracking of satellites allows deducing the atmospheric drag similarly (but less precisely) using the standard TLE data, containing the orbital state of satellites (see Subsection 4.1.1). Satellites with mass spectrometers like *Atmospheric Explorer* allow measuring the molecular species densities. And, Satellites with ultraviolet imaging capabilities allow for deriving temperature and composition profiles in the upper atmosphere. Subsection 2.1.2 investigates the phenomena involved in atmospheric variations. Subsection 2.1.3 explains how the phenomena are modeled using proxy indices for representing atmospheric behavior. Finally, Sections 2.1.4 to 2.1.6 introduces the implemented models and most important model families. [5, 10, 11]

2.1.2. Phenomena affecting the Thermospheric Density

The density in the thermosphere is subject to a great variety of influences, changing its composition in vertical and horizontal cross-sections. Figure 2.1 shows, for example, the vertical density variation at a **fixed point in time and a fixed location**. Looking at it, one notices that overall, the density is decreasing with increasing altitude. However, the concrete values can vary dramatically by more than a magnitude depending on external influences like solar activity or day-night-cycle. Figure 2.2 shows the density for a **fixed location and a fixed altitude** given a specific date. It illustrates that the density is - again - variable by more than magnitude given the solar and geomagnetic activity over time. Lastly, Figure 2.4 and Figure 2.3 show the horizontal decomposition for a **fixed altitude and a fixed point in time**. It also illustrates the upper atmosphere's wind velocity, whose strength strongly depends on solar activity.

The following sections introduce the significant individual effects causing these deviations.

Diurnal and Seasonal Variations

The diurnal density variation (i.e., the day-to-night variation) results from the uneven heating of the Earth by the Sun. Figure 2.1 and Figure 2.2 display the day-to-night difference, respectively, by the dashed lines in the former and the grey band in the latter. Maximum density is reached around two hours after noon, while the minimum is approached during nighttime. The transition in between follows a smooth decrease. The diurnal density variation can be around a "factor of 5 at 500 km and higher [due to] heating at the bottom of the thermosphere generally" [5], causing the entire thermosphere above this layer to expand. The diurnal variation is also subject to seasonal changes because the changing Sun-Earth geometry causes the sub-solar point to move over the course of a year. [5]

Solar Activity Variation

The Sun also causes the second contribution to the atmospheric variations similar to the diurnal and seasonal variations. However, the root cause does not result from the revolution nor the rotation of the Earth, but rather from the processes in the Sun itself. In contrast to solar radiation in the visible spectrum, the extreme ultra violet (EUV) radiation and X-ray wavelengths are highly variable. The solar cycle spanning from solar minima to solar minima approximately covers 11 years, with the Sun's magnetic field changing its polarity at the solar maxima in between them. The solar cycle can be monitored by measuring the amount of sunspots. A sunspot indicates an active region. Thus, the peak number of solar spots happens at the maxima. As of writing, the current solar cycle 25 has begun in December 2019 "when the 13-month smoothed sunspot number fell to 1.8" [12]. The maxima is expected to happen in 2025. The EUV output is on the lower end at solar minima and is usually relatively stable. The opposite holds for the solar maxima when the Sun's emitation of EUV is highly variable and more potent following the Sun's periodic 27-day rotation. Overall, the density can alter by up to a factor of 20 at 500 km and higher, with higher variability in the outer layers of the atmosphere. [5]

Geomagnetic Activity Variation

Ejections of coronal mass and solar flares originating from the Sun launch significant quantities of charged particles into space. Earth's magnetic field deflects most of the particles when they arrive. However, a fraction of them may "enter through the so-called polar cusps and via so-called reconnection processes at the dayside magnetopause" [5]. These particles then cause geomagnetic storms, which are admirable by the appearance of the Aurora. The additional energy inside the polar thermosphere and ionosphere (i.e., the thermosphere and exosphere contain ionized atoms and free electrons, thereby the name) causes density variations of up to one order of magnitude. [5]

Semiannual Variation

The semiannual density variation is the fourth influence. In 1961, Paethold and Zschörner [13] first discovered that the thermospheric density has its maxima in the months around April and October and its minima around January and July [14]. However, the concrete amplitude and timing of this periodic variation differ annually. The recognized physical theory behind it was proposed by Fuller-Rowell [15] in 1998. "The global-scale, interhemispheric circulation at solstice acts like a huge turbulent eddy in mixing the major hemispheric species" [15], which leads to less increased molecular nitrogen and oxygen densities and reduced atomic oxygen. This effect leads to a lower density scale at a given height and compresses the atmosphere. Recent work by Jones et al. [16] supports the hypothesis, and Bowman et al. [17] also found a correlation to solar activity. "During solar maximum, the semiannual variation can be as small as 30% at $220\,km$, and as high as 250% near $800\,km$" [17]. In contrast, at solar minimum, the variation is only around 70% in $800\,km$.

2.1.3. Utilized Proxy Indices

Obviously, the day-to-night-time density variation can be simply modeled by using the local solar time at a given longitude. The previous Subsection 2.1.2 has shown that the main drivers for thermospheric temperature and density are the influx of heat through solar EUV radiation conditioned by solar activity and due to charged particles related to geomagnetic activity. However, modeling the density variations caused by geomagnetic storms and solar activity is more complex. In practice, one uses a so-called proxy index, which is easily measurable and shows a similar behavior to the property one desires to model.

Solar Activity Indices

Most of the atmosphere models use the $F_{10.7}$ solar flux index in order to model solar activity (if they model solar activity at all). $F_{10.7}$ abbreviates $10.7\,cm$ solar radio flux. Its physical units are solar flux units with $1\,sfu = 1 \times 10^{-22}\,Wm^{-2}Hz^{-1}$. The $F_{10.7}$ adopts values ranging from $70\,sfu$ during solar minima to around $370\,sfu$ during maxima [5]. Radio telescopes have measured the $F_{10.7}$ solar flux daily since 1947, with the earliest measurements dating back to the 1930s [18]. This long track of records is one of its most significant advantages. Earth's atmosphere does not absorb this solar flux originating from the solar transition region/ the cool corona, making the $F_{10.7}$ suitable measurable on the surface level. Additionally, this "dependence on few processes, combined with its localized

formation in the cool corona, i.e., a region that is closely coupled with magnetic structures responsible for creating the XUV-EUV irradiances, makes this a good generalized solar proxy for thermospheric heating" [19]. Also, given that EUV radiation can not be measured on the ground since it is absorbed in the atmosphere, causing heating, one seeks to model [18]. Overall, primarily due to its availability, the $F_{10.7}$ is nearly used widely across all empirical atmospheric density models, with different indices being used appearing some decades ago [18, 19]. In many models, the $F_{10.7}$ is utilized in an average form over the last three solar rotations spanning 81 days. It is usually denoted as $\bar{F}_{10.7}$

Thanks to satellites like Solar and Heliospheric Observatory (SOHO), Geostationary Operational Environmental Satellite (GOES), or Solar Radiation and Climate Experiment (SORCE) in orbit, it has become possible to measure the Lyman-α ($121\,nm \leq \lambda < 122nm$ with λ as the wavelength), EUV ($10\,nm \leq \lambda < 121nm$), coronal X-rays ($0.1\,nm \leq \lambda < 0.8\,nm$), soft X-ray ($0.1\,nm \leq \lambda < 10\,nm$), and far ultraviolet ($122\,nm \leq \lambda < 200\,nm$) radiation directly. All of them affect thermospheric heating - predominantly due to the EUV radiation - and originate from different parts of the Sun. Thus, this affects the density variation. This development led to the appearance of new indices capable of more directly representing the phenomena. These are also measured in solar flux units. The JB2008 density model employs these new indices [20]:

- $S_{10.7}$ representing EUV from the chromosphere of the Sun. It has been available daily since 1996.
- $M_{10.7}$ representing FUV from the photosphere and lower chromosphere. It has been available daily since 1991.
- $Y_{10.7}$ representing X-rays and UV from the chromosphere, transition region, and hot corona. It has been available daily since 1991.

For the precise construction details specification, it is referred to Tobiska et al. [19]. These indices solve the apparent issue that the $F_{10.7}$ has become a limiting factor for the obtainable accuracy in empirical atmosphere models [5].

Another approach is undertaken by de Wit and Bruinsma [21]. They replaced the $F_{10.7}$ solar flux by the $30\,cm$ solar radio flux, denoted with F_{30} for the Drag Temperature Model 2013 (DTM2013). The approach is grounded on an extensive database because other centimetric wavelengths - such as the F_{30} - have also been continuously monitored since the 1950s. The F_{30} solar flux contains "a relatively larger proportion of emissions coming from solar features such as plages, faculae and hot coronal loops, and consequently correlates better" [21] with the actual UV emission. For that purpose, they scale the F_{30} to the $F_{10.7}$ range to replace the latter seamlessly. This approach reduced the error of their density predictions [21].

Geomagnetic Activity Indices

Either the index K_p ("planetarische Kennziffer") or the a_p represents the geomagnetic activity in all empirical atmosphere models implemented throughout the thesis. The a_p index is a derivative of the K_p index. The relation between the two numbers is provided by Table 2.1. The K_p index introduced and standardized by Bartels et al. in 1949 is given in discrete steps and is not associated with any physical unit. It is used "to monitor subauroral

K_p	**0o**	**0+**	**1-**	**1o**	**1+**	**2-**	**2o**
a_p	0	2	3	4	5	6	7
Freq. [%]	3.55	7.17	8.58	8.87	8.85	8.62	8.49
K_p	**2+**	**3-**	**3o**	**3+**	**4-**	**4o**	**4+**
a_p	9	12	15	18	22	27	32
Freq. [%]	8.03	7.40	6.56	5.74	4.59	3.68	2.71
K_p	**5-**	**5o**	**5+**	**6-**	**6o**	**6+**	**7-**
a_p	39	48	56	67	80	94	111
Freq. [%]	2.13	1.47	1.09	0.74	0.52	0.36	0.26
K_p	**7o**	**7+**	**8-**	**8o**	**8+**	**9-**	**9o**
a_p	132	154	179	207	236	300	400
Freq. [%]	0.18	0.13	0.11	0.07	0.06	0.04	0.01

Table 2.1.: The geomagnetic K_p index and its relation to the a_p index [22]

geomagnetic disturbance on a global scale" [22]. The mean value K_p is computed from the single results of 13 participating observatories at subauroral latitudes, each providing a K variance. At each station, the horizontal magnetic components are split into a regular non-K-variation and irregular variations, so-called K variation, with the latter expressed in a K index every three hours. Here, K variation describes geomagnetic activity or disturbance caused by "solar particle radiation" [22] within 3 hours. As mentioned, the a_p index is derived using Table 2.1. It has a physical unit: nanoTesla nT. Similar to the three-hourly K_p index, one can derive a three-hourly a_p index. Eight a_p values form the global planetary average index A_p. [22, 5]

Another geomagnetic index employed by the JB2008 density model is the Disturbance Storm Time (Dst) index [20]. The Dst index indicates the geomagnetic activity at low latitudes on the Earth's surface. During a geomagnetic storm, the magnetic field becomes southward-directed due to the highly energized ring current. "The terrestrial ring current is an electric current flowing toroidally around the Earth, centered at the equatorial plane at altitudes of $\sim 10,000 - 60,000\,km$" [23]. It is calculated hourly using the measurements of four off-equatorial observatories. Its physical unit is nanoTesla nT, similar to the a_p index. Thus, it provides a higher resolution index for magnetic storms, increasing the obtainable accuracy more than the raw usage of just K_p or a_p. It allows for a more precise segregation of storm phases. [19, 23, 24]

The concrete integration in JB2008 is demonstrated by Subsection 2.1.5.

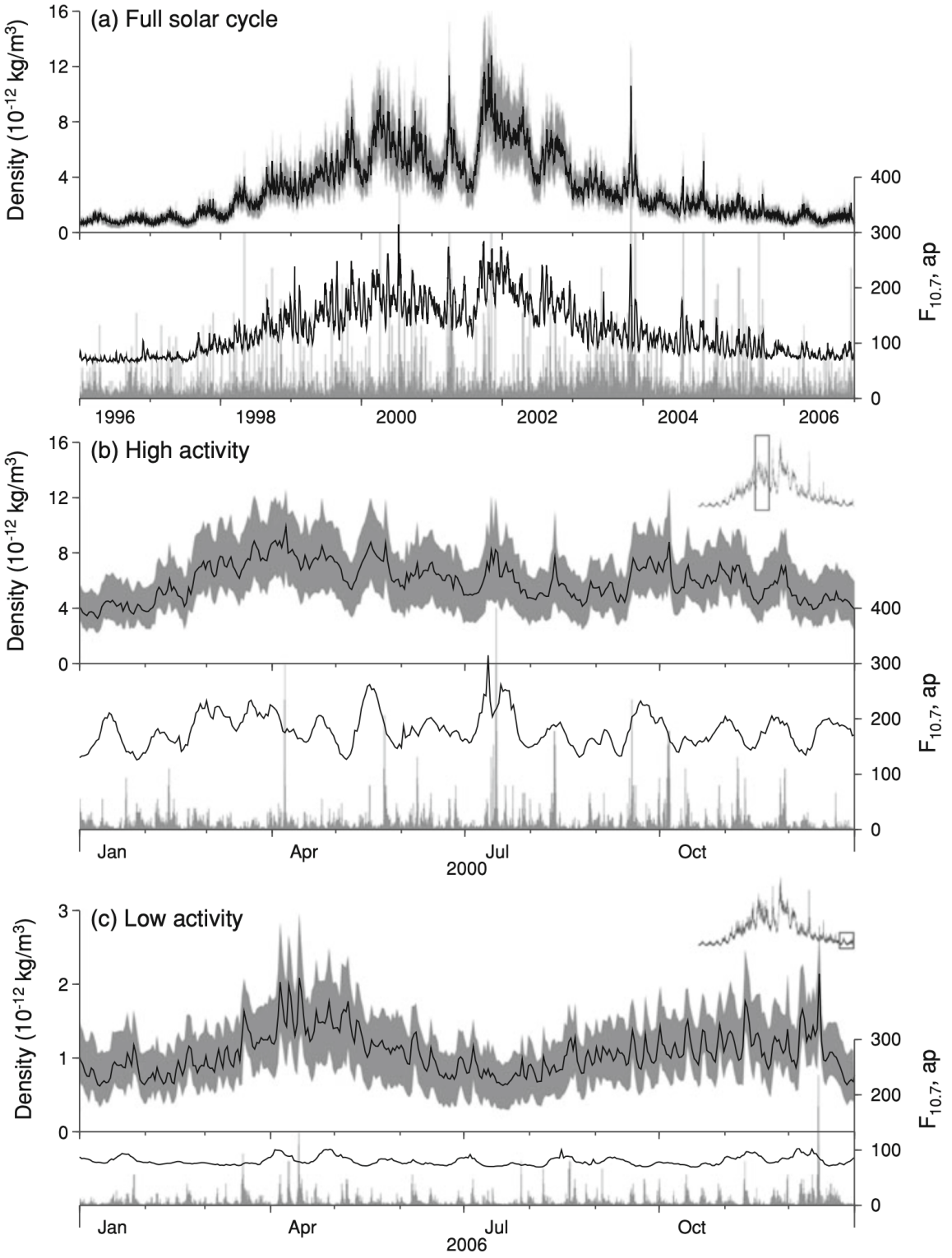


Figure 2.2.: Density average ρ as black line with the daily maxima and minima as gray area in the respective upper plots. $F_{10.7}$ solar flux as black line and geomagnetic a_p index as gray spikes in the respective bottom plots. The total diagram covers the time frame in 400 km height above Delft, The Netherlands. The MSIS00 produced the density values; from [5]

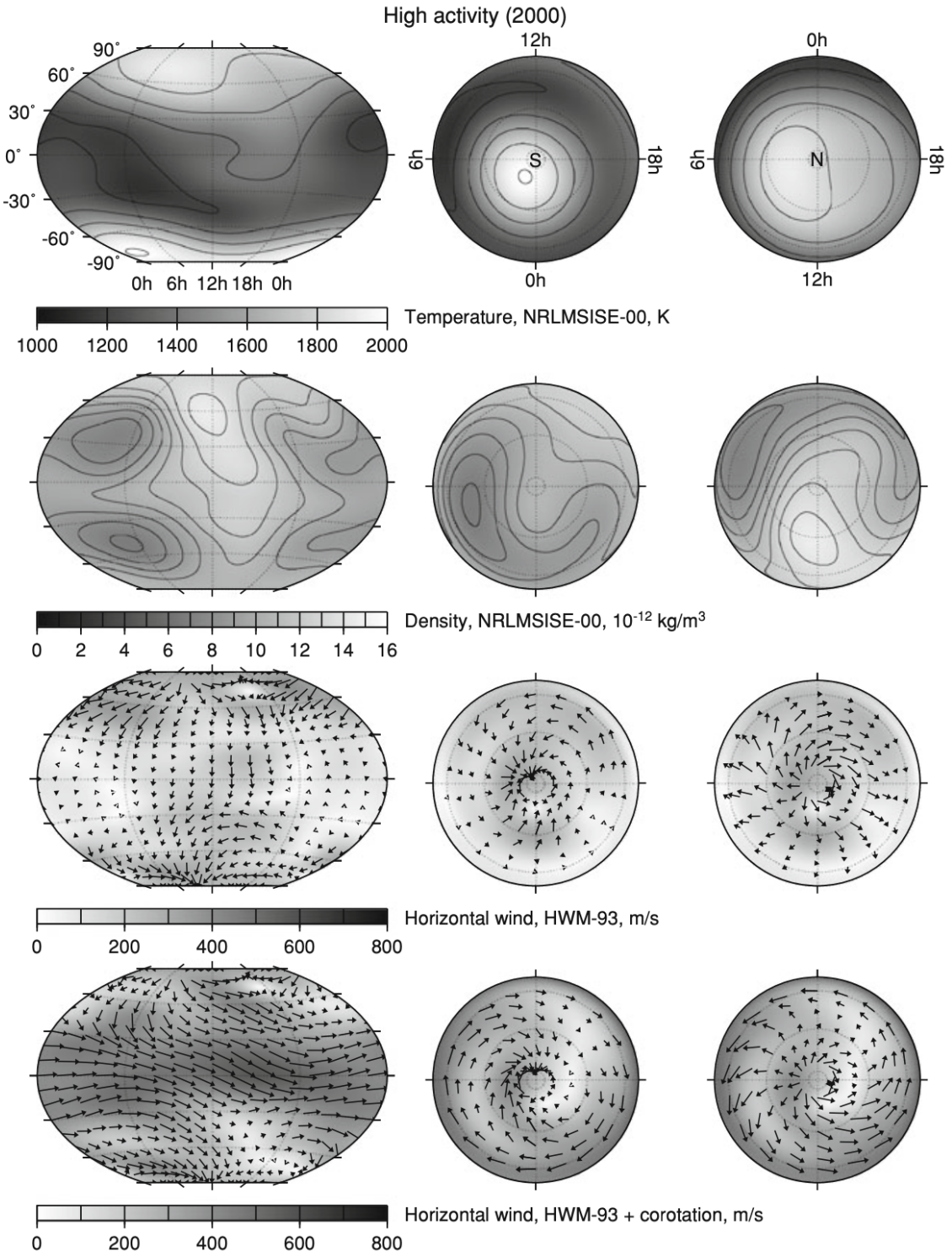


Figure 2.3.: Density (MSIS00) and Wind (HWM93) at 400 km altitude on July 15, 2000 at 18:00 UTC with $F_{10.7} = 213.1$ and $a_p = 400$ (High Activity); from [5]

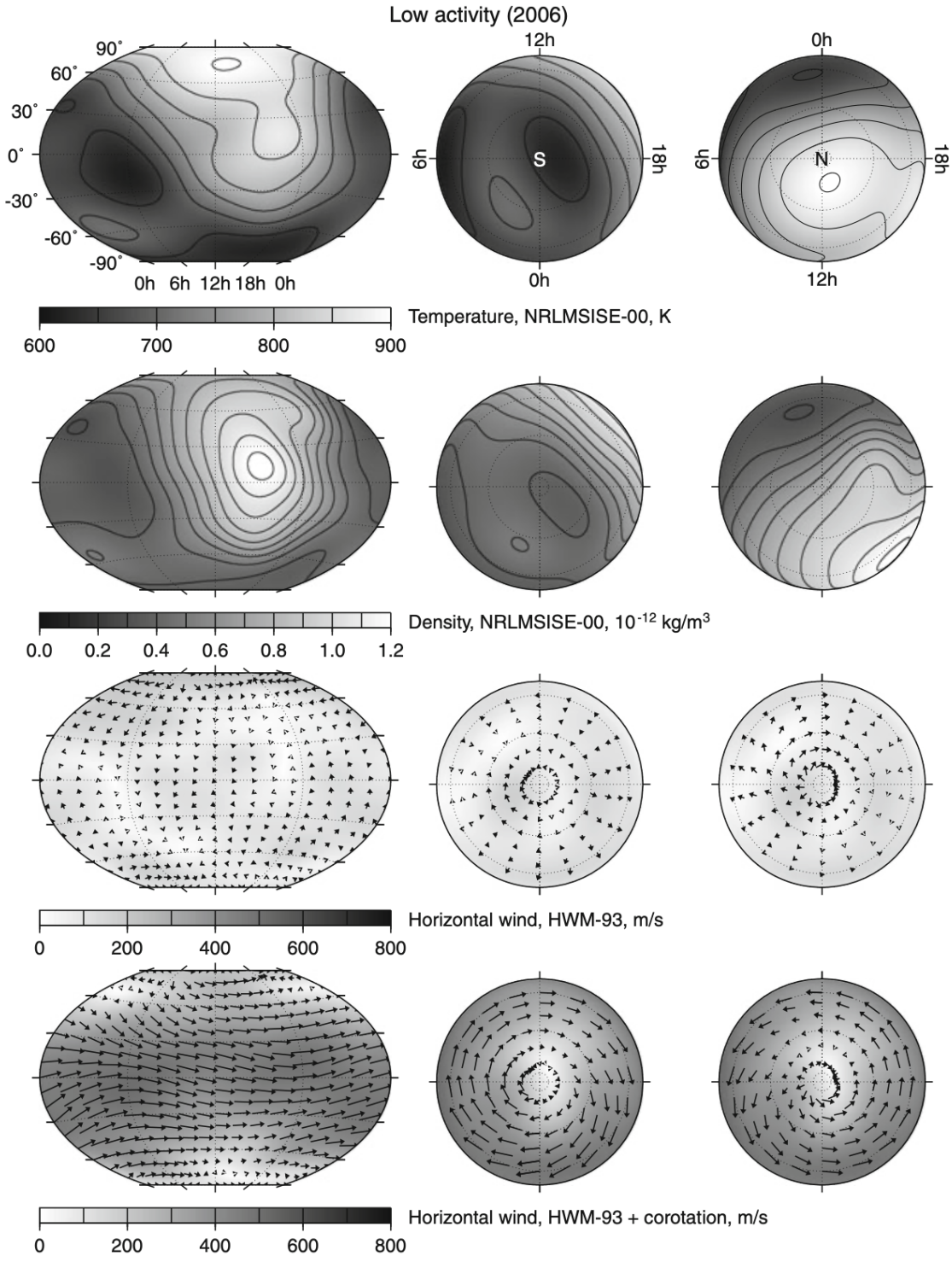


Figure 2.4.: Density (MSIS00) and Wind (HWM93) at 400 km altitude on July 15, 2006 at 18:00 UTC with $F_{10.7} = 70.2$ and $a_p = 3$ (Low activity); from [5]

2.1.4. Overview of Empirical Atmosphere Models

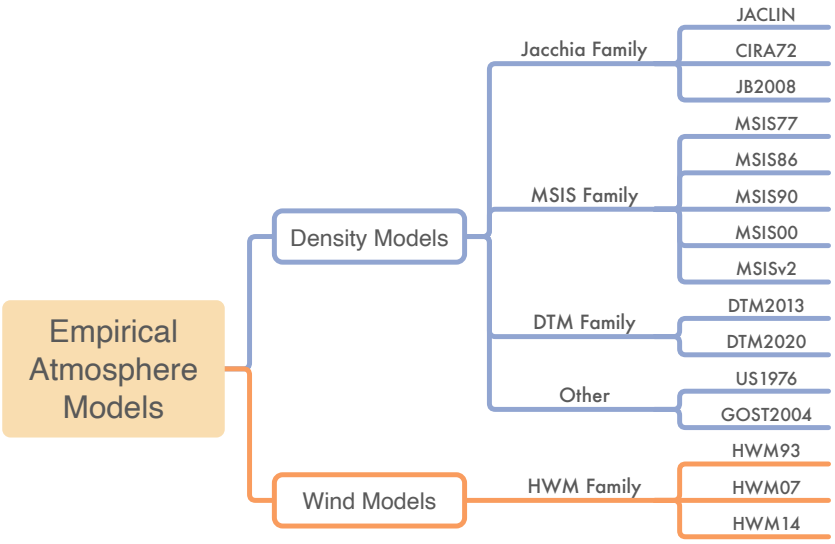


Figure 2.5.: Overview of empirical atmospheric models included in this thesis.

As part of this thesis, multiple atmospheric density and wind models are merged into the software `godot` via a plugin. It goes too far to present every model in detail, but rather, the coming sections will focus on presenting the three prominent families. The models of a family usually do not differ much from generation to generation but rather tune the fitting given the availability of new data. This data can include new density sets or better proxies modeling the complex energetic input factors that influence the thermal state of the atmosphere. Figure 2.5 shows the models integrated into the latter presented `godotAtmsoph` plugin.

The oldest generation of empirical density models dates back to the 1960s. The Jacchia family finds its roots in this time frame with the Jacchia-64 model [25], and the empirical model by Harris and Priester [26]. The Jacchia family is in the later given implementation represented by the Jacchia-71/ Lineberry Model (JACLIN), the COSPAR International Reference Atmosphere of 1972 (CIRA72), and the JB2008 with the latter of particular interest, as it is the target of the optimization process in Chapter 9. In this thesis, the models of the MSIS family are all denoted with MSIS, which stands for Mass Spectrometer and Incoherent Scatter radar, giving hints about the data utilized for generating the model series. Further, they are often referred to as *NRLMSISE*, which is the combination of US Naval Research Laboratory (NRL) and Mass Spectrometer and Incoherent Scatter radar (MSIS). Subsection 2.1.6 presents details about this model family. The DTM family is given by the two models: DTM2013 and Drag Temperature Model 2020 (DTM2020). They are presented in Subsection 2.1.7.

The two remaining models are the Upper Earth Atmosphere Density Model for Ballistic Support of AES Flight (GOST2004) [27] and US Standard Atmsophere of 1976 (US1976) [28].

GOST2004 is a Russian model for the upper atmosphere until a height of 1500 km derived from satellite drag from the Cosmos satellites (1964-2000) [9]. The latter, US1976, technically covers the full atmosphere. However, the implementation integrated into `godotAtmsoph` only provides density to a maximum height of 150 km.

Lastly, the Horizontal Wind Models, whose members are Horizontal Wind Model 1993 (HWM93), Horizontal Wind Model 2007 (HWM07), and Horizontal Wind Model 2014 (HWM14), provide output for zonal and meridional wind speeds in the upper atmosphere. They are aligned to the specification and inputs of the MSIS density models since they were created alongside. [29, 30]

2.1.5. The Jacchia Model Series: JB2008

This section presents the procedure of the JB2008 model, which comprises the Jacchia-Bowman 2006 (JB2006) and older fragments from the Jacchia model series. The JB2008 is largely based on CIRA72, which integrates the diffusion equations using the Jacchia 71 temperature formulation. However, several components were replaced by the formulations of the Jacchia 70 model to better include the latter explained correction terms. [31]

The JB2008 was "adopted as part of the COSPAR International Reference Atmosphere" and it is the recommended model for Earth's upper atmosphere according to ISO 1422 for the purpose of satellite drag calculation [3].

The first step is the computation of the exospheric temperature T_∞ comprising the culmination point temperature T_c and two temperature correction terms denoted with ΔT_c. T_c can also be referred to as the night-time minimum of the global exospheric temperature field. It is computed by

$$T_c = 392.4 + 2.227\bar{F}_S + 0.298\Delta F_{10.7} + 2.59\Delta S_{10.7} + 0.312\Delta M_{10.7} + 0.178\Delta Y_{10.7} \tag{2.5}$$

with

$$\bar{F}_S = \bar{F}_{10.7} \times W_T + \bar{S}_{10.7} \times (1 - W_T) \text{ where } W_T = \sqrt[4]{\bar{F}_{10.7}/240} \tag{2.6}$$

and the delta values $\Delta F_{10.7}, \Delta S_{10.7}, \Delta M_{10.7}, \Delta Y_{10.7}$ as the differences between daily and 81-day average values. The first of the two correction terms ΔT_c results from the diurnal density variation given by the formulation in Subsubsection 2.1.5. It is modeled using the solar activity indices $F_{10.7}, S_{10.7}, M_{10.7}, Y_{10.7}$. The second correction originates from the geomagnetic activity. It is modeled using the Dst index. The concrete translation of Dst to ΔT_c is given by Bowman et al. [20]. Simplified, this translation involves calculating the difference between the one-hour intervals and scaling, whose concrete parameters depend on a fitting to available data. It is not necessary to compute it manually since this part of the correction terms is provided as an index file similar to the $S_{10.7}, M_{10.7}, Y_{10.7}$ as presented in Subsection 4.1.4.

Next, the inflection point temperature T_x must be computed given the exospheric temperature T_∞. The inflection point resides at an altitude of 125 km and represents the lower end of the temperature profile. It is computed (with the concrete coefficients inserted from the `Fortran` implementation) by

$$T_x = 444.3807 + 0.0238T_\infty + 392.8292\exp(-0.0021357T_\infty) \tag{2.7}$$

Given the complete temperature profile T_∞ and T_x, the procedure now computes the single species number densities by integrating the diffusion equation while incorporating correction factors $\Delta\log(\rho)$ for the semiannual density variation (given by Subsubsection 2.1.5) and the seasonal-latitudinal variation (given by Jacchia 70 [32]).

Subsubsection 2.1.5 presents the high altitude density correction scaling factor. It is applied to the final ρ of all previous calculations by a single multiplication.

This collected description of the algorithm is compiled by the author as the concrete instructions are spread over multiple sources of the JB2006 [31], JB2008 [20], Jacchia 70 [32]. At the same time, some were only found inside the implementation.

Diurnal Density Correction

The diurnal density correction is conducted by computing a temperature correction ΔT_c. This offset is then added to T_c for the classic Jacchia density computation. The coefficients B_i and C_i utilized in the polynomials are given by Bowman et al. [31]. However, the equations of the available implementation of JB2008 do not fit the equations reported in [31] nor in [20]. Thus, the equations presented in this section are reconstructed from the present `Fortran` implementation, which is also incorporated in `godotAtmsoph`. Note that the coefficient C_2 is never used across the `Fortran` implementation (so, B_2 utilized next to the C_i is not a printing error). This might be an implementation bug since C_2 has an assigned value by the fitting process of Bowman et al. [20]. In this thesis, we stick to the given original implementation.

$$\begin{aligned} F &= (F_{10.7} - 100)/100 \\ \theta &= (\text{local solar time})/24 \\ \phi &= \cos(\text{latitude}) \\ h &= \text{height(km)} \end{aligned}$$

1. For an altitude $120\,km < h < 200\,km$

$$\begin{aligned} X = C_{17} + \theta\phi\left[C_{18} + C_{19}\theta + C_{20}\theta^2\right] + \\ F\phi\left[C_{21} + C_{22}\theta + C_{23}\theta^2\right] \end{aligned} \tag{2.8}$$

$$\begin{aligned} Y = C_1 + F\left[B_2 + C_3\theta + C_4\theta^2 + C_5\theta^3 + C_6\theta^4 + C_7\theta^5\right] + \\ \theta\phi\left[C_8 + C_9\theta + C_{10}\theta^2 + C_{11}\theta^3 + C_{12}\theta^4\right] + \\ \phi\left[C_{13} + C_{14}F + C_{15}F\theta + C_{16}F\theta^2\right] \end{aligned} \tag{2.9}$$

$$\begin{aligned} \Delta T_c = z^2(3X - Y) + z^3(-2X + Y) \\ \text{with } z = (h - 120)/80 \end{aligned} \tag{2.10}$$

2. For an altitude $200\,km < h < 240\,km$

$$\begin{aligned}\Delta T_c = H\left[C_1 + B_2F + C_3F\theta + C_4F\theta^2 + C_5F\theta^3 + C_6F\theta^4 + C_7F\theta^5\right] + \\ H\theta\phi\left[C_8 + C_9\theta + C_{10}\theta^2 + C_{11}\theta^3 + C_{12}\theta^4\right] + \\ H\phi\left[C_{13} + C_{14}F + C_{15}F\theta + C_{16}F\theta^2\right] + \\ C_{17} + C_{18}\theta\phi + C_{19}\theta^2\phi + C_{20}\theta^3\phi + C_{21}F\phi + C_{22}\theta F\phi + C_{23}\theta^2F\phi\end{aligned} \tag{2.11}$$

with

$$H = (h - 200)/50 \tag{2.12}$$

3. For an altitude $240\,km < h < 300\,km$, one interpolates between the known previous formulation

$$\begin{aligned}X_1 = H\left[C_1 + B_2F + C_3F\theta + C_4F\theta^2 + C_5F\theta^3 + C_6F\theta^4 + C_7F\theta^5\right] + \\ H\theta\phi\left[C_8 + C_9\theta + C_{10}\theta^2 + C_{11}\theta^3 + C_{12}\theta^4\right] + \\ H\phi\left[C_{13} + C_{14}F + C_{15}F\theta + C_{16}F\theta^2\right] + \\ C_{17} + C_{18}\theta\phi + C_{19}\theta^2\phi + C_{20}\theta^3\phi + C_{21}F\phi + C_{22}\theta F\phi + C_{23}\theta^2F\phi\end{aligned} \tag{2.13}$$

$$\begin{aligned}Y_1 = C_1 + F\left[B_2 + C_3\theta + C_4\theta^2 + C_5\theta^3 + C_6\theta^4 + C_7\theta^5\right] + \\ \theta\phi\left[C_8 + C_9\theta + C_{10}\theta^2 + C_{11}\theta^3 + C_{12}\theta^4\right] + \\ \phi\left[C_{13} + C_{14}F + C_{15}F\theta + C_{16}F\theta^2\right]\end{aligned} \tag{2.14}$$

with H given by

$$H = 40/50 \tag{2.15}$$

and the later used formulation for heights above $300\,km$

$$\begin{aligned}X_2 = B_1 + B_{19}\phi + F\left[B_2 + B_3\theta + B_4\theta^2 + B_5\theta^3 + B_6\theta^4 + B_7\theta^5\right] + \\ \theta\phi\left[B_8 + B_9\theta + B_{10}\theta^2 + B_{11}\theta^3 + B_{12}\theta^4\right] + \\ H\phi\left[B_{13} + B_{14}\theta + B_{15}\theta^2 + B_{16}\theta^3 + B_{17}\theta^4 + B_{18}\theta^5\right]\end{aligned} \tag{2.16}$$

$$Y_2 = \phi\left[B_{13} + B_{14}\theta + b_{15}\theta^2 + B_{16}\theta^3 + B_{17}\theta^4 + B_{18}\theta^5\right] \tag{2.17}$$

here, with H is given by

$$H = 300/100 \tag{2.18}$$

then final ΔT_c is then given by

$$\begin{aligned}\Delta T_c = X_1 + Y_1 + z^2(3X_2 - Y_2 + 3X_1 - 2X_2) + z^3(X_2 - X_1 - Y_1 - Y_2) \\ \text{with } z = (h - 240)/60\end{aligned} \tag{2.19}$$

4. For an altitude $300\,km < h < 600\,km$ one uses solely the X_2 from the previous step (3):

$$\begin{aligned}\Delta T_c = B_1 + B_{19}\phi + F\left[B_2 + B_3\theta + B_4\theta^2 + B_5\theta^3 + B_6\theta^4 + B_7\theta^5\right] + \\ \theta\phi\left[B_8 + B_9\theta + B_{10}\theta^2 + B_{11}\theta^3 + B_{12}\theta^4\right] + \\ H\phi\left[B_{13} + B_{14}\theta + B_{15}\theta^2 + B_{16}\theta^3 + B_{17}\theta^4 + B_{18}\theta^5\right]\end{aligned} \tag{2.20}$$

with H given by

$$H = h/100 \tag{2.21}$$

5. For an altitude $600\,km < h < 800\,km$ one uses X_2 and Y_2 from the previous step (3):

$$\begin{aligned}X = B_1 + B_{19}\phi + F\left[B_2 + B_3\theta + B_4\theta^2 + B_5\theta^3 + B_6\theta^4 + B_7\theta^5\right] + \\ \theta\phi\left[B_8 + B_9\theta + B_{10}\theta^2 + B_{11}\theta^3 + B_{12}\theta^4\right] + \\ H\phi\left[B_{13} + B_{14}\theta + B_{15}\theta^2 + B_{16}\theta^3 + B_{17}\theta^4 + B_{18}\theta^5\right]\end{aligned} \tag{2.22}$$

$$Y = \phi\left[B_{13} + B_{14}\theta + b_{15}\theta^2 + B_{16}\theta^3 + B_{17}\theta^4 + B_{18}\theta^5\right] \tag{2.23}$$

with H given by

$$H = 600/100 \tag{2.24}$$

The final ΔT_c is defined as

$$\begin{gathered}\Delta T_c = X + Yz + z^2\left(-\tfrac{3}{4}X - Y\right) + z^3\left(\tfrac{1}{4}X + \tfrac{1}{4}Y\right) \\ \text{with } z = (h - 600)/100\end{gathered} \tag{2.25}$$

Semiannual Density Correction

The semiannual density variation Δ_{SA} is given by

$$\Delta_{SA} log_{10}(\rho) = F(z) \cdot G(t) \tag{2.26}$$

with $F(z)$ representing the amplitude as a function of altitude where with $z = h/1000$. And, $G(t)$ represents the normalized periodic average density variation as a function of time t. [20]. The two functions are given by

$$F(z) = B_1 + B_2\bar{F}_{SMJ} + B_3\bar{F}_{SMJ} + B_4 z^2\bar{F}_{SMJ} + B_5 z\bar{F}^2_{SMJ} \tag{2.27}$$

$$\begin{aligned}G(t) = C_1 + C_2\sin(\omega) + C_3\cos(\omega) + C_4\sin(2\omega) + C_5\cos(2\omega) + \\ \bar{F}_{SM}\left\{C_6 + C_7\sin(\omega) + C_8\cos(\omega) + C_9\sin(2\omega) + C_10\cos(2\omega)\right\}\end{aligned} \tag{2.28}$$

with

$$\bar{F}_{SMJ} = 1.00\bar{F}_J - 0.70\bar{S}_J - 0.04\bar{M}_J \tag{2.29}$$
$$\bar{F}_{SM} = 1.00\bar{F}_{10.7} - 0.75\bar{S}_{10.7} - 0.37\bar{M}_{10.7} \tag{2.30}$$
$$\omega = 2\pi\vartheta \tag{2.31}$$
$$\vartheta = (t - 1.0)/365.0 \tag{2.32}$$
$$t = \text{ day of year} \tag{2.33}$$

where $\bar{F}_J, \bar{S}_J, \bar{M}_J$ are the July averages of the 81-day centered $\bar{F}_{10.7}, \bar{S}_{10.7}, \bar{M_{10.7}}$ and B_i and C_i as coefficients given by Bowman et al. [20]. $\bar{F}_{SMJ}$ in $F(z)$ is based on the yearly semiannual amplitudes observed from 1997 to 2006. [20]

High Altitude Correction

The high altitude correction factor F_ρ of JB2008 for altitudes $h > 1500\,km$ is given by

$$F_\rho(h) = C_1 + C_2\bar{F}_{10.7} + C_3 h + C_4 h\bar{F}_{10.7} \tag{2.34}$$

In the transition regime from $1500\,km > h > 1000\,km$ the following spline function is utilized

$$F_p(H) = 1 + \{3F_{1500} - 500x - 3\}\,H^2 + \{500x - 2F_{1500} + 2\}\,H^3 \tag{2.35}$$

with

$$x = \frac{\delta F_{1500}}{\delta h} = 500(C_3 + C_4\bar{F}_{10.7}) \tag{2.36}$$
$$F_{1500} = \text{ density factor at } 1500\,km \tag{2.37}$$
$$H = (h - 1000)/500 \tag{2.38}$$

and the fitted coefficients C_i given by Bowman et al. [31].

2.1.6. The MSIS Model Series: MSIS00 and MSISv2

Alan Hedin created the first MSIS model in the late 1970s. The MSIS models were initially only based on the mass spectrometer and incoherent scatter radar observations, which has the advantage of maintaining "independent observations of both temperature and number densities for the atmospheric constituents" [5]. MSIS also provides the species densities and not only the total mass density.

The recent models of the MSIS series are MSIS00 [33] and NRLMSIS 2.0 (MSISv2) [34]. The latter has been updated to MSISv2.1 by adding the species density of nitric oxide (NO). However, the primary formulations of MSISv2 remained the same. [34] The models span from the ground upwards to the exobase. They also include accelerometer-derived density data [5].

The models compute the atmospheric density given the geomagnetic a_p index (including the three-hourly values up to 72 hours before) and the $F_{10.7}$ proxy for the solar activity. In MSISv2, the temperature is calculated using a "linear combination of cubic B-splines [..]

below 122.5 km and a Bates thermospheric temperature profile [..] above 122.5 km"[35]. The density is computed using the number densities of the single species whose profile is "parameterized assuming hydrostatic balance in the lower and middle atmosphere and species-by-species hydrostatic equilibrium [..] in the upper thermosphere" [35].

Since it has no added value for the understanding of this thesis to unfold the concrete equations, the interested reader is referred to the original literature by Emmert et al. [35, 34].

2.1.7. The DTM Series : DTM2013 and DTM2020

1978 was the year in which the first Drag Temperature Model was released based on observational data of satellite drag and atmospheric temperature. Over the years, the model's foundation was consequently extended by the addition of more satellite drag, accelerometer, and mass spectrometer data. [5]

The two latest revisions, both included in `godotAtmosph`, are DTM2013 and DTM2020. DTM2013 was first released as DTM2012. It enriches the previous iteration by the high-resolution density data of *CHAMP* and *GRACE*. DTM2013 is an updated version incorporating the *GOCE* densities and the alternative solar activity index F_{30}. [36, 21]

The most recent iteration DTM2020 exists in three variations whose backbone are the densities from *GOCE*, *CHAMP*, and *Swarm A*:

- An operational version using the established solar and geomagnetic indices $F_{10.7}$ and K_p and the algorithm of DTM2013 [37]
- An intermediate version using F_{30} and K_p indices and the algorithm of DTM2013 [36]
- A research version using the - not yet widely as of writing - indices F_{30} and Hpo (a higher resolution K_p index) and a new algorithm [37, 38]

Overall, the DTM2020 produces densities which are around $20\% - 30\%$ smaller compared to previous models. These results are concurrent with the results of MSISv2 despite it "using the assimilation of data in the mesosphere instead of the accelerometer-inferred densities" [37] like DTM2020. `godotAtmosph` integrates the operational first version of DTM2020. Equation 2.39 shows the fundamental equation for calculating the density ρ at a given height h.

$$\rho(h) = \sum_i \rho_i(120\,km) f_i(h) \exp(G_i(L)) \tag{2.39}$$

The density ρ is accumulated over the partial densities of the i single species densities. The latter $f_i(h)$ results from integrating the differential equation of the diffusive equilibrium, which yields the densities propagated to h from $120\,km$. The density variations of Subsection 2.1.2 are integrated via $G_i(L)$ given the environmental parameters L such as latitude, local solar time, and activity. [37, 36]

2.2. Satellite State Modeling

This Section 2.2 sets the previously explained atmosphere models into the referential context. It introduces the available reference frames to describe spatial coordinates and a concrete

point in time. The knowledge is fundamental as the satellite state propagation is often numerically integrated using inertial systems, while the atmosphere usually requires positions with latitude and longitude fixed to the surface. Hence, they require Earth's rotation to be resolved within the reference frame.

2.2.1. Fundamental Definition: System vs. Frame

Before describing how to represent the state and properties of a satellite, one first requires a referential context in which to describe the state. Subsection 2.2.2 introduces the three in the context of this work utilized spatial frames, as well as the fundamental time frames.

Before going into details, it is first necessary to clarify the terminology of a reference system and a reference frame. A reference system specifies how to form a celestial coordinate system in its completeness. This specification covers the origin and the specification of the fundamental planes and axes while also specifying the underlying required models, e.g., the algorithms to work with quantities within the system. In contrast, a reference frame is the concrete instantiation of a reference system and realizes it by specifying concrete, identifiable fiducial points at a fixed point in time. [39]

2.2.2. Coordinate Reference Systems

This paragraph shall give a brief introduction to give the reader only an overall understanding as this is sufficient for understanding this thesis; for a complete mathematical description, refer to references [39, 40, 41] for details. Figure 2.6 outlines the reference systems, frames and intermediate steps presented in this Subsection 2.2.2.

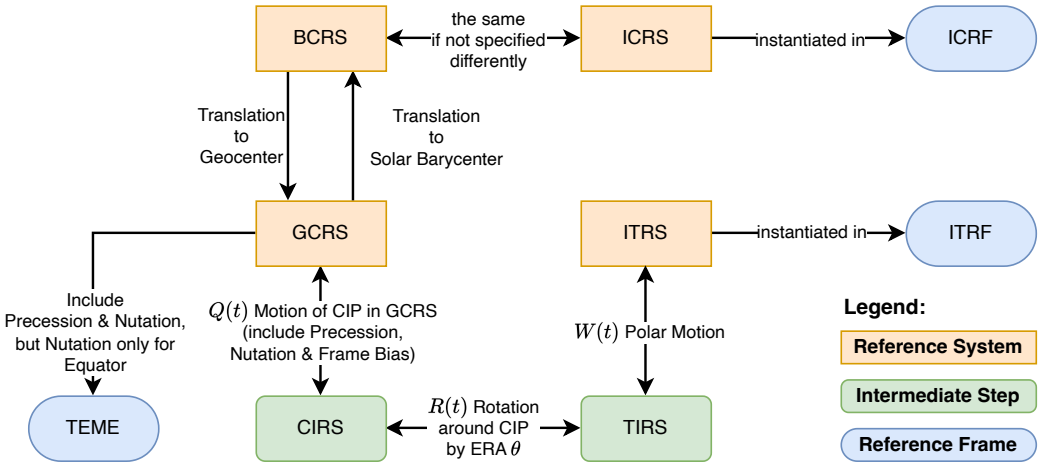


Figure 2.6.: Overview of reference systems, frames and intermediate steps

Reference System Families

To begin with resolution B1.3 of the year 2000, the International Astronomical Union (IAU) defined two coordinate systems: the Barycentric Celestial Reference System (BCRS) and the Geocentric Celestial Reference System (GCRS). Both define a system of space-time

coordinates for, respectively, the solar system and the Earth "within the framework of General Relativity, by specifying the form of the metric tensors for each and the 4-dimensional space-time transformation between them."[39] The BCRS has its origin in the barycenter of the Solar System. The Solar System's barycenter is the center of mass of all its enclosing bodies. In contrast, the GCRS has its origin in the geocenter. Thus, the BCRS is well-suited for describing positions and motions outside the context of Earth, while the GCRS should be favored suited for describing objects near Earth.

The axes of BCRS and GCRS do not have a formal orientation. Further, they are described as kinematically non-rotating ("space-fixed") [39]. This means that for both systems, no rotation exists with respect to distant objects in the universe (i.e., objects so far away that they appear to be fixed using humanity's available instruments [41]).

These two systems span the families to define the following reference system.

Celestial Reference Systems

Although the BCRS's axes are not formally defined, if not otherwise stated, they usually align with the specification of the International Celestial Reference System (ICRS) axes for all practical applications. The ICRS formalizes the BCRS by using a set of benchmark objects, observable at radio wavelength, whose adopted coordinates effectively define the direction of the ICRS axes. The ICRS axes closely approximate the mean Earth equator and equinox (Earth's equinox is the moment when the Sun is directly above the Earth's equator) of January 1, 2000, 12.00 h TT (J2000) to within 0.02 arcseconds. However, there is actually no date associated with the ICRS since its axes are kinematically non-rotating, and the defining sources are so far away that their motion is negligible, as previously already stated. [39]

The concrete realization of the ICRS is given by a concrete set of 212 defining radio catalog sources. The realization is called the International Celestial Reference Frame (ICRF), more specifically ICRF1 [39]. The IAU adopted the current realization ICRF3, covering an extended and more precise catalog of defining sources in August 2018 [42]. The base plane of the ICRF is the equatorial plane at J2000, with the x-axis being the quasar 3C273, the z-axis as the normal to the equatorial plane, and the y-axis as the cross product of x and z [41].

Terrestrial Reference Systems

Section 2.1 presents several atmosphere models whose evaluation depends on the concrete position on Earth with respect to its surface. GCRS does not account for Earth's rotation. Thus, coordinates in this celestial reference frame are unsuitable as input for these atmosphere models. So, GCRS needs to be converted beforehand to a reference frame accounting for the rotation. This is solved in the International Terrestrial Reference System (ITRS), which finds its realization in the International Terrestrial Reference Frame (ITRF) using the results of several terrestrial measurement stations across the Earth which provide a list of precise coordinates fixing the reference frame [43]. The current realization is ITRF2020 [44]. Another widely-used realization of the ITRS is ,e.g., GPS [40].

ITRS co-rotates with the Earth and further accounts for:

- *Precession and Nutation* (forced rotation): Earth's axis of rotation is not fixed in space due to the gravitational attraction of the Moon, Sun and other major planets. Instead, it rotates about the pole of the ecliptic (i.e., the orbital plane of the Earth around the Sun). Precession represents the secular component with a period of 26000 years and nutation represents the periodic component with a period of 18.6 years of this movement [45]. The forced rotation can also be described as the motion of the Celestial Intermediate Pole (CIP) in the GCRS [39]. The CIP is the shared z-axis (along the poles) utilized for transforming ITRS to GCRS and vice-versa.
- *Pole Motion* (free rotation): "Due to the structure of Earth's distribution of mass and its variation, Earth's axis of rotation is not fixed in relation to Earth's crust" [45]. This movement has a period of 430 sidereal days, which is referred to as the Chandler period [45]. The free rotation can also be described as "the motion of Earth's pole with respect to the ITRS"'[39].

The following paragraph investigates one of the two possible transformations from ITRS to GCRS. The transformation from ITRS to GCRS is trivial using the respective inverse operations. Figure 2.7 displays the two ways of performing the mentioned conversion. The main emphasis is on the transformation based on the CIO-based transformation defined by Equation 2.40.

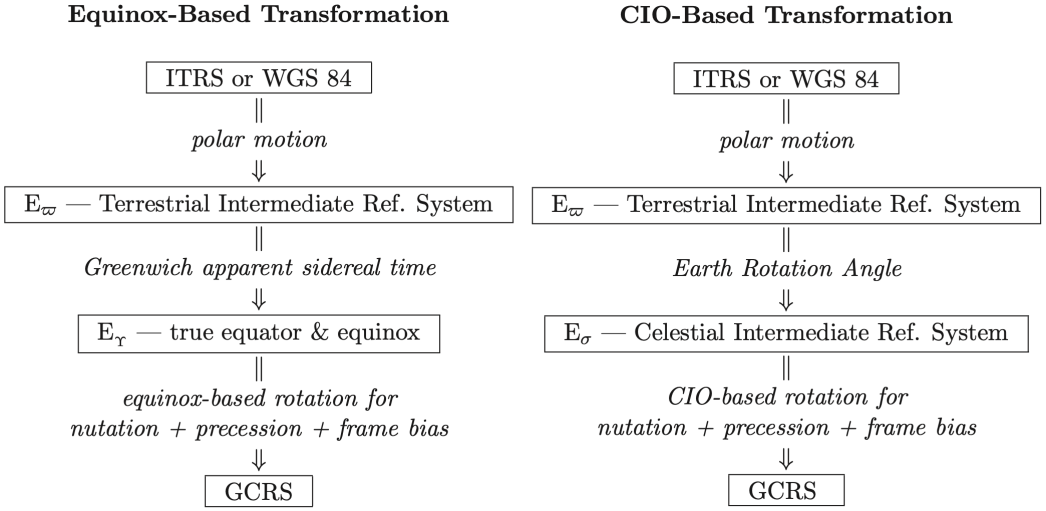


Figure 2.7.: The two ways to transform coordinates from ITRS to GCRS; from [40]

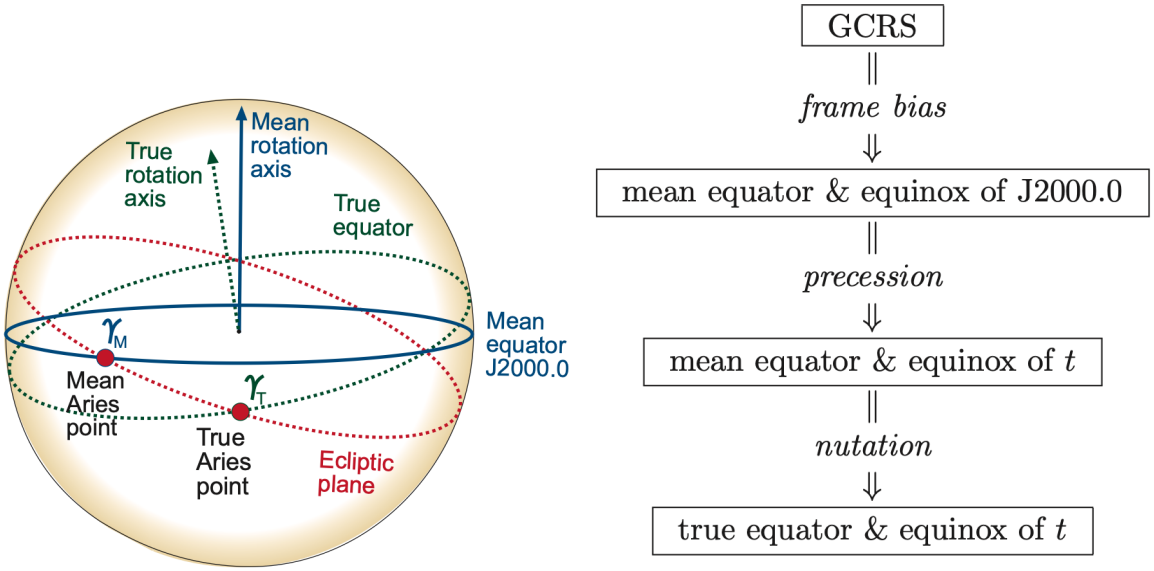
$$\overrightarrow{r_{GCRS}} = Q(t)R(t)W(t) \cdot \overrightarrow{r_{ITRS}} \tag{2.40}$$

$\overrightarrow{r_{GCRS}}$	is the coordinate vector in the GCRS
$\overrightarrow{r_{ITRS}}$	is the coordinate vector in the ITRS
$W(t)$	is the rotation ITRS to TIRS E_{ϖ}
$R(t)$	is the rotation TIRS E_{ϖ} to CIRS E_{σ}
$Q(t)$	is the rotation CIRS E_{σ} to GCRS

The first out of three rotations is conducted with the polar motion (wobble) matrix $W(t)$. It relates the ITRS and the Terrestrial Intermediate Reference System (TIRS) - also denoted with E_{ϖ} - and arises from the polar motion (which is also called wobble). The TIRS has its origin in the Terrestrial Intermediate Origin (TIO), and the z-axis is the CIP. "The TIO was originally set at the ITRF origin of longitude and throughout 1900-2100 stays within 0.1 mas of the ITRF zero meridian" [39]. Thus, $W(t)$ rotates the axes to align with the CIP

The second rotation uses the matrix $R(t)$. It relates TIRS and Celestial Intermediate Reference System (CIRS) - also denoted as E_{σ} - by using the Earth Rotation Angle (ERA) θ at epoch t to incorporate the sidereal rotation of Earth. CIRS has its origin in the Celestial Intermediate Origin (CIO), which corresponds to the non-rotating origin of the GCRS [39]. The ERA θ is the angle along the CIP between TIO and CIO positively in retrograde direction. It is linearly time-dependent to Universal Time No. 1 (UT1) (see Subsection 2.2.3). This middle of three rotations also demonstrates how the CIP clearly separates the nutation and precession from polar motion [46].

The third rotation uses the matrix $Q(t)$. It relates CIRS and GCRS and accounts for the effects of nutation and precession.



(a) Illustration of the mean equator and mean equinox (= Mean Aries Point) γ_M of J2000 and the true equator and true equinox (= True Aries Point) γ_T at a given time t [45]. The equinox γ is the intersection between the ecliptic and the Earth's equator.

(b) The inclusion of the frame bias converts GCRS to the mean equator and equinox of J2000. The inclusion of precession at epoch t yields the mean coordinates at t. The inclusion of the even smaller oscillating nutation at t yields the true coordinates at t. [40]

Figure 2.8.: Illustration of true and mean coordinates given a reference epoch t

The whole procedure is summarized in Figure 2.7 on the left side. The second procedure differs from the explained CIO-based transformation starting from step two. It is displayed on the right side of Figure 2.7. While the CIO-based procedure utilities the ERA θ, which is defined using UT1, the equinox-based transformation uses the Greenwich Apparent Sidereal Time (GAST) to represent the Earth's angle of rotation, i.e. the angle between

true equinox and TIO. For the definition of GAST, refer to Subsection 2.2.3. The result of this transformation yields the coordinates in true equinox and equator E_γ at epoch t [40]. Figure 2.8a illustrates the meaning of true and mean elements in the orbital plane, while Figure 2.8b summarizes the concrete steps required to obtain one of them. "True" means that nutation and precession are covered, while "Mean" implies only the inclusion of precession at a given epoch t.

The third step in the equinox-based transformation is similar and accounts for precession and nutation. However, the concrete operations differ. GAST is a sidereal time. Thus, nutation and precession are already in parts included in the second step. This is also the main conceptual difference of the CIO-based transformation when compared to the equinox-based transformation. In the former case, all rotations are independent, whereas in the later step two and three are coupled.

True Equator Mean Equinox

Subsection 4.1.1 presents the Two Line Element Set (TLE). This format specifies the orbital elements in the True Equator Mean Equinox (TEME) reference frame. The main goal of the TEME reference frame at the time of construction was a reduction of computations, especially trigonometric computations since they were a "formidable challenge to early digital computers" [47] and key to propagating many objects. It is an Earth-centered inertial (ECI) reference system [47]. Given the name, the position of the equator is given by the true elements (including precession and nutation), whereas the equinox is defined only by the precession. This concept is referred to as *Uniform Equinox of Date* in literature [47, 48].

For example, in order to convert to ITRF from TEME, one needs to add the polar motion, and it requires a rotation using the Greenwich Mean Sidereal Time (GMST). The procedure is widely implemented and not subject to further explanation. For details, refer to [47, 48].

However, it is unclear if the TEME frame is "of date" (meaning that the epoch of the frame corresponds to the associated ephemeris generation time) or "of epoch" (meaning that the epoch of the frame is held constant). "Researchers generally believe the 'of date' option is correct, but confirmation from official sources is uncertain, and others infer that the 'of epoch' is correct" [48]. TEME is a military development and this subject has never been clarified. As the following study does not implement the TEME conversion but rather relies on the provided one by the `godot` library, this issue is not further of interest.

2.2.3. Timescales

Most of the previously presented atmosphere models require their time in Coordinated Universal Time (UTC), while some require a different input ,e.g., GOST2004 requires GMST. This section is going to present UTC and the common time frames when working with celestial or terrestrial reference frames.

Measuring the Date

The Gregorian calendar, which we currently use, is based on the birth of Christ. It defines a year as 365.2425 days. Thus, the year has 365 full days in a year and a leap day is added every 4 years to compensate for the reminder. However, the leap day is omitted every 100 years except for every 400 years, where it is added back again. There is no year zero in the

Gregorian calendar, and time is divided into Before Christ (BC) and Anno Domini (AD) periods. [41]

In contrast, the discipline of astronomy uses the Julian Date (JD). Its point of origin is 12h noon on January 1, 4713 BC. The JD does not count in years but only in days since its origin. Thus, the JD does not have consistency problems like the Gregorian calendar introducing leap days. Equation 2.41 shows the conversion to JD, given a year $y \in [1901, 2099]$, month m, day d, hour h and second s in the Gregorian system [41]

$$\begin{aligned} JD = 367 \cdot y - floor\left\{1.75 \cdot \left[y + floor\left(\frac{m+9}{12}\right)\right]\right\} \\ + floor\left(\frac{275 \cdot m}{9}\right) + d + 1721013.5 + \frac{\frac{s/60+m}{60} + h}{24} \end{aligned} \tag{2.41}$$

For example, the widely used standard reference epoch J2000 has the Julian Date 2451545.0. In order to tighten the numbers and simplify handling JDs, several different reference origins were introduced, as shown in Equations 2.42–2.44.

$$MJD = JD - 2,400,000.5 \quad \text{(with origin November 17, 00:00 h, 1858)} \tag{2.42}$$

$$MJD1950 = JD - 2,433,282.5 \quad \text{(with origin January 01, 00:00 h, 1950)} \tag{2.43}$$

$$MJD2000 = JD - 2,451,544.5 \quad \text{(with origin January 01, 00:00 h, 2000)} \tag{2.44}$$

Measuring the Time

Measuring time requires a periodic steady, periodic process since periods are countable [41]. As of now, one can classify three different periodic processes in use for measuring time. Table 2.2 shows the commonly used periodic processes: Earth's rotation, Earth's revolution around the Sun, and atomic oscillators.

When using Earth's rotation, one measures the angle between a reference median (e.g., Greenwich) and the meridian containing a celestial reference. This celestial reference can be, e.g., the Sun, in which case one speaks of solar time, or the Aries point, in which case one speaks of sidereal time. The three Universal Times (UTs) use Earth's rotation and as the second reference median Greenwich next to the Sun as celestial reference point.

Similarly to mean and true equinox, one can distinguish between mean and true solar time. The UTs use the mean solar time which uses a fictitious mean Sun that moves along the equator of Earth with a uniform speed over the whole course of a year. UT0 is then the mean solar time at the Greenwich meridian without a correction of the polar motion. UT1 includes the effect of polar motion. UT2 removes periodic seasonal variations of Earth's rotation from UT1. The previously introduced ERA θ depends on UT1. This relation is shown in Equation 2.45 [39]. In practice, only UT1 is used, while the other two are a legacy of the past. [45]

$$\theta = 0.7790572732640 + 1.00273781191135448 \cdot (JD(UT1) - 2451545.0) \tag{2.45}$$

Figure 2.9 exemplifies GMST, GAST, Local Mean Sidereal Time (LMST) and Local Apparent Sidereal Time (LAST). These time frames do not use the movement of the Sun over the Earth's equator. Instead, they measure the hour angle between the reference

Periodic Process	Time Frames	
Earth's rotation	Solar Time Frames	Sidereal Time Frames
	Universal Time (UT0, UT1, UT2)	Greenwich Mean Sidereal Time (GMST), Greenwich Apparent Sidereal Time (GAST)
Earth's revolution	Terrestial Dynamic Time (TDT) ≡ Terrestrial Time (TT), Barycentric Dynamic Time (TDB)	
Atomic oscillators	International Atomic Time (TAI), Coordinated Universal Time (UTC)	

Table 2.2.: Timescales and their underlying periodic process [45]

meridian and the vernal equinox. A mean sidereal day is shorter than a mean solar day by $24^h/365.2422 \approx 3^m56^s$ "due to the relative movement between Sun and Earth as a consequence of its annual translation" [45].

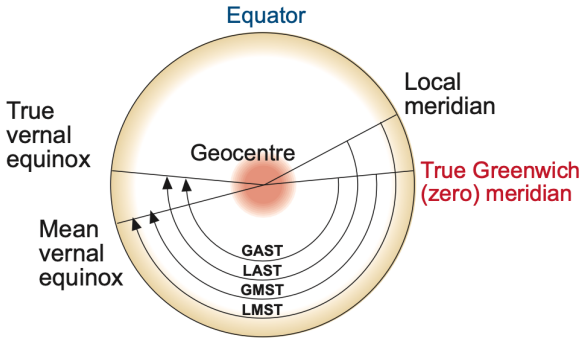


Figure 2.9.: GMST and GAST are both the hour angle between the Greenwich meridian and respectively the mean or true equinox. LMST and LAST are the equivalents using the local meridian where being measured as first reference. [45]

The Earth's revolution timescale was initially realized with the Ephemeris Time (ET) and later replaced by two relativistic timescales: TDB and TDT. Both of them are based on the ET second, which is defined as the fraction of the tropical (solar) year as shown in Equation 2.46. TDB "is an inertial time in the Newtonian sense and provides the time variable in the equations of motion for the ephemerides related to the cent[er] of gravity of the Solar System" [45]. At the same time, TDT is only quasi-inertial with respect to Earth and utilized as the independent variable of the satellite's equation of motion.

$$\text{ET second} = \frac{1}{31\,556\,925.9747} \tag{2.46}$$

TDT was with IAU resolution A4 replaced by TT. However, from the definition side, TT and TDT are equivalent. Further, beginning in 1967, the ET second (which also served as SI second until then) was replaced by the TAI second. The TAI second is defined as "*the*

duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the Caesium 133 atom" [45]. The period is not chosen randomly. Instead, it makes the ET second and the TAI second equivalently long. Thus, the definition of the SI second changed, but its length remained the same in 1967.

Equation 2.47 shows the relation between TDT, TT and TAI. The only difference between the former and TAI lies in the different starting offsets.

$$TDT \equiv TT = TAI + 32.184s \tag{2.47}$$

Equation 2.48 exemplifies the conversion between TDT and TDB. This conversion is not further discussed, as TDB is in the subsequent chapters only required to be defined but not used. For details, refer to sources [45, 39, 40].

$$\begin{aligned} TDB &= TDT + 0^s_.001658 \ \sin(g + 0.0167 \ \sin g) \\ \text{with } g &= \frac{2\ \pi\ (357^o_.528 + 35\,999^o_.050\ T)}{360^o} \\ T &= \frac{JD - 2451545.0}{36525} \end{aligned} \tag{2.48}$$

Finally, UTC is a hybrid timescale. Its second is based on TAI. However, it is synchronized to the rotation-based timescale UT1 by the irregular introduction of leap seconds to be never more than 0.9s out of the scope of UT1. This relation is displayed in Equation 2.49. [39]

$$UTC = TAI - \text{leap seconds} = UT1 \pm 0.9s \tag{2.49}$$

2.2.4. Satellite State

Given the in Subsection 2.2.2 presented reference frames, one requires three elements in order to describe the position $\vec{r}$ of a spacecraft inside the frame. In order to describe the orbit one requires three additional elements, namely the first derivatives providing the velocity $\vec{v} = \dot{\vec{r}}$. These six elements precisely specify the orbit in a three-dimensional coordinate system. Given an initial state $(\vec{r}_0, \vec{v}_0)$ at an epoch t, it is possible to determine the future motion of the spacecraft in phase space using the two first-order differential vector equations based on Newton's gravitation equation of motion. [41] Equation 2.50 depicts this with μ as the gravitational constant of the central body and r as the distance between the centers of mass.

$$\dot{\vec{r}} = \vec{v}, \dot{\vec{v}} = -\frac{\mu}{r^3} \cdot \vec{r} \tag{2.50}$$

The disadvantage of this representation is that each of the six elements constantly changes. The Keplerian elements give a better, more illustrative representation.

Figure 2.10 shows the Keplerian elements and how they affect the orbit of a satellite around the Earth. Semi-major axis a and eccentricity e are the two elements describing the shape of an orbit and are, therefore, called metric elements. a is, as the name suggests, half of the major-axis of an ellipsoid, whereas the eccentricity reflects how circular the orbit is (see Equation 2.51).

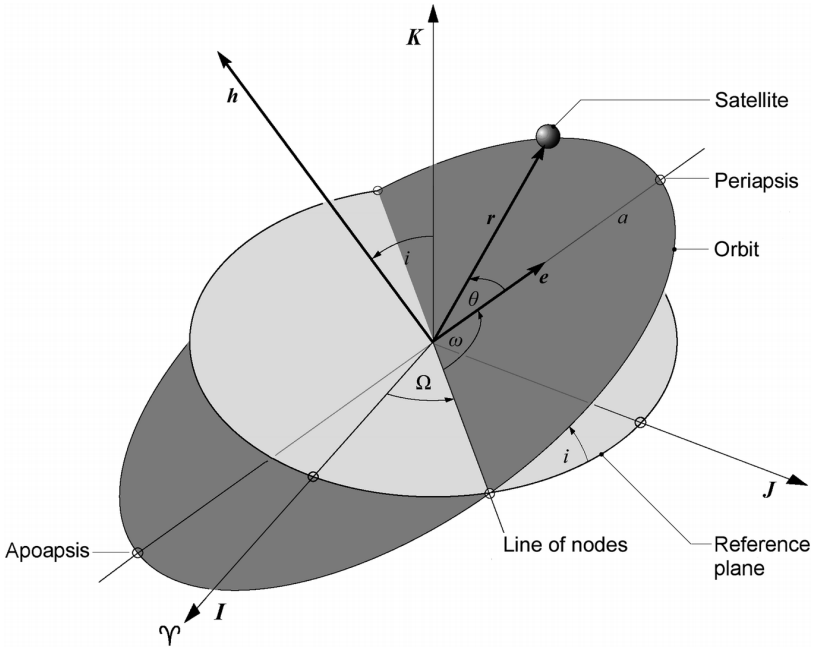


Figure 2.10.: The chart shows the semi-major axis a, the inclination i, the right ascension of ascending node Ω, the argument of periapsis ω and the true anomaly θ; from [41]

$$shape = \begin{cases} \text{circular} & \text{if } e = 0 \\ \text{elliptic} & \text{if } 0 < e < 1 \\ \text{parabolic} & \text{if } e = 1 \\ \text{hyperbolic} & \text{if } e > 1 \end{cases} \tag{2.51}$$

The following three elements characterize the orbit's orientation around a celestial body; hence, they are called angular elements. The inclination i is the angle "between the angular momentum vector h and the z-direction of the reference frame" [41] or the angle between the reference plane, commonly the equatorial plane, and the orbital trajectory plane. In contrast to the inclination, the right ascension of ascending node (RAAN) Ω describes the "angle between the line from the origin O of the reference frame to the vernal [equinox] and from O to the ascending node" [41]. The last of the three elements is the argument of periapsis ω, which is the angle "in the orbital plane between the line of nodes and the periapsis measured in the direction of the motion" [41].

These are the five time-independent orbital elements. The sixth element is the true anomaly θ, which is time-dependent. It reflects the angle between the periapsis and the current position $\vec{r}$. Alternatively, one can also specify the mean anomaly M given in Equation 2.52 whose relation to the mean motion n is given in Equation 2.53.

$$M = n \cdot (t - t_p) \text{ with } t_p = \text{point in time at periapsis} \tag{2.52}$$

$$n = \frac{2\pi}{T} \overset{elliptic}{=} \sqrt{\frac{\mu}{a^3}} \text{ with T = Orbital Period}, \mu = \text{gravitational parameter} \tag{2.53}$$

For a detailed description of the conversion between cartesian state vectors and Keplerian elements, refer to [41].

2.3. Orbit Propagation with Perturbations

A satellite orbiting a body whose mass distribution is radially symmetric follows a conic section called Keplerian orbit [6]. This orbit is clearly defined by Equation 2.50. However, in practice, additional forces are affecting a satellite around, e.g., the Earth. These forces are "called perturbing forces, and one speaks of perturbed Keplerian orbits" [6]. Equation 2.54 adds the perturbation $\vec{a}_p$ into the equation.

$$\dot{\vec{r}} = \vec{v}, \dot{\vec{v}} = -\frac{\mu}{r^3} \cdot \vec{r} + \vec{a}_p \tag{2.54}$$

These perpetuating forces can be of various origins and add together in $\vec{a}_p$. To mention a few:

- the oblateness/ non-radially-symmetric mass distribution of Earth causing perturbations in the gravity field
- other celestial bodies' attraction (especially due to the Moon and the Sun)
- atmospheric drag
- solar radiation forces
- electromagnetic forces (an electrically charged satellite is affected by Earth's magnetic field)

In this thesis, only the atmospheric drag is further investigated (see Subsection 2.4.1). For a precise mathematical formulation of the other remaining perpetuating influences, refer to sources [41, 6, 49], as the details are not required to understand the accomplished assignment further.

It is not possible to solve Equation 2.54 using analytical methods anymore. In order to solve the problem, three techniques have been established. The *special perturbation methods* solve the differential equation using numerical methods. The theory is discussed in Subsection 2.3.1. For example, the later in detail presented `godot` provides propagation by employing numerical methods [50]. In contrast, the *general perturbation methods* solve the propagation problem by utilizing approximate analytical methods. A well-established general propagator designed for use in combination with TLE data is the Simplified General Perturbations 4 (SGP4) model [51]. SGP4 and the general perturbation theory are examined in Subsection 2.3.2.

The third option are *semi-analytical* orbit propagation methods [52]. They combine the analytical propagation's advantage of fast runtime with the accuracy of numerical

propagation. Subsection 2.2.2 discusses precession and nutation. Similarly, the orbital state of a satellite can be split into a secular, long-term (the mean-motion) and an osculating, short-term motion. The idea is to propagate the slow dynamics by numerical means with a large step size and apply the short-period terms afterward at the final epoch in order to recover the entire state. One realization of this procedure is the *Draper Semi-analytical Satellite Theory* (DSST). [53, 54]

2.3.1. Special Perturbation Theory

Implementation Methods

The special perturbation methods perform the computation by a numerical step-by-step integration process. This chapter first investigates the available numerical methods for performing the propagation while focusing on the numerical integration procedures in the second half. The "three classical methods are known as *Cowell's method*, *Encke's method* and the *method of variation of orbital elements*" [6].

Cowell's method is the simplest one because it is just the plain numerical realization of Equation 2.54. Thus, it can be directly implemented and does not require any previous assumptions. However, it also does not take advantage of, e.g., splitting the equation into the "base" Keplerian orbit and the small perpetuating forces. Rather, the sum of all acceleration is integrated together in one step, leading to the requirement of small integration steps - ultimately increasing the numerical error due to floating point arithmetic. [6]

Encke's method resolves this by only integrating the difference between the major acceleration and all perpetuating forces. In other words, only the difference from a given reference trajectory (given by Equation 2.54 without $\vec{a}_p$) is numerically integrated. This way, the method is about ten times faster [49]. However, the method loses its advantage if the perpetuating acceleration does not vary but accumulates over time. In this case, the reference orbit must be adapted ("rectification"). [6]

The *method of variation of orbital elements* (also known as the "method of variation of parameters") uses the orbital elements and considers the perpetuated orbit as a continuous sequence of Keplerian orbits. Thus, it can be compared to *Encke's method*, which also uses a single (or a set if rectification is required) of Keplerian reference orbits. Depending on how the set of differential equations yielded by this approach is evaluated, the method falls in the category of special perturbation or general perturbation techniques. [6, 49]

Cowell's method is the most widely used today [55], given that it does not come with any presumptions and can be notably straightforwardly implemented.

Numerical Integration Methods

A great variety of numerical integration procedures exists for solving differential equations. The analyzed literature agrees that the Gauss-Jackson method is the most suitable if the integration can be executed with a near-constant step size. However, no definite best method can be named as soon as one starts to vary the step length. [56] This is reflected by a query among members of the IAU showing that every method, from Runge-Kutta, multi-step, and extrapolation methods, is in use [57].

`godot` provides namely two numerical integration methods: The Runge-Kutta 8(7) method and an Adams-Bashforth-Moulton predictor-corrector model. The latter multi-step procedure

is of variable order (in `godot` from 1 to 12) and variable step-size, both being determined during runtime [58]. The former Runge-Kutta 8(7) is of eighth order, with an error estimator of seventh order. It is based on the Runge-Kutta method but improved by Fehlberg in order to estimate the error using an interpolation of higher order without excessive computational overhead. This reduced overhead is achieved by embedding the evaluations of the error estimator of higher order into the actual calculation, effectively re-using the evaluations of the lower order. This error estimator allows for an adaptive step, i.e., an optimal step size can be calculated automatically. The coefficients and equations can be found in the publications by Fehlberg [59, 60]. `godot` further provides Runge-Kutta 8(7) by Fehlberg in a different flavor using the improved coefficients by Verner [61]. Further, polynomial interpolation of order 7 or 8 between propagation steps is offered for dense output by `godot`.

2.3.2. General Perturbation Theory

This Subsection 2.3.2 first explores some options for using general perturbation theory to propagate a satellite. The second half focuses on the widely-used SGP4. Subsection 2.3.1 introduced the *method of variation of orbital elements*. Simplified, applying it on Equation 2.54 but with Keplerian elements rather than cartesian coordinates yields the *Lagrange's planetary equations*. These cannot be evaluated by a closed-form solution. Nevertheless, an approximate analytical evaluation is possible in case of small perturbations. [6, 41]

Another approach providing more flexibility is given by Hoots et al. [62, 63]. They are using the *method of averaging* to remove the dependence of the fast variable mean anomaly. Finally, the approach yields an analytical formulation only requiring numerical measures once at startup. Further, the final analytical form is independent of the utilized density model (However, it is not entirely free since the density values need to be known at initialization time)

Hoots also created the Simplified General Perturbations 8 (SGP8) [64] model based on his previous work [65]. The SGP8 model improves SGP4 regarding atmospheric decay. However, "there is no evidence to suggest that SGP8/SDP8 was implemented for operational TLE formation" [48]. So, SGP4 is still the reference for a general perturbation model. SGP4 was developed in 1970. It is purposed for near-Earth satellites. The drag is based on a power density model of the atmosphere. Overall, the TLE data published is specifically designed to be used in combination with the SGP4. [51]

2.4. Reentry Modeling

This section's first half briefly discusses the foundations and requirements of modeling atmospheric drag and its impact on a satellite in Subsection 2.4.1, while the second half in Subsection 2.4.3 finalizes Chapter 2 by gluing everything together.

2.4.1. The Impact of Atmospheric Drag

The determination of the atmospheric drag requires two major operations. First, one must know the atmospheric state at the target body's location. Second, one requires the knowledge of how the atmosphere interacts with the target. The former operation requires, e.g., one of the in Section 2.1 presented atmospheric density models, while the latter requires information

about the attributes of the respective target satellite. Figure 2.11 shows this interplay between satellite and atmosphere. It illustrates the exponential increasing decelerating force with decreasing altitude resulting from the constant exchange of momentum and energy between satellite and atmosphere.

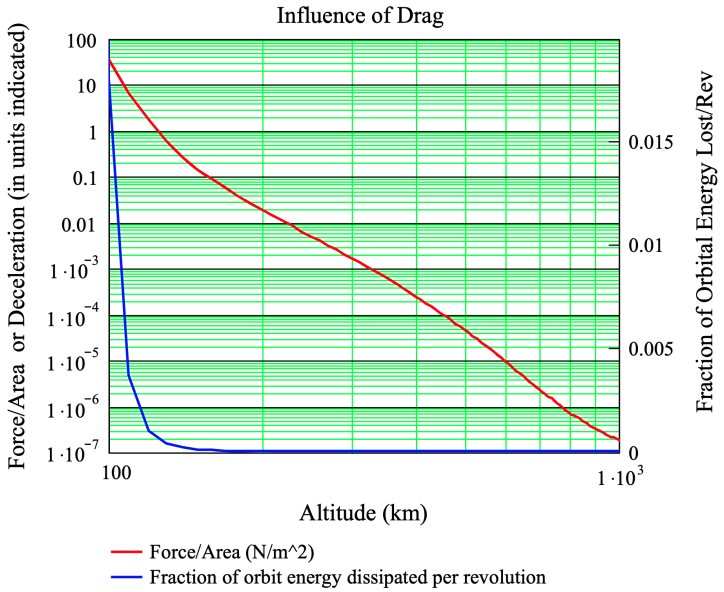


Figure 2.11.: Influence of atmospheric drag for a 1000 kg satellite with 100 m^2 drag area, and $C_D = 2.2$ in a Harris-Priester model atmosphere; from [18]

As Equation 2.55 illustrates, the satellites mass m, the drag area A, the drag coefficient C_D and the current state of motion, i.e. the relative velocity to the atmosphere $\vec{v}_{rel}$ must be known. One could assume that the evaluation is trivial due to the clearly defined equation. However, all parameters of Equation 2.55 vary over time and are not necessarily well-known from the beginning. [18]

$$\vec{a}_{drag} = -\frac{1}{2}\rho\frac{C_D A}{m} \cdot v_{rel}^2 \frac{\vec{v}_{rel}}{|\vec{v}_{rel}|} \tag{2.55}$$

Equation 2.55 covers three subproblems, two already mentioned at the beginning of the section. First, The density ρ variation and its involved factors are subject of Section 2.1. As shown, the density does not only depend on the time but also on other variables like EUV radiation, $F_{10.7}$, K_p and a_p. Each of these variables can introduce new uncertainties, e.g., limited prediction capabilities a priori or limited measurement capabilities a posteriori.

Second, the relative velocity $\vec{v}_{rel}$ is composed of not only the satellite's velocity. Commonly, one assumes that the lower atmosphere rotates with Earth, allowing a vector summation as shown in Equation 2.56. In the case of the upper atmosphere, wind models are required to determine the possibly significant effect of winds of more than several hundreds m/s onto the relative velocity $\vec{v}_{rel}$. Section 2.1 introduced several wind models like the HWM14. However, there remains a large uncertainty [18].

$$\vec{v}_{rel} = \vec{v}_{sat} + \vec{v}_{atmos} \tag{2.56}$$

Third, various variable properties are inherent to the satellite, like drag area, mass, and attitude, to name a few. These are often also combined and summarized in the ballistic coefficient B displayed in Equation 2.57 in order to reduce the number of uncertain parameters and concentrate on a single coefficient [66].

$$B = \frac{m}{C_D A} \tag{2.57}$$

In summary, finding a universally suitable solution for atmospheric drag is complex, especially since these problems condition and influence each other. Section 2.1 explains that errors in deriving density from acceleration measurements originate from uncertainty in the proper modeling of the ballistic coefficient and unknown atmospheric winds. This section shows the other side of the medal. Errors in choosing the suitable ballistic coefficient with the goal of ultimately determining the drag acceleration originate from errors in the underlying atmosphere models (wind and density). The subsequent Subsection 2.4.2 provides a more detailed investigation of how to determine the ballistic coefficient.

2.4.2. Modeling the Ballistic Coefficient

It becomes evident that the ballistic coefficient B behaves like a scaling factor when inserted into Equation 2.55.

In order to estimate the ballistic coefficient overall, two approaches can be utilized. Given the satellites' geometry, one can estimate the ballistic coefficient by analytical or numerical modeling of the aerodynamics. For example, the panel method realizes this approach. It splits the satellite's geometry into multiple smaller elements. For each element, the aerodynamic contribution can then be calculated in an analytical fashion. Thereby, it is computationally efficient, but it is unable to account for "particle–particle interactions or multiple particle reflections from surfaces" [66]. A computationally expensive approach to computing the satellite aerodynamic forces lies in the Monte Carlo simulation of rarefied atmospheric flows using a high-fidelity model of the satellite. [66, 67]

Alternatively, one can find the ballistic coefficient by using satellite tracking data over a given time span [67]. An optimal ballistic coefficient is calculated for every interval. In the second step, these values are averaged to find an overall suitable value. This approach is similar to deriving the density from the acceleration of in-flight satellites but does not require special knowledge about the shape of the satellite [5]. This fact is advantageous when working with debris whose precise geometry is unknown. Thus, it is also the technique used in this thesis, next to setting default values. It follows the so-called shooting method by [4, 2].

The main idea is minimizing the error between the observed and propagated state. Thus, one starts at epoch t_0 and propagates to $t_0 + \Delta t$, while the orbital state is known for both timestamps. One then compares the expected state at $t_0 + \Delta t$ and repeats the procedure. The procedure is repeated for every pair $(t_0, t_0 + \Delta T)$ until a user-defined accuracy criterion (e.g. a maximal standard deviation of 15%) is reached. The final B is the mean value of all calculated candidate ballistic coefficients. The framework in which the procedure is embedded is called Lifetime Assessment of Catalogued Objects (LASCO). It uses the above-described procedure if neither mass m nor drag area A are known. Because if they are known, the ballistic coefficient is simply calculated with Equation 2.57 and a $C_D = 2.2$.

Case ↓	Prerequisites	Description
1	Drag area A and mass m are known	$B = \frac{m}{A \cdot C_D}$ with $C_D = 2.2$
2	Perigee height $< 1000\,km$	Iteratively improve B by fitting the results of step-wise propagation to the observed data
3	Otherwise	$B^{-1} = 0.011\ \mathrm{m}^2/\mathrm{kg}$

Table 2.3.: Summary of the Ballistic Coefficient Determination as given by Bunte et al. [4]. One takes the first applicable case.

If not, the procedure is used for a perigee altitude smaller than $1000\,km$. Otherwise, the influence of the atmosphere is too small. In this final case, a default $B^{-1} = 0.011\ \mathrm{m}^2/\mathrm{kg}$ is assumed. [4, 2]. Table 2.3 summarizes the described procedure.

2.4.3. Orbital Lifetime Assessment and Reentry Evaluation

Reentry prediction or the broader topic of orbital lifetime assessment can be split into two phases: Long- and short-term reentry prediction. The former uses analytical propagators, as semi-analytical or even numerical propagators would be computationally too expensive. In cases where the reentry is more than a thousand years in the future, other methods, like analytical formulations, are considered to estimate the orbital lifetime [4]. Further, one must consider that every method degrades with increasing time due to accumulating errors. This holds especially true for numerical propagators. Therefore, our solution for reentry optimization, as presented in Part III, using `godot` is only suitable for short-term reentry predictions. It is used in the first place to get a rough estimate of the reentry epoch by only using, e.g., simple aerodynamics based only on the first perigee altitude and a simplistic density model or mean solar activity. Depending on the result of this propagation or orbit of the satellite (e.g., a high eccentricity), the techniques of short-term reentry prediction are used to gain a more accurate picture. This included the incorporation of, e.g., solar radiation pressure, diurnal density terms, or the overall use of a more complex atmosphere model. [4, 2]

Short-term reentry predictions can further be subdivided into automatic and manually conducted simulations. While the former is based on routines and available data, the latter can be more sophisticated - using dedicated sensing only for the reentry - with the goal of getting a reentry epoch and, additionally, even an estimate for the reentry location.

The prediction error E_p utilized in this context is listed in Equation 2.58 [2] and depends on the epoch of prediction, the predicted reentry epoch, and the knowledge of the actual reentry.

$$E_p = \frac{\mathrm{Epoch}_{\text{predicted reentry}} - \mathrm{Epoch}_{\text{actual reentry}}}{\mathrm{Epoch}_{\text{prediction}} - \mathrm{Epoch}_{\text{actual reentry}}} \tag{2.58}$$

The values of E_p are typically quite high, with errors of more than 80% a month before the actual reentry not being a rarity. Such a number is not surprising given the uncertainties in modeling the ballistic coefficient, atmospheric density, atmospheric wind, and predicting

the space weather *a priori*. [4, 2] Even with the previously mentioned manual efforts, values of 20% are considered state-of-the-art, with errors of below 10% - in case of the Gravity field and steady-state ocean circulation explorer (GOCE) - being exceptionally good [68].

3. Related Work

Section 2.1 gives examples of empirical data utilized in the process of deriving a new atmospheric model involving the choice of suitable proxy indices to model observed behavior. However, a new model, including a complex set of equations, does not necessarily need to be invented. Instead, re-fitting an existing model to new data by modifying its coefficients is also an option.

Searching across the literature, one can mainly find two distinct approaches to predict thermospheric density without the need to create an explicitly new set of equations. One is the re-calibration of empirical models, while the other uses neural networks as universal function approximators. Licata et al. follow the later approach of using machine learning operating on the density data from the HASDM SET database (see Section 3.4 and Subsection 4.1.3) [69]. However, this Chapter 3 only focuses on the primary approach of improving existing atmospheric models by continuously monitoring the state of the atmosphere given the available satellite data, which often comes with several challenges. For example, Subsection 2.4.2 teased the problem that the density ρ can only be accurately improved if other parameters, namely the ballistic coefficient, are well-known. The section will delve into the underlying techniques that form the basis of model calibration step by step.

3.1. The Basic Idea of Model Calibration

Figure 3.1 demonstrates the basic idea of calibrating an empirical atmosphere model. Rather than fixing the whole model at creation time t_0, one keeps the models parameterized given a set of coefficients. A fitting routine is then placed to periodically adapt the model's coefficients, given the recent observational data. Ideally, this process should run in near real-time to allow for the most accurate predictions. Hence, potential biases and shortcomings introduced due to a former overfitt to the reference data can be countered by continuous re-calibration. [70]

3.2. Early Empirical Model Calibration

Marcos et al. [71] were the first to publish the concept of empirical density calibration [5] using the tracking data of four satellites in orbits between 400 to 500 km during 1988-1989. Their basic ansatz to improve the density model lies in the comparison of true and fluctuating ballistic coefficients. They define the true ballistic coefficient by mass, drag area, and drag coefficient (inverse of Equation 2.57) and as to be known. This coefficient is set in relation to the fluctuating ballistic coefficient, which results from the step-wise orbit determination process using special perturbations and the Jacchia 70 density model. The resulting quotient allows them to determine if the predicted density of the model is under/ well/ over the actual value, i.e., the density model error. The improved model then allows for more reliable

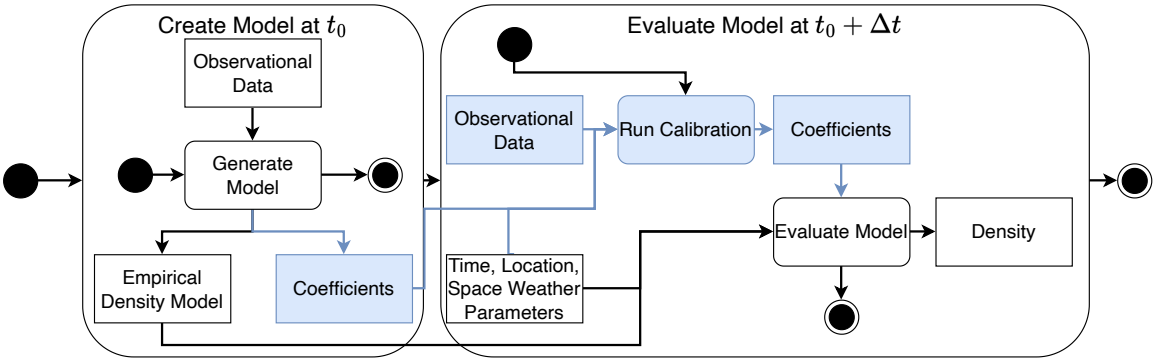


Figure 3.1.: The difference between a traditional empirical density model and a calibrated density model (including the blue lines) as UML Activity Diagram.

orbital state and decay predictions. Further, the results demonstrate that the quotient correlates with the in Subsection 2.1.2 reported phenomena of atmospheric density variation. [71]

Several of the in Section 2.1 presented models have already been subject to approaches for dynamical calibration. In the early 2000s, a collaborative US-Russian project aimed at the creation of height-dependent scaling factors with the ultimate goal of improving the density of the Russian GOST and the American MSIS00 models. Cefola et al. [72] used 250 LEO objects with perigees below 600 km and available TLEs. Generally, the experiment was designed to estimate the ballistic coefficients of selected satellites to conclude the density model error given their true ballistic values. The experiment involved the propagation of the satellites and, basically, the error estimation between uncorrected and corrected density models. They concluded that the accuracy can be improved using the TLE data. However, the approach does not generalize into the far future and only improves the prediction capabilities up to the order of several days into the future when compared to an uncalibrated model. From the computational perspective, it is interesting that they could not finalize the analysis for the MSIS00 model because its runtime is two orders of magnitudes greater than the Russian GOST model. An argument also used in the context of this work as an optimization of JB2008 would not be possible if the evaluation of single points took long. However, as of writing, all tested atmospheric density models can be executed fast with typical hardware equipment. A runtime comparison is presented later in Figure 7.1. We can confirm that MSIS00's evaluation takes more runtime than the evaluation of GOST2004. However, we don't have the concrete model implementations nor the hardware of two decades ago.

3.3. Deriving Densities from TLE Data

Two other relevant works are conducted by Shi et al. [73] and Doornbos et al. [70, 5]. The former scales the MSIS00 density model's output, while the latter works with multiple models such as MSIS-86/CIRA 1986 Neutral Thermosphere Model (MSIS86), MSIS00 and CIRA72 (a Jacchia model) and multiple approaches for the optimization. However, when using TLE data, both approaches make use of the density derivation algorithm described by

Picone et al. [74]. The idea is that the TLE data encompasses only the mean elements of the orbit. Thus, if no gravitational perturbations are included, only atmospheric drag and Solar Radiation Pressure (SRP) remain as the main drivers of a semi-major axis change. Given the assumption that solar radiation pressure is negligible below a certain altitude threshold, one can bring the semi-major axis change into relation to the atmospheric drag, i.e., the density or the ballistic coefficient. For instance, Gondelach et al. also have used this method to estimate the ballistic coefficient [75]. Shi et al. report this altitude threshold starting at approximately $600\,km$ since beginning from $800\,km$ SRP may be comparable to drag in size.

So, the mean semi-major axis a_M change is given by

$$\frac{d}{dt}a_M = -\frac{a_M^2}{\mu}B\rho F v^3 + \frac{d}{dt}a_M\Big|_{SRP} \tag{3.1}$$

with the ballistic coefficient B, the density ρ, the satellites velocity v, the gravitational constant μ and the dimensionless wind factor F. The wind factor is given by

$$F = \frac{|v - v_w|^2}{v^2} e_{v-v_w} e_v \tag{3.2}$$

with wind velocity v_w, and the unit vectors e of v and $v - v_w$. Equation 3.2 can be approximated with a relative error of less than 3% when Equation 3.2 is evaluated with HWM93 [74]. The approximation is given subsequent by Equation 3.3

$$F \approx \left(1 - \frac{rw}{v}\cos(i)\right)^2 \tag{3.3}$$

with the distance from Earth's center r, the angular velocity of Earth's rotation w, and the inclination of the orbit i. Sometimes F is trivially assumed to be 1. Equation 3.1 can be rewritten using $a_M = \sqrt[3]{\mu/n_M^2}$ where n_M is the mean motion. This reformulation yields Equation 3.4 given by

$$\begin{aligned}\rho_O(t_{ik}) &= \frac{\frac{2}{3}\mu^{\frac{2}{3}} B\left[n_M(t_k) - n_M(t_i)\right]}{\left[\frac{1}{2}n_M(t_k) + \frac{1}{2}n_M(t_i)\right]^{\frac{1}{3}} \int_{t_i}^{t_k} F v^3 dt} \\ &= \frac{\frac{2}{3}\mu^{\frac{2}{3}} B\left[n_M(t_k) - n_M(t_i)\right]}{\left[\frac{1}{2}n_M(t_k) + \frac{1}{2}n_M(t_i)\right]^{\frac{1}{3}} \sum_{t=t_i}^{t_k} F_t v_t^3 \Delta t}\end{aligned} \tag{3.4}$$

with the observational density ρ_O derived from the TLE elements at epochs t_i and t_k. It can be quickly evaluated using the n_M values from the TLE and the velocities given the SGP4 propagator. On the other hand, the model density ρ_M for the same time step is given by

$$\begin{aligned}\rho_M(t_{ik}) &= \frac{\int_{t_i}^{t_k} \rho F v^3 dt}{\int_{t_i}^{t_k} F v^3 dt} \\ &= \frac{\sum_{t=t_i}^{t_k} \rho_t F_t v_t^3 \Delta t}{\sum_{t=t_i}^{t_k} F_t v_t^3 \Delta t}\end{aligned} \tag{3.5}$$

with ρ as the density from the employed density model. Similarly to Equation 3.4, Equation 3.5 can be solved using the density model's output along the trajectory and the SGP4's results. The ratio λ given by

$$\lambda(t_{ik}) = \frac{\rho_O(t_{ik})}{\rho_M(t_{ik})} \tag{3.6}$$

can then be used to scale the density model. Some problems still need to be solved. The scaling factor λ has no associated height, latitude or longitude. It just refers to a coarse step between epochs t_i and t_k. For instance, Shi et al. solve this by introducing an equivalent height, basically weighting Equation 3.5. [5, 70, 73, 74]

In the context of this work, we stick to the raw $\lambda(t_{ik})$. We will try to step-wise decrease the ratio to 1 by optimizing either the ballistic coefficient B or the density ρ by respectively modifying the diurnal density coefficients of JB2008. While optimizing the ballistic coefficient yields good results, optimizing the diurnal coefficients does not work as initially hoped. These results are presented in Chapter 10, and the underlying fitness functions are investigated by Chapter 9.

3.4. The High Accuracy Satellite Drag Model

The High Accuracy Satellite Drag Model (HASDM) is based on the JB2008 density model and operated by the United States Space Force. It is arguably "the world's most accurate" [3] model for the purpose of forecasting the thermospheric density. It brings the average density modeling error down to $6 - 8\%$ "across all heights from 200 to 800 km" [76].

As mentioned, the model is based on the JB2008. Thus, it uses the new solar indices described in Subsection 2.1.3 and the Jacchia 70 model as a base. The calibration algorithm is called Dynamic Calibration Atmsophere (DCA). It uses the trajectory data from 75-80 satellites in various different LEOs (nowadays potentially more, i.e., the precise number is unknown) to estimate 13 correction coefficients for the inflection temperature T_x and the nighttime minimum exospheric temperature T_c. Effectively, these two corrections to T_x and ΔT_c are realized by spherical harmonic expansion [77]. T_x is corrected by 2×2 (9 coefficients) and ΔT_c is corrected using 1×1 (4 coefficients) [78]. It is a weighted least square differential correction "that simultaneously solves for global density corrections and a state vector for each calibration satellite" [79].

The density correction reflects the "dynamic changes in the diurnal and semi-diurnal [density] variations" [79]. Its calibration data originates directly from the American Space Surveillance Network (SSN) without further preprocessing. This is a significant fact that allows them to run the calibration once every three hours. The calibration coefficients can be predicted up to three days into the future as a function of solar and geomagnetic activity. This prediction procedure is included in HASDM, allowing to propagate satellites respectively for the next interval. [3, 77, 76, 79, 78, 5]

HASDM is not available to the scientific community, nor are the calibrated coefficients and precise SSN data. However, the computed density values were released to the scientific community as part of the SET HASDM Database. The database is presented in detail in Subsection 4.1.3.

4. Implementation Theory & Data Methodology

This Chapter 4 introduces the employed services and origins of the data utilized over the course of experiments. Additionally, Chapter 4 explains the reasoning behind the decision to use a specific source.

4.1. Data Sources

This section briefly describes the various data sources employed in this work while also discussing the applicability and limitations.

4.1.1. Space-Track and the Two Line Element Set (TLE)

The Two Line Element Set (TLE) data utilized in this work is provided by the 18th Space Defense Squadron via their public service Space-Track[1].

Line Number	NORAD Catalog ID	Classification	International Designator	Epoch	First Derivative of the Mean Motion	Second Derivative of the Mean Motion	Starred Ballistic Coefficient in Earth Radii^{-1}	Ephemeris Type	Element Set Number	Checksum % 10
1	43600	U	18066A	23209 .73257513	.24348956	20945-5	58706-4	0	999	9
2	43600		96 .7031	214 .90230058712	264 .1723	95 .35041	6 .56201943		2855	8 1

Line Number	NORAD Catalog ID	Inclination (in degree)	Right Ascension of the Ascending Node (in degree)	Eccentricity (assumed decimal point in front)	Argument of Perigee (in degree)	Mean Anomaly (in degree)	Mean Motion (in revolutions per day)	Revolution Number at epoch	Checksum % 10

Table 4.1.: Example TLE of the satellite *Aeolus*. The single entries are gray-shaded in an alternating fashion. White columns are not used.

A TLE consists of two lines, each with 69 characters. The U.S. Strategic Commands constructs the TLEs by estimating the orbital elements in the TEME reference frame (see Subsection 2.2.2) using observational tracking data and the SGP4 model (see Subsection 2.3.2) [70, 5]. As discussed in Subsection 2.2.2, there are some unknown factors involved when using TLE, e.g., that it is unknown to which reference epoch TEME refers. We ignore these problems. In the following, we assume that a precision of just over one kilometer [48] for the input data is precise enough for the use-case of atmospheric optimization.

[1] `https://www.space-track.org`, last accessed: 17.12.2023

Table 4.1 gives an example for a TLE of the satellite *Aeolus*. Most of the entries are self-explanatory. Thus, the following paragraphs focus only on some rather than all. The unique identifier of a satellite, which is also utilized in the later provided implementation, is the NORAD Catalog ID where NORAD stands for North American Aerospace Defense Command. In contrast, the International Designator provides a more illustrative way of identifying a satellite. In order of appearance, it comprises the last two digits of a satellite's launch year, the 3-digit number of its launch in the given year, and a fragment identifier of the initial launch vessel. The epoch of the TLE consists of the last two digits of the year, the Day of Year (DOY), and the fraction of the epoch's day.

As previously mentioned, the TLE is specifically designed to be used with the SGP4 model. It models the atmospheric drag given the starred ballistic coefficient B^* from the TLE. It is given in inverse Earth Radii R_E^{-1}. In Table 4.1, $B^* = 58706 \cdot 10^{-4} R_E^{-1}$. It is an adjusted value of the actual ballistic coefficient B using the reference value of the atmospheric density ρ_0 at one Earth radius [48]. The B^* is estimated by fitting SGP4 to the estimated orbit. It compensates for SGP4 model deficiencies and cannot be directly interpreted as a physical quantity. The direct use in non-SGP4-based models is therefore not recommended. However, one can use the B^* to generate an initial guess of the actual ballistic coefficient B using Equation 4.1, as e.g. conducted by Gondelach et al. [75].

$$B = \frac{R_E \cdot \rho_0}{2B^*} \text{ with } \rho_0 = 2.461 \times 10^{-5} \frac{kg}{m^2} \tag{4.1}$$

This work employs the starred ballistic coefficient and the relation given by Equation 4.1 in order to determine an initial ballistic coefficient in case satellite characteristics from the Database and Information System Characterising Objects in Space (DISCOS) are not available. Equation 4.1 has its flaws since unmodeled forces can lead to inconsistent values like a negative B^* inside the TLE [80]. In this case, the default value of $B^{-1} = 0.011 m^2/kg$ of the shooting method is taken as initial value. The subsequent Subsection 4.1.2 presents a potential data source for these values.

4.1.2. DISCOS and Satellite Characteristics

The Space Debris Office of the ESA maintains DISCOS. It is publicly accessible via a Web Interface[2]. The database behind aggregates the characteristics of objects originating from a variety of different catalogs and sources, such as Space-Track's TLE data, NASA's history of on-orbit satellite fragmentations, ESA's own data, and launch information from providers. [81] Thus, it facilitates the retrieval of physical attributes like mass and drag area of a satellite due to its unified interface to otherwise distributed data.

As presented in Subsection 2.4.2, LASCO and its underlying mechanic to determine the ballistic coefficient ("shooting-method") rely on DISCOS for the retrieval of drag area and mass [81]. This work uses DISCOS in its newest iteration DISCOS3 [82], which improved the power of the employed underlying relational model and its API to retrieve the satellite characteristics in order to calibrate the ballistic coefficient.

[2] `https://discosweb.esoc.esa.int`, last accessed: 20.12.2023

4.1.3. The HASDM Density Set

Section 3.1 introduces the idea of continuously improving the density prediction by continuously calibrating the model using recent observational data. It also introduced HASDM and the idea behind the model. The model itself is classified and not available to the scientific community. However, the computed density data was released to the public in the date range from the 1st of January, 2000, to the 31st of December, 2019, with a step size of three hours and for altitudes from 175 km to 825 km. Thus, it covers the partial solar cycle 23 and the full cycle 25. The density was evaluated for the grid in steps of 25 km altitude, 10° of latitude, and 15° of longitude. *Space Environment Technologies* hosts this HASDM density set[3]. [3]

Initially, it was planned to use the HASDM density set also to improve the coefficients of the JB2008 model in this work. However, this plan was dropped in favor of laying a stronger focus on the calibration using TLE data. Instead, we use the HASDM density data only as validation data set across the implementation and to evaluate the performance of the improved density model. In view of the fact that HASDM is calibrated near real-time, its data can be considered a gold standard of long-term density data. These concrete use-case scenarios are presented later in Chapters 7 and 10.

4.1.4. Solar Indices

Subsection 2.1.3 introduces the proxy indices utilized in the subsequently presented implementation.

This work employs the solar and geomagnetic activity data, *SOLMAG*, provided publicly by the *European Space Agency's Space Debris Office*[4] for all atmosphere models except the JB2008. The concrete index file is named `fap_day.dat`.

The JB2008 is fed using the solar activity and geomagnetic activity data provided by *Space Environment Technologies*[5]. The geomagnetic Dst is not directly utilized, but rather the already precomputed and provided ΔT_c from it. The concrete index files are `SOLFSMY.txt` and `DTCFILE.txt`.

Another topic to be discussed is the use of interpolation between the measurements of these indices since all these indices are provided in discrete time steps. This question especially holds for the geomagnetic indices because they are provided in a three-hourly fashion (or even one-hourly in the case of Dst). Opinions differ on this. Some argue that interpolation conflicts with the original definitions of models and indices. Others, on the other hand, believe that a continuous representation works better with the natural rhythm. Since continuous functions define the density in these models, continuous input values should also be necessarily sufficient. [18]

In this thesis, the discrete index values are used. Interpolation is only performed for the ΔT_c values since they are already a product of the actual discrete Dst index. This decision does not imply that the model input requirements are not fulfilled, e.g., MSIS requires an average of eight 3-hourly a_p from 12 to 33 hours prior to the current time. However, when computing such an average, the index is not interpolated to the true $12, 15, 18, .., 33$ hours

[3] `https://spacewx.com/hasdm/`, last accessed: 20.12.2023
[4] `https://sdup.esoc.esa.int/solmag/`, last accessed: 11.12.2023
[5] `https://sol.spacenvironment.net/JB2008/index.php`, last accessed: 11.12.2023

before but is used in the respective available discrete values.

This issue also goes hand in hand with using the 81-day center or the moving 81-day average. This thesis generally uses the 81-day moving average. This average has the advantage of data availability, especially for a priori prediction. An exception is the JB2008 model, where the centered averages and all inputs are distributed in the above-mentioned separate index files, ready to be directly used. Another exception is the GOST2004 model, which requires as input a specially weighted 81-day average assigning the most recent day a weight of 1.0 and gradually decreasing the weight while going into the past to a weight of 0.5.

4.2. ESA's flight dynamics library: Godot

`godot` is the flight dynamics library of the European Space Agency (ESA) employed and developed at European Space Operations Centre (ESOC). `godot` is an abbreviation of *General Orbit Determination and Optimisation Toolkit*. Regarding functional requirements, it was designed to perform "orbit related computations for estimation, optimization, and analysis of orbits for mission analysis and in-flight operations" [58]. In contrast, the main non-functional requirements focus especially on extensibility and modularity in order to provide a high-level system of abstraction to solve arbitrary problems with low-effort additions to the core system. It is written in `C++` with a separate `pybind11` interface exposing its functionality to `Python`. The `Python` interface is called `godotpy`. `godot` is operational software. Thus, it is actively in use and the contributions to the core are administered by ESOC [83]. However, as mentioned, the massive focus on modularity facilitates the process of writing extensions independent of the actual core library. These so-called plugins are shared libraries linked to `godot` at runtime extending, e.g., its selection of atmospheric model like the plugin presented in Chapter 5. The extension functionality is largely based on the creational *Prototype Design Pattern*, i.e., new implementations from a plugin are registered in `PrototypeRegistry` from which they are then accessible to the client user. [50, 58, 83]

The development is grounded on experience with previous software in the area of flight dynamics like *NAPEOS*, *AMFIN*, and *INTNAV*. The formerly mentioned legacy software is utilized to verify the correctness of the new implementation. The reference test cases of *NAPEOS*, which include a series of propagation scenarios around Earth, were utilized in the context of this work to verify the correctness of the new atmosphere system. This testing procedure is detailed in Chapter 6.

4.2.1. Godot's Architecture

Figure 4.1 provides a high level UML component overview of `godot`. It is compiled from the source code as of the time of writing. Given the diagram, `godot` could also be described as an instance of a layered architecture style. The foundation, marked in blue, are the `core` components. Independent libraries built for a single purpose like `tempo`, which offers timescale and time-system data types and their associated behavior (the realization of Subsection 2.2.3). Alternatively, they include wrappers facilitating the use of, e.g., the linear algebra library `eigen`[6], as it is the case for the `linalg` component. Further, namely

[6] `https://eigen.tuxfamily.org/`, last accessed: 10.01.2024

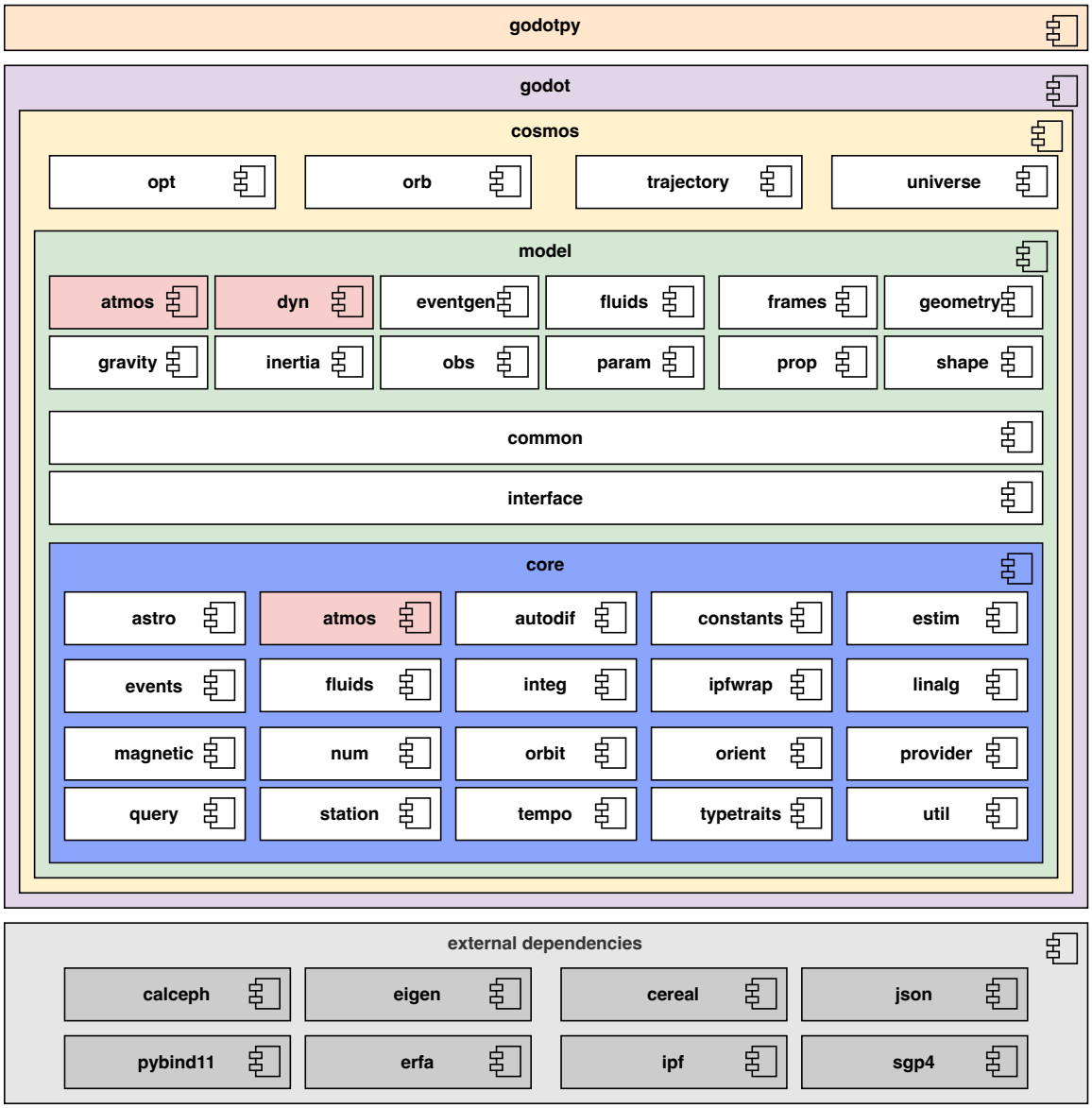


Figure 4.1.: A UML Component Diagram of `godot` as of version 1.3. The interfaces in-between the components are omitted on purpose to reduce complexity. Instead, the components are vertically ordered in a layered manner with different colored components using the layer beneath/ inside. For example, `model` uses the components of `core`. The component `model` contains the components `interface` and `common`. Both of them are displayed enlarged because they respectively define and implement the core design philosophy: The `TimeEvaluables`. Components colored in red are the primary target of the implementation presented in Part II.

to mention due to them being used in this work are: `astro`, `autodif`, `atmos`. The `astro` component provides functionality regarding astrodynamic-related quantities like coordinate transformations or state representation algorithms. The `autodif` cleverly makes use of

C++'s expression templates to provide computational capabilities for determining partial derivatives. It also interfaces to `eigen` using the custom type `xdouble`, which tracks the gradient. Finally, `atmos` provides the means of determining atmospheric density and winds before it is replaced by the new `godot::model::atmos`, presented in Section 5.2, one layer above. The old design of `atmos` is examined in detail by Subsection 4.2.2.

The mid layer is the `model` layer. It can be characterized as the most fundamental layer from a software design point of view. It contains the definitions (component `interface`) and basic implementations (component `common`) of the `TimeEvaluable` class. The more significant share of parameters in `godot` are modeled as `TimeEvaluable`. It provides an overloaded method taking an `Epoch` or `XEpoch` and calculates the value in dependence of that epoch. If the epoch argument is a `XEpoch`, gradients are tracked, i.e., `autodif` is employed. Figuratively spoken, a chain exists through a whole computational process capable of tracking the derivatives if desired. Another central aspect of the `TimeEvaluable` implementation contract is `Events`. A class usually needs to implement both interfaces. By implementing `Event`, the values are "observed" (in the figurative sense), and,e.g., a propagation can be interrupted if, e.g., zero values appear. Other central components of the `model` layer are `atmos`, `dyn`, `frames`, `geometry`. The `frames` component keeps track of the reference systems, axes, and points. Figuratively exemplified, the points do not directly result in, e.g., `Vector3TimeEvaluable`, which yields the cartesian position. Instead, the properties are stored in an abstract way, like the concept of pointers in computer memory. Thus, a created point is like a null pointer in the beginning. This behavior is, for example, of great use in the Dynamics component `dyn`. At creation time, the point to be propagated is yet unknown. Thus, our `AtmosphericDrag` is instantiated by a `Vector3TimeEvaluable` with the actual two evaluation points being a placeholder. The aliasing to the actual points is deferred until the concrete evaluation happens. The `geometry` component provides the classes to create the aforementioned `Vector3TimeEvaluable` from concrete axes and two points. The dynamics component, abbreviated with the namespace `dyn`, contains the various dynamics like - previously mentioned - `AtmosphericDrag` or SRP. Finally, the `atmos` component of the `model` layer is one final implementation artifact of this work and is presented in Chapter 5.

The top layer is called `cosmos`. It unifies the layers below and provides the means to set up a universe of forces and orchestrate a propagation with the whole model of `godot` as defined by the universe. It also provides the capabilities to perform optimization and estimation using `pagmo` [84]. We do not make use of the latter due to five reasons. First, our data for optimization resides mainly on the `Python` side. Second, an explorative implementation is more straightforward and realized via `Python` and allows for faster prototyping of approaches. Third, the documentation of how to perform and use the optimization library still needed to be fully fleshed out at the time of writing. Fourth, making all coefficients of JB2008 optimization and exposing them turned out to be a difficult task, which was overcome by injecting the parameter change past `godot` via a dedicated `Python` binding. Fifth, we use the optimization that relies on the `Trajectory` class, which adds the propagated properties to the universe instance. This accumulates over time in memory. We switched to the `BallisticPropagator`, which does not insert its data into the universe and provides a more flexible lightweight approach in terms of exchanging the atmospheric model. Nonetheless, the `Trajectory` is class is also utilized due to being more robust in terminating when a reentry event occurs, as discussed in Subsection 8.3.1. The implementation and optimization

is presented in detail in Part III of this work.

4.2.2. The Issues of the Current Atmospheric Component

The state of the atmospheric component previous to this thesis resides among the `core` libraries of `godot`. This work fundamentally revised this component, as presented in Section 5.2. Figure 4.2 depicts the original implementation. It basically uncovers five distinct anti-patterns of software engineering:

- The `AtmosphericDrag` has, in theory, full access to the frames system of the universe violating *data encapsulation* best practices. Thus, an unnecessary *coupling* to the `frames` component.
- The `Atmosphere` class stores in its base variant properties inherent to the celestial body, violating the principle of *separation of concerns* and enforcing the existence of these might-not-be-needed attributes in its subclasses.
- The `Atmsophere` class calculates density and wind violating the *single responsibility principle*.
- The subclasses of `Atmosphere` are by design forced to implement wind, density, and the partial derivatives for both. However, not all of them do. Instead, they return plain zero values if the functionality is not provided. This circumstance violates the contract obligated by the superclass. Thus, it violates *Liskov's substitution principle*.
- The subclass `Nrlmsise00Atmosphere` stores several parameters as plain values violating `godot`'s own `TimeEvaluable` based core design philosophy. Further, it handles file input for solar and geomagnetic activity on its own, violating again *separation of concerns*.

The subsequent in Chapter 5 presented implementation refactors the approach in order to comply with `godot`'s clever design principle of `TimeEvaluables` and bring the atmospheric component in line with other components of its kind.

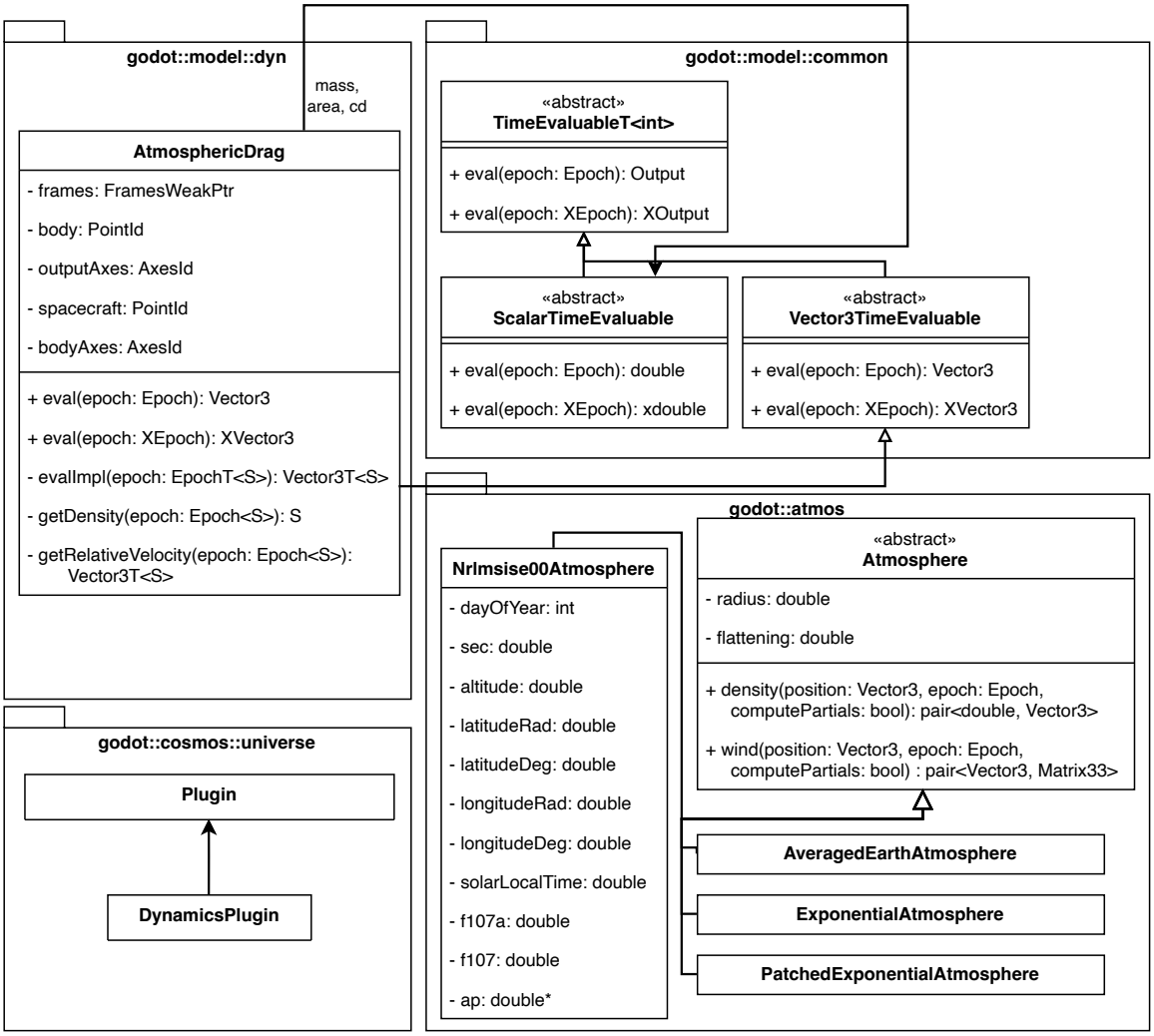


Figure 4.2.: A UML Class Diagram of the showing the interplay between `godot::atmos` and `godot::model::dyn`. The implementation details of `Atmosphere`'s subclasses are hidden with exception of the class `Nrlmsis00Atmsophere`.

Part II.

The Godot Atmosphere Plugin

5. Architecture

The previous Subsection 4.2.2 showed the issues of the initial atmospheric component in `godot`. This Chapter 5 presents the `godotAtmsoph` plugin and the refactoring inside `godot` required to make the new implementation compatible. The engineering process converging to this solution consisted of two steps. In the first iteration, which is not discussed in detail, the old atmospheric system, precisely as depicted by Figure 4.2, was kept. A new atmosphere called `FlexAtmosphere` was subclassed from `Atmsophere` and added to the inheritance hierarchy. It administered its own registry for density and wind models, which could be arbitrarily switched and combined, as they were stored as `std::function`.

This first approach was given up in favor of a complete overhaul of the overdue atmospheric component. Thus, the second approach presented in this Chapter 5 does not only create an extension to `godot`'s atmosphere system, but instead also redesigns the base system inside `godot`'s core to the TimeEvaluable-based approached presented in Chapter 5.

The verification process is written down in Chapter 6, while Chapter 7 presents the results and required configurations obtainable with the work presented in the subsequent sections.

In the following, if we refer to atmosphere models, we imply density and wind models. *Design Pattern* and *Structural Choices* are emphasized in *italics*.

5.1. Overview of Components

Figure 5.1 portrays the realization of the density and wind model inclusion as a UML Component Diagram. Each outermost component ensembles its own `CMake` target with the exception of the `atmosph` component, i.e., these components are independent libraries (even though the linkage type differs). The `atmosph`'s components are, respectively, individual static libraries. This process facilitates the exclusion of atmosphere models with a more restrictive copyright.

Overall, the `atmosph` component ensembles the density and wind models, which are completely written in `Fortran`, even the newer ones like MSISv2 published in 2021. The concrete `Fortran` standard differs from model to model. Some of them already use Fortran-90, while others use the Fortran-77. The component also includes for each atmosphere model function a wrapper written in `Fortran`, which makes use of the Fortran-ISO-C-Binding functionality to provide a binding to the `C/C++` world. `Fortran` 2003 introduced the ISO-C Binding to the Fortran standard.[1] It provides type safety for interoperability with `C/C++` and clarity about the function name and how to reference it from `C/C++`. The inner components of `atmosph` are named after the models they respectively contain, with the exception of the `sdo-atmosph` component. The content of this component originates from an internal repository of ESA's Space Debris Office (SDO) and encloses all remaining density and wind

[1] `https://gcc.gnu.org/onlinedocs/gfortran/ISO_005fC_005fBINDING.html`, last accessed: 10.01.2024

models listed in Figure 2.5. The `jb2008mod` component contains the latter for optimization purposes employing the density model.

The single components of `atmosph` are statically linked to `godotAtmosph`, which itself consists of four major components:

- The `input` provides a `TimeEvaluable`-based access to solar or geomagnetic data stored in index files and is investigated by Section 5.3
- The `model` component uses the models from `atmosph` and is investigated by Section 5.4
  - The `density` component encloses each density model into a `ScalarTimeEvaluable` and provides the concrete inputs to it
  - The `wind` component encloses each wind model into a `Vector3TimeEvaluable` and provides the concrete inputs to it
  - The `common` component provides shared methods utilized across `density` and `wind` component to bring the inputs into the requested format (e.g. correct delay for solar geomagnetic activity data)
- The `util` provides functionality non-specific to `godotAtmosph` like string manipulation or higher level functions for caching (similarly to e.g. the pythonic `cached` decorator)
- The `plugin` does not contain any functionality, but it only purpose is to register the density and wind evaluables in `godot::model::atmos`

The above ensemble of `godotAtmosph` is then dynamically linked to `godot`. The interior of the component `godot`, as depicted in Figure 5.1, demonstrates which components are required by `godotAtmosph` as dependencies. These include functionality for timescales, coordinate transformation, constants, and the partial derivative library. Next, it shows how the `godotAtmosph` component interfaces to the new atmosphere component in `godot`, which is detailed in the subsequent Section 5.2.

A fourth component resides on the left side of Figure 5.1: `godotAtmosphPython`. This module provides direct access to the atmosphere models integrated into `godotAtmosph` while still preserving the unified interface, i.e., only position, epoch, and solar geomagnetic activity index files are required for calculation without the overhead of taking care of `godot`'s internal systems. Further, the transformations and everything else are taken care of by `godotAtmosph` and hidden. Section 5.5 provides a detailed look at this module and why it was made in the first place.

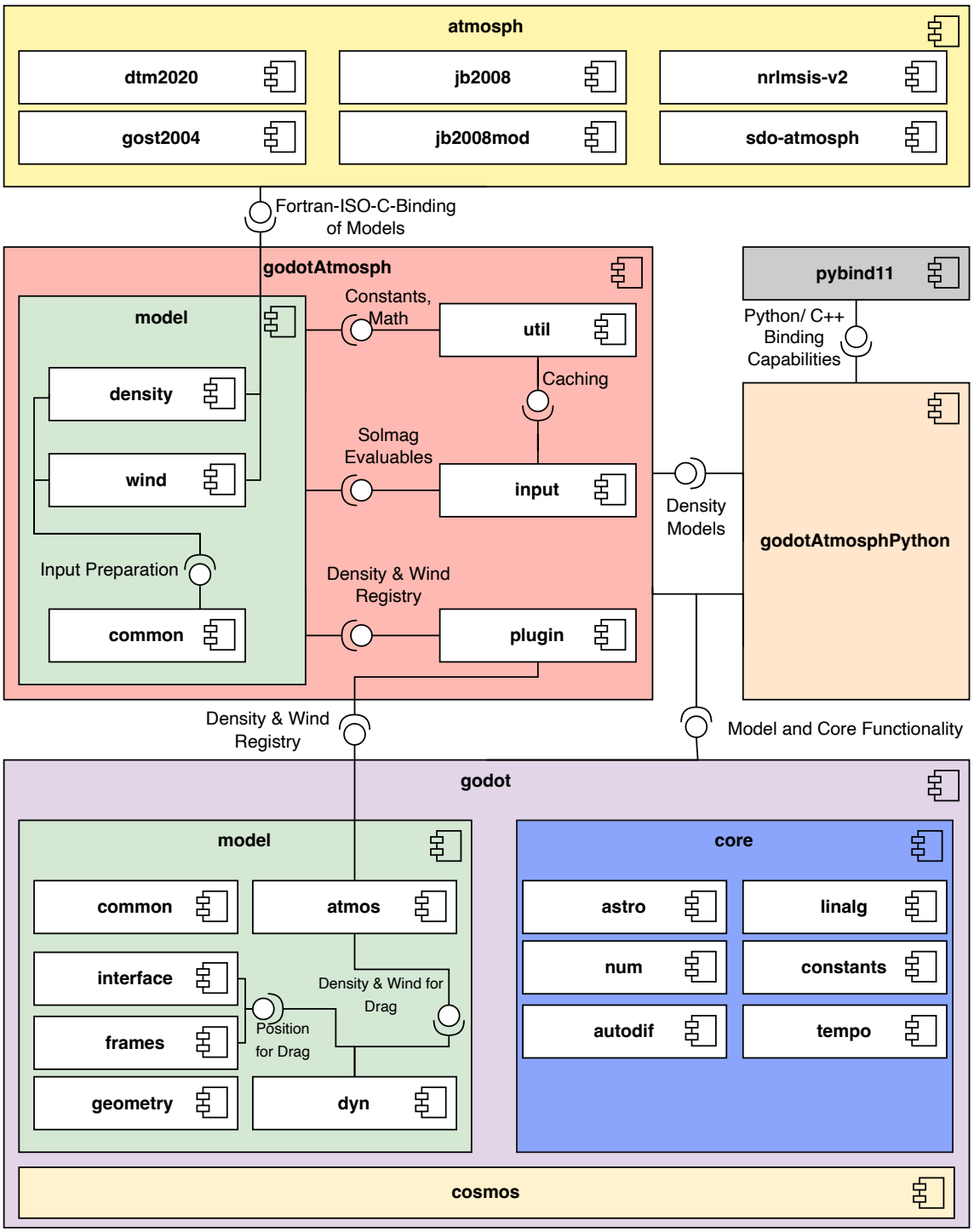


Figure 5.1.: A UML Component Diagram of `godotAtmsoph`. The bottom shows only the utilized components of `godot` (colors match with Figure 4.1), while the upper two third show the actual `godotAtmsoph` plugin, the statically linked `atmosph` library, and the custom Python Binding.

5.2. The Refactored Model in Godot

Before godotAtmosph can be described in detail, a detailed look at the redesigned atmospheric component of godot is necessary.

We describe the new atmosphere component in godot depicted in Figure 5.2. It contrasts the former architecture presented in Figure 4.2 and shows the improved refactored solution. The structure of Figure 4.2 persists only in some small remnants. The former Atmosphere class and all subclasses are split into a DensityModel and WindModel - with the latter being omitted in case the Atmosphere only implemented an empty method. The densities and wind models are administered by their respective registries in the new AtmosphericFeaturesPlugin. AtmosphericFeaturesPlugin provides a DensityModelRegistry and WindModelRegistry following the *Prototype Pattern*. The new density and wind evaluables defined by godotAtmosph are registered in it and ready to be instantiated from there. Thereby, they are available to the dynamics (denoted with dyn) component and ultimately inside a propagation/ the godot::cosmos library. The new dynamic AtmosphericFeaturesDrag uses the DensityModel and WindModel. It is named AtmosphericFeaturesDrag instead of the already utilized name AtmosphericDrag since the old system will continue to co-exist for some time to facilitate the transfer to the new system. The derivatives and integration to autodif are provided by numerical means in the top superclass DensityModel or WindModel.

AtmosphericFeaturesDrag does not require any direct access to the frames system. Instead, it directly gets an Vector3TimeEvaluable called position using godot's geometry component from the model layer. Hence, this resolves the unnecessary exposure of the whole frames system into a small dynamic.

Overall, all problems listed in Subsection 4.2.2 related to *separation of concern*, *single responsibility* and of high importance *Liskov's substitution principle* are solved by these new abstract interfaces. Further, the new approach blends in seamlessly with godot's core design philosophy of using TimeEvaluables for every parameter, including density and wind now being a TimeEvaluables as well.

5.3. Input of Godot-Atmosph

Before continuing with the model of godotAtmosph, a glance over input is required due to the dependency direction in-between. Figure 5.3 shows a UML Class Diagram of the input component and its major white-box dependencies (due to inheritance) into godot. The input component provides the means to model geomagnetic and solar indices by means of a ScalarTimeEvaluable. Thus, we adapt these indices into the universal system of godot.

Listing 1 gives an example of a typical index file. These standard text files splitting the values by white spaces are widely employed across the scientific community, at least from the investigative perspective of this work. In our use case, three such text files needed to be integrated, as mentioned in Subsection 4.1.4: fap_day.dat, SOLFSMY.txt, and DTCFILE.txt. The files' schemas are known at compile-time and do not change. The resolution of this engineering problem is the class IndexReader<T...> using a variadic template. It can be arbitrarily typed. Columns that are not needed can be denoted with the empty struct Discard<size_t> with size_t being the number of consecutive columns to

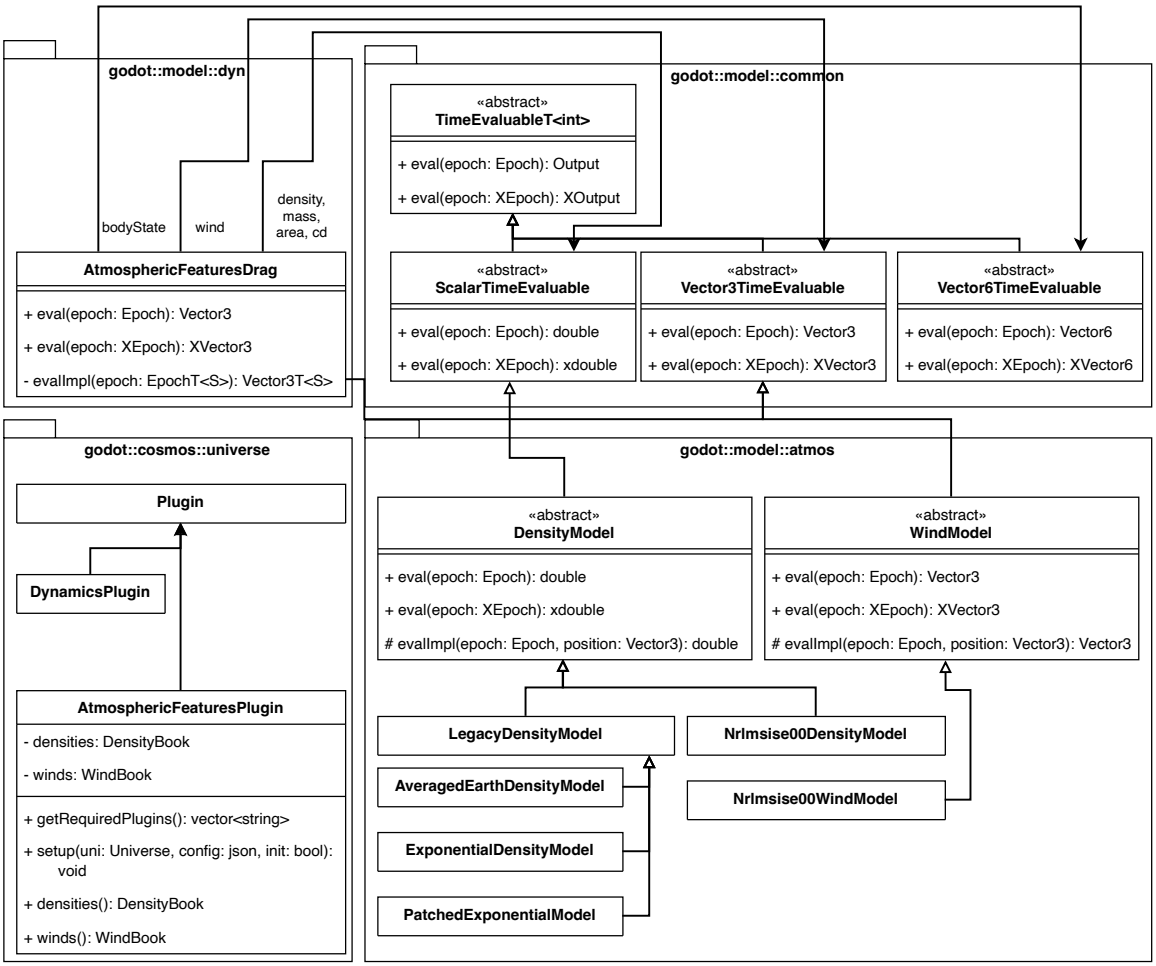


Figure 5.2.: A UML Class Diagram of the namespace `godot::model::atmos` and the major additions around presenting the new representation of atmospheric drag.

```
1    #d/mm/yyyy F10 F3M SSN Ap  3-hr Kp-Indices   last update: 23-APR-2023
2    #--------- --- --- --- --- ---------------   27d forecast -> nominal
3    01/04/1957 218 208 198 023 4o4+4o3+3o3+4-4-
4    02/04/1957 202 207 215 015 4+2-3o3o2+2+3+3+
5    ...
```

Listing 1: An example of the `fap_day.dat` containing the solar and geomagnetic indices distributed by ESA's SDO

discard. For example, in Listing 1, the sunspot number `SSN` column is discarded. Listing 2 displays the full instantiation. Every type must provide an `istream` operator. In the case of `godot::tempo::Date`, the `istream` operator was extra implemented as part of `godotAtmosph`. A further advantage, the methods `getColumnWise()` and `getRowWise()` are cached. Thus, repeated access - if required - does not imply further file reads, greatly

improving the runtime.

```
class SdoSolmagReader
  : public IndexReader<godot::tempo::Date,
    double, double, Discard<1>, double, DailyApIndexInstream>;
```

Listing 2: The declaration of the `SdoSolmagReader`

The common superclass `IndexReaderBase` solves the problem that templating is static and does not allow for polymorphism. The only method the subclasses need to implement is `getEvaluables()` due to the fact that the names cannot automatically be determined, as well as which kind of `ScalarTimeEvaluable` shall be utilized. The `IndexReader<T...>` is the heart of the `input` component and was used due to its versatility also in the context of testing, as density data is distributed in index files similarly to the one presented in Listing 1, too.

Figure 5.3 also depicts the class `IndexTimeEvaluable`. It offers a `TimeEvaluable` based access to the underlying data. It only stores the starting epoch and the step size (in seconds) to reduce the memory footprint acting like a *flyweight*. The two subclasses basically either return the last discrete value (`DiscontinousIndexTimeEvaluable`) or a linearly interpolated value (`ContinousIndexTimeEvaluable`). In September 2023, there was a discussion to include this functionality into core `godot`. However, at that time, internal plans were already on the way to include an even more universal `TimeSeries` class. Thus, the functionality depicted here is exclusive to `godotAtmosph`.

The last class of `input` is the `IndexTimeEvaluableBuilder`. It facilitates the creation of the actual `TimeEvaluable` employing the *Builder Pattern*. The client only needs to hand over the file paths. The rest is taken care of, including caching the evaluables and error handling in case of duplicated data. Hence, from the outside, the `IndexTimeEvaluableBuilder` could either be described as an instance of a `Proxy Pattern` or a `Singleton`.

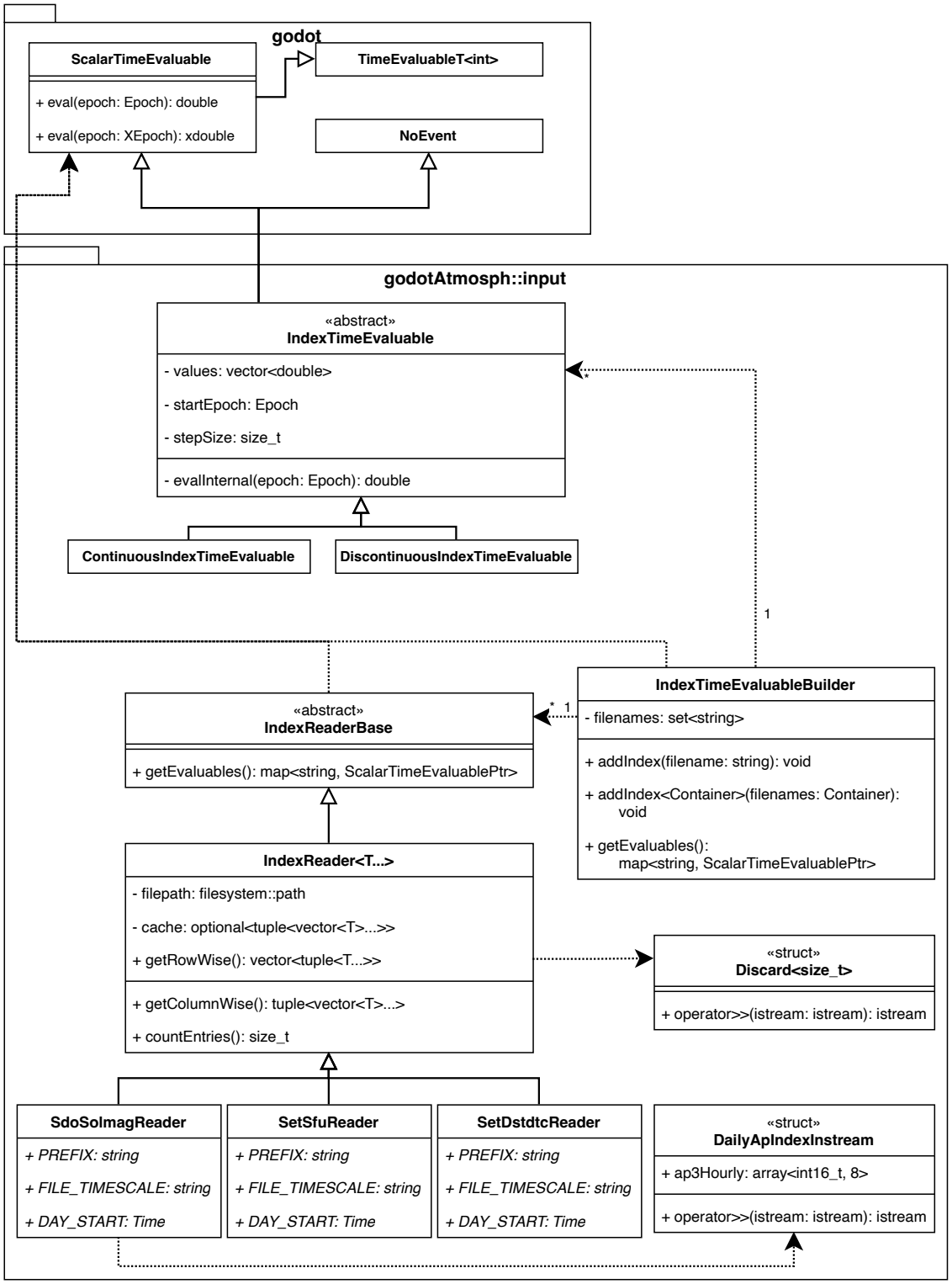


Figure 5.3.: A UML Class Diagram of the namespace `godotAtmosph::input`. Types are abbreviated using the namespaces `std` and the respective ones from `godot`. `Templating` is depicted as inheritance (e.g. the `Reader` all inherit from `IndexReader<T...>`, while in practice the concrete templating creates for each subclass its own distinct superclass).

5.4. Model of Godot-Atmosph

Figure 5.4 depicts the `godotAtmosph::model` and how it relates to `godot::model::atmos`, previously seen in Figure 5.2.

In order to obtain a degree of independence, `godotAtmosph` subclasses its own `DensityModel` and `WindModel`. This *bridge* ensures that `godotAtmosph` keeps working even if the actual base class should change with a newer version.

Figure 5.4 lists the concrete density and wind models inheriting from their abstract counterparts. Each of these subclasses overwrites the `evalInternal(epoch, position)` method. In each of these methods:

1. The model specific transformations are conducted, e.g., including the weighting of solar and geomagnetic activity data exclusive to that specific model
2. The Fortran-ISO-C function of `atmosph` is called
3. The output is converted into the right unit

The `MSIS` and `HWM` models are unique, as they provide the same signature over multiple model generations with the only distinct characteristic of the employed floating point precision. In these cases, a standard template superclass is introduced while the subclass is templated with its concrete `std::functional<FloatType(...)>` calling the respective model.

The `common` module contains a selection of utility or often used shortcuts required for the models but not provided by `godot` at the implementation time. To summarize, the `model` component of `godotAtmosph` mainly serves the purpose of transforming a signature like Listing 3 to a more usable method like Listing 4 by converting all different inputs and outputs to the concrete required ones. Further, the `model` provides the integration into `godot`'s `TimeEvaluable`-based system.

```
1 extern "C" void c_jb2008(
2   const double &amjd, const double *sun, const double *sat,
3   const double &f10, const double &f10b, const double &s10,
4   const double &s10b, const double &xm10, const double &xm10b,
5   const double &y10, const double &y10b, const double &dstdtc,
6   double *temp, double *rho
7 );
```

Listing 3: The signature of JB2008's Frotran-ISO-C Interface.

```
1 double evalImpl(const Epoch &epoch, const Vector3 &position);
```

Listing 4: The signature of a JB2008 evaluation in `godotAtmosph`.

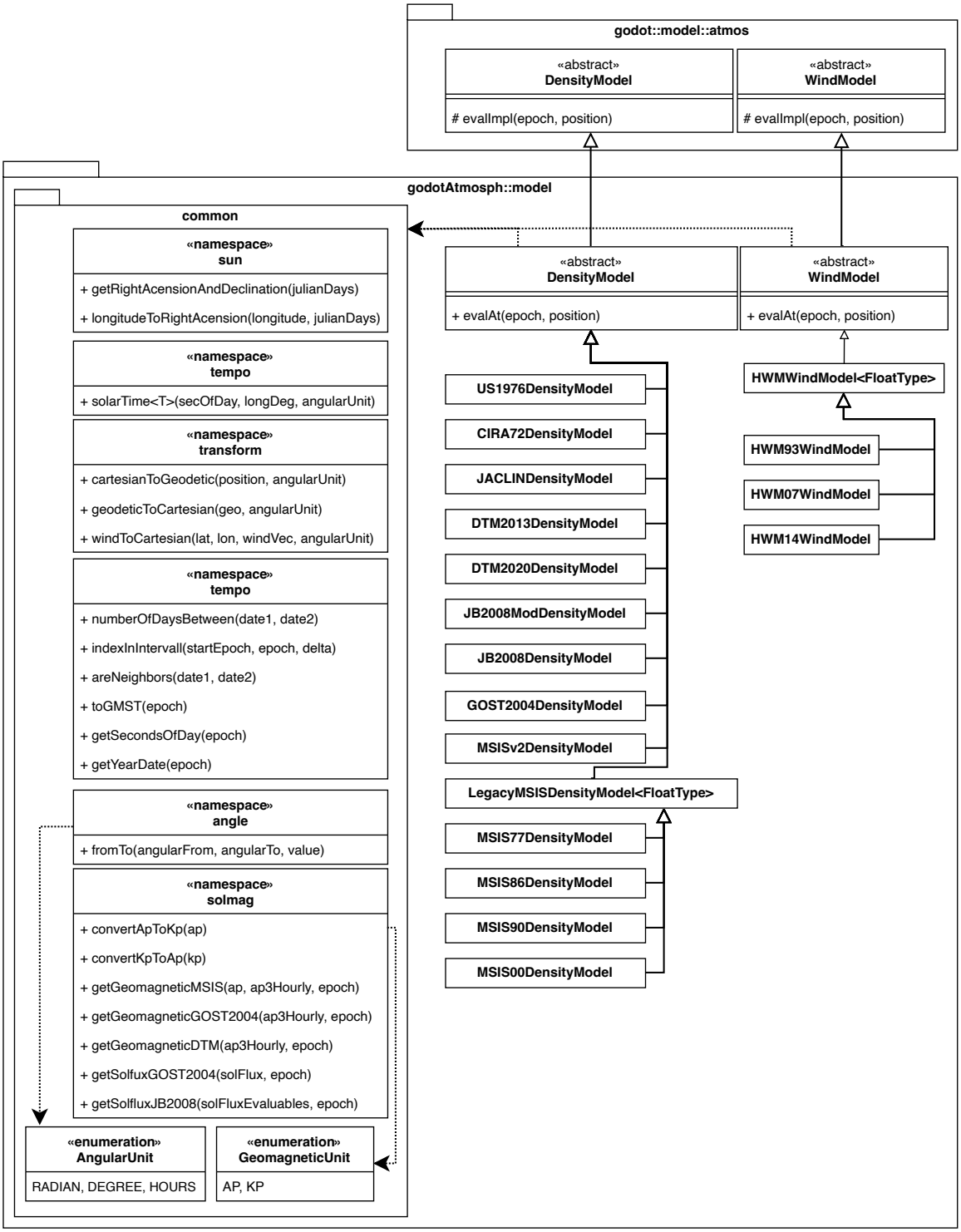


Figure 5.4.: A UML Class Diagram of the namespace `godotAtmosph::model` and the core inheritance relation to the new namespace `godot::model::atmos`. Types are omitted for clarity. `Templating` is collapsed and depicted as inheritance (e.g. `LegacyMSISDensityModel<FloatType>` represents `float` and `double` template instantiation). Free namespace functions are denoted in classes with the stereotype `namespace`. Each concrete `DensityModel` and `WindModel` implements its own `evalInternal(epoch, position)` method.

5.5. Python Binding of Godot-Atmosph

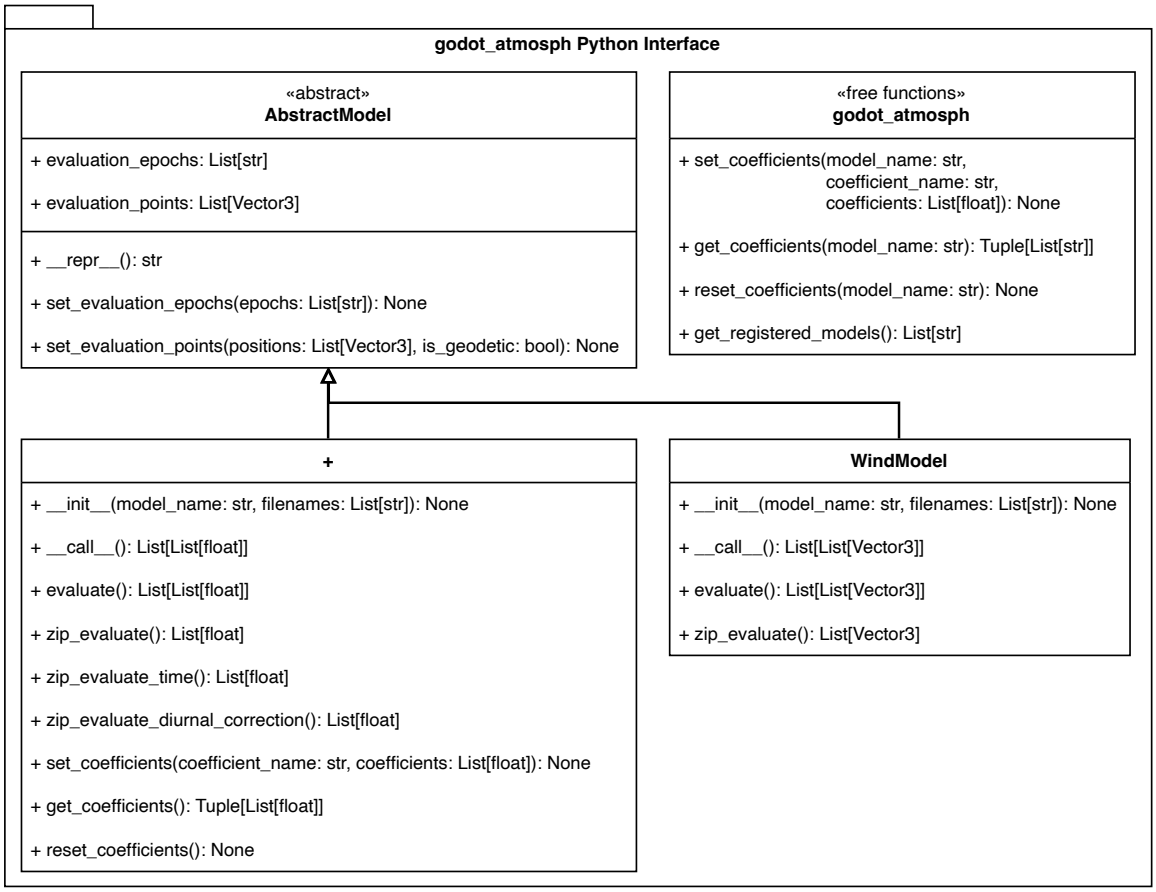


Figure 5.5.: A UML Class Diagram of the independent Python binding of `godotAtmosph`. The type names are slightly adapted to improve readability (e.g. `Vector3` is a list of size 3 and value `float`).

The described `C++` implementation does not extend `godotpy` by any means and is subject to `godot`'s architectural constraints. This circumstance is accompanied by the following two issues:

1. The new atmosphere models dynamically linked as plugin `godotAtmosph` to `godot` are only available in the context of a `Universe`. However, there is no way to instantiate the model directly via `godotpy`. One needs to instantiate the `Universe`, and then, one can get the density evaluable from the evaluables book of the `Universe`. In order to evaluate the model at a given position, one is now required to manually intervene in the frames system and align the placeholder positions of the density model to the desired location. This is a tedious procedure.
2. Second, while in theory, an extension of `godotpy` would be possible, given that `DensityModel` and `WindModel` are now part of `godot` itself. The realization still would require a substantial effort. Further, it is questionable if an independent atmosphere

model implementation only for the purpose of wind and density evaluation fits into the design approach of core `godot/` `godotpy`.

To summarize, the atmosphere models are easily used in the bigger context of satellite propagation, i.e., a `Universe` set-up in `godot`. However, single access to evaluate the density or wind at a specific point without the need for boilerplate code is not possible.

Hence, this work is accompanied by a small `Python` binding named `godot_atmosph`, which offers the same unified, easy-to-use interface as in the `C++` (see Listing 4) version to quickly evaluate an arbitrary atmosphere model at a given epoch and position. Figure 5.5 displays the `Python` binding exposing the `C++` functionality to `Python`. It consists of an abstract base class `AbstractModel`, which stores the evaluation points and evaluation epochs. The storage of points inside the `C++` class enables fast subsequent evaluations, especially regarding an optimization process where grid points remain constant, i.e., the setup time amortizes to a constant cost in the beginning. The need to transfer the epoch and point vectors repeatedly to the evaluation function is omitted. This is of particular importance as epochs are handed over as strings (to expose as little as possible from `godot`) and only converted to type `godot::tempo::Epoch` on the `C++` side.

The two subclasses `DensityModel` and `WindModel` respectively specialize the base class for either density or wind evaluation. The function `evaluate` or `__call__` returns the evaluated results at the cartesian product of epochs and points while `zip_evaluate` returns the inner product of the input. The density subclass also enables the manipulation of the coefficients of the underlying density model. This is only implemented for the manipulable version of the JB2008 model, abbreviated with JB2008 Modified (JB2008MOD). In every other case, the methods throw an exception.

The free functions in `godot_atmosph` enable the manipulation of the coefficients of a JB2008MOD created in the context of a `godot Universe` by effectively injecting the correct parameters during runtime in an instance of JB2008MOD. This technique is obviously not optimal. However, we can save the runtime of re-creating a fully populated universe instance with an JB2008MOD instance containing a different set of parameters. This time savings are beneficial for small propagation of around several hours to a few days where the initializing takes more time than the propagation. Secondly, the approach allows for faster prototyping, as new coefficients can quickly be exposed without the need to modify overlaying client code. For example, a coefficient modeled as TimeEvaluable would be required to be modified in either `C++` or `Python` context with associated new production code. The current setup allows this to be just a different string for `coefficient_name` in `godot_atmosph` without the need to add anything different in client code on top.

Two functions remain to be explained. The method `zip_evaluate_time()` enables to efficiently measure the runtime of the raw Fortran atmosphere model (e.g., Listing 3) using a singleton recorder instance. It returns the time for each zipped evaluation in microseconds. A preprocessor macro `MEASURE_RUNTIME_OR_NOP(fun)` inserts time measurements into the singleton when set to recording. The macro only adds $O(1)$ realized by an additional condition to the whole evaluation when disabled and nothing at all when the time measurement is not compiled. It is utilized in the context of Section 7.2.

The method `zip_evaluate_diurnal_correction` enables the raw evaluation of JB2008 diurnal density correction function with the current set of coefficients, positions, and epochs. Chapter 8 explains why this functionality is required.

6. Verification & Validation

The implementation is continuously integrated into the *GitLab*[1] instance of ESA. The implementation around `godotAtmosph` encloses an extensive amount of unit tests, which can be classified into three categories.

First, the standard unit tests exist for the in Chapter 5 presented classes. These tests verify the correctness of every class, including, but not limited to, the I/O operations, the solar and geomagnetic indices processing, and the correctness of the Fortran-ISO-C binding.

The second category of tests verifies the correctness of the atmosphere model integration. Further, they re-validate the correctness of the atmospheric models as a whole, that the implementation is correct. The publishing authors of MSISv2[2] and HWM14[3] distribute reference data with their models. Thus, a parameterized test checks the implementation of all released data points in both cases. In case of DTM2013[4], JB2008[5], MSIS00[6], MSISv2[7] and HWM14[8] and online evaluation service exists. This service is kindly made available by the Community Coordinate Modeling Center (CCMC). We utilized this service to produce reference data. In an initial verification of the MSISv2 and MSIS00 model, we found deviations when compared to the CCMC data. We found our implementation correct, and the CCMC's containing slight deviations when compared to the previously mentioned reference data - luckily available for MSISv2. In 67 of 200 cases, the CCMC'S data deviated by at least 1% from ours and the reference of the MSISv2's authors. In 13 out of these 67 cases, the deviation lied between 5% and 9%. Sometime later in the year, the CCMC's results were aligned to ours. These tests are performed with a relative epsilon of 10^{-2} to 10^{-4} depending on the reference data's precision, which is usually given by an analogous amount of digits. Every other density model, including the above mentioned again, is further compared to 600 random samples from the HASDM density set with a generous epsilon for the purpose of checking that the magnitude is similar. The later test also revealed an implementation error in MSIS86 leading to `nan` values when compiled in `Release` mode. The root cause was found to be non-initialized variables. This bug was fixed in the MSIS86 version in `godotAtmosph`.

Both of these first two test categories are written using GoogleTest[9].

The third category is written using `pytest`. It is an integration test based on the *NAPEOS* test case (see Section 4.2). In *NAPEOS*, a spacecraft (*Cryosat-2*) is propagated around the Earth for twelve hours, and the final positions are compared. We get identical results

[1] `https://gitlab.esa.int/sdo/godot-atmosph/`, last accessed: 11.01.2024
[2] `https://map.nrl.navy.mil/map/pub/nrl/NRLMSIS/NRLMSIS2.0/`, last accessed: 11.01.2024
[3] `https://map.nrl.navy.mil/map/pub/nrl/HWM/HWM14/`, last accessed: 11.01.2024
[4] `https://ccmc.gsfc.nasa.gov/models/DTM~2013/`, last accessed: 11.01.2024
[5] `https://ccmc.gsfc.nasa.gov/models/JB2008~2008/`, last accessed: 11.01.2024
[6] `https://ccmc.gsfc.nasa.gov/models/NRLMSIS~00/`, last accessed: 11.01.2024
[7] `https://ccmc.gsfc.nasa.gov/models/NRLMSIS~2.0/`, last accessed: 11.01.2024
[8] `https://ccmc.gsfc.nasa.gov/models/HWM14~2014/`, last accessed: 11.01.2024
[9] `https://github.com/google/googletest`, last accessed: 11.01.2024

to the previous implementation with the new implementation using MSIS00 and the same parameter set. The epsilon for *NAPEOS* is $5 \cdot 10^{-4}$. The other density and wind models are also tested for completeness in the integration test. However, those tests are conducted with a decently bigger test epsilon of $5 \cdot 10^{-3}$.

The changes in `godot` itself are tested by their already existing test toolchain. The existing test base of `godot::atmos` has been migrated and refactored to fit the new structure of `godot::model::atmos`.

7. Use Cases, Results & Discussion

This Chapter 7 presents runtime measurements of the aforementioned implementation, and results obtainable using the presented `godotAtmosph` Plugin.

7.1. Use Case Example of Atmospheric Models in Godot

```
"atmospheric_features": {
  "density":
  [
    {
      "name": "MyDensity",
      "type": "JB2008",
      "config": {
        "solmag":
        [
          "path/to/DTCFILE.TXT",
          "path/to/SOLFSMY.TXT"
        ]
      }
    }
  ],
  "wind":
  [
    {
      "name": "MyWind",
      "type": "HWM14",
      "config":
      {
        "solmag": "path/to/fap_day.dat"
      }
    }
  ]
}
```

Listing 5: An example of how to use the new atmospheric models in the context of a `godot` universe configuration to instantiate a concrete `Universe`.

```
"dynamics":
[
  {
    "name": "EarthAtmosphere",
    "type": "AtmosphericFeaturesDrag",
    "config": {
      "point": "Earth",
      "axes": "ITRF",
      "density": "MyDensity",
      "wind": "MyWind",
      "mass": "SC_mass",
      "area": "SC_drag_area",
      "cd": "SC_drag_cd"
    }
  }
]
```

Listing 6: This example assembles a density and wind model to a dynamic in the context of a `godot` universe configuration. The parameters `wind` and `density` are configured in the `atmospheric_features` plugin (see Listing 5). The parameter `axes` is defined in the `frames` plugin. The parameter `point` is implicitly added to the `frames` plugin by adding the Earth to the `bodies` plugin. The remaining parameters are defined in the `spacecraft` plugin. Details can be found in the documentation [58] or the Software Specification Document [50].

Section 5.2 presented the changes in `godot`. Listing 5 demonstrates how the new density and wind models are incorporated into the elementary universe configuration of `godot`, which specifies the equally named `Universe` instance. A density model is created by selecting a type and - if required - handing over a list of files from which the respective evaluables for solar and geomagnetic activity can be created.

The density and optionally the wind model are then embedded into a dynamic of type `AtmosphericFeaturesDrag` as shown by Listing 6. The latter also requires mass, drag area, and the coefficient of drag to be handed over. These properties are defined in the `SpacecraftPlugin`, which is not displayed here. It is sufficient to know that the `SpacecraftPlugin` defines all properties inherent to a spacecraft, e.g., like characteristics or the propulsive system [58]. The name of the dynamics (here: `EarthAtmosphere`) can then be utilized to, e.g., propagate a spacecraft, by defining the boundary conditions of the respective trajectory in a separate configuration file. For interested reader is referenced to the official documentation [58].

7.2. Runtime Measurements

Section 5.5 presents the `Python` interface. Figure 7.1 compares the runtime in seconds for 100 000 randomly generated pairs of points around the globe with random epochs between

the years 2000 and 2020. It also demonstrates the overhead added to the raw Fortran evaluation required for input and output transformation/ solar and geomagnetic indices preparation. Overall, the density models perform similarly given the complete comparison, with no model being a full magnitude faster or slower. However, the situation changes dramatically when only observing the raw Fortran runtime. The raw Fortran runtime is measured by the dedicated time measurement function of a `DensityModel`, as presented in Section 5.5. MSISv2 performs the worst, probably due to the model's build-up of several spline functions being expensive. It is also significantly slower than its predecessors, which makes sense, as MSISv2 is built from the ground new, whereas its predecessors are increments of each other. This fact is reflected in the step-wise increase from generation to generation.

GOST2004 is among the fastest to evaluate. However, a significant bulk of time is spent in input preprocessing. This can be explained by the fact that GOST2004 requires a special form of weighted solar $F_{10.7}$ index. For each evaluation, this requires iterating over the last 81 days of solar activity. Nonetheless, due to `godot`'s `TimeEvaluable`-based design, this is not efficiently possible even though the values are stored consecutively in memory. A `TimeEvaluable` only provides the `evaluate` function taking a single epoch. Thus, iterating over the last 81 days requires the repeated build of full `Epoch` objects with whom to call the `evaluate` method. Removing this single calculation significantly improves the runtime of GOST2004 by more than a third.

The same problem solidifies in the evaluation of the MSIS models but is less severe. Dedicated averages require the last eight three-hourly a_p values.

The problem could be resolved in three ways:

1. The caching of already computed solar and geomagnetic activity data, mainly the averages
2. The full precomputing of the averages and storing them in a new index file in the persistent memory
3. A more efficient bulk/ range-based access for `TimeEvaluables`

The first idea is realized in the existing implementation of MSIS00 in `godot`. It achieves better runtime performance around factor two in a satellite propagation scenario. However, it would fail in the here presented runtime comparison, as we sample randomly unique non-consecutive epochs. Thus, a cache for already-seen data would not help in this concrete situation. Nevertheless, it would be beneficial because real applications usually require consecutive temporal and spatial data.

The second idea would be overall the fastest, but somehow also counters the idea of a general purpose software like `godot` when one requires a unique file for every use case.

The third idea is currently being discussed internally to be integrated into `godot`'s core but was not finished at the time of writing. The introduction of a `TimeSeries` class, as mentioned in Section 5.3. Also, a self-brewed solution only for `godotAtmosph` was found to be unneeded with a core solution on the horizon. Moreover, the runtimes measured here are sufficiently good for the use cases of this work.

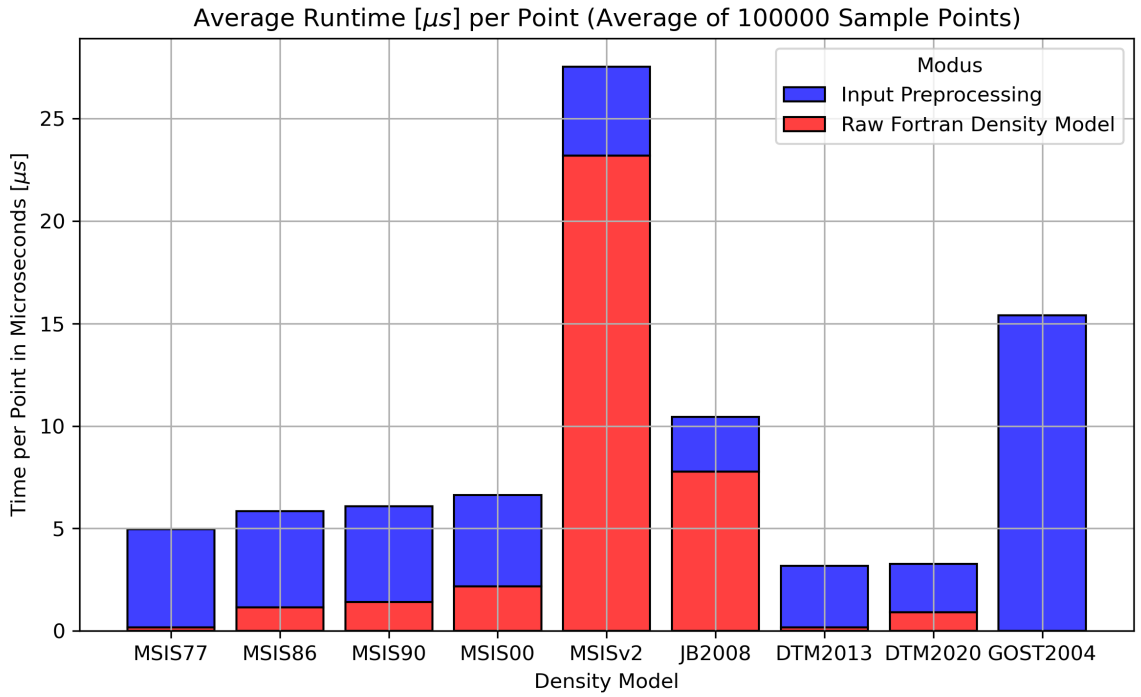


Figure 7.1.: Average Runtime Comparison between the major density model. The measurement has been conducted on the laptop specified in Appendix. The Python Interface/ Atmosphere Models have been compiled with Apple Clang 15. The plot shows that the input processing takes in several cases the greater share of runtime (especially to mention e.g. GOST2004) than the actual density model function evaluation.

7.3. Global Density and Wind Model Evaluation

Chapter 3 and Chapter 4 introduced HASDM. It is often refereed to as "gold standard" of the atmospheric density models. While the model itself is disclosed, the density data is available to the public from 2000 to 2019. Figure 7.2 plot the relative error of each density model compared to the HASDM database using respectively one million randomly sampled points in the years 2000, 2006 and 2019.

2000 and 2006 are chosen because they represent years of high and low solar activity. 2019 is chosen since it is the most recent one available in the HASDM database. Further, none of the models (with exception of DTM2020 and MSISv2) could purpose 2019's data for fitting. All box-plots include the mean and standard deviations, but exclude outliers and are cut-off at 2.5.

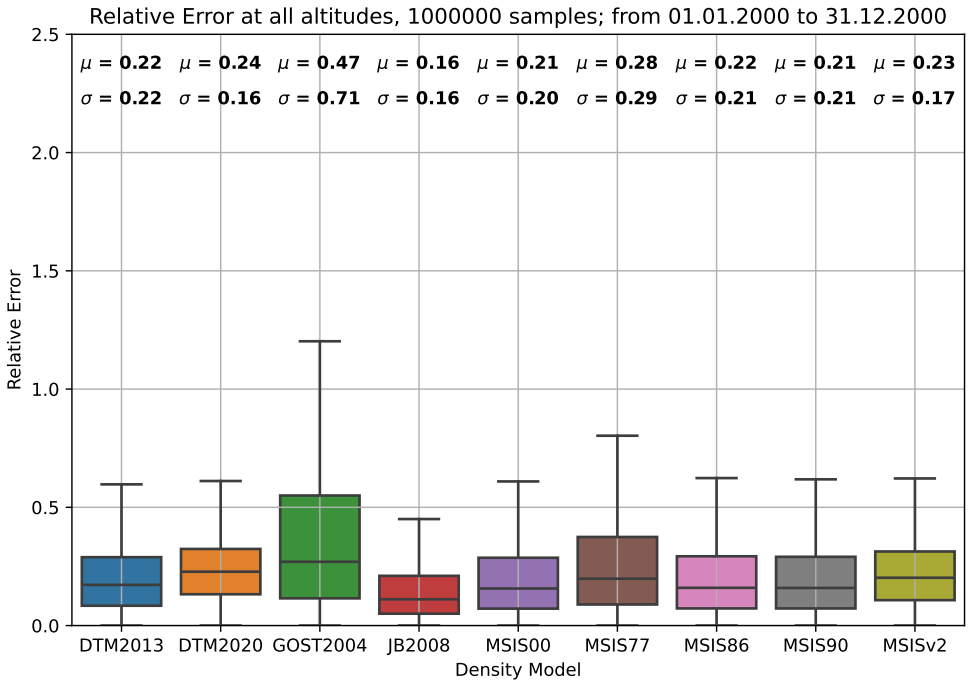
The more recent models perform most of the time better than the older ones. JB2008 performs overall the best. This performance is not surprising because HASDM is based on JB2008 with the only difference being the coefficient utilized. However, it also part of the justification why we will focus on JB2008 in context of Part III's optimization.

Lastly, the Figures 7.3 and 7.4 show the global density situation in altitude of 400 km. The dates are not randomly sampled, but fixed to the ones in Chapter 2. The pairs of Figures 2.3

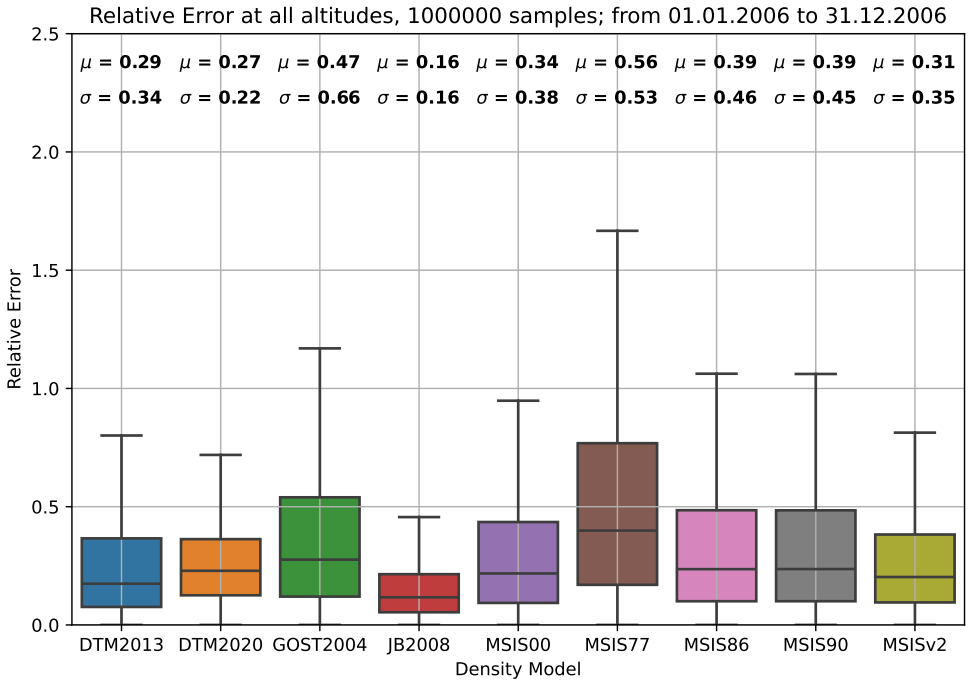
and 7.3 and Figures 2.4 and 7.4 depict respectively the same situation. Figures 2.3 and 2.4 utilize MSIS00 and one can partially see the similarities to Figures 7.3d and 7.4d using MSIS00.

The HASDM Reference uses interpolation in between grid points in Figures 7.3 and 7.4. This visibly explains the round shapes in the global density map. New models, namely JB2008, MSISv2, MSIS00, DTM2013, and DTM2020, are closer to the HASDM reference. The relative errors in Figure 7.2 confirm this. Further, the older MSIS models from 1977, 1986, and 1990 have a distinctly different density distribution. This is also mirrored in the bigger relative errors of Figure 7.2. An outlier is the Russian GOST2004, which has its density maxima at very different positions than all other models thourghout Figures 7.3 and 7.4 Nonetheless, as no reference or description exists to the author's knowledge, we cannot say if this might be due to wrong inputs in the implementation or if this is just the model. All statements are consistent through low and high solar activity in Figures 7.3 and 7.4.

For completeness, Figure 7.5 shows the wind models comparing the situation for high and low solar activity.

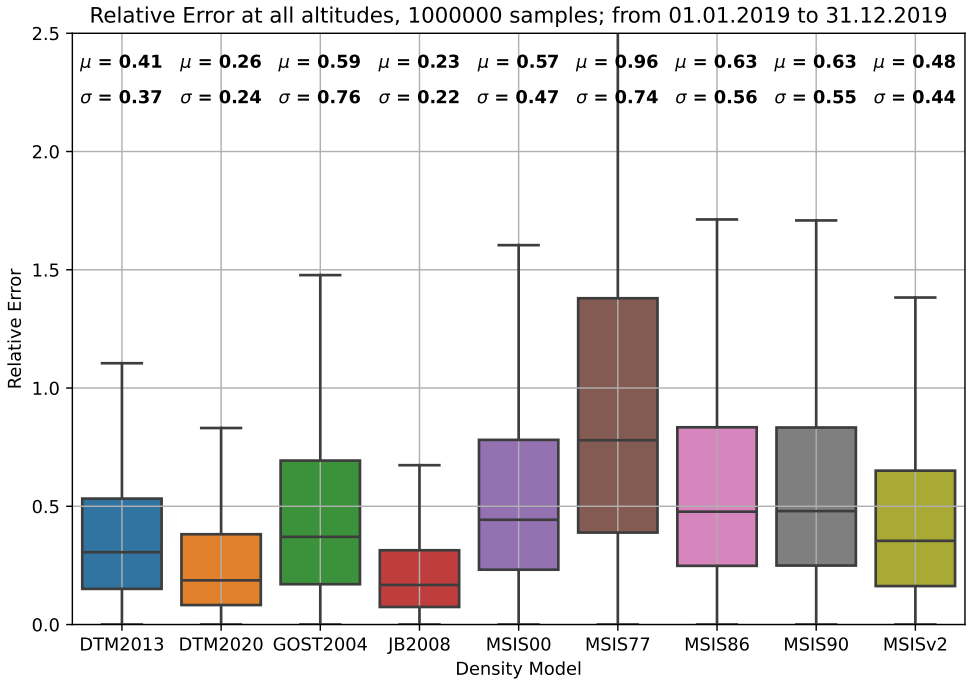


(a) 2000 - High Solar Activity



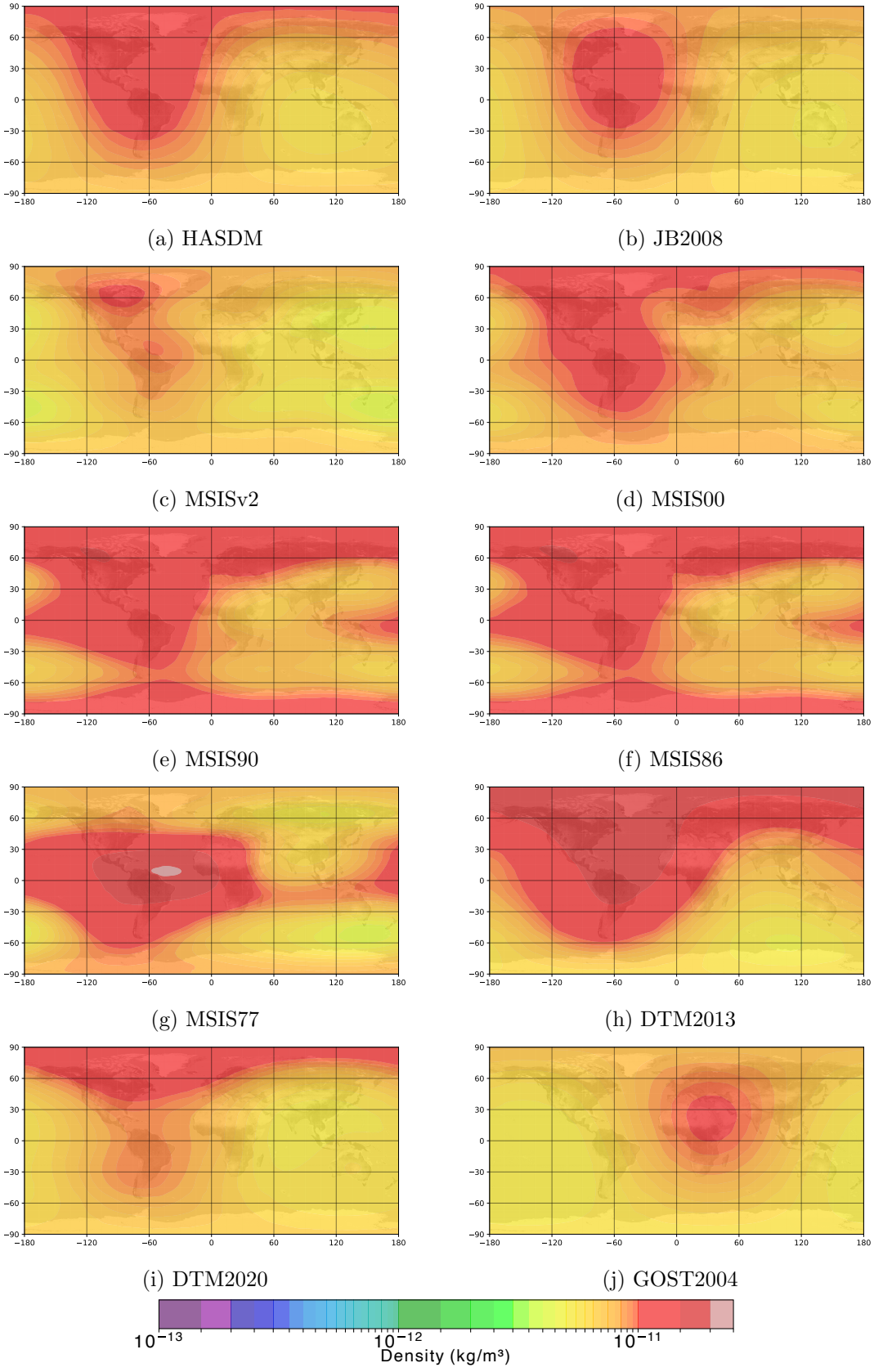
(b) 2006 - Low Solar Activity

Figure 7.2.: Description on the next page



(c) 2019 - Low Solar Activity. All models expect of DTM2020, MSISv2 were fitted earlier

Figure 7.2.: The box plots show the distribution of the respective density model's relative error to the HASDM reference. For each model, one million reference points were randomly sampled from the HASDM density dataset. The plots are cut off at 2.5 for better readability and comparability, and outliers are not shown. As compensation, mean μ and standard deviation σ are given, revealing, e.g., the existence of significant outliers for GOST2004. Three years are shown: 2000 and 2006 to demonstrate the performance of each model during low and high solar activity, as well as 2019, whose data was unavailable when some of the models were constructed. Each box spans from Q_1 (median of the lower half of the dataset) to Q_3 (median of the upper half of the dataset). Q_2 marks the median with a horizontal line.



(a) HASDM (b) JB2008

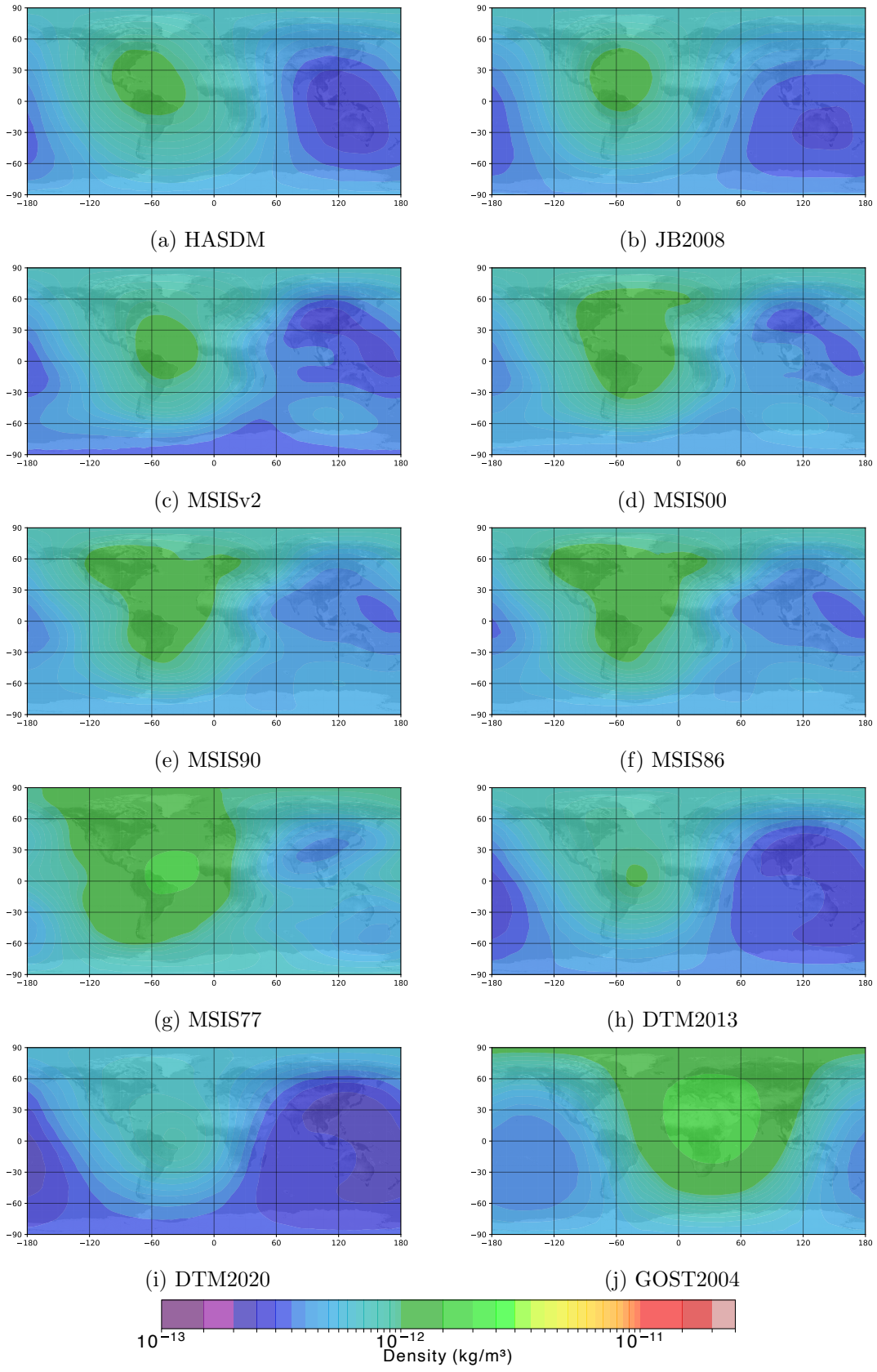
(c) MSISv2 (d) MSIS00

(e) MSIS90 (f) MSIS86

(g) MSIS77 (h) DTM2013

(i) DTM2020 (j) GOST2004

Figure 7.3.: Atmospheric density in an altitude of 400 km on 15.07.2000 at 18:00 UTC.



(a) HASDM (b) JB2008

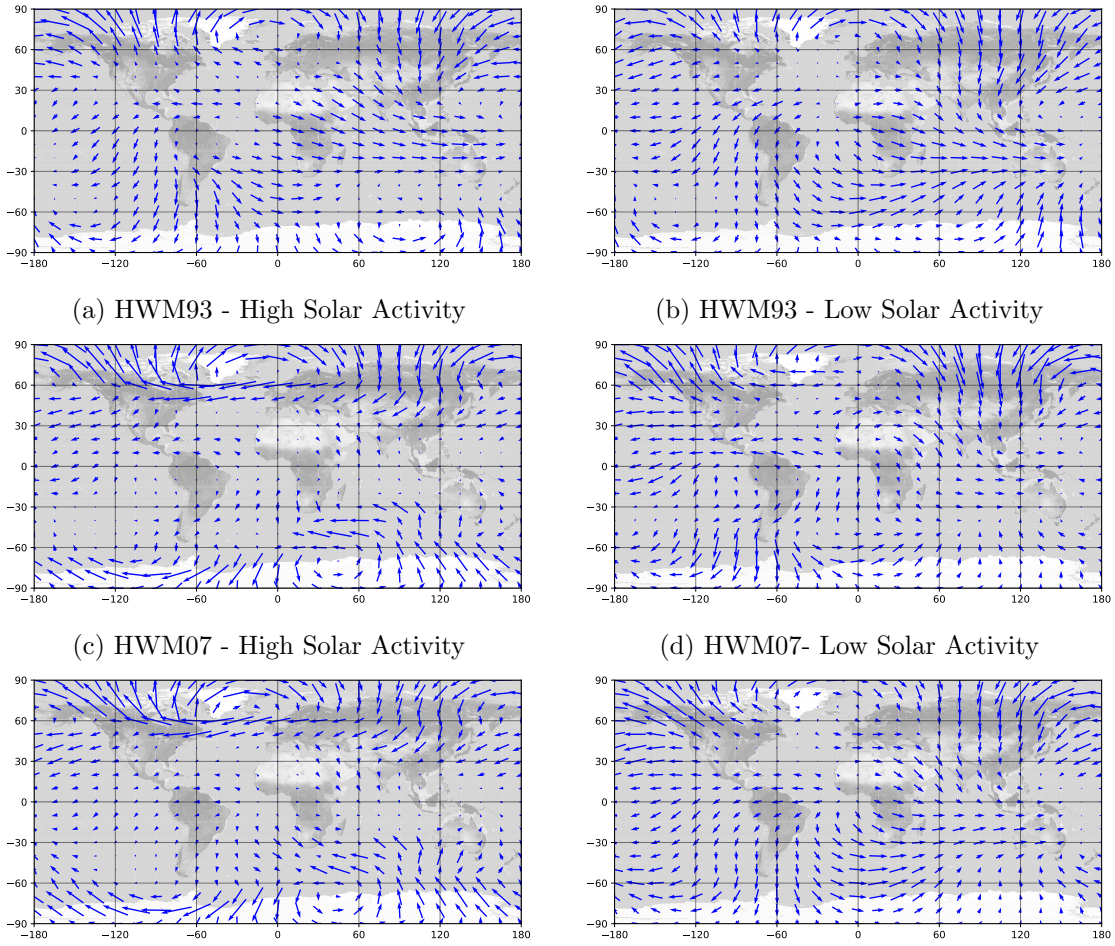
(c) MSISv2 (d) MSIS00

(e) MSIS90 (f) MSIS86

(g) MSIS77 (h) DTM2013

(i) DTM2020 (j) GOST2004

Figure 7.4.: Atmospheric density in an altitude of 400 km on 15.07.2006 at 18:00 UTC.



(a) HWM93 - High Solar Activity

(b) HWM93 - Low Solar Activity

(c) HWM07 - High Solar Activity

(d) HWM07- Low Solar Activity

(e) HWM14 - High Solar Activity

(f) HWM14 - Low Solar Activity

Figure 7.5.: Atmospheric wind in an altitude of 400 km on 15.07.2000 or 15.07.2006 at 18:00 UTC. 2000 was a year of high solar activity. 2006 was a year of low solar activity.

7.4. Reproducibility of the Plots

The subsequent Part III introduces the Atmosphere & Reentry Optimization Framework. It provides a wide variety of features, such as the capability to plot results automatically. All plots of Chapter 7 are fully made with the framework and the configuration presented in Listing 7. The subsequent Part III explains the Atmosphere & Reentry Optimization Framework and how this tight integration of subsystems is achieved to enable the user to perform complex operations, including I/O with a single configuration file.

```yaml
---
atmosopt:
  hasdm:
    altitudes: [ 175, 300, 400, 500, 600, 700, 800 ]
    years: [ 2000, 2006, 2019 ]
    save_path: "./path/to/hasdm"

  model:
    density_ground_truth: "./path/to/hasdm"

  plot:
    - name: density
      with_model: model
      params:
        density_model: HASDM
        epoch: 2000-07-15T18:00:00.000
        altitude: 400
      save_path: "./path/to/Density_HASDM_2000_400km.pdf"
    - name: wind
      with_model: model
      params:
        wind_model: HWM14
        epoch: 2000-07-15T18:00:00.000
        altitude: 400
      save_path: "./path/to/Wind_HWM14_2000_400km.pdf"
# ... other plots of type "density" or "wind"
    - name: density_relative_error
      with_model: model
      params:
        sample_size: 1000000
        seed: 42
        from: 2000-01-01T00:00:00
        to: 2000-12-31T23:59:00
        plot_mean_std: True
      save_path: "./path/to/Density_RMSE_2000_all.png"
# ... other plots of type "density_relative_error"
    - name: runtime
      with_model: model
      params:
        sample_size: 100000
      save_path: "./plots/runtime.pdf"
```

Listing 7: `YAML` configuration for the plots of Chapter 7 using `atmosopt`

Part III.

The Atmosphere & Reentry Optimization Framework

8. Architecture & Application

This chapter introduces the architecture of the Atmosphere & Reentry Optimization Framework, which is in the following abbreviated with the name of the Python package `atmosopt`. An example configuration file accompanies the explanations to continuously illustrate how the configuration file is mirrored in the architecture and vice-versa. Beforehand, the subsequent section presents a high-level overview of components and the leitmotif.

In the following, if we refer to atmosphere models, we **only** refer to density models since wind models are not of further interest in the context of `atmosopt`. *Design Pattern* and *Structural Choices* are emphasized in *italics*.

8.1. Overview of Components

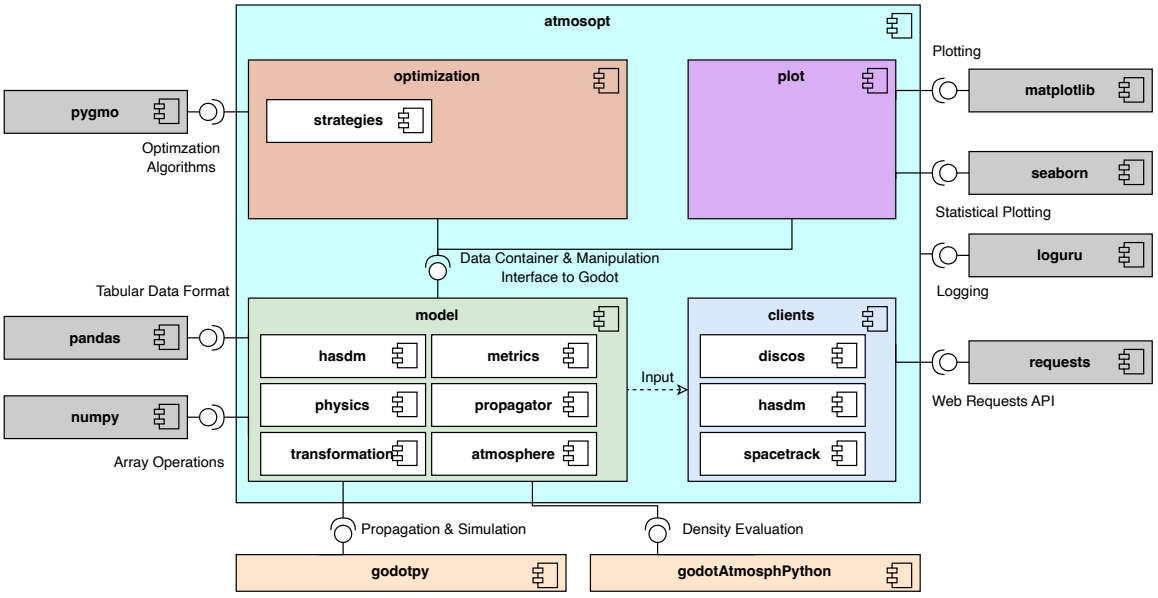


Figure 8.1.: A UML Component Diagram of `atmosopt` displaying its major dependencies in gray. The inner components are colored differently for better visibility. The `model` is central and unifies the capabilities of `godot` with the automatic retrieval of data. The relation from `model` to `clients` is realized by a higher level driver program of `atmosopt`. However, there exists no actual inter-dependency between the plain two components. The connections only serves the understandability.

The Atmosphere & Reentry Optimization Framework breaks down into four sub-components:

- The `clients` component handles the raw input retrieval from publicly available sources. It is subdivided into three independent modules:

  - The `spacetrack` component handles the retrieval of satellite state data from Space-Track (see Subsection 4.1.1)
  - The `discos` component handles the retrieval of satellite characteristics from ESA's DISCOS (see Subsection 4.1.2)
  - The `hasdm` component handles the retrieval of density data from the SET HASDM Database (see Subsection 4.1.3)
- The `models` component is the central storage container and provides (parallel) propagation, conversion, and more functionality. Its foundation consists of the two classes `SatelliteCollection` and `Satellite`
  - the `hasdm` component provides an efficient storage container for the SET HASDM Data
  - The `propagator` component provides the interface to one of `godot`'s core competences: Satellite Propagation
  - The `atmosphere` component provides a unified interface to the `godot_atmosph` Python binding
  - The remaining component `transformation`, `metrics`, `physics` provide minor utility functionality
- The `optimization` component provides the capabilities to assemble a ballistic coefficient or atmospheric optimization problem by arbitrarily combining satellites, epoch ranges, and concrete strategies into a single problem. Further, it enables the parallel execution of these tasks.
- The `plot` component provides ways to visualize the interior of the `mode` component

The core philosophy of `atmosopt` is the complete decoupling of `optimization`, `plot` and `clients`. The only interplay occurs via the `model`. The `model` contains the satellites and density data provided as instantiated objects to the other components on request. The output of the `optimization`, `plot`, and `clients` is written to the hard drive. An exchange of optimized parameters from `optimization` to `plot` only happens via file paths if `atmosopt` is utilized like an executable. This design choice allows for interruption of the process without losing intermediate optimization results. If `atmosopt` is utilized as a library, functional interfaces allow the user to hand over parameters within Python from one component to another.

Thus, a configuration file consists of six independent steps:

1. Download data from Space-Track
2. Download data from DISCOS
3. Download data from the SET HASDM Database
4. (Lazily) assemble the model(s) as specified by the user by consolidating the data from step 1 to 3
5. Run all specified optimizations and save the result of each single optimization task, making it available as input for the next optimization in order
6. Run all requested plotting activities

The configuration file is executed by the `Runner` class. Due to the independence of each step, it is possible to, e.g.,

- fully ignore/ omit certain parts like the download parts if the files are already present
- specify multiple models with different input files
- reuse the results of previous optimizations
- recover gracefully as every single (end-)result is saved to persistent memory
- incrementally add full steps or single tasks to a configuration file while still reusing already computed results (e.g., starting with ten satellites and adding run by run more satellites)
- easily conduct comparisons between results of different methods

For instance, the configuration presented in Section 7.4 demonstrates a configuration entirely omitting satellite data and optimization tasks. In the following, a configuration file, utilized in Chapter 10, accompanies the descriptions and rationale of the architectural design step by step. The configuration leads to the results given by Figure 10.3.

8.2. Clients Component

The `clients` component is split into three subcomponents, namely `discos`, `hasdm`, `spacetrack`. They all provide a similar handling experience. From the first view on Figure 8.2, it is tempting to add abstract classes for the `Client`, `Error`, and `Query` class. It would make sense structurally, but it is pointless from an operative point of view since the `accept` methods have highly different return types. If one had introduced an abstract superclass, all subclasses would have violated the *contract*, which they should obligate. Thus, we remain at this clean triple split. All clients are implemented as singleton. Thus, a repeated instantiation will return the same instance.

For Space-Track, an excellent implementation exists installable via `pip`[1]. For this work, we decided to *reinvent the wheel* for one reason. The class hierarchy can be directly re-found in the configuration depicted in Listing 8. The configuration parsing is basically for free as a side effect of the class hierarchy. The Space-Track client implements rate limiting and lazily authenticates the user on the first actual request. Six methods are implemented listed in `SpaceTrackMethod` with three of them of particular importance and also used in Listing 8. `GENERAL_PERTURBATION_HISTORY` returns the historic TLE data, `DECAY` returns the decay date(s) and their creation times, while `TRACKING_AND_IMPACT_PREDICTION_MESSAGE` is superior and also contains the concrete latitude and longitude of reentry. The former two are `None` for active satellites. The `SATCAT` is helpful to get an initial estimate of satellites fulfilling launch and decay criteria without the need to process TLE state data.

The `discos` client returns the satellite characteristics, given a set of Norad Catalog IDs, including mass and drag area boundaries - if available. This information is one option for the initial ballistic coefficient of every satellite.

[1] `https://github.com/python-astrodynamics/spacetrack`, last accessed: 08.02.2024

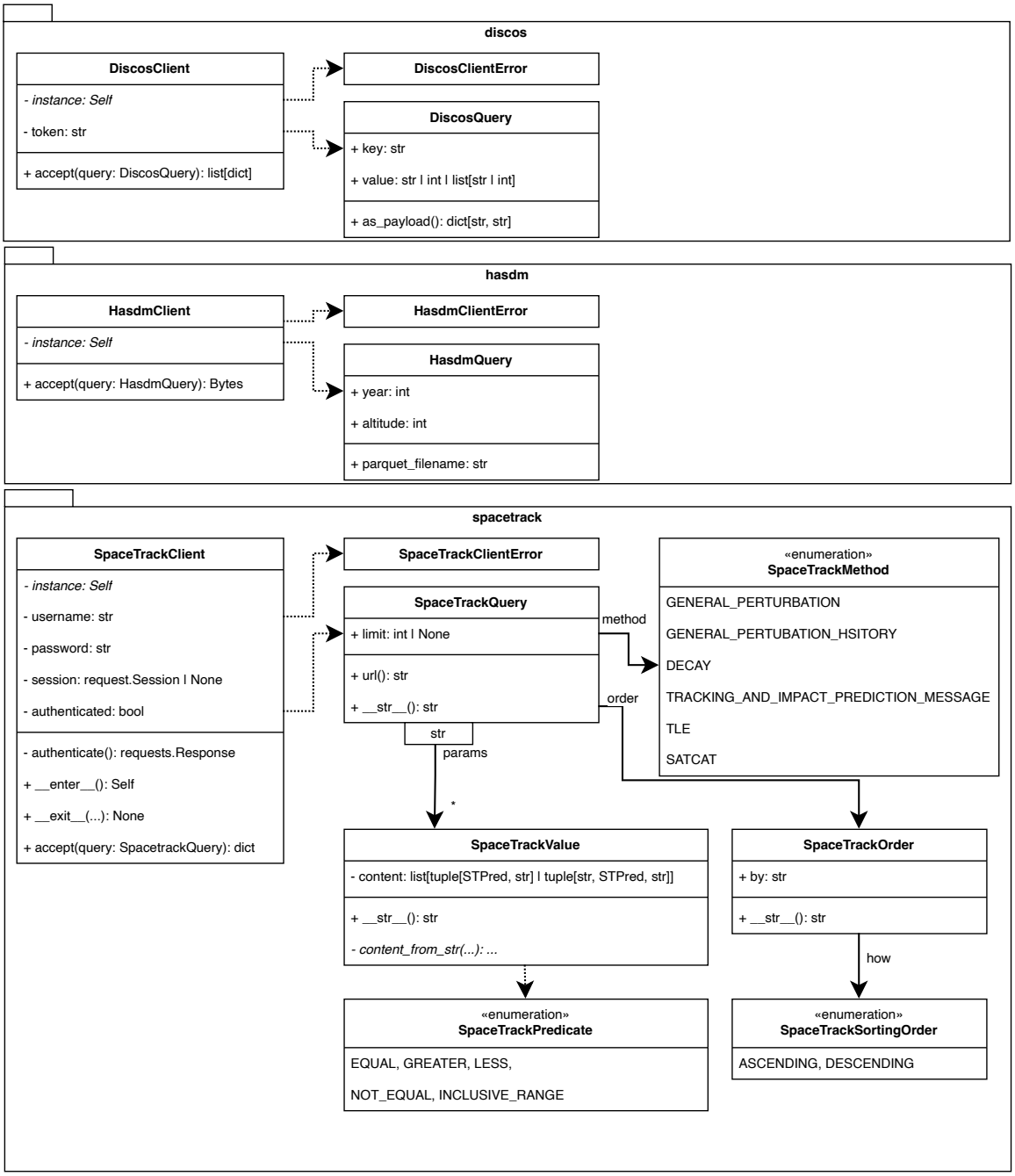


Figure 8.2.: A UML Class Diagram of the `clients` component of `atmosopt`. It illustrates the three major clients capable of retrieving data from Space-Track, DISCOS, and the HASDM SET Density Database. The handling from a user-centric perspective is similar, with the main entrance point being the `accept(..)` method, consuming a query.

The `hasdm` client accepts a request for a single altitude $\in \{175, 300, 400, 500, 600, 700, 800\}$

and year $\in [2000; 2019]$. The client returns a byte object representing the content of a zip-compressed text file. The bytes object is written to a temporary directory, uncompressed, and further prepared for use in the implementation by the mentioned `Runner` instance. Details about the HASDM processing can be found in Subsection 8.3.2.

```yaml
---
atmosopt:
  spacetrack:
    credentials:
      username: <e-mail>
      password: <password>
    run:
      - name: Satellites decayed in 2019-05, and available in 2019-04 for optimization
        query:
          method: "gp_history"
          params:
            NORAD_CAT_ID: [22503, 24905, 25171, 34294, 38743, 42706, 42712, 42715, 42820,
                           42821, 42822, 42823, 42824, 43033, 44121, 44123, 44128, 44136,
                           44144, 44145, 44148, 44163, 44168, 44169, 44171 ]
            EPOCH: "2019-03-01T00:00:00.000--2019-07-01T00:00:00.000"
          order:
            by: "EPOCH"
            how: "desc"
        save_path: "./2019-05-04/input/tle_data.json"
      - name: Reentry Data
        query:
          method: "decay"
          params:
            NORAD_CAT_ID: [22503, 24905, 25171, 34294, 38743, 42706, 42712, 42715, 42820,
                           42821, 42822, 42823, 42824, 43033, 44121, 44123, 44128, 44136,
                           44144, 44145, 44148, 44163, 44168, 44169, 44171 ]
        save_path: "./2019-05-04/input/decay_data.json"
      - name: TIP
        query:
          method: "tip"
          params:
            NORAD_CAT_ID: [22503, 24905, 25171, 34294, 38743, 42706, 42712, 42715, 42820,
                           42821, 42822, 42823, 42824, 43033, 44121, 44123, 44128, 44136,
                           44144, 44145, 44148, 44163, 44168, 44169, 44171 ]
        save_path: "./2019-05-04/input/tip.json"

  discos:
    credentials:
      token: <discos-api-token>
    run:
      - norad_cat_id: [22503, 24905, 25171, 34294, 38743, 42706, 42712, 42715, 42820,
                       42821, 42822, 42823, 42824, 43033, 44121, 44123, 44128, 44136,
                       44144, 44145, 44148, 44163, 44168, 44169, 44171 ]
    save_path: "./2019-05-04/input/discos.json"

  hasdm:
    altitudes: [ 175, 300, 400, 500, 600, 700, 800 ]
    years: [ 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
             2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019 ]
    save_path: "./data/hasdm"
```

Listing 8: Configuration of the clients to download Satellite State & Characteristics and the HASDM Set Density Data. The configuration is case-insensitive. The configuration of Space-Track and DISCOS client consists of a list of queries, while the HASDM client takes several altitudes and years and builds queries by forming the cartesian product.

8.3. Model Component

```yaml
---
atmosopt:
  model:
    satellite_ground_truth:
      satellite_data:
        - "./2019-05-04/input/tle_data.json"
      decay_data:
        - "./2019-05-04/input/decay_data.json"
        - "./2019-05-04/input/tip.json"
      attribute_data:
        - "./2019-05-04/input/discos.json"
    density_ground_truth: "./data/hasdm"
```

Listing 9: Configuration of a model. A model is not related to any of the inputs. An arbitrary JSON file with the same format could be insert into any of the lists. The model configuration requires the files specifying the input satellites to be categorized by their content: state data, decay data, and satellite characteristics. The density data is specified by a directory containing the HASDM Parquet files (see Subsection 8.3.2). The top-level entry must not be named "model", but only must contain the string.

The `model` component is the core component of `atmosopt` It consolidates the available data and provides methods to work with the data. Its setup is by design as simple as possible, and the consolidation happens fully automatic. Listing 9 illustrates the simple configuration. A model only requires the orbital state data (TLEs) downloaded from Space-track in JSON format. Optionally, one can add decay data, attribute data, and a density ground truth, with the latter being a directory of HASDM Density Data Files.

Figures 8.3 and 8.4 provide a UML Class Diagram of the `model` component. First, Figure 8.3 illustrates what previously has been visible in Listing 9. A `Model` object consists of `SatelliteCollection` and a `DensityContainer`.

The `SatelliteCollection` merges the files of a group to a single *pandas* `DataFrame` with the exception of the decay data. The decay data is stored in a dictionary mapping Norad Catalog IDs to `DecayData` objects if a decay data could be derived from either the TLE state data (e.g., the last state may contain a reentry date) or the most recent reentry message. The `SatelliteCollection` ignores columns of the TLE data that are not required by the implementation.

A `Satellite` is lazily constructed on request via `__getitem__(..)`. The `Satellite` instances are cached. A shallow copy of the cached instance is returned in case of repeated access. A shallow copy is sufficient because the `Satellite`'s state data is private and not mutable from outside. Robustness testing revealed one case in which Space Track returned every TLE entry twice. Thus, during construction, the state data for the requested satellite is checked for duplicates. Further, the initial ballistic coefficient is determined by either using DISCOS's data with Equation 2.57 or the starred ballistic coefficient of the latest TLE with Equation 4.1. Figure 9.1 illustrates this as a UML Activity Diagram.

The `Satellite` class offers mainly two functionalities:

- Retrieving the observed TLE state, either raw or resolved using `godot`'s polynomial SGP4 interpolation
- Propagating the satellite state with one of the available propagators, namely by one of `godot`'s numerical propagation procedures

The first function is realized by the method `known_state`, while the second is realized by the method `propagated_state`. In case no SGP4 resolution is requested, the raw TLE state is returned. However, in any other case, one of the implemented propagators is required. These are presented in the subsequent Subsection 8.3.1. Both methods allow the arbitrary decoration of the output with additional information or switching the units or reference frame. This is outfactored in the `SatelliteStateDecorator`. The class builds upon a given state `DataFrame` like a *Builder* while providing a similar handling as chained *Decorator* functions. Hence, it can be described as a mixture of both software patterns.

Figure 8.4 presents the other content of the `module`. These are mainly grouped utility functions or wrapper functions. For instance, the `transformation` module provides pseudo vectorized access to `godot`'s conversion functionality, while `atmosphere` provides an interface to the Python binding `godot_atmosph`. The orange emphasized contents are subject to the next Subsection 8.3.1.

8.3.1. Propagator Component - The Interface to Godot

Figure 8.5 states the `propagator` component and its relation to the outer scope. The component implements four different propagators. All of them inherit from the `AbstractPropagator`.

The two simple propagators are the `KeplerPropagator` and the `SGP4Propagator`. The former only requires an initial state and propagates this state without any perturbation by neglecting perturbations. The latter adds a TLE-backed point to the frames system of the singleton `godot Universe` instance. The `SGP4Propagator` uses a simple `Vector6TimeEvaluable` constructed between the freshly created TLE point and the Earth's center to return the satellite state at a given epoch.

The more interesting parts are, without doubt, the two propagators using the full dynamic model of `godot`. Both of them use the same configuration for a `godot Universe` and deliver similar results. However, the implementation works differently.

Before going into details, a few words about the universe configuration utilized in the context of this work. The spacecraft is affected by Earth, Sun, and Moon. For the Earth, we use a gravity model using spherical harmonics expansion up to order and degree 12. The concrete model is called "EIGEN-05C", and details can be found in the explanations by Förste et al. [85]. The dynamics are enriched only by atmospheric drag, simulated with the respective density model. Hence, solar radiation pressure and other factors are excluded from the study.

The `GodotBallisticPropagator` uses the singleton instance `GODOT_UNIVERSE_INSTANCE` since it relies on the `BallisticPropagator` of `godot`. It allows the propagation of the state consisting of position and velocity, and it does not insert its results into the `Universe` instance. Thus, repeated utilization with the same instance is possible without any pollution. This is sufficient for a typical use case and saves the runtime overhead of repeated re-creation

of the full `Universe` with all its enclosed dynamic models. However, there is one drawback in the case of atmospheric reentries. Here, the propagator needs to stop the propagation out-of-schedule beforehand. In `godot`, this is realized via interrupting events that trigger when a condition reaches a (lower than) zero value. However, in one of our test cases with around 60 satellites, `godot` started to freeze nondeterministically with an enabled reentry event in rare cases (1 out of 120 propagation).

Given this problem, we created a second propagator using `godot`'s robust `Trajectory` class. The configuration is created via `create_trajectory_cfg(..)`. For the computation, the `Trajectory` class requires a `Universe` instance in which the propagation data is inserted. Thus, a re-propagation requires each time a fresh new `Universe` instance. The trajectory is configured with the final epoch and propagated. In case of an error likely due to a reentry event, the propagation is retried with a reentry altitude similar to the previously mentioned events. In contrast to the former approach, this is more expensive due to the `Universe` creation and potential second propagation. Nevertheless, it is robust, and no errors have occurred. At the time of writing, it is not yet possible to propagate a reentry using a single condition like: Stop the propagation below $100\,km$ or at epoch t like it is possible with the events system. However, the current work proved itself to be good enough. Both propagators offer methods to reconfigure the atmospheric coefficients in case the density model is JB2008MOD.

The propagators allow the selection of the integration method and the maximal amount of integration steps. In our use cases, the `Adams` integration method was usually by about factor two faster. Hence, the `Adams` integrator is utilized across all experiments in Chapter 10. When propagation spans multiple weeks, nearly all experiments utilize a step size of at least one million integration steps.

8.3.2. HASDM Component

Figure 8.3 introduces the `hasdm` component. It contains an efficient container that lazily loads the requested year and altitude data. Further, it provides several functions to store the data even more compressed. The zip files are extracted to the textiles by the `Runner`. Subsequently, they are read with *pandas*, and unnecessary data is dropped, while each column uses the minimal sufficient data type for storage. For instance, the latitude can be conveniently stored in an 8-bit type. The standard HASDM density text file has a size of around 412 MB, and the corresponding DataFrame with minimal types only has 120 MB.

The *Apache Parquet*[2] proved to be the most efficient way among the tested formats to store the data on a hard drive. Further, the loading time to retrieve the data was negligible fast from an solid state drive. Using Parquet, the original index file of 412 MB can be efficiently reduced to only 12 MB since it is capable of choosing an optimal compression scheme per column[3] and our data being redundant except for the density column. For instance, the latitude, longitude, or epochs are the same for many rows. The `DensityContainer` starts with a directory containing these parquet files and loads them on-demand.

[2] `https://parquet.apache.org`, last accessed: 09.02.2024
[3] `https://parquet.apache.org/docs/overview/motivation/`, last accessed: 14.02.2024

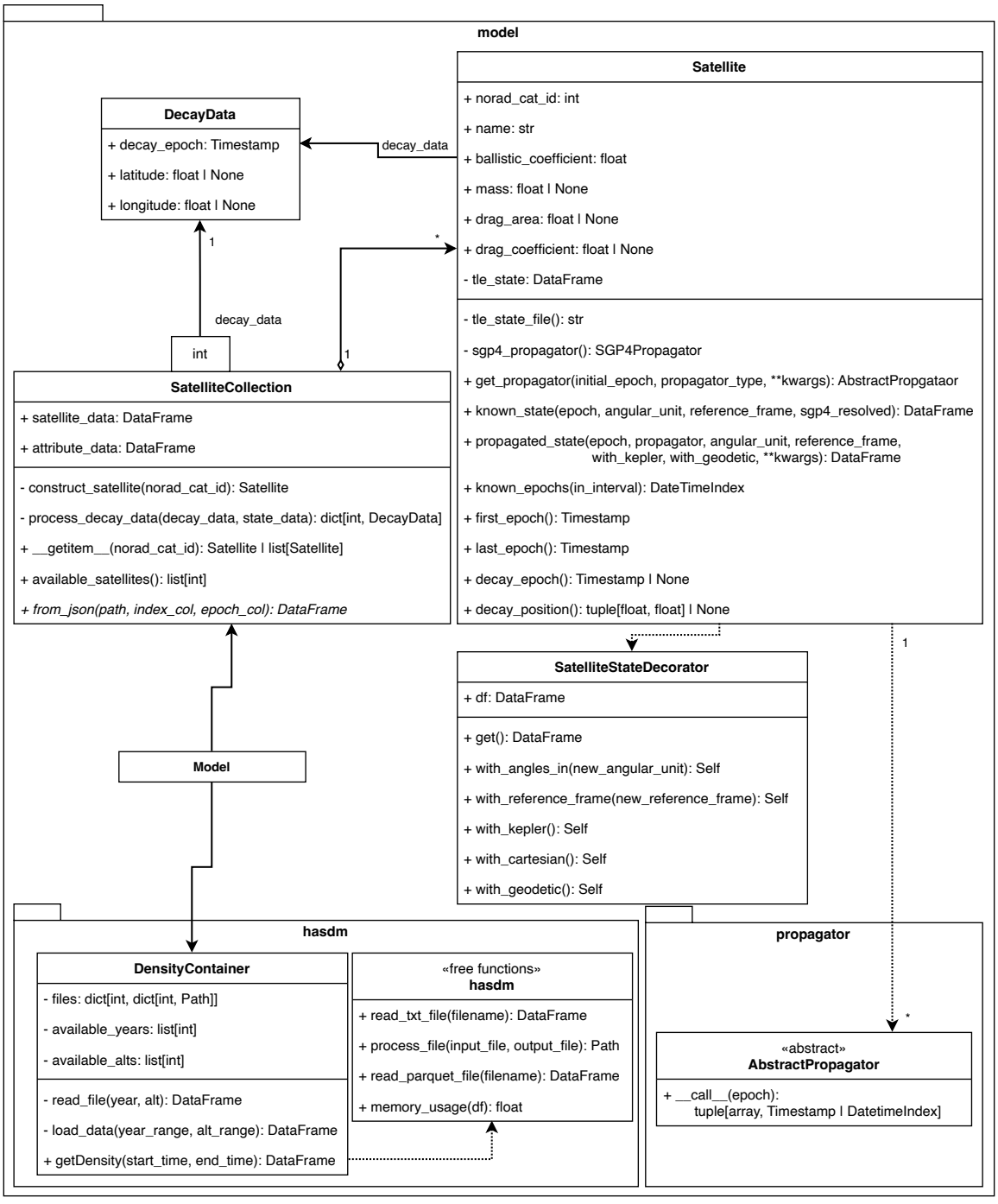


Figure 8.3.: A UML Class Diagram of the `model` component of `atmosopt` showing the central classes. Types are omitted for function arguments to improve readability. The core ensembles are `Satellite` and `SatelliteCollection`. The former yields the latter in a lazy fashion. The `Satellite` offers unified access to `godot` propagation and state conversion. The calls are respectively delegated to a `Propagator` object or `SatelliteStateDecorator`.

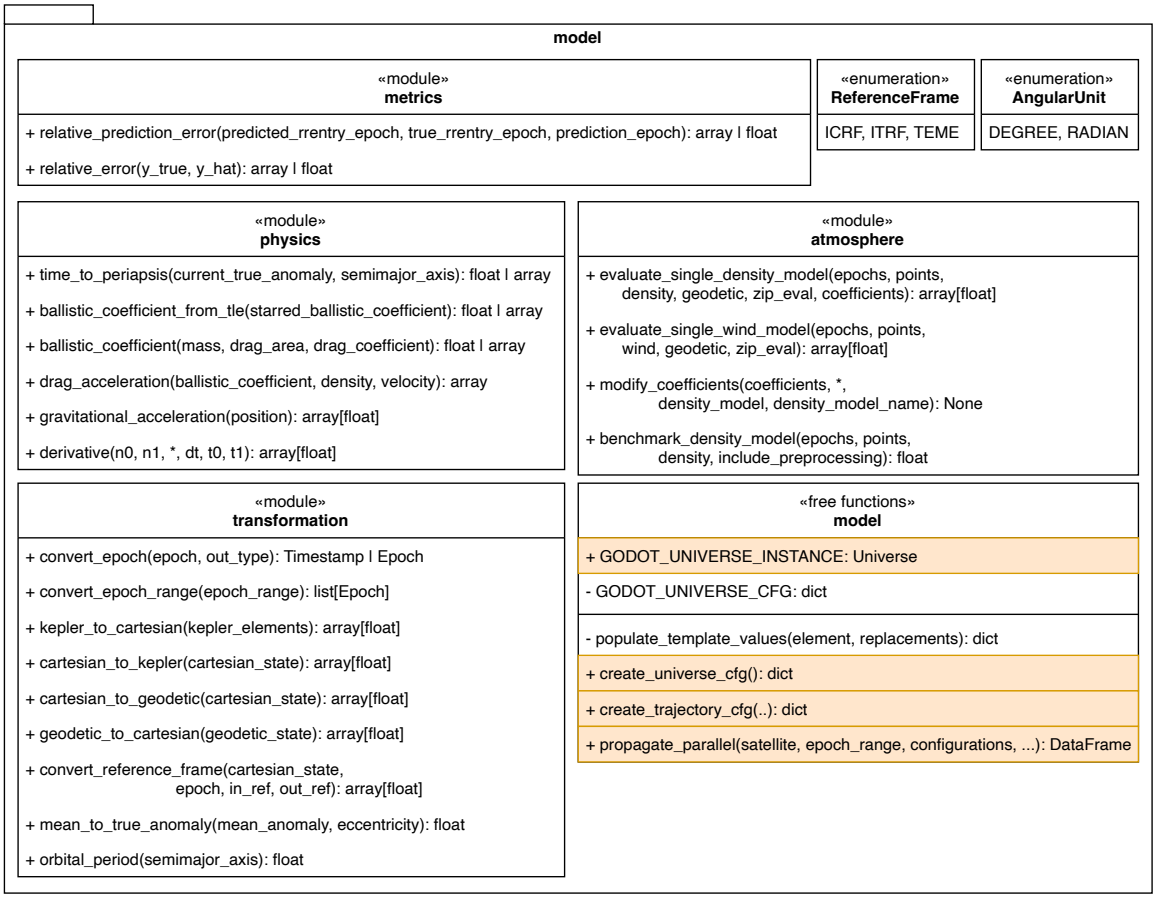


Figure 8.4.: A UML Class Diagram of the `model` component of `atmosopt` showing the utility of various categories. The diagram emphasizes the interfaces to `godot` utilized by `atmosopt.model.propagator` in orange. Types are omitted for function arguments to improve readability.

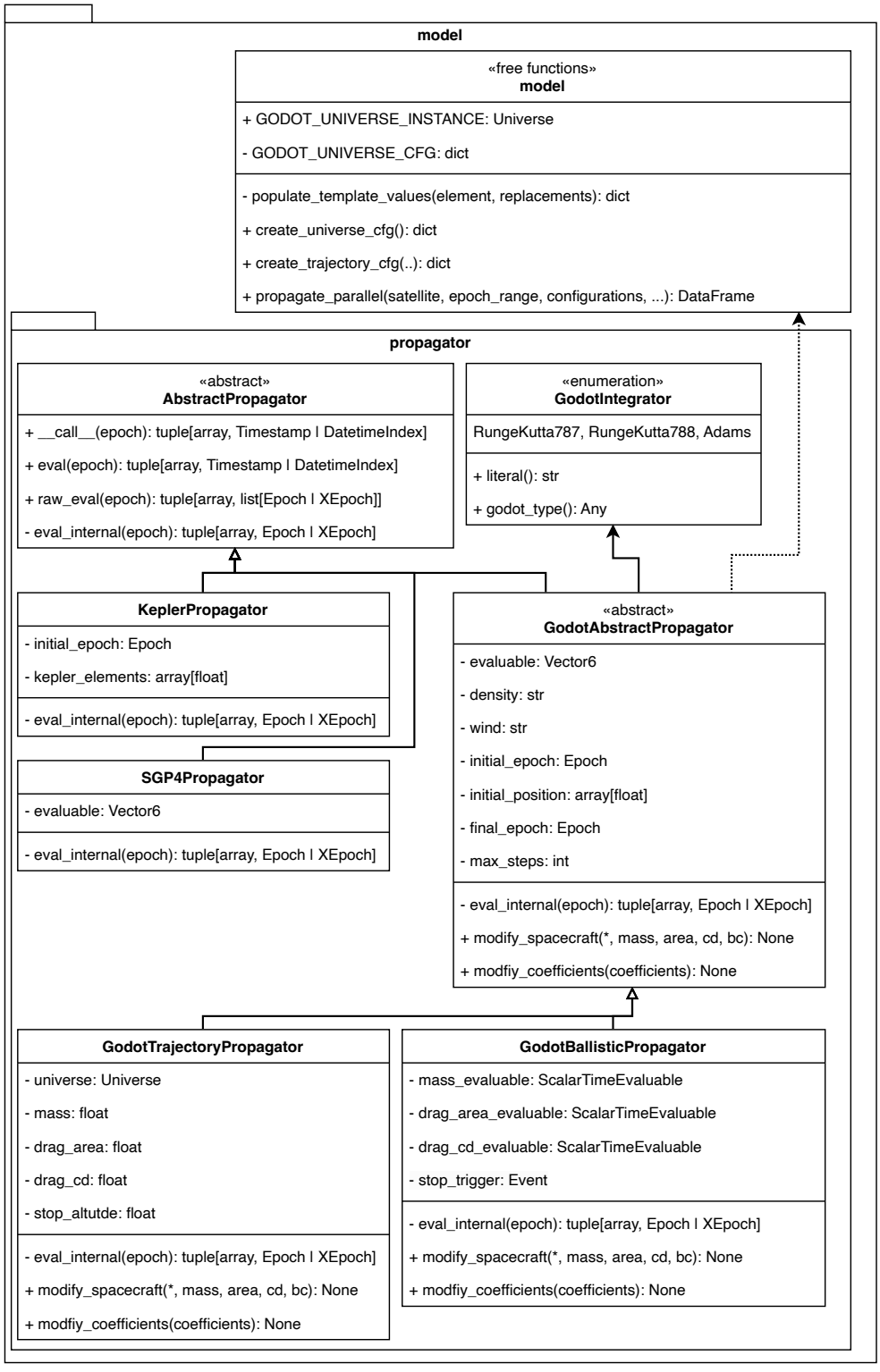


Figure 8.5.: A UML Class Diagram of the `propagator` component of `atmosopt.model` showing the available propagators. The `GodotBallisticPropagator` uses a singleton `Universe` instance `GODOT_UNIVERSE_INSTANCE`, whereas the `GodotTrajectoryPropagator` needs to re-create the instance for every propagation via `create_universe_cfg()`.

8.4. Optimization Component

The `optimization` component has been the most challenging design. The framework's vision is that several Norad Catalog IDs, previously assembled from several building blocks, are sufficient to start an optimization run. Multiple questions were open beforehand:

- How to work with arbitrary data when the amount and quality are unknown?
- How to integrate multiple different optimization strategies without creating clutter while maintaining a degree of independence in between?
- How to allow the user to mix combinations arbitrarily?
- How to allow the chaining of multiple optimizations?

This section answers these questions and explains how the software architecture works as an enabler to achieve the necessary degree of flexibility. Figure 8.6 shows the `optimization` component and its content. The description follows Figure 8.6 top-down, with the starting points marked in orange.

The `OptimizationSpecification` class reflects the example configuration presented in Listing 10. An optimization task consists of a human-readable name, a model instance made of satellite data and density data, one or several Norad Catalog IDs, optimizations that shall be applied beforehand to the existing model instance, a list of one or multiple optimization intervals, and finally, parameters specifying the concrete optimization strategy. The `execute(..)` function calls `orchestrate_optimization(..)` with the aforementioned parameters. Hence, `OptimizationSpecification` is the object-oriented way to run an optimization task, while `orchestrate_optimization(..)` offers the same functionality via a functional interface.

The `ParameterRepository` is filled with the results of previous optimizations and, depending on the chosen settings, utilized to patch the model instance by updating ballistic coefficients/ or providing a set of density coefficients.

Next, there is the class `AbstractProblem`. It describes - in our context - always either a ballistic coefficient optimization or density coefficient optimization. The first subclass is `ProblemTemplate`. It is a composite of multiple different strategies. A `Strategy` is a single part of the optimization function. Each `Strategy` has one satellite and one epoch range. Depending on the concrete subclass, the `Strategy`'s `__call__(..)` function expects a candidate ballistic coefficient or candidate density coefficients and computes the fitness value for its satellite and epoch range. The `__call__(..)` function, i.e., its evaluation, is the variable and distinction characteristic between all strategies. They can be subdivided into three classes using propagation, the drag acceleration along a reference trajectory, or TLE-derived densities. Chapter 9 elaborates on the details of each strategy implemented throughout this work.

The `ProblemTemplate` calculates its fitness by combining the fitness values of its contained strategies. This accumulation function is configurable, similar to combining multiple strategies with the same goal into one `ProblemTemplate`. To summarize, the `ProblemTemplate` is a *composite* of multiple replaceable *strategies*.

In order to facilitate the creation of a `ProblemTemplate`, the `ProblemBuilder` has been created. It builds the problem by adding the cartesian product of satellites, epoch ranges, and

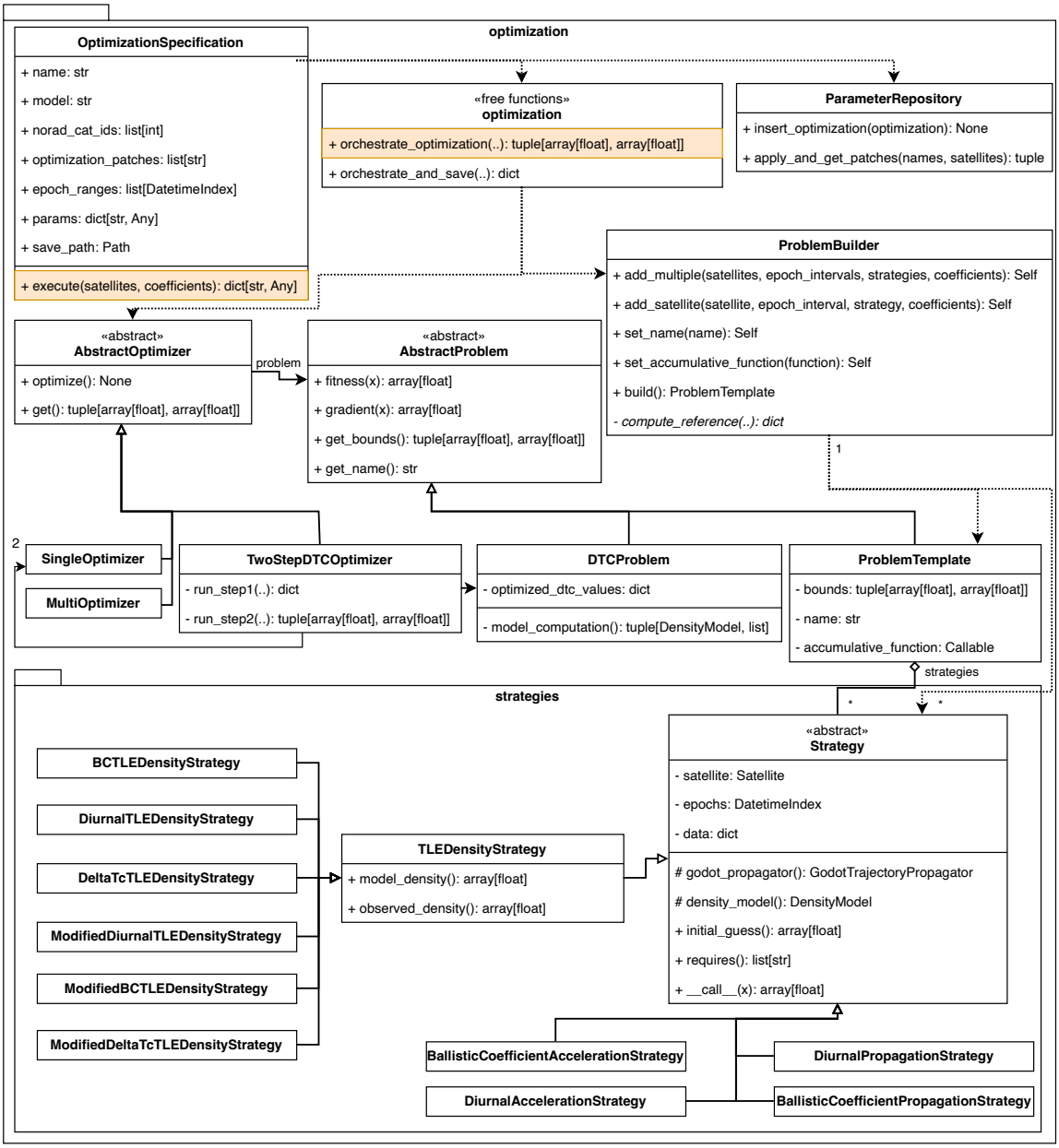


Figure 8.6.: A UML Class Diagram of the `optimization` component of `atmosopt` showing the strategies and implemented approaches of optimization. The diagram emphasizes the objected oriented entry and functional entry into the optimization control flow. Types are omitted for function arguments to improve readability. The diagram partially hides public methods and simplifies the content of certain classes to focus on the core design. The diagram's core is the various strategies representing a subset of a problem consisting of satellite, epoch range, and solution approach, which can be arbitrarily combined via the class `ProblemTemplate`.

concrete strategic methods. Including other satellites in the ballistic coefficient optimization of a single satellite does not make sense. Thus, the logic beforehand knows to exclude the satellites from the cartesian product and create a problem instance for each satellite using the `ProblemBuilder`. For instance, Listing 10 shows a configuration of three optimization tasks. The first two of them optimize the ballistic coefficient. Hence, the following problems are created:

BC-PROP For every satellite, one ballistic coefficient optimization problem with a single strategy "bc-prop" mapping to the class `BallisticCoefficientPropagationStrategy`.

BC-MIXED For every satellite, one ballistic coefficient optimization problem with two strategies of classes `BCTLEDensityStrategy` and `ModifiedBCTLEDensityStrategy` where both fitness functions are merged via the operation "sum".

JB2008-DTC-Diurnal-TLE One problem atmosphere diurnal coefficient optimization problem containing all satellites.

The following steps involve *pygmo* [84]. Each problem is wrapped into `pygmo.problem` and is optimized with a subclass of `AbstractOptimizer`. Subclasses of `AbstractProblem` qualifies themselves as `pygmo.problem` due to the presence of `fitness` and `get_bounds` method. The subclasses of `AbstractOptimizer`, in turn, use the `AbstractProblem` and evolve its internal population of candidate solution vectors.

To begin, the `SingleOptimizer` is the most straightforward case. It uses one of the available algorithms in *pygmo*, with a starting population of initial candidate solutions, and tries to minimize the fitness function. Depending on the option "gradient", an algorithm omitting or using the gradient is chosen. In our implementation, we fixed the algorithms to be *Extended Ant Colony Optimization Algorithm*[4] or in case gradient is enabled to *Sequential Quadratic Programming (SQP) Algorithm*[5]. The former uses a heuristic approach, whereas the latter uses the derivatives to find the optimum. It was not examined if these algorithms are the best choices. In order to compensate for that, the `MultiOptimizer` was created, which uses multiple algorithms for the minimization and allows the sharing of solutions between each algorithm's single candidate solution pool. However, extensive experimenting has yet to be conducted due to time constraints. The investigation of alternative fitness functions using different methods to compare ground truth and model was prioritized - more on this topic in Chapter 9.

The "JB2008-DTC-Diurnal-TLE" in Listing 10 has a unique role in the implementation. It is a two-step optimization procedure, which was created to reduce dimensionality (see Chapter 9). As such, it uses two instances of the `SingleOptimizer`. The first optimizes the ΔT_c of the JB2008 model for each configured trajectory. The second problem uses these optimal values to operate on the diurnal density function directly and optimizes its coefficients to fit the ΔT_c values. Since the latter problem `DTCProblem` does not use satellites at all, it does not fit into the inheritance hierarchy of `Strategy`.

In conclusion, the key takeaway of this section is that any combination of strategies (consisting of satellite, time range, and method) is possible due to the generality-trimmed architecture.

[4] `https://esa.github.io/pygmo2/algorithms.html`, last accessed: 09.02.2024
[5] `https://nlopt.readthedocs.io/en/latest/NLopt_Algorithms`, last accessed: 09.02.204

```yaml
---
atmosopt:
  optimization:
    - name: "BC-MIXED"
      with_model: "model"
      for_satellites: [22503, 24905, 25171, 34294, 38743, 42706, 42712, 42715, 42820,
                       42821, 42822, 42823, 42824, 43033, 44121, 44123, 44128, 44136,
                       44144, 44145, 44148, 44163, 44168, 44169, 44171 ]
      in_time_range:
        - from: "2019-04-06T00:00:00.000"
          to: "2019-04-30T23:59:59.000"
      params:
        population_size: 30
        generations: 100
        method: ["bc-tle", "bc-mod-tle"]
        accumulative_function: "sum"
        gradient: True
        bounds: [[1e-8], [400]]
      save_path: "./2019-05-04/optimization/bc_tle_mixed.npy"
    - name: "BC-PROP"
      with_model: "model"
      for_satellites: [22503, 24905, 25171, 34294, 38743, 42706, 42712, 42715, 42820,
                       42821, 42822, 42823, 42824, 43033, 44121, 44123, 44128, 44136,
                       44144, 44145, 44148, 44163, 44168, 44169, 44171 ]
      in_time_range:
        - from: "2019-04-06T00:00:00.000"
          to: "2019-04-30T23:59:59.000"
          frequency: "3h"
      params:
        population_size: 15
        generations: 10
        method: "bc-prop"
        gradient: True
        bounds: [[1e-8], [400]]
      save_path: "./2019-05-04/optimization/bc_prop.npy"
    - name: "JB2008-DTC-Diurnal-TLE"
      with_model: "model"
      with_patches: [ "BC-MIXED" ]
      for_satellites: [22503, 24905, 25171, 34294, 38743, 42706, 42712, 42715, 42820,
                       42821, 42822, 42823, 42824, 43033, 44121, 44123, 44128, 44136,
                       44144, 44145, 44148, 44163, 44168, 44169, 44171 ]
      in_time_range:
        - from: "2019-04-06T00:00:00.000"
          to: "2019-04-30T23:59:59.000"
      params:
        population_size: 15
        generations: 10
        method: "dtc-diurnal-tle"
        gradient: True
        alpha: 0.2
      save_path: "./2019-05-04/optimization/jb2008_dtc_diurnal_tle.npy"
# ... other configurations
```

Listing 10: Configuration of the optimization. Each optimization requires a model, and optionally a list of optimizations to be applied beforehand. Each optimization requires satellites, a epoch range and a list of methods.

8.5. Plotting Component

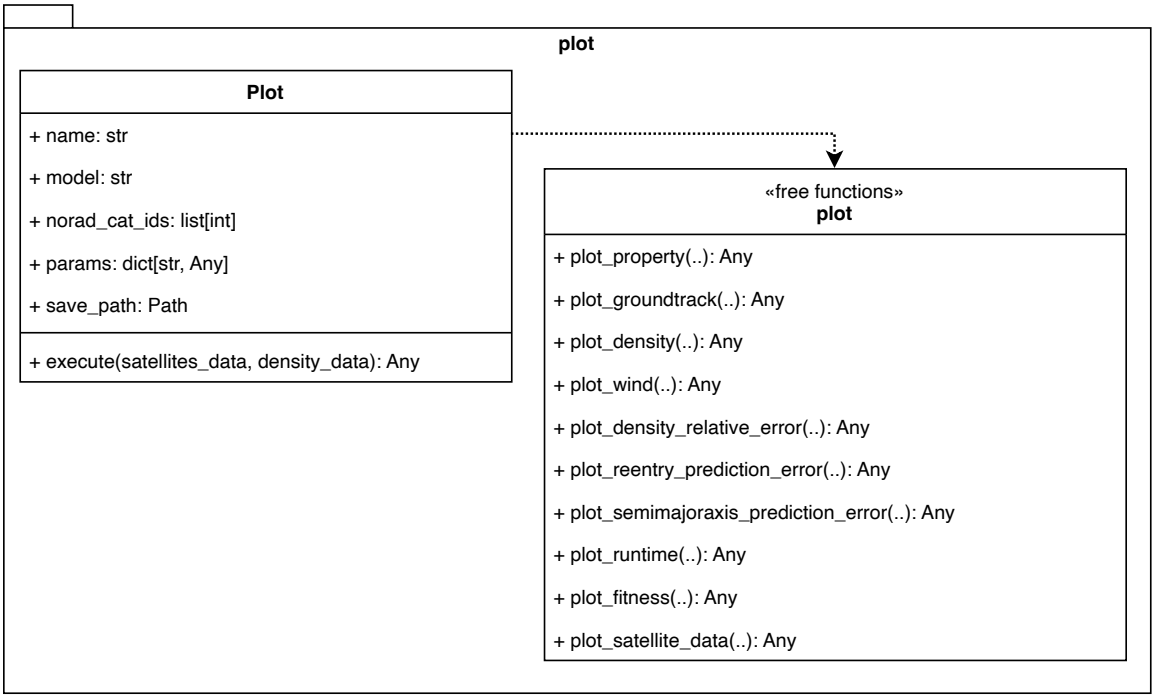


Figure 8.7.: A UML Class Diagram of the `plot` component of `atmosopt` showing the object oriented and functional interface to the plotting capabilities. Function arguments are mostly omitted to improve readability.

The last component `plot` is kept simple. It consists of a number of plotting functions, as illustrated in Figure 8.7. Similarly, to the `optimization` component, a object oriented and a functional interface exists. The former to be used via the configuration file (for instance in Listing 11), while the latter is to be preferred from within Python.

```yaml
---
atmosopt:
  plot:
    - name: density_relative_error
      with_model: model
      params:
        sample_size: 100000
        seed: 42
        from: 2019-04-01T00:00:00
        to: 2019-04-30T23:59:00
        plot_mean_std: True
        density_models: [ "JB2008" ]
        configurations:
          JB2008MOD:
            diurnal: "./2019-05-04/optimization/jb2008_dtc_diurnal_tle.npy"
      save_path: "./2019-05-04/output/jb2008mod_density_relative_error_30d.pdf"
    - name: reentry_prediction_error
      with_model: model
      params:
        norad_cat_id: [22503, 24905, 25171, 34294, 38743, 42706, 42712, 42715, 42820,
                       42821, 42822, 42823, 42824, 43033, 44121, 44123, 44128, 44136,
                       44144, 44145, 44148, 44163, 44168, 44169, 44171 ]
        initial_epoch: 2019-05-01T00:00:00.000
        overhang: "62D"
        stop_altitude: 100
        max_steps: 1000000
        uniform_color: True
        save_data: True
        configurations:
          NO-OPT:
            density: "JB2008"
          BC-PROP:
            density: "JB2008"
            ballistic_coefficient_source: "./2019-05-04/optimization/bc_prop.npy"
# ... other configurations
          JB-BC-MIXED:
            density: "JB2008MOD"
            density_coefficients:
              diurnal: "./2019-05-04/optimization/jb2008_dtc_diurnal_tle.npy"
            ballistic_coefficient_source: "./2019-05-04/optimization/bc_tle_mixed.npy"
      save_path: "./2019-05-04/output/prediction_error_start2.pdf"
```

Listing 11: Configuration of the plotting. The configuration specifies for each plot the type (denoted with name), the model and the parameters. The parameters usually specify the density models to use, the epoch range, and the satellites to be incorporated into the plot. Plots taking satellites usually also accept single IDs instead of lists. The configurations specify as a dictionary all propagation cases with results from previous optimizations. It contains the special case `ballistic_coefficient_source`. In this case, for every single satellite propagation, the Norad Catalog ID is appended to the file name. When these files are created, this convention is applied in the `optimization` component, too.

9. Optimization Strategies & Discussion

Figure 8.6 showed the strategies implemented throughout this work. These were the focus of this last part of the work. Thus, the primary assumption is that not the optimization algorithm is key, but the fitness function and the underlying method are needed to generate the required data. This chapter outlines the three methods to generate a fitness value. Fitness based on the comparison of

1. the semi-major axes given the TLE reference data and propagated results using the atmospheric model in `godot`
2. the drag acceleration along a reference trajectory spanned by the SGP4 propagation (computed by subtracting the gravitational acceleration) to the acceleration computed with the density of the atmospheric model
3. the TLE derived observed density to the model density (the approach by Picone et al. presented in Section 3.3)

Equation 2.55 states that atmospheric drag is computed by density and ballistic coefficient. Hence, all approaches are **applicable for both optimization goals** where only the free parameter varies.

Figure 9.1 demonstrates the overall structure of a ballistic coefficient optimization. The structure optimizing the diurnal coefficients B_i, C_i is identical, without the ballistic coefficient initialization. All fitness functions are presented as summations over the respective grid points. However, different reducer functions, for instance, *min* or *max* norm, are also conceivable. In our manual test cases, namely, the satellites with IDs 43600, 35867, and 32764, the summation was usually the best choice. Fregat, the satellite with Norad Catalog ID 35867, is also the object of interest in Section 10.1.

9.1. Ballistic & Diurnal Coefficients Optimization by Propagation

The first strategy uses the propagation capabilities of `godot`. Equation 9.1 shows the elementary fitness function utilized for a single strategy consisting of one satellite and an epoch interval with a configurable frequency of grid points. The relative error of the semi-major axes of propagated a_{prop} and SGP4 reference trajectory a_{ref} is computed and accumulated as fitness value. Thus, the optimization goal is the minimization of the relative semi-major axis difference at the grid points. If the propagation terminates earlier, indicating too much dag, the propagated semi-major axes a_{prop} of the missing entries are set to 0. The semi-major axis a_{prop} depends on propagation results computed with a solution candidate ballistic coefficient or diurnal coefficients.

$$\sum_{\forall \text{Epochs}} \left| \frac{a_{ref} - a_{prop}}{a_{ref}} \right| \tag{9.1}$$

In case the gird points lie on the TLE observation points, the method corresponds to the *shooting method*, presented in Subsection 2.4.2. Under the condition of a suitable parameter choice, namely the step size between temporary gird points, this method should always converge towards the optimal ballistic coefficient B. The method can be used with a gradient-based function, which approximates the derivative numerically using a finite difference. Gradient-based procedures are only advisable when the optimization goal is the ballistic coefficient. In the case of the 42 diurnal coefficients, the gradient's numerical evaluation becomes too expensive runtime-wise. The same fact holds for optimizing the diurnal coefficients without gradient when multiple satellites are part of the fitness function.

Each strategy evaluation implies one propagation with the full `godot` dynamic model over multiple weeks. Potentially, if the satellite is close to reentry, a second propagation is required (as discussed in Subsection 8.3.1).

For 15 satellites, a ballistic coefficient optimization with gradient takes around 20 minutes on the hardware specified in Appendix using as many parallel processes as available with Python's multiprocessing. In our case, this corresponds to ten processes.

When the same experiment is conducted for a diurnal optimization, the optimization does not finish in hours, even without a gradient. While the evaluation remains of the theoretical same size, i.e., 15 strategies, they are now part of the same problem. The major difference is the missing parallelization. The aforementioned scenario allows parallelization over every single problem. One could parallelize the evaluation of the single strategies. However, Python's multiprocessing limits parallelism to full process parallelism, which induces much overhead when starting new processes. For each iteration, all processes would be required to join for the evaluation of the shared fitness function in every single iteration - only to be re-created with a new work distribution afterward. This approach was tested but found to be not more performant than keeping everything in a single process. The scheduling and process creation induce too much overhead. Further, the propagator in its current state is not thread-safe. Thus, an approach different from process parallelism is not possible. The best solution would be a complete solution on the C++ side, migrating large parts of the current Python setup. On the C++ side, a thread-safe solution is possible. Further, the C++ side can take full advantage of less overhead inducing thread-level parallelism. To summarize, the propagation strategy is suitable for one-dimensional problems like the ballistic coefficient but is too expensive runtime-wise for higher-dimensional problems or problem combinations.

To overcome the issue, the idea was to search for algorithmic alternatives that do not require expensive propagation or propagation only once in the beginning. This investigation led to the subsequent presented idea, in Section 9.2.

9.2. Ballistic & Diurnal Coefficients Optimization by Acceleration

The second optimization strategy aims to compare the drag accelerations. Therefore, we make the strong assumption that the spacecraft is only affected by gravitation and drag acceleration. Other perturbations are ignored by assuming they are zero.

The spacecraft's trajectory is defined by `godot`'s polynomial spline SGP4 interpolation incorporating the TLE grid points in between the start and end epochs. The reference acceleration $\vec{a}_{ref}$ along the trajectory is computed using a numerical finite difference. Given

the initial assumption, the expected reference drag $\vec{a}_{drag}$ is assumed to be the total acceleration affecting the spacecraft minus the gravitation. Since we do not want to propagate, we assume $\vec{a}_{grav}$ to be defined as given by Equation 2.50. The actual drag of our model is computed along the reference trajectory using the selected density model and ballistic coefficient. The whole relation is given by Equation 9.2. Depending on the optimization goal, either the ballistic coefficient B is variable or the density ρ parameterized by the diurnal coefficients.

$$\sum_{\forall \text{Epochs}} \left| \frac{\vec{a}_{drag} - \frac{-0.5\rho|\vec{v}|}{B}}{a_{drag}} \right| \text{ where } \vec{a}_{drag} = \vec{a}_{ref} - \vec{a}_{grav} \tag{9.2}$$

The strategy is similar to the approach presented in Section 3.3 by Gondelach et al. with one big exception. They use a fully dynamic model and propagate with it to generate new candidate solutions. During testing, this approach initially showed some good results. This initial success turned out to be misleading. With an increasing number of satellites, it became apparent that the optimization always optimized towards the interval's bounds. This is a reason why the ballistic coefficient's optimization's bounds are set to $B \in \left[10^{-8}, 400\right]$ where the bounds are comparably low and high in Chapter 10. Hence, a misguided optimization scheme stands out quickly due to the extreme values appearing in the result. Also, a modification only using the prograde component of the drag did not help. The initial assumption may be too strong.

To summarize, this acceleration-based idea did not bring any success, but it put the investigation on the right track to a better working approach. The usage of these strategies in the implementation framework is discouraged.

9.3. Ballistic & Diurnal Coefficients Optimization using TLE Derived Densities

The third approach derives an atmospheric density from the TLE itself. It is the approach by Picone et al., introduced in Section 3.3. Equation 9.3 shows the fitness function utilized in the context of this work. Its central core is the scaling factor $\lambda(t_{ik})$ computed from the observed TLE density $\rho_O(t_{ik})$ (see Equation 3.4) and the model density $\rho_M(t_{ik})$ (see Equation 3.5). The observed and modeled density match if the $\lambda(t_{ik})$ is one. A deviation marks that the model density ρ_M is either too high or too low.

$$\sum_{\forall \text{TLE Intervals } t_{ik}} (1 - \lambda(t_{ik}))^2 \tag{9.3}$$

One can use this approach to optimize the model density, respectively, the coefficients of the density-producing model. Further, optimizing the ballistic coefficient is also possible since ρ_M depends on the ballistic coefficient B.

Chapter 10 shows that the computation of TLE Densities produces acceptable results with low runtime costs compared to the techniques involving propagation with the full dynamic model of `godot`. Hence, it is suitable for continuous optimization of large amounts of data.

In the implementation, we also make use of a slightly modified procedure.

- Equation 3.4 is multiplied by $6.66231957e-11$, which results from consequent straight usage of the TLE unit format, e.g., mean motion in $\frac{rev}{day}$, instead of SI units.
- The velocity in both Equation 3.4 and Equation 3.5 is only utilized linearly and in $\frac{km}{s}$ instead of $\frac{m}{s}$.

This modification was originally an implementation error. However, it turned out to work better, i.e., producing smaller errors in all observed experimental cases compared to the unmodified version. Hence, we left the modified version as an option to remain in the implementation framework.

9.4. Diurnal Coefficients Optimization - Resolving The Problem of Dimensionality

As previously discussed, the runtime cost has been a primary concern in optimizing the diurnal coefficients throughout this work. All the above-presented strategies, even the ones involving propagation with the full dynamic model, are suitable for optimization with numerical gradient computation under the condition that the optimization variable is one-dimensional. The diurnal coefficients of the JB2008 model have a dimensionality of 42. Optimization with a numerical gradient gets too costly runtime-wise. The optimization does not necessarily have to be conducted in as little time as possible. A diurnal density correction ideally should run, as the names suggest, daily - hence, potentially overnight using a set of calibration satellites like in the case of HASDM.

As mentioned, a migration of large parts of the implementation to C++ to better exploit thread-level parallelization and to tighten the connection to `godot` capable of providing gradients would be one suitable option. This option was rejected since this implies the transferal of large systems of the existing framework. Further, this approach is less flexible regarding changes, given that implementation needs to comply with its architecture and enforced constraints. Finally, the learning curve to get into `godot`'s problem optimization module is steep. Instead of searching for a more performant solution, the algorithmic side was improved by reducing dimensionality.

The JB2008 diurnal density variation is realized by an offset ΔT_c computed in its own function given by Equations 2.8–2.25. It uses the fitted diurnal coefficient B_i and C_i. It depends on the solar activity $F_{10.7}$, the local solar time θ, the latitude ϕ and the height h. We first optimize JB2008 regarding the output ΔT_c of the diurnal density function. This procedure yields a mapping of satellite, TLE interval start and end to optimal offset $\{satellite, t_i, t_k\} \rightarrow \Delta T_c$. Thereby, we reduce the expensive full-density evaluation to a one-dimensional problem. This optimization can be executed with numerical gradient information with any of the above techniques. Due to the good results, we use the modified TLE-density-based strategy for this first step, even though any strategy would be applicable. Using these optimal ΔT_c, we optimize the computational cheap Equations 2.8–2.25 in a second step, which explains why the diurnal function was exposed from Fortran to Python in Section 5.5. Since the function resides inside the `DensityModel` class, we consistently take advantage of caching computation points and epochs on the C++ side. The UML Activity Diagram in Figure 9.2 illustrates the entire procedure.

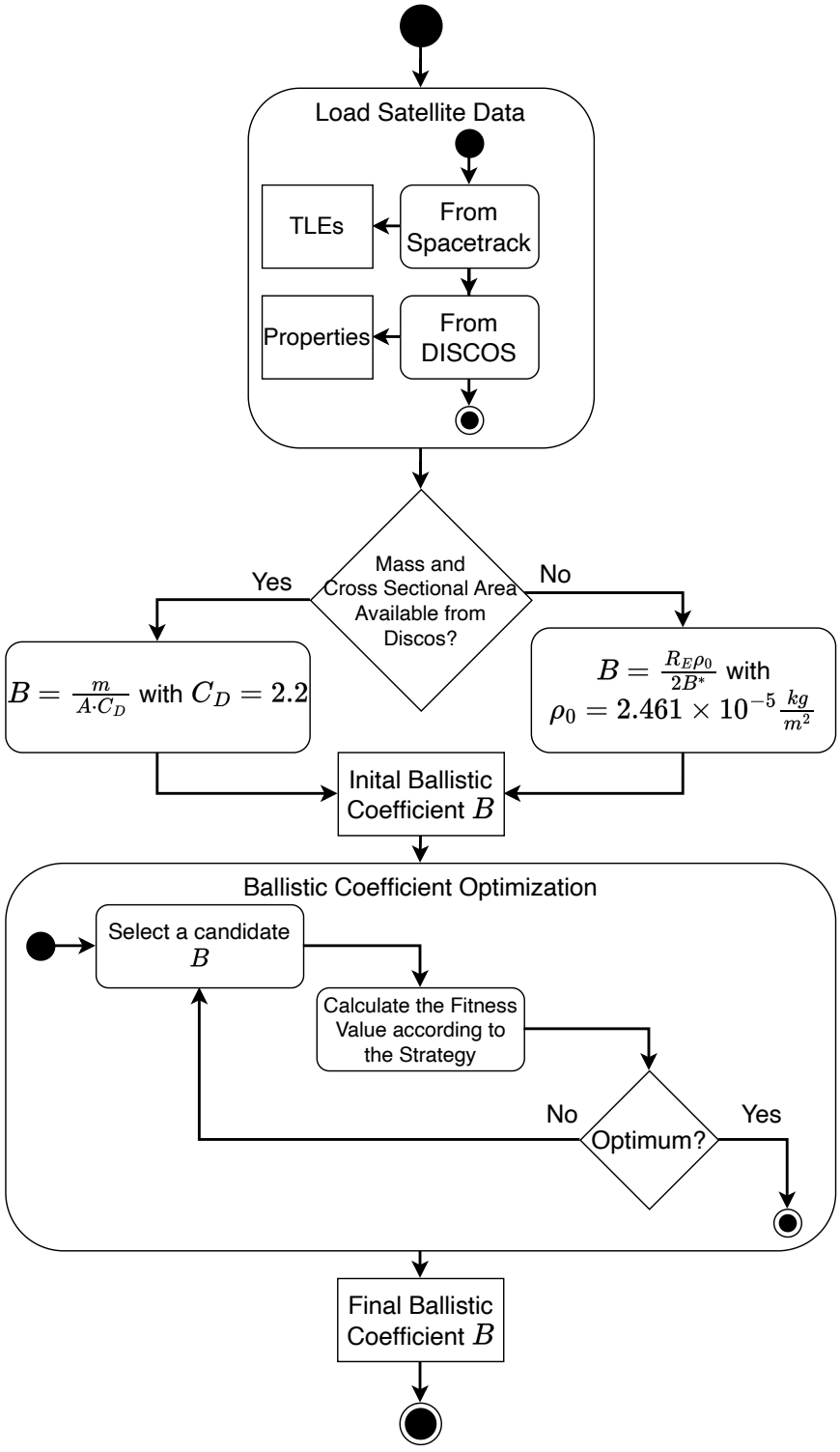


Figure 9.1.: A UML Activity Diagram displaying the determination of the final value as ballistic coefficient B for the atmospheric optimization process. The optimization of the diurnal coefficients B_i, C_i works similar by replacing the ballistic coefficient B with them in the diagram. The initial value is taken from the definition of the JB2008 model.

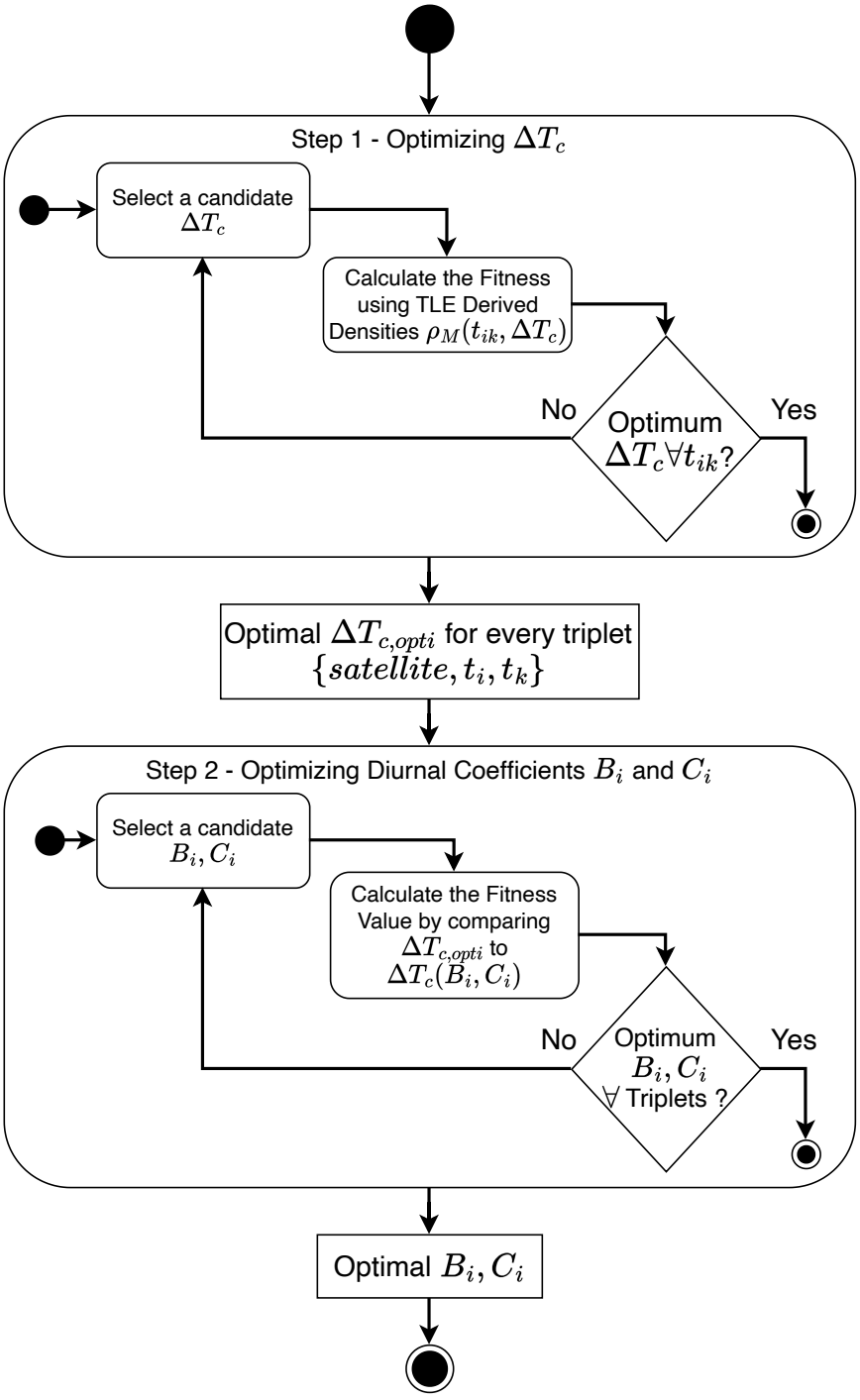


Figure 9.2.: A UML Activity Diagram displaying the two step optimization scheme to find suitable diurnal coefficients B_i, C_i. The first step only requires a single ΔT_c and a gradient can be computed by numerical means with only one additional evaluation. The second step can also be evaluated with a numerical gradient since the underlying 42-dimensional function is computationally cheap compared to the full JB2008 density model.

10. Results & Discussion

This chapter contains two examples. Section 10.1 focuses on qualitatively improving a single historic reentry of the debris fragment Fregat. Section 10.2 takes a more quantitative approach by comparing the implemented techniques for multiple satellites in 2014, 2018, and 2019. Like all plots of this work, the plots of this chapter are the automatic output of the `atmosopt` framework. All results are based on historic reentry events. Thus, we always have the actual solar and geomagnetic activity data available. Further, all results use the JB2008 without any diurnal coefficient correction if not differently specified in the plot.

The reentry prediction error is slightly adapted compared to Equation 2.58, as given by Equation 10.1. We set a minimal value of one day in the denominator. In the large-scale quantitative comparison, the propagation of all satellites starts at a uniform epoch. However, the amount of satellites can contain (varies) some satellites that actually decay on the first day of propagation. In such cases, a minor deviation by only a few minutes leads to extremely high relative errors. The maximum constraint introduced by Equation 10.1 prevents this from happening.

$$E_p = \frac{\text{Epoch}_{\text{predicted reentry}} - \text{Epoch}_{\text{actual reentry}}}{\max\left(\text{Epoch}_{\text{prediction}} - \text{Epoch}_{\text{actual reentry}}, 1\,\text{Day}\right)} \tag{10.1}$$

Given Equation 10.1, a negative error indicates a too-early decay. A positive error indicates a reentry event happening too late. The negative error is bounded to minimally -1 in case the target reenters immediately on propagation start. Although the positive error is unbounded in theory, in practice, the plots presented here always limit the maximal propagation duration so that it does not stretch into the far future until the propagator times out. Hence, the error is bound. Chapter 8 illustrated that the implementation framework support arbitrary combinations of strategies. In this Chapter 10, we decided to use the approaches given by Table 10.1. This includes one application of the acceleration strategy of Section 9.2, four applications of the propagation strategy of Section 9.1 and three use cases of the TLE-density-based approach given in Section 9.3. Due to runtime constraints, this Chapter 10 does not contain an atmosphere optimization with one of the base strategies, but only the reduced two step optimization of Section 9.4 using ΔT_c as intermediate value.

10.1. Optimizing a single Reentry Prediction: Fregat

This Section 10.1 optimizes the reconstruction of the reentry of the satellite Fregat with Norad Catalog ID 35867. It decayed on 13.03.2021 at 09:56 UTC somewhere over the Pacific Ocean, given its Tracking and Impact Prediction (TIP) message.

Figure 10.1 shows the results using ballistic coefficient optimization along the data available from 13.01.2021 to 13.02.2021 and all implemented techniques. The propagation starts on

Abbreviation	Method	Goal	Frequency	Grad.	Runtime (hh:mm:ss)
BC-ACC	Acceleration	$B \in [10^{-8}; 400]$	1 Minute	True	00:00:02
BC-PROP-S	Propagation		1 Hour		00:05:27
BC-PROP			3 Hours		00:06:15
BC-PROP-L			6 Hours		00:04:50
BC-PROP-XL			24 Hours		00:04:27
BC-TLE	TLE-Derived Density		10 seconds Integration Step-size $\forall$ TLE intervals $[t_i, t_k]$		00:00:47
BC-TLE-MIXED					00:02:30
BC-TLE-MOD					00:00:46
JB-BC-MIXED	Diurnal Two Step Optimization B_i, C_i indirect via ΔT_c with previously optimized B by BC-TLE-MIXED	B_i, C_i with maximally $\pm 20\%$ deviation to initial values			03:28:59 (for $N = 45$ satellites given in Figure 10.5)
NO-OPT	No Optimization, B from DISCOS or TLE, Standard B_i, C_i of JB2008				

Table 10.1.: The methods and their specification utilized in this Chapter 10. The runtime is reported for the procedures in Figure 10.1 for the satellite Fregat with exception of the JB-BC-MIXED procedure. It would make little sense to try an atmosphere optimization with a single satellite and only one month of data. Hence, the runtime is reported for the procedure in Figure 10.5 with $N = 45$ satellites.

13.02.2021 in Figure 10.1 and is cut off on 13.05.2021 if no reentry occurred until then. The optimization took place with all available data ranging 13.01.2021 until 13.02.2021.

DISCOS contains characteristic data for Fregat. Thus, the initial values are pretty good, and we achieve, even with no optimization, a reentry error of only -18.6%.

We employed the propagation strategy with different interval sizes. This interval size for the fit is essential, as Lemmens et al. [2] found out. In this scenario, `BC-PROP-L` performs the best with an interval frequency between grid points of six hours. It achieves a relative reentry prediction of 8.7%. The worst propagation strategy is `BC-PROP-XL` with an error of 97.2%. Overall, this strategy contrasts the "shooting method" in Subsection 2.4.3, which uses a variable interval based on the distance between known TLE epochs. Due to the generality of `atmosopt`, we decided not to include this concrete strategy here but rather make the interval size a configurable parameter. This allows us to conduct experiments similar to Lemmens et al. [2], who searched for an optimal fitspan $\Delta T \in [5; 110]$ and interval size $\delta t \in [1; 36]$ to optimize B and reduce the reentry prediction error depending on the orbital region of the satellite. They report median reentry errors ranging from 14% to 134%. Hence, our relative errors conform to the values found in the literature.

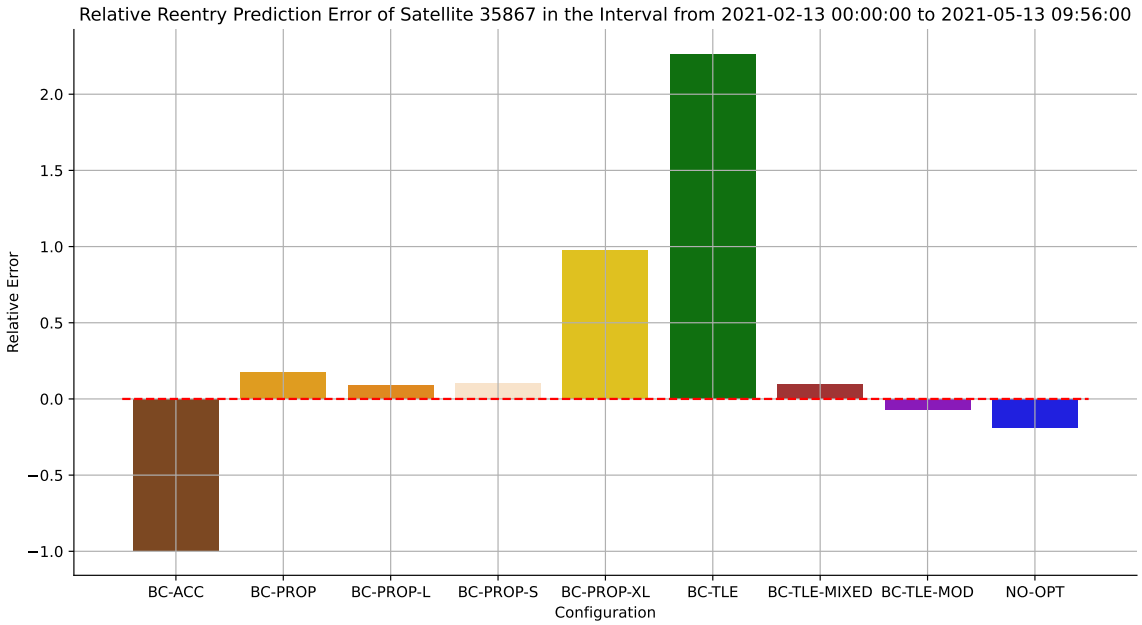
As previously reported, Figure 10.1 shows that `BC-ACC` optimizes towards the bounds of the interval. This misguided optimization leads to an extreme value of the ballistic coefficient. Figure 10.1b illustrates the relative error. The satellite immediately decays when propagation starts.

The original TLE density-based approach `BC-TLE` performs interestingly the worst with an error of 226%. Our modified version `BC-TLE-MOD` produces overall the best results and decreases the error to -6.9%. A accumulative mixture of both mentioned techniques `BC-MIXED` shifts the prediction towards a positive error of 9.2%.

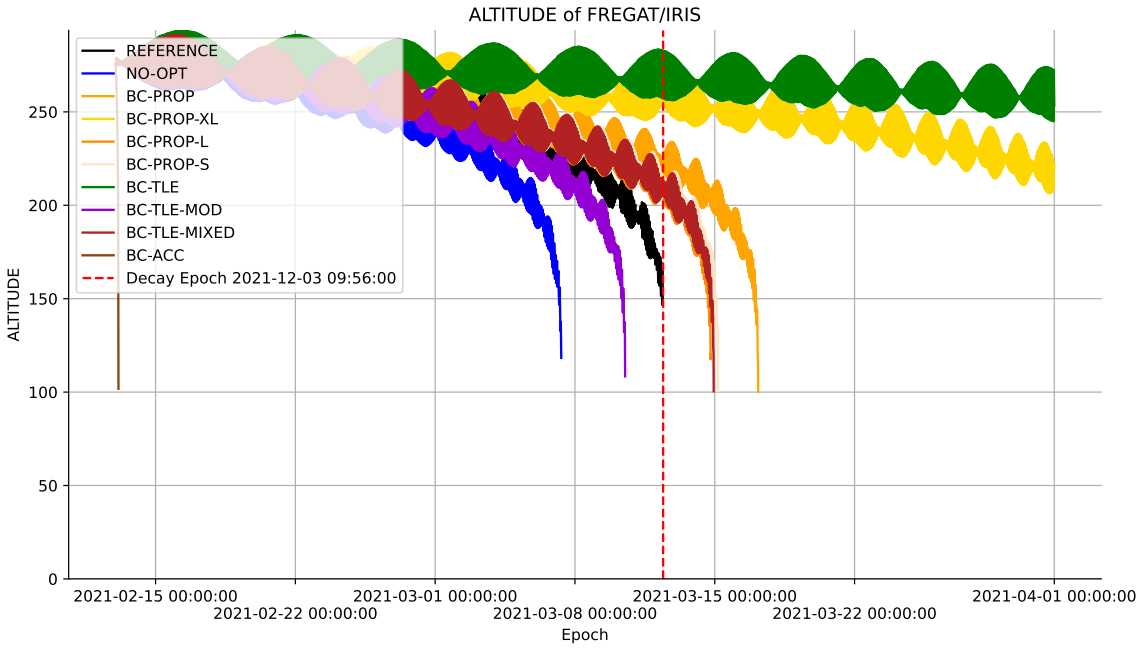
Table 10.1 also shows the runtime results. The mixed `BC-TLE-MIXED` procedure has approximately 2-3 times the runtime of the single TLE-density based procedures. Given that, it contains two of them and needs to reduce the result, this seems to be reasonable. Overall, the `BC-TLE-MOD` produces a good result while being considerable faster than `BC-PROP` or its kind of strategies.

Virgili et al. conducted the reentry of Fregat similarly to us with multiple atmospheric models and approaches [86]. Their relative errors 25 days before reentry range from 10% to 30% with a single exceptional good result for the DTM2013 at around 3%. Consequently, our results fit the reported values, with our best case being `BC-TLE-MOD` with a relative error of -6.9%.

Figure 10.2 shows a ground-track using the best B from `BC-TLE-MOD`. The propagation starts on the day of reentry at midnight. In theory, one would fit here again using data from March closely before the reentry. Nonetheless, we used the previously best B from the optimization.



(a) The Reentry Prediction Error. The propagation was capped until 13.05.2021. It started one month before on 13.02.2021.



(b) The Altitude of Fregat. The propagation was capped to the 01.04.2021

Figure 10.1.: Plots of the satellite Fregat 35867. The ballistic coefficient optimization has been run from 13.01.2021 to 13.02.2021. The actual reentry happened on the 13.03.2021 at around 09:56:00.

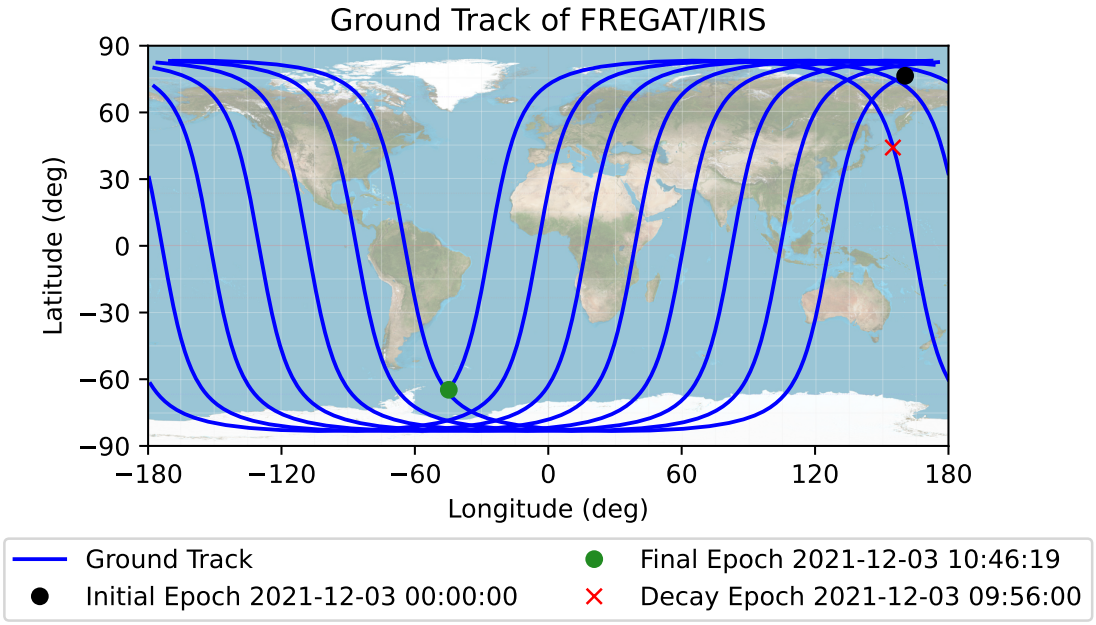


Figure 10.2.: The ground track of Fregat's reentry beginning at 00:00:00 UTC on the day of reentry. We use the best ballistic coefficient from `BC-TLE-MOD`

10.2. Large-Scale Optimization Techniques Comparison

A quantitative comparison of more than 60 satellites available in August 2019 with their ballistic coefficients optimized yielded no difference compared to a scenario without any optimization. The performance was then measured by checking the semi-major axis error between improved prediction with the optimized ballistic coefficient and the reference trajectory derived from the TLEs. The experiment showed no significant difference using ballistic coefficient optimization compared to no optimization. This was later tracked down to the concrete satellite selection. All 60 satellites still had a lifetime of at least multiple months, and their altitudes were too high. Thus, atmospheric drag did not play a significant role for any of the satellites, leading to no differences in the results. Another source of distortion holds for satellites that conduct maneuvers. Space-Track does not provide details about conducted maneuvers. Our selection of satellites might always contain some satellites whose orbit is influenced by maneuvers. These distort the optimization and lead to wrong optimization results.

The first problem can be countered by selecting only satellites that are known to be decayed anytime soon after the optimization interval. This usually implies that atmospheric drag has a sufficiently significant impact on the satellite's orbital state. However, the second problem of distortions due to maneuvers might still exist in the data presented hereafter.

Figures 10.3 to 10.5 show the same results but different satellites at different epochs: 2014, 2018, and 2019. In all cases, every satellite was optimized with all available TLE data

for the whole month before reentry. The start prediction epoch for the relative prediction error is the first day after the optimization epoch, i.e., the first day of the month in which the satellites decayed. The propagation for the relative reentry error is capped at 62 days after the satellite with the last decay date actually decayed. Due to runtime constraints, only `BC-PROP` and `BC-PROP-L` are evaluated. Further, we optimize the JB2008's diurnal coefficients using the two-step optimization scheme from Section 9.4. The JB2008's diurnal optimization uses the satellites with optimized ballistic coefficients from `BC-TLE-MIXED`.

The year 2014, depicted in Figure 10.5, is chosen for being the year of high solar activity during solar cycle 24, while the year 2019, depicted in Figure 10.3 is chosen for being the year of low solar activity. Finally, Figure 10.4 depicts the situation during a different year's season.

In all case, all potential satellites from Space-Track fulfilling the following criteria have been selected:

- The satellite has one TLE available before the optimization interval and at least 2 TLEs inside the optimization interval (e.g., for Figure 10.5, all satellites have at least two TLEs in March 2014, and one TLE available in February 2014)
- The satellite decayed in the month after the optimization interval (e.g., for Figure 10.5, all satellites decayed in April 2014)

The number of satellites fulfilling the selected criteria varies from scenario to scenario—for instance, $N = 17$ in 2018 and $N = 45$ in 2014.

The results in Figures 10.3 to 10.5 show that, usually, every method has its outliers. Each method performs differently for every satellite. This way, the median $|m|$ is included in the Figures 10.3 to 10.5, which is also commonly used across literature in this area due to being more stable towards outliers. Figures 10.3 to 10.5 contain the mean $|\mu|$, standard deviation σ, and median $|m|$ values to support the visual distinction. In order to not produce distorted results, mean $|\mu|$ and median $|m|$ value are computed from all errors $|E_p|$ without sign.

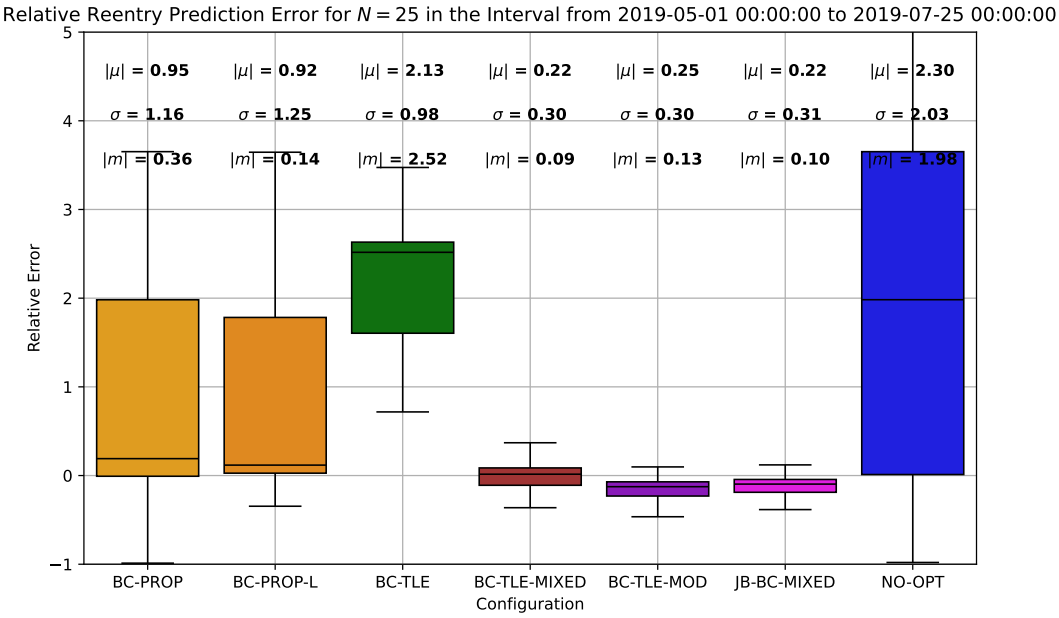
The propagation method `BC-PROP-L` performs well in 2019 with a median of only 14%, depicted in Figure 10.3. Also, its 3-hour companion `BC-PROP` performs decently. Nonetheless, both methods struggle in the 2014 scenario, depicted in Figure 10.5 with median errors above 237%. As mentioned, Lemmens et al. [2] report that the frequency is critical to an optimal ballistic coefficient optimization. Thus, in our cases, the method might be non-suitable parameterized. One can also argue that the difference lies in solar activity, as 2019 and 2018 are years of low solar activity while 2014 is a year of high solar activity. More experiments would need to be conducted to make a strong statement about solar activity.

Similarly to the results observed for Fregat, the base `BC-TLE` method performs in all cases worse compared to the modified variant. In contrast, the modified approach performed in 2019 excellently with an `BC-TLE-MIXED` achieving a relative error median of 9%, and even the mean value is small with only an error of 22%. These are exceptionally good results comparable to the state-of-art in reentry prediction, as discussed in Subsection 2.4.3. The median for the combined technique `BC-TLE-MIXED` remains good with a value of 24% in 2014, depicted by Figure 10.5, but drops to a value of 43% in November 2018. The 2018 scenario depicts a period of different semiannual density variation, as discussed in Subsection 2.1.2. Again, to make a strong claim about the relation to semiannual density variation, the experiment should be repeated for every year available. Nonetheless, given the

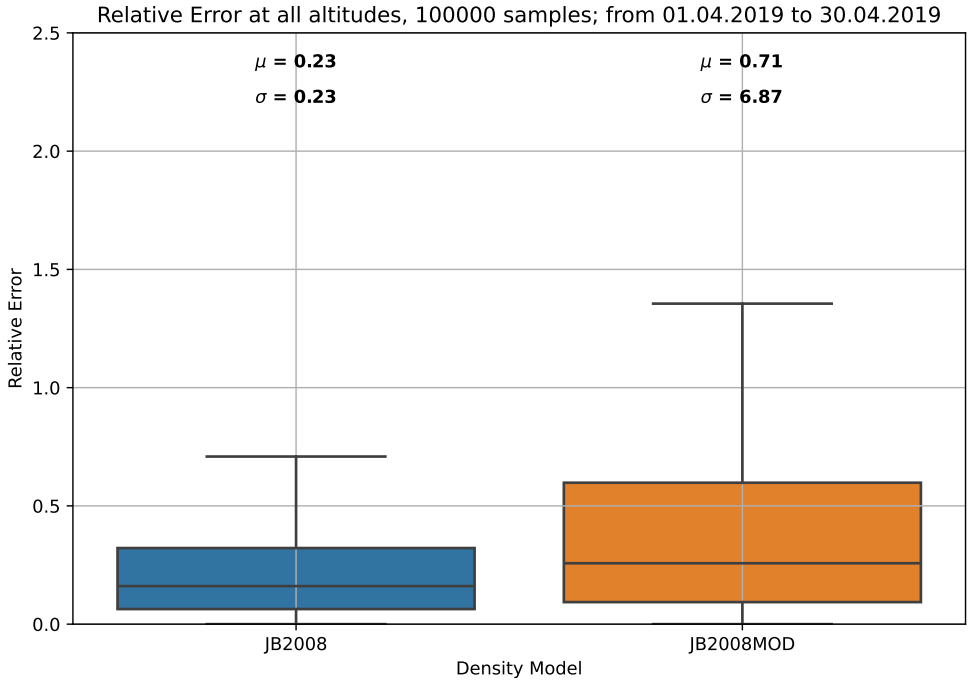
accuracy and runtime of `BC-TLE-MIXED`, one concludes that this is overall the best approach developed in this thesis.

The diurnal atmospheric optimization of JB2008 was only conducted with the two-step optimization scheme. Given that the procedure `BC-TLE-MIXED` is the foundation of the approach, one would expect further improvement. Nonetheless, in all observed cases, the accuracy of the reentry prediction worsens. Figures 10.3 to 10.5 also depicts the optimized density model using the optimized set of diurnal coefficients and the base JB2008 model. Both of them are compared to the HASDM data set inside the optimization interval. The modified JB2008 is worse in all cases and produces even unrealistic density value in large amounts in Figure 10.4, leading to extremely high mean relative errors. This behavior indicates that the density was not optimized, but rather, the optimization converged towards a different goal. Over the course of this work, multiple such optimizations were conducted. However, similarly to these results, none of them could improve the global density prediction. Some improved at least the reentry error, but most did little to nothing compared to the good results achieved with the ballistic coefficient optimization.

To conclude, the diurnal atmospheric optimization of JB2008 with the chosen methods did not work. This remains the subject of future work, even though it is considerably simplified due to the software framework on which one can build. In contrast, the more significant topic of reentry optimization overall can be treated as a success. One has to restrict that all results reported here are conducted a posterior, with geomagnetic and solar activity being known. We achieve reentry errors comparable to the current state of the art using an efficient propagation-less method to optimize the ballistic coefficient in an interval before the re-entry.

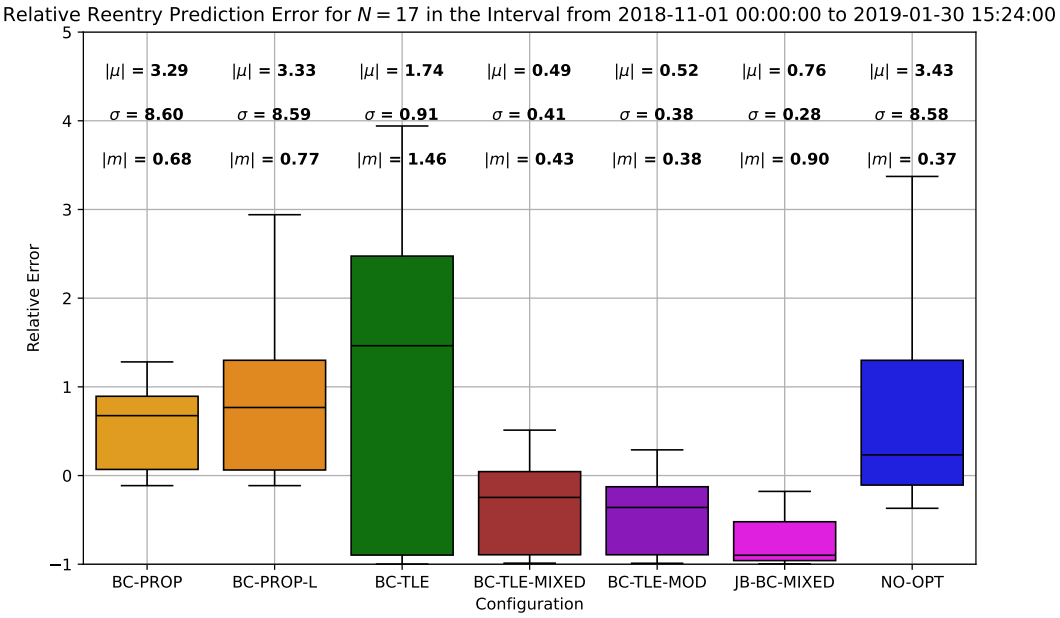


(a) The Reentry Prediction Error, Epoch of Prediction: 2019-05-01 00:00 UTC

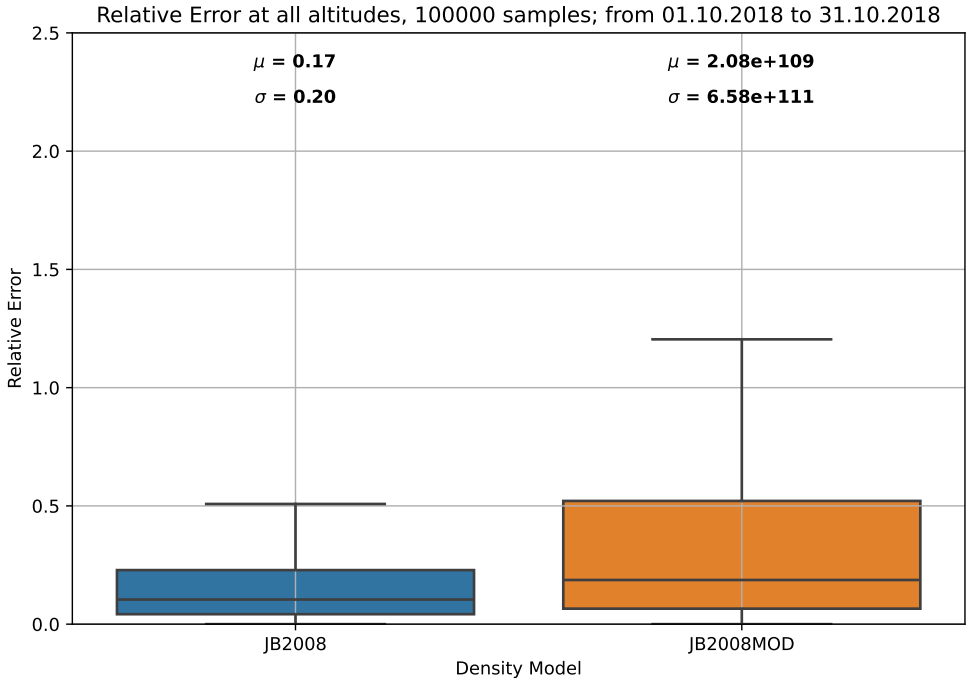


(b) Standard JB2008 vs. JB2008 with modified diurnal coefficients inside the optimization interval

Figure 10.3.: The optimization includes all satellites ($N = 25$) available via *Space-Track* which decayed in May 2019. All satellite have at least two TLE states during the optimization interval in April, and minimally one TLE before and after the optimization interval in April 2019.

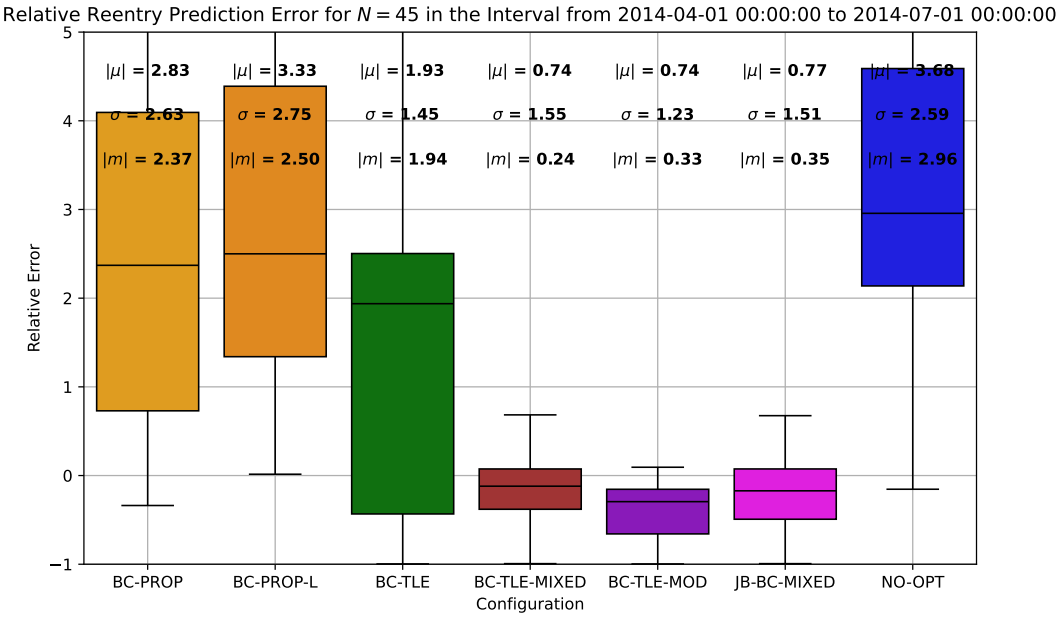


(a) The Reentry Prediction Error, Epoch of Prediction: 2018-11-01 00:00 UTC

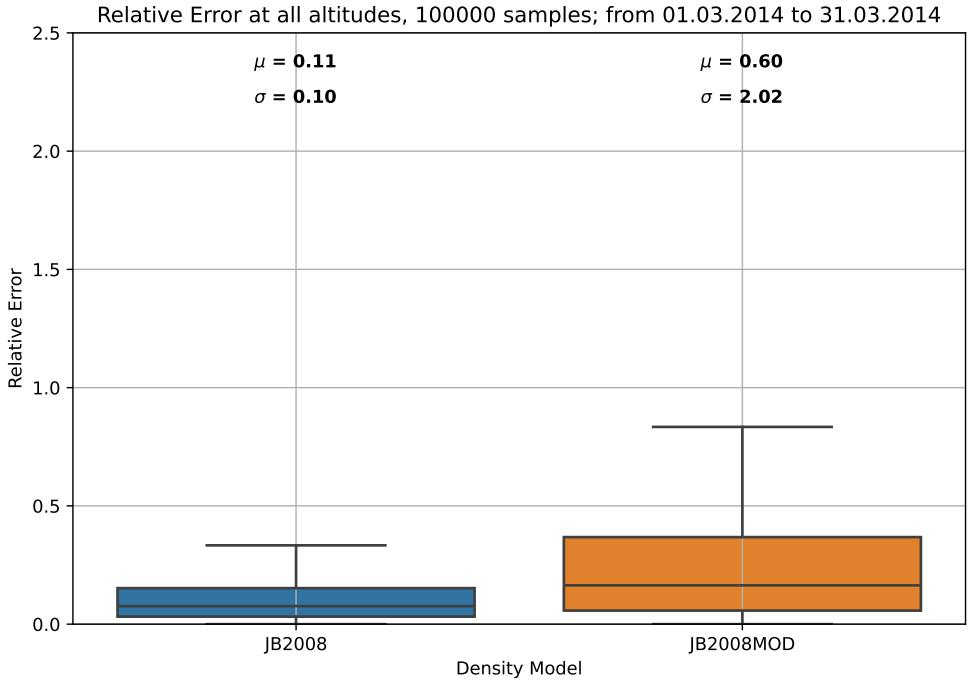


(b) Standard JB2008 vs. JB2008 with modified diurnal coefficients inside the optimization interval

Figure 10.4.: The optimization includes all satellites ($N = 17$) available via *Space-Track* which decayed in November 2018. All satellite have at least two TLE states during the optimization interval in October, and minimally one TLE before and after the optimization interval in October 2018.



(a) The Reentry Prediction Error, Epoch of Prediction: 2014-04-01 00:00 UTC



(b) Standard JB2008 vs. JB2008 with modified diurnal coefficients inside the optimization interval

Figure 10.5.: The optimization includes all satellites ($N = 45$) available via *Space-Track* which decayed in April 2014. All satellites have at least two TLE states during the optimization interval in March, and minimally one TLE before and after the optimization interval in March 2014.

Part IV.

Conclusion and Future Work

11. Conclusion & Summary

Over the course of this thesis, we presented the topic of atmospheric reentry in the context of two software projects aiming to extend atmosphere-related functionality in `godot` and to enhance the predictability of the atmosphere on satellite states with `atmosopt`. Part I presented the background of major density model families: DTM, Jacchia, and MSIS. It laid the foundation for modeling reentries and which properties are of significant interest when modeling atmospheric drag. These are namely the ballistic coefficient and the atmospheric density. It also provided a first look at the later incorporated approach by Picone et al. [74], which was utilized to derive density values from TLE values.

Next, the key contributions of this work regarding Part II are:

- The refactoring of the atmospheric core component of the flight dynamics library `godot` of the European Space Agency by refactoring a separate wind and density `TimeEvaluable` class
- The integration of several well-known historic and current density & wind models as Plugin `godotAtmosph` given a uniform straightforward to use interface easily extendable by new models and sources of solar activity data
- A slim python wrapper to evaluate an arbitrary density or wind model

These additions lay the foundation for the `godot` community to more easily conduct experiments with atmospheric density models, while the `godotAtmosph` plugin already allows from now on to propagate anything with a wide variety of established models - even more easily with the framework implemented in Part III. Chapter 7 has presented some maps of the global atmospheric density distribution during high and low solar activity years. The JB2008 model comes the closest to predicting the density of the HASDM SET Density Database. Part III stands out for the creation of a comprehensive framework for the goal of reentry prediction with a focus on JB2008 including:

- A unified approach to access and efficiently store TLE data, satellite characteristics and SET HASDM density data
- A model capable of quickly propagating and predicting a satellite trajectory without much configuration overhead
- An optimization framework allowing the arbitrary combination of strategies, satellite inputs, and time ranges, which can be easily configured to optimize the ballistic coefficient or diurnal coefficients of JB2008 quickly
- A wide selection of plots to quickly visualize the data directly configurable from the configuration file
- Overall, an effortless way to make a reentry prediction

Both parts are written in a way that facilitates future extension and modifications. The overall optimization of the diurnal coefficients of JB2008 has not been successful. The density data produced by the optimized JB2008MOD fits the satellites utilized for optimizing but does not generalize to the global density model. In most cases, propagation with the optimized coefficients even leads to degraded reentry prediction performance, indicating an inadequate selection of input satellites or an insufficiently designed fitness function. So, while this part did not bring the desired success, the reentry optimization does. We have implemented three ways to optimize satellite drag: propagation, acceleration, and TLE-derived densities. Especially the last method proved highly valuable due to the low runtime costs and the associated speed to optimize a large mass of satellites simultaneously compared to the existing shooting method using propagation. The median reentry error of the sub-method with best accuracy `BC-TLE-MIXED` lies at around 9% in 2019, comparable to the state-of-the-art reentry error obtainable with public data and without dedicated resources. The method generalizes well to data in 2014, with a median of 24%, but struggles on the satellite selection in November 2018, with a median reentry error of 43%. Nonetheless, even the latter values can be considered respectable given the usual reentry prediction errors from literature.

To conclude, the novelty of this approach lies in something other than the utilized strategies. These existed beforehand. The novelty is the software framework, which fulfills the entrance vision of "all one needs is a satellite ID" - the implementation framework with minimal configuration automatically determines the rest. Further, the implementation is not yet another script working for a dedicated preselected quantity of satellites but designed to function and deliver for any input, allowing to try out methods and arbitrary parameter combinations - simply implementable as a new strategy - on a large scale.

12. Future Work

While the thesis is split into two major parts, so can be future work divided into two parts.

The development of the `godotAtmosph` plugin, presented in Part II, is software-wise limited by the capabilities and architectural constraints of `godot`. Two major improvements can be identified.

First, Section 7.2 showed the high runtime overhead enforced by input processing compared to the raw Fortran atmosphere model. The introduction of per-atmosphere model caching could improve the runtime for timeline-wise continuous propagation. This is a low-effort addition. However, it needs to be implemented for every atmosphere model individually because the data required to be cached varies. Another improvement is the introduction of ranged-based access to a `TimeEvaluable` in `godot`. This core feature is internally discussed at the time of writing and hopefully available in the future due to its significant usefulness in the context of consecutive solar and geomagnetic index data. When this update is available, an update to `godotAtmosph` would also be beneficial.

Second, the atmosphere models do not implement a derivative. The current and previous implementations in `godot` compute the derivative by a simple numerical finite difference. This process could be improved as there are libraries that can do this task with less overhead. Thus, a valuable contribution could be the replacement of these code fragments using, e.g., *Enzyme AD*[1], which provides an automatic derivative for the Fortran implementations [87]. If this is correctly set up, it would also be beneficial for computing partial derivatives regarding properties other than the position, such as the solar indices or the coefficients of an atmosphere model.

The other area of improvement is `atmosopt`, presented in Part III. The current implementation is primarily based in Python, with the only exceptions being the propagation and atmosphere evaluation. This decision had the great advantage that ideas could be quickly prototyped as convenient I/O functions are directly usable from the language standard or reputable libraries like *numpy* or *pandas*. However, the decision also limits the obtainable performance significantly, as Python's GIL does not allow proper parallelism. Further, all user-defined problems in *pygmo* are by default not thread-safe, disallowing any other parallelization than the one on the process level. Therefore, an improvement of the implementation could focus on two areas. Either the capabilities of `godot`'s `Problem` class are further investigated and made usable. Or alternatively, the `atmosopt`'s optimization module is largely migrated into `godot_atmosph`. The primary beneficiaries of both approaches are the optimization strategies involving propagation with `godot`'s full dynamic model. In contrast, the other strategies would only benefit from faster setup times because propagation only happens once in the beginning.

The latter idea promises quicker realization as no considerations regarding `godot`'s architecture need to be made beforehand `godot` would be only used as a library. Instead,

[1] `https://github.com/EnzymeAD/Enzyme`, last accessed: 07.01.2024

pygmo's C++ backbone *pagmo* is utilized together with thread-level parallelism for, e.g., the propagation. The optimization would then be a single function call delegating to C++ and returning the results of the optimization. This way, most components of `atmosopt` would remain the same, with only notable changes inside the internals of the `optimization` component.

The former idea of using `godot`'s `Problem` has already been investigated on the surface level in the context of optimizing the ballistic coefficient. The main challenge is the automatic setup of the problem's configuration. This configuration process has improved with `godot` 1.5. However, a more extensive package of work would still be required to achieve a similar degree of customizability and adaptability to arbitrary tasks. This contribution would then result in a new ensemble of `AbstractOptimizer` and `AbstractProblem`. More work is required in order also to optimize the diurnal coefficients. They need to be incorporated into the `TimeEvaluable`-based system. Further, the configuration of the problem would be complex as many trajectories need to be combined in a single problem - something better implemented via builder/ assembler interface in core `godot`.

While the above suggestions focus only on implementation ideas, other directions are also possible. Here, an investigation of the available strategies and parameters could be desirable. While this work implemented an extensive framework, the large-scale try-out phase came short, with only a limited selection of strategy combinations presented. However, the implementation invites us to try the optimization with different methods, to vary the fitness function, or to explore different scenarios. The possibilities are endless, and the implementation framework is designed to be built upon and to be used.

Part V.

Appendix

Computational Hardware

All experiments are conducted on an Apple MacBook Pro with

- M1 Pro Chip with 10 (8 performance, 2 efficiency) cores
- 16 GB of LPDDR5 RAM

List of Acronyms

AD Anno Domini

BC Before Christ

BCRS Barycentric Celestial Reference System

CCMC Community Coordinate Modeling Center

CEP Celestial Ephemeris Pole

CIO Celestial Intermediate Origin

CIP Celestial Intermediate Pole

CIRA72 COSPAR International Reference Atmosphere of 1972

CIRS Celestial Intermediate Reference System

DTM2013 Drag Temperature Model 2013

DTM2020 Drag Temperature Model 2020

DCA Dynamic Calibration Atmsophere

DISCOS Database and Information System Characterising Objects in Space

DOY Day of Year

Dst Disturbance Storm Time

ECEF Earth-centered Earth-fixed

ECI Earth-centered inertial

ERA Earth Rotation Angle

ESA European Space Agency

ESOC European Space Operations Centre

ET Ephemeris Time

EUV extreme ultra violet

GAST Greenwich Apparent Sidereal Time

GCRS Geocentric Celestial Reference System

GMST Greenwich Mean Sidereal Time

GOCE Gravity field and steady-state ocean circulation explorer

GOES Geostationary Operational Environmental Satellite

GOST2004 Upper Earth Atmosphere Density Model for Ballistic Support of AES Flight

HASDM High Accuracy Satellite Drag Model

HWM07 Horizontal Wind Model 2007

HWM14 Horizontal Wind Model 2014

HWM93 Horizontal Wind Model 1993

IAU International Astronomical Union

ICRF International Celestial Reference Frame

ICRS International Celestial Reference System

ITRF International Terrestrial Reference Frame

ITRS International Terrestrial Reference System

J2000 January 1, 2000, 12.00 h TT

JACLIN Jacchia-71/ Lineberry Model

JB2006 Jacchia-Bowman 2006

JB2008 Jacchia-Bowman 2008

JB2008MOD JB2008 Modified

JD Julian Date

LASCO Lifetime Assessment of Catalogued Objects

LAST Local Apparent Sidereal Time

LEO Low Earth Orbit

LMST Local Mean Sidereal Time

MSIS Mass Spectrometer and Incoherent Scatter radar

MSIS00 NRLMSIS-00

MSIS77 MSIS Neutral Thermosphere Model of 1977

MSIS86 MSIS-86/CIRA 1986 Neutral Thermosphere Model

MSIS90 Neutral Atmosphere Empirical Model from the surface to lower exosphere MSISE90

MSISv2 NRLMSIS 2.0

NORAD North American Aerospace Defense Command

NRL US Naval Research Laboratory

SDO Space Debris Office

SGP4 Simplified General Perturbations 4

SGP8 Simplified General Perturbations 8

SOHO Solar and Heliospheric Observatory

SORCE Solar Radiation and Climate Experiment

SRP Solar Radiation Pressure

SSN Space Surveillance Network

TAI International Atomic Time

TDB Barycentric Dynamic Time

TDT Terrestial Dynamic Time

TEME True Equator Mean Equinox

TIO Terrestrial Intermediate Origin

TIP Tracking and Impact Prediction

TIRS Terrestrial Intermediate Reference System

TLE Two Line Element Set

TT Terrestrial Time

US1976 US Standard Atmsophere of 1976

UT Universal Time

UT1 Universal Time No. 1

UTC Coordinated Universal Time

List of Figures

List of Tables

Bibliography

[1] E. S. D. Office, "ESA'S ANNUAL SPACE ENVIRONMENT REPORT," tech. rep., ESA Space Debris Office, 2023.

[2] S. Lemmens, B. Bastida Virgili, V. Braun, T. Flohrer, Q. Funke, H. Krag, F. Mclean, and K. Merz, "From end-of-life to impact on ground: An overview of ESA's tools and techniques to predicted re-entries from the operational orbit down to the Earth's surface," in *Proceedings of the 6th International Conference on Astrodynamics Tools and Techniques, Darmstadt, Germany*, 2016.

[3] W. K. Tobiska, B. R. Bowman, S. D. Bouwer, A. Cruz, K. Wahl, M. D. Pilinski, P. M. Mehta, and R. J. Licata, "The SET HASDM density database," *Space Weather*, vol. 19, no. 4, p. e2020SW002682, 2021.

[4] K. Bunte, H. Sdunnus, J. Mandeville, and H. Klinkrad, "Ballistic parameter and lifetime assessment for catalogued objects," in *Space Debris*, vol. 473, pp. 781–786, 2001.

[5] E. Doornbos, *Thermospheric density and wind determination from satellite dynamics*. Springer Science & Business Media, 2012.

[6] K. F. Wakker, "Fundamentals of astrodynamics," *TU Delft Repository, Delft*, 2015.

[7] J. Emmert, "Thermospheric mass density: A review," *Advances in Space Research*, vol. 56, no. 5, pp. 773–824, 2015.

[8] S. Elvidge, H. C. Godinez, and M. J. Angling, "Improved forecasting of thermospheric densities using multi-model ensembles," *Geoscientific Model Development*, vol. 9, no. 6, pp. 2279–2292, 2016.

[9] C. He, Y. Yang, B. Carter, E. Kerr, S. Wu, F. Deleflie, H. Cai, K. Zhang, L. Sagnières, and R. Norman, "Review and comparison of empirical thermospheric mass density models," *Progress in Aerospace Sciences*, vol. 103, pp. 31–51, 2018.

[10] L. Yin, L. Wang, W. Zheng, L. Ge, J. Tian, Y. Liu, B. Yang, and S. Liu, "Evaluation of empirical atmospheric models using Swarm-C satellite data," *Atmosphere*, vol. 13, no. 2, p. 294, 2022.

[11] S. Bruinsma, C. Siemes, J. T. Emmert, and M. G. Mlynczak, "Description and comparison of 21st century thermosphere data," *Advances in Space Research*, 2022.

[12] N. W. Service, "Hello Solar Cycle 25." `https://www.weather.gov/news/201509-solar-cycle`, last accessed: 05.12.2023, September 2020.

[13] H. Paetzold and H. Zschörner, "An annual and a semiannual variation of the upper air density," *Geofisica pura e applicata*, vol. 48, pp. 85–92, 1961.

[14] D. R. Weimer, M. Mlynczak, J. Emmert, E. Doornbos, E. Sutton, and L. Hunt, "Correlations between the thermosphere's semiannual density variations and infrared emissions measured with the SABER instrument," *Journal of Geophysical Research: Space Physics*, vol. 123, no. 10, pp. 8850–8864, 2018.

[15] T. Fuller-Rowell, "The "thermospheric spoon": A mechanism for the semiannual density variation," *Journal of Geophysical Research: Space Physics*, vol. 103, no. A3, pp. 3951–3956, 1998.

[16] M. Jones Jr., J. T. Emmert, D. P. Drob, J. M. Picone, and R. R. Meier, "Origins of the Thermosphere-Ionosphere Semiannual Oscillation: Reformulating the "Thermospheric Spoon" Mechanism," *Journal of Geophysical Research: Space Physics*, vol. 123, no. 1, pp. 931–954, 2018.

[17] B. R. Bowman, W. K. Tobiska, and M. J. Kendra, "The thermospheric semiannual density response to solar EUV heating," *Journal of Atmospheric and Solar-Terrestrial Physics*, vol. 70, no. 11-12, pp. 1482–1496, 2008.

[18] D. A. Vallado and D. Finkleman, "A critical assessment of satellite drag and atmospheric density modeling," *Acta Astronautica*, vol. 95, pp. 141–165, 2014.

[19] W. K. Tobiska, B. R. Bowman, and S. D. Bouwer, "Solar and geomagnetic indices for the JB2008 thermosphere density model," 2008.

[20] B. Bowman, W. K. Tobiska, F. Marcos, C. Huang, C. Lin, and W. Burke, "A new empirical thermospheric density model JB2008 using new solar and geomagnetic indices," in *AIAA/AAS astrodynamics specialist conference and exhibit*, p. 6438, 2008.

[21] T. D. de Wit and S. Bruinsma, "The 30 cm radio flux as a solar proxy for thermosphere density modelling," *Journal of Space Weather and Space Climate*, vol. 7, p. A9, 2017.

[22] J. Matzka, C. Stolle, Y. Yamazaki, O. Bronkalla, and A. Morschhauser, "The Geomagnetic Kp Index and Derived Indices of Geomagnetic Activity," *Space Weather*, vol. 19, no. 5, p. e2020SW002641, 2021.

[23] I. A. Daglis, R. M. Thorne, W. Baumjohann, and S. Orsini, "The terrestrial ring current: Origin, formation, and decay," *Reviews of Geophysics*, vol. 37, no. 4, pp. 407–438, 1999.

[24] C. Pardini, K. Moe, and L. Anselmo, "Thermospheric density model biases at the 23rd sunspot maximum," *Planetary and Space Science*, vol. 67, no. 1, pp. 130–146, 2012.

[25] L. G. Jacchia, "Static diffusion models of the upper atmosphere with empirical temperature profiles," *SAO Special Report*, vol. 170, 1964.

[26] I. Harris and W. Priester, "Time-dependent structure of the upper atmosphere," *Journal of the Atmospheric Sciences*, vol. 19, no. 4, pp. 286–301, 1962.

[27] P. Cefola, I. Volkov, and V. Suevalov, "Description of the Russian upper atmosphere density model GOST-2004," *37th COSPAR Scientific Assembly*, vol. 37, p. 476, 2008.

[28] N. Oceanic and A. Administration, *US standard atmosphere*. National Oceanic and Atmospheric Administration, 1976.

[29] D. P. Drob, J. T. Emmert, J. W. Meriwether, J. J. Makela, E. Doornbos, M. Conde, G. Hernandez, J. Noto, K. A. Zawdie, S. E. McDonald, *et al.*, "An update to the Horizontal Wind Model (HWM): The quiet time thermosphere," *Earth and Space Science*, vol. 2, no. 7, pp. 301–319, 2015.

[30] D. Drob, J. Emmert, G. Crowley, J. Picone, G. Shepherd, W. Skinner, P. Hays, R. Niciejewski, M. Larsen, C. She, *et al.*, "An empirical model of the Earth's horizontal wind fields: HWM07," *Journal of Geophysical Research: Space Physics*, vol. 113, no. A12, 2008.

[31] B. R. Bowman, W. K. Tobiska, F. A. Marcos, and C. Valladares, "The JB2006 empirical thermospheric density model," *Journal of Atmospheric and Solar-Terrestrial Physics*, vol. 70, no. 5, pp. 774–793, 2008.

[32] L. G. Jacchia, "New static models of the thermosphere and exosphere with empirical temperature profiles," *SAO special report*, vol. 313, 1970.

[33] J. Picone, A. Hedin, D. P. Drob, and A. Aikin, "NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues," *Journal of Geophysical Research: Space Physics*, vol. 107, no. A12, pp. SIA–15, 2002.

[34] J. T. Emmert, M. Jones Jr, D. E. Siskind, D. P. Drob, J. M. Picone, M. Stevens, S. M. Bailey, S. Bender, P. F. Bernath, B. Funke, *et al.*, "NRLMSIS 2.1: An empirical model of nitric oxide incorporated into MSIS," *Journal of Geophysical Research: Space Physics*, vol. 127, no. 10, p. e2022JA030896, 2022.

[35] J. T. Emmert, D. P. Drob, J. M. Picone, D. E. Siskind, M. Jones Jr, M. Mlynczak, P. F. Bernath, X. Chu, E. Doornbos, B. Funke, *et al.*, "NRLMSIS 2.0: A whole-atmosphere empirical model of temperature and neutral species densities," *Earth and Space Science*, vol. 8, no. 3, p. e2020EA001321, 2021.

[36] S. Bruinsma, "The semi-empirical thermosphere model DTM2012," in *6th European Conference on Space Debris*, vol. 723, p. 39, 2013.

[37] S. Bruinsma and C. Boniface, "The operational and research DTM-2020 thermosphere models," *Journal of Space Weather and Space Climate*, vol. 11, p. 47, 2021.

[38] C. Boniface and S. Bruinsma, "Uncertainty quantification of the DTM2020 thermosphere model," *Journal of Space Weather and Space Climate*, vol. 11, p. 53, 2021.

[39] G. H. Kaplan, "The IAU resolutions on astronomical reference systems, time scales, and Earth rotation models," *arXiv preprint astro-ph/0602086*, 2006.

[40] B. Luzum and G. Petit, "The IERS Conventions (2010): Reference systems and new models," *Proceedings of the International Astronomical Union*, vol. 10, no. H16, pp. 227–228, 2012.

[41] U. Walter, *Astronautics - The Physics of Space Flight*. Springer, third ed., 2018.

[42] P. Charlot, C. Jacobs, D. Gordon, S. Lambert, A. de Witt, J. Böhm, A. Fey, R. Heinkelmann, E. Skurikhina, O. Titov, *et al.*, "The third realization of the international celestial reference frame by very long baseline interferometry," *Astronomy & Astrophysics*, vol. 644, p. A159, 2020.

[43] C. Boucher and Z. Altamimi, "ITRS, PZ-90 and WGS 84: current realizations and the related transformation parameters," *Journal of Geodesy*, vol. 75, pp. 613–619, 2001.

[44] Z. Altamimi, P. Rebischung, X. Collilieux, L. Métivier, and K. Chanard, "ITRF2020: An augmented reference frame refining the modeling of nonlinear station motions," *Journal of Geodesy*, vol. 97, no. 5, p. 47, 2023.

[45] J. S. Subirana, J. J. Zornoza, and M. Hernández-Pajares, *GNSS Data Processing, Volume I: Fundamentals and Algorithms*. European Space Agency, 2013.

[46] N. Capitaine, "The Impact of the New IAU Resolutions on ICRF Definition and Realization," in *Reference Frames for Applications in Geosciences*, pp. 165–173, Springer, 2013.

[47] J. Seago and D. Vallado, "Coordinate frames of the us space object catalogs," in *Astrodynamics Specialist Conference*, p. 4025, 2000.

[48] D. Vallado, P. Crawford, R. Hujsak, and T. Kelso, "Revisiting spacetrack report# 3," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, p. 6753, 2006.

[49] R. R. Bate, D. D. Mueller, J. E. White, and W. W. Saylor, *Fundamentals of astrodynamics*. Courier Dover Publications, 2020.

[50] G. Team, "GODOT Software Specification Document," tech. rep., European Space Agency, 2022.

[51] T. Kelso, F. Hoots, and R. Roehrich, "Spacetrack report no. 3-models for propagation of norad element sets," *NASA, Tech. Rep*, 1988.

[52] R. Flores, B. M. Burhani, and E. Fantino, "A method for accurate and efficient propagation of satellite orbits: A case study for a molniya orbit," *Alexandria Engineering Journal*, vol. 60, no. 2, pp. 2661–2676, 2021.

[53] P. Cefola, J. San-Juan, S. Setty, R. Proulx, *et al.*, "Review of the Draper semi-analytical satellite theory (DSST)," in *Proceedings of 18th Australian International Aerospace Congress*, 2019.

[54] J. San-Juan, R. López, S. Setty, A. M., and P. J. Cefola, "Current status of the Draper Semi-analytical Satellite Theory (DSST)." GODOT Community Workshop, July 2023.

[55] R. Razali and M. S. Kamal Adnan, "Mathematical Modeling and software development on pertubation effects towards small satellite mission on low earth orbit (LEO)," *International Journal of Applied Science nad Technology*, vol. 1, no. 1, pp. 54–57, 2011.

[56] K. Fox, "Numerical integration of the equations of motion of celestial mechanics," *Celestial mechanics*, vol. 33, pp. 127–142, 1984.

[57] O. Montenbruck, "Numerical integration methods for orbital motion," *Celestial Mechanics and Dynamical Astronomy*, vol. 53, pp. 59–69, 1992.

[58] G. Team, "GODOT documentation," 2023.

[59] E. Fehlberg, "Classical fifth-, sixth-, seventh-, and eight-order Runge-Kutta formulas with stepsize control," *NASA Technical Report TR R-287*, October 1968.

[60] E. Fehlberg, "Klassische Runge-Kutta-Formeln fünfter und siebenter Ordnung mit Schrittweiten-Kontrolle," *Computing*, vol. 4, no. 2, pp. 93–106, 1969.

[61] J. Verner, "Jim Verner's Refuge for Runge-Kutta Pairs." `https://www.sfu.ca/~jverner/`, last accessed: 03.12.2023, 2023.

[62] F. HOOTS, "An analytical satellite theory using gravity and a dynamic atmosphere," in *Astrodynamics Conference*, p. 1409, 1982.

[63] F. R. Hoots and R. G. France, "An analytic satellite theory using gravity and a dynamic atmosphere," *Celestial Mechanics*, vol. 40, no. 1, pp. 1–18, 1987.

[64] F. Hoots, "A short efficient analytical satellite theory," *Journal of Guidance, Control, and Dynamics*, vol. 5, no. 2, pp. 194–199, 1982.

[65] F. R. Hoots, "Theory of the motion of an artificial earth satellite," *Celestial Mechanics*, vol. 23, no. 4, pp. 307–363, 1981.

[66] P. M. Mehta, S. N. Paul, N. H. Crisp, P. L. Sheridan, C. Siemes, G. March, and S. Bruinsma, "Satellite drag coefficient modeling for thermosphere science and mission operations," *Advances in Space Research*, 2022.

[67] G. March, E. Doornbos, and P. Visser, "High-fidelity geometry models for improving the consistency of CHAMP, GRACE, GOCE and Swarm thermospheric density data sets," *Advances in Space Research*, vol. 63, no. 1, pp. 213–238, 2019.

[68] S. Lemmens, B. B. Virgili, T. Flohrer, F. Gini, H. Krag, and C. Steiger, "Calibration of radar based re-entry predictions," in *Proceedings of the 5th International GOCE User Workshop. ESA Publications Division, European Space Agency, Noordwijk, The Netherlands*, 2014.

[69] R. J. Licata, P. M. Mehta, W. K. Tobiska, and S. Huzurbazar, "Machine-learned HASDM thermospheric mass density model with uncertainty quantification," *Space Weather*, vol. 20, no. 4, p. e2021SW002915, 2022.

[70] E. Doornbos, H. Klinkrad, and P. Visser, "Use of two-line element data for thermosphere neutral density model calibration," *Advances in Space Research*, vol. 41, no. 7, pp. 1115–1122, 2008.

[71] F. A. Marcos, M. Rendra, J. Griffin, J. Bass, D. Larson, and J. Liu, "Precision low earth orbit determination using atmospheric density calibration," *The Journal of the astronautical sciences*, vol. 46, pp. 395–409, 1998.

[72] P. Cefola, R. Proulx, A. Nazarenko, and V. Yurasov, "Atmospheric density correction using two line element sets as the observation data," *Advances in the Astronautical Sciences*, vol. 116, pp. 1953–1978, 01 2003.

[73] C. Shi, W. Li, L. Min, Q. Zhao, and J. Sang, "Calibrating the scale of the NRLMSISE00 model during solar maximum using the two line elements dataset," *Advances in Space Research*, vol. 56, 03 2015.

[74] J. Picone, J. Emmert, and J. Lean, "Thermospheric densities derived from spacecraft orbits: Accurate processing of two-line element sets," *Journal of Geophysical Research: Space Physics*, vol. 110, no. A3, 2005.

[75] D. J. Gondelach, R. Armellin, A. A. Lidtke, *et al.*, "Ballistic coefficient estimation for reentry prediction of rocket bodies in eccentric orbits based on TLE data," *Mathematical Problems in Engineering*, vol. 2017, 2017.

[76] B. R. Bowman and M. F. Storz, "High Accuracy Satellite Drag Model(HASDM) review," *Advances in the Astronautical Sciences*, vol. 116, pp. 1943–1952, 2003.

[77] S. Casali and W. Barker, "Dynamic calibration atmosphere (DCA) for the high accuracy satellite drag model (HASDM)," in *AIAA/AAS astrodynamics specialist conference and exhibit*, p. 4888, 2002.

[78] B. Bowman and M. Storz, "Time series analysis of HASDM thermospheric temperature and density corrections," in *AIAA/AAS astrodynamics specialist conference and exhibit*, p. 4890, 2002.

[79] M. F. Storz, B. R. Bowman, M. J. I. Branson, S. J. Casali, and W. K. Tobiska, "High accuracy satellite drag model (HASDM)," *Advances in Space Research*, vol. 36, no. 12, pp. 2497–2505, 2005.

[80] D. A. Vallado and P. J. Cefola, "Two-line element sets–practice and use," in *63rd International Astronautical Congress, Naples, Italy*, pp. 1–14, 2012.

[81] T. Flohrer, S. Lemmens, B. Bastida Virgili, H. Krag, H. Klinkrad, E. Parrilla, N. Sanchez, J. Oliveira, and F. Pina, "DISCOS-current status and future developments," in *Proceedings of the 6th European Conference on Space Debris*, vol. 723, pp. 38–44, 2013.

[82] F. McLean, S. Lemmens, Q. Funke, and V. Braun, "DISCOS 3: An improved data model for ESA's database and information system characterising objects in space," in *7th European Conference on Space Debris*, vol. 11, pp. 43–52, 2017.

[83] G. Team, "GODOT Introduction and System Architecture." GODOT Community Workshop, July 2023.

[84] F. Biscani and D. Izzo, "A parallel global multiobjective framework for optimization: pagmo," *Journal of Open Source Software*, vol. 5, no. 53, p. 2338, 2020.

[85] C. Förste, F. Flechtner, R. Schmidt, R. Stubenvoll, M. Rothacher, J. Kusche, H. Neumayer, R. Biancale, J.-M. Lemoine, S. Loyer, *et al.*, "EIGEN-05C-A new global mean

Gravity Field Model from Combination of Satellite Mission and Altimetry/Gravimetry Surface data," in *General Assembly European Geosciences Union (EGU)(Vienna, Austria 2007)*, 2007.

[86] B. B. Virgili, S. Lemmens, and E. Stevenson, "The impact of the new NRLMSIS 2.0 on re-entry predictions," in *Proceedings of 8th European Conference on Space Debris (Virtual), Darmstadt, Germany*, pp. 20–23, 2021.

[87] W. Moses and V. Churavy, "Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 12472–12485, Curran Associates, Inc., 2020.