State-of-the-art study of Python libraries for orbit propagation, comparison of poliastro & Tudatpy with CelestLab & GMAT

Arnaud Muller
Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, SWITZERLAND

As part of an effort to streamline the space mission design workflow of the CREME satellite mission into a single-language process, this study provides a list of open-source Python libraries capable of orbit propagation and a comparison of their feature set. Two libraries selected as prime candidates are compared across three different force models with trusted reference tools based on their numerical results, computation time and ease of use. Both are found to be suitable options for different use cases, with faster and more modular computations being their strongest advantages, whereas the reference tools remain particularly compelling for their accessibility when used independently.

Nomenclature

The following acronyms, abbreviations and symbols are used throughout this document:

AOP	=	Argument of perigee
CSV	=	Comma-separated values
Ecc	=	Eccentricity
ECEF	=	Earth-centered Earth-fixed
ECI	=	Earth-centered inertial
EOM	=	End of mission
GS	=	Ground station
GUI	=	Graphical user interface
Inc	=	Inclination
J2,3,4+	=	Earth geopotential model zonal coefficients
JVM	=	Java virtual machine
LEO	=	Low Earth Orbit
MA	=	Mean anomaly
RAAN	=	Right-ascension (longitude) of the ascending node
RK4	=	Runge-Kutta 4
SGP4	=	Simplified general perturbations propagator
SMA	=	Semi-major axis
SRP	=	Solar radiation pressure
SSO	=	Sun-synchronous orbit
TA/TAN	=	True anomaly
TLE	=	Two-line element set

I. Introduction

As part of the ongoing development of the CREME (Cubesat Radiation Environment Monitoring Experiment) satellite mission [1], this study aims to provide an overview of orbit propagation Python libraries able to replace other tools in the space mission design workflow.

This programming language is becoming increasingly popular in education and research environments, both for its ease of use and significant capabilities, notably thanks to numerous open-source libraries. Therefore, replacing other tools by open-source Python libraries allows for more streamlined and modular workflows as well as more verifiable results.

First, a brief state-of-the-art comparison of existing libraries is provided. Libraries providing SGP4 [2] as only propagator are not considered in this study, as its context is preliminary mission design and not operations, where TLEs are more prevalent. Then, the rest of the study compares two Python libraries selected for their ease of use and feature set with two reference tools widely used in the French aerospace academic community. The results comparison is not aimed at finding one library to be more accurate than others, but rather as a verification that specific features yield similar results. Finally, recommendations are given on which library is best suited to various hypothetical situations and user profiles.

II. Python libraries

A. Overview of available libraries

The following python libraries were found to be capable of propagating orbits: *poliastro* [3], the *TU Delft Astrodynamics Toolbox in Python "Tudatpy"* [4], *Skyfield* [5], *Orbit Predictor* [6], *orbitdeterminator* [7], the *Orekit Python wrapper* [8], and *orbdetpy* [9].

poliastro is an open source (MIT) pure Python orbital mechanics library developed by Juan Luis Cano Rodríguez and an international community. It was historically focused on interplanetary applications, explaining its currently limited capabilities in Earth-specific applications. Such features are on the roadmap for a future `1.0` release. As a library focused on ease of use, its high level API handles physical quantities with units using `astropy.units`.

Tudatpy is an open source (BSD-3) Python library that exposes the powerful set of C++ libraries provided by the *TU Delft Astrodynamics Toolbox "Tudat"* [10]. This interface enables the acceleration of *Tudat* simulations in a language more widely used in education, data science and machine learning. *Tudatpy* and *Tudat* are mainly developed for research purposes by a dedicated team at the *Technische Universiteit Delft* in the Netherlands.

Skyfield is an open source (MIT) pure Python astronomy library developed by Brandon Rhodes and an international community. It is focused on position computations of stars, planets and Earth-orbiting satellites. As its only orbit propagation feature uses *SGP4* to propagate TLEs, it is not considered further in this study.

Orbit Predictor is an open source (MIT) Python library developed by *Satellogic*. Its main propagation feature uses SGP4 to propagate TLEs like *Skyfield*, but it also provides two numerical propagators based on Keplerian and perturbed (J2 only) force models. At the time of writing (June 2022), the latest release dated from September 2020.

orbitdeterminator is an open source (MIT) Python library for satellite orbit determination developed by *AerospaceResearch*, a small community of space experts and enthusiasts. It implements an SGP4 propagator as well as an RK4 propagator using Cowell's formulation with a central force, J2 perturbation, and drag.

The *Orekit Python wrapper*, referred to as *"Orekit (py)"* in this document, is the official open source (Apache-2.0) interface that enables the powerful Java libraries provided by *Orekit* [11] to be used in a Python environment. It has been developed by the *Swedish Space Corporation* since 2014 and was still being updated at the time of writing.

orbdetpy is an open source (GPLv3) Python library for satellite orbit determination developed by the *Advanced Sciences and Technology Research in Astronautics "ASTRIA"* laboratory at the *University of Texas, Austin* in the USA. It is based on *Orekit* and can be used as an alternative to the official Python interface.

The main features of the these libraries available at the time of writing are compared in Table 1.

Another relevant project is *Astrodynamics with Python "AWP"* [12]. It is not a library, but rather a set of Python tools and scripts for propagating orbits and designing space missions with the main purpose of accompanying YouTube tutorials. As such, it is not included further in this study but remains a compelling option to learn and experiment with orbital mechanics.

B. Libraries selected

Due to time constraints, only two Python libraries of all those presented in II.A are selected for a more detailed analysis in the rest of this study: *poliastro* and *Tudatpy*.

This selection is based essentially on their large feature set, ease of use and quality documentation. Indeed, both integrate well with a Python working environment, which was found to be more complicated for Java-based libraries due to the interaction with a JVM. The other two libraries are focused on SGP4 propagation and only provide very basic perturbations for the customizable propagator. Additionally, both *poliastro* and *Tudatpy* are in active development at the time of writing, with questions and bug reports resolved within days and new features released frequently. Finally, these two libraries are built using a very different approach, *poliastro* being fully developed in Python for ease of use and quick implementation, while *Tudatpy* is an interface to a faster, more complex C++ library developed for research. Both fit very well with the context of the CREME mission, i.e. an education and research environment.

	poliastro	Tudatpy	Orbit Predictor	Orbitdeterminator	orbdetpy	Orekit (py)
Propagation	Python	C++	Python/C++(SGP4)	Python/C++(SGP4)	Java	Java
Keplerian	Yes	Yes	Yes	Yes	Yes	Yes
J2	Yes	Yes	Yes	Yes	Yes	Yes
J3	Yes	Yes	No	No	Yes	Yes
J4+	No	Yes	No	No	Yes	Yes
Drag	Yes	Yes	No	Yes	Yes	Yes
3rd body	Yes	Yes	No	No	Yes	Yes
SRP	Yes	Yes	No	No	Yes	Yes
SGP4	No	No	Yes	Yes	No	Yes
Other pert.	No	Yes	No	No	Yes	Yes
Eclipses	Yes	Yes	Yes	No	No	Yes
GS visibility	No	No	No	No	Yes	Yes
ECEF export	No	Yes	Yes	Yes	Yes	Yes

Table 1. Features comparison of the main Python libraries suitable for orbit propagation.

III. Reference tools

Two reference tools widely used in French aerospace academia are used to compare results from *poliastro* and *Tudatpy* against: *CelestLab* [13] and *GMAT* [14].

A. CelestLab

Developed by CNES, the French space agency, *CelestLab* is a free *Scilab* [15] toolbox for spaceflight dynamics. With a large feature set and various propagators and force models, its main application is trajectory and mission analysis. Additionally, its ease of use in a scientific programming language similar to MATLAB [16] makes it a popular tool in education and research.

Six orbit propagation models are provided with *CelestLab*, of which one is SGP4 for TLE propagation. The other five models are compared in Table 2, of which three are selected for use as references in this study.

The first selected, Keplerian, accounts only for the central attraction of the Earth.

The second one, Lyddane, is based on Brouwer theory [17] modified by Lyddane [18] and takes into account the effects of zonal harmonics up to J5. Its implementation in *CelestLab* includes secular and long-period effects on mean elements.

Finally, the third one is *STELA*, which is provided in the *CelestlabX* [19] extension and as a standalone software. It is widely used for long-term orbit propagation and re-entry prediction as it was originally developed to ensure missions launched after 2021 respected the *Loi sur les opérations spatiales* [20] [21], a French law on space operations. It imposes constraints on EOM operations by prohibiting cluttering of LEO and GEO, instead imposing a timely re-entry or decommissioning in graveyard orbits. The STELA working principle is based on a semi-analytic extrapolation method, taking into account the effects of Earth, Sun and Moon gravity, atmospheric drag and solar radiation pressure. These forces are averaged over one orbit before being integrated, ensuring that the integration is fast and suited for long-term propagation. However, it is not as accurate over short durations.

B. General Mission Analysis Tool (GMAT)

The other reference software used is the *General Mission Analysis Tool (GMAT)*, which is developed by a team of NASA, private industry, public, and private contributors. Contrarily to *CelestLab*, it does not

Force model	Earth gravity	3rd body	Drag	SRP
Keplerian	Point mass	No	No	No
Secular J2	Only up to secular effects of J2 on AOP, RAAN and MA	No	No	No
Eckstein-Hechler	Zonal harmonics up to J6, no tesseral terms	No	No	No
Lyddane	Zonal harmonics up to J5, no tesseral terms	No	No	No
STELA	Degree and order up to 15	Moon & Sun	Yes	Yes

Table 2. Comparison of the force models available in *CelestLab*.

work based on a set of predefined force models, but is fully customizable. An extensive set of perturbations is available, as well as numerous input and output data options.

GMAT can be used through a GUI, which saves all the mission parameters to a script, or by editing said script directly and running it with a terminal.

IV. Methodology

To compare the selected Python libraries with *CelestLab* and *GMAT* in identical situations, a workflow defined by input-files, a single propagation code per library as well as a single results analysis script was implemented. All the output files are named following the same convention, which includes all the input names, ensuring that results loaded simultaneously during the analysis are reliably comparable.

Specifying which input files and force model to use for each computation is done at the beginning of the Python and *CelestLab* codes. In the case of GMAT, one script is created for each spacecraft, and the force model as well as propagation dates are selected for each computation using the GUI.

A. Input data

The input data used by all libraries consists of CSV files with static specifications of the spacecraft, orbit, start/end dates with time step, and ground station.

Two very different spacecraft are used in this study. The first one is CREME, a 3U cubesat. The second one, Sunjammer, is a large and very light solar sail inspired by a technology demonstrator developed by NASA [22]. Their specifications are listed in Table 3.

	CREME	Sunjammer
Mass	4 kg	32 kg
Drag coefficient	2.2	1.28
Reflectivity coefficient	1.5	1.8
Drag area	$0.033\,\mathrm{m}^2$	$1444\,\mathrm{m}^2$
SRP area	$0.033\,\mathrm{m}^2$	$1444\,\mathrm{m}^2$

Table 3. Spacecraft specifications used.

	CREME	LEO
Epoch	2023-01-01T00:00:00Z	2023-01-01T00:00:00Z
SMA	6978.14 km	7100 km
Ecc	0	2×10^{-3}
Inc	97.8 °	98.277 102 °
AOP	0 °	90 °
RAAN	190.0962 °	20 °
TA	0.1 °	10 °

Table 4. Initial osculating orbits used.

The targeted orbit for CREME is a 600 km SSO, whereas Sunjammer was planned to be sent near the Earth-Sun Lagrange Point L1. However, this solar sail is only considered in order to detect drag and solar radiation pressure effects. As drag can only be observed within the high atmosphere, Sunjammer was propagated in the Low Earth Orbit (LEO) defined in Table 4.

B. Force models

As both *poliastro*, *Tudatpy* and *GMAT* can be configured with a fully custom force model, the three *CelestLab* models selected in III.A are used as a basis, as detailed in Table 5.

Library	Force model	Earth gravity	3rd body	Drag	SRP	CelestLab model
poliastro	keplerian	Point mass	No	No	No	Keplerian
	earthZonalJ3	Zonal harmonics up to J3	No	No	No	No
	complete	Zonal harmonics up to J3	Moon & Sun	Yes	Yes	No
Tudatpy	keplerian	Point mass	No	No	No	Keplerian
	earthZonalJ5	Zonal harmonics up to J5	No	No	No	Lyddane
	complete	Degree and order up to 15	Moon & Sun	Yes	Yes	STELA

Table 5. Force models implemented in *poliastro* and *Tudatpy* with the closest *CelestLab* model.

Naturally, the Keplerian model can be matched perfectly by all three other libraries. On the other hand, the Lyddane model can only be matched by *Tudatpy* and *GMAT* with the custom `earthZonalJ5` model, as *poliastro* provides geopotential perturbations only up to J3. Therefore, a slightly simplified intermediate model called `earthZonalJ3` is used with *poliastro* instead.

Finally, the STELA model cannot be perfectly matched with any other library, due to its singular computation method. The included perturbations can however be matched by *Tudatpy* and *GMAT* with the `complete` model, whereas *poliastro* has a slightly different one with geopotential perturbations only up to J3.

To handle the differences between similar models of *poliastro* and *Tudatpy* in result comparisons, *GMAT* is configured with twice as many models, which are configured differently to match with the former and the latter respectively.

Despite configuring all these tools with similar force models, the intrinsic physical models of their implementation can still differ. For example, Table 6 lists all the geopotential and atmospheric models used, where *GMAT* is configured with a different atmospheric model for comparisons with *poliastro* or *Tudatpy* to limit the impact of differences. Additionally, differences in the solar activity predictions, Moon and Sun ephemerides, or in the implementation of the SRP or drag perturbations can also lead to slight differences in the results when using the `complete` models.

	Geopotential	Atmospheric
poliastro	J2/J3 [23]	Jacchia77 [24]
Tudatpy	GOCO05c [25]	NRLMSISE-00 [26]
CelestLab	EGM96 [27]	NRLMSISE-00 [26]
GMAT (p)	EGM96 [27]	JacchiaRoberts [28]
GMAT (T)	EGM96 [27]	MSISE-90 [29]

Table 6. Geopotential and atmospheric models used. *GMAT (p)* and *GMAT (T)* represent different configurations of *GMAT* to be compared with *poliastro* and *Tudatpy* respectively.

	Propagation	Integrator
poliastro	Cowell	D&P 853
Tudatpy	Cowell	D&P 87
CelestLab	Analytical	/
STELA	Semi-analytical	/
GMAT (p)	Cowell	D&P 853
GMAT (T)	Cowell	D&P 87

Table 7. Propagation methods and integrators used.

C. Propagation

Although *GMAT* and the Python libraries can be configured to use Cowell's propagation method [30], the *CelestLab* models use analytical—and semi-analytical in the case of STELA—methods.

Additionally, a specificity of *poliastro* is that propagation is not performed with a configurable time step. Indeed, the state is automatically propagated using the Dormand & Prince integration method of order 8(5,3) [31] until a termination event and the state at every requested epoch is computed afterwards via interpolation. For the configurable propagators, a constant time step of 10 s is used to keep computation times below a few hours while achieving a sufficient accuracy for the CREME mission design requirements. Since *poliastro* can only be configured with D&P 853 as integrator with Cowell's method whereas *Tudatpy* does not have this exact integrator implemented, the closest one available, D&P 87 [32], is used. As both are available in GMAT, configurations to be compared with *poliastro* and *Tudatpy* use one or the other accordingly. Table 7 summarises the propagation methods and integrators used throughout all the configurations.

D. Results processing

All results are stored in CSV files – or text files for GMAT – following the same content convention. They include at each row the elapsed time, ECI, ECEF, and osculating Keplerian state.

As the raw output of the propagation with all the libraries and tools does not follow the same convention, conversions have to be performed before saving the results to standardize them. In the case of *poliastro*, the semi-latus rectum is converted to semi-major axis and the ECEF states are computed using the *astropy* library. In the case of *CelestLab*, the mean Keplerian elements are converted to osculating states and the mean anomaly to true anomaly.

V. Results

The main results of this study are plots highlighting the difference in evolution of the ECI and Keplerian states of CREME and Sunjammer between the Python libraries and the reference tools. Figures 1 and 2 present examples of such plots, all the others are in the Appendix. In the case of the `keplerian` model, only the ECI state difference in evolution is considered as the Keplerian state remains constant as expected with all tools, notwithstanding negligible numerical errors.

Table 8 summarizes the results by compiling all the orders of magnitude of difference between the CREME ECI position and velocity computed with the Python libraries and with the reference tools.

VI. Discussion

A. Results analysis

Figures 3 and 4 show the ECI state difference between results computed with the `keplerian` model with the Python libraries and *GMAT*, *CelestLab* respectively. Both *poliastro* and *Tudatpy* present a similar deviation with respect to *GMAT* of approximately 10^{-3}m for the position and 10^{-6}m s^{-1} for the velocity after 10 minutes. The deviation of *Tudatpy* results is smooth, whereas results from *poliastro* present small variations introduced by its different integration method. However, both Python libraries present the same sawtooth-like deviations with respect to *CelestLab*, due to its very different computation method. The order of magnitude of difference remains approximately 10^{-3}m for the position and 10^{-6}m s^{-1} for the velocity after 10 minutes.

Figures 5, 13, 23 and 6, 14, 24 show the ECI state difference between results computed with the `earthZonal` model with the Python libraries and *GMAT*, *CelestLab* respectively. Over 10 minutes, Figure 5 demonstrates that *Tudatpy* is very close to *GMAT*, while *poliastro* deviates by an order of magnitude of approximately 10^{1}m for the position and 10^{-2}m s^{-1} for the velocity. This observation still stands for durations of an hour and a day visible in Figures 13 and 23. On the other hand, none of the Python libraries is as close to *CelestLab* as *Tudatpy* is to *GMAT*. Indeed, Figure 6 shows deviations of a similar order of magnitude for both Python libraries with respect to *CelestLab* in ECI position and velocity. However, it should be noted that *poliastro* is consistently closer to *CelestLab* than *Tudatpy* despite having an `earthZonal` model limited to J3 instead of J5. These observations remain valid for a duration of an hour visible in Figure 14, but does not for an even longer duration of a day visible in Figure 24. Indeed, over longer durations the deviation with respect to *CelestLab* of *poliastro* grows larger than that of *Tudatpy*, most likely due to the growth in the difference caused by the simpler `earthZonal` model.

Figures 9, 17, 25 and 10, 18, 26 show the ECI state difference between results computed with the `complete` model with the Python libraries and *GMAT*, *CelestLab* respectively. Over 10 minutes, Figures 9 and 10 show that no Python library matches perfectly

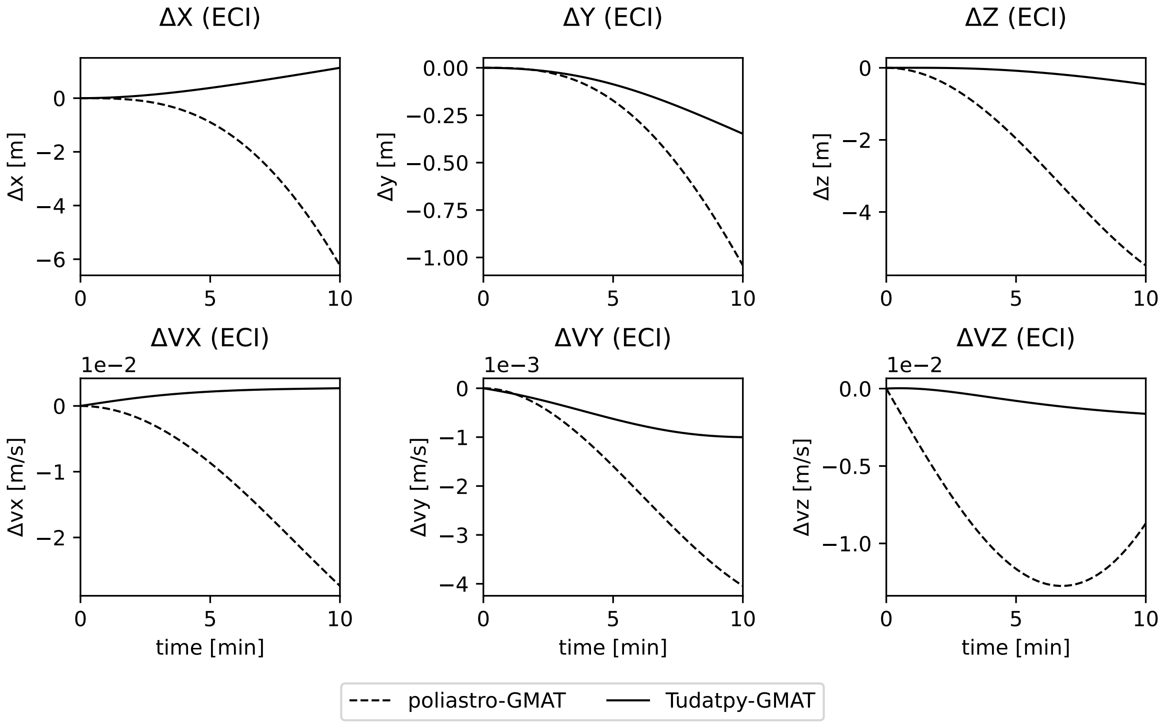

Figure 1. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 10 minutes with the `complete` model.

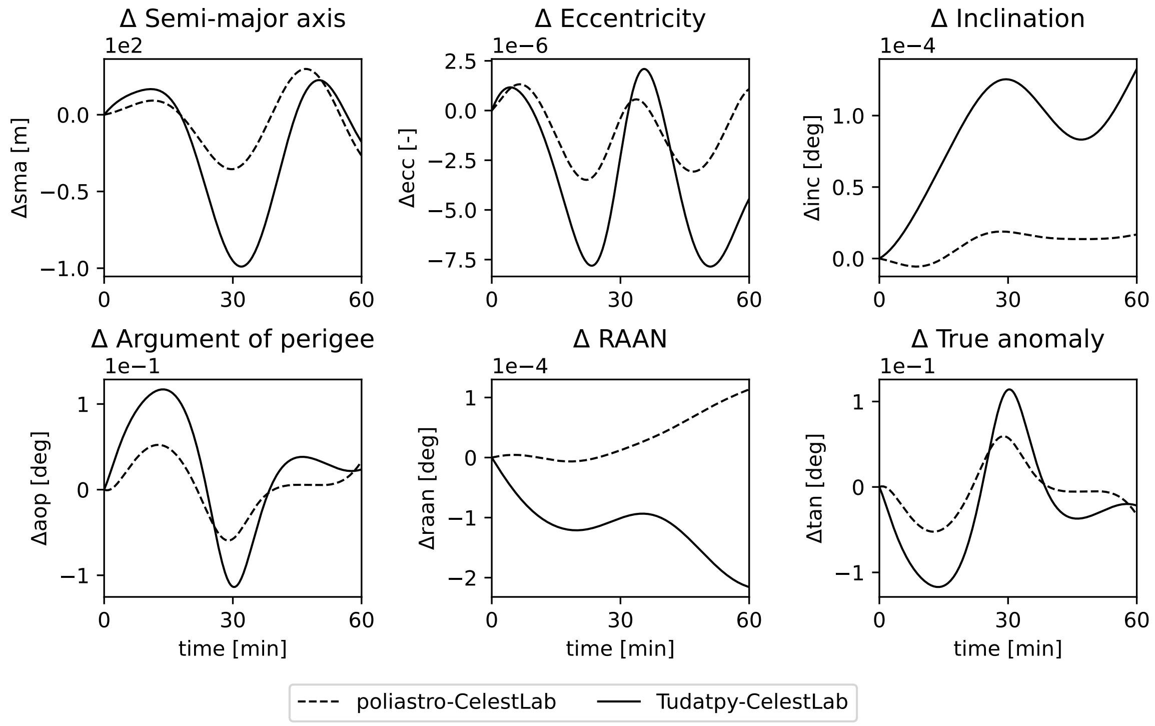

Figure 2. Difference between the Sunjammer Keplerian state computed with Python libraries and with *CelestLab* over 1 hour with the `earthZonal` model.

Force model	After	poliastro-CelestLab	poliastro-GMAT	Tudatpy-CelestLab	Tudatpy-GMAT
keplerian	10 min	10^{-3}m, 10^{-6}m s^{-1}	10^{-3}m, 10^{-6}m s^{-1}	10^{-3}m, 10^{-6}m s^{-1}	10^{-3}m, 10^{-6}m s^{-1}
	1 hour	10^{-2}m, 10^{-5}m s^{-1}	10^{-2}m, 10^{-5}m s^{-1}	10^{-2}m, 10^{-5}m s^{-1}	10^{-2}m, 10^{-5}m s^{-1}
	1 day	10^{-1}m, 10^{-4}m s^{-1}	10^{-1}m, 10^{-4}m s^{-1}	10^{-1}m, 10^{-4}m s^{-1}	10^{-1}m, 10^{-4}m s^{-1}
earthZonal	10 min	10^{1}m, 10^{-2}m s^{-1}	10^{1}m, 10^{-2}m s^{-1}	10^{1}m, 10^{-2}m s^{-1}	10^{-2}m, 10^{-4}m s^{-1}
	1 hour	10^{1}m, 10^{-2}m s^{-1}	10^{2}m, 10^{-1}m s^{-1}	10^{1}m, 10^{-2}m s^{-1}	10^{0}m, 10^{-3}m s^{-1}
	1 day	10^{3}m, 10^{0}m s^{-1}	10^{3}m, 10^{0}m s^{-1}	10^{2}m, 10^{-1}m s^{-1}	10^{1}m, 10^{-2}m s^{-1}
complete	10 min	10^{0}m, 10^{-2}m s^{-1}	10^{1}m, 10^{-2}m s^{-1}	10^{1}m, 10^{-1}m s^{-1}	10^{0}m, 10^{-3}m s^{-1}
	1 hour	10^{0}m, 10^{-2}m s^{-1}	10^{1}m, 10^{-2}m s^{-1}	10^{3}m, 10^{0}m s^{-1}	10^{1}m, 10^{-2}m s^{-1}
	1 day	10^{3}m, 10^{0}m s^{-1}	10^{2}m, 10^{-1}m s^{-1}	10^{4}m, 10^{1}m s^{-1}	10^{2}m, 10^{-1}m s^{-1}

Table 8. Order of magnitude of difference in CREME ECI position and velocity between *poliastro*, *Tudatpy* and *CelestLab*, *GMAT*.

a reference tool, but the observations on short durations with the `earthZonal` model remain applicable. Indeed, *Tudatpy* is closer to *GMAT*whereas *poliastro* is closer to *CelestLab*. After 10 minutes, *Tudatpy* remains within approximately 10^{0}m of *GMAT* for the position and 10^{-3}m s^{-1} for the velocity whereas *poliastro* reaches deviations of approximately 10^{1}m for the position and 10^{-2}m s^{-1}. With respect to *CelestLab*, *poliastro* remains within approximately 10^{0}m for the position and 10^{-2}m s^{-1} for the velocity whereas *Tudatpy* reaches deviations of approximately 10^{1}m for the position and 10^{-1}m s^{-1} for the velocity. These observations still stand with longer durations of one hour and one day visible in Figures 17, 18 and 25, 26 respectively.

Overall, both Python libraries remain within a level of deviation with respect to the reference tools compliant with the CREME mission design requirements, as precise positioning is not of utmost importance for this mission. However, *Tudatpy* has proven measurably closer to *GMAT* than *poliastro* is to *CelestLab* , yielding *Tudatpy* a slight advantage at this stage.

B. Computation speed

Table 9 compiles the average computation speed over three runs of the propagation step only across *poliastro*, *Tudatpy*, *CelestLab*, and the two configurations of *GMAT*. These measurements highlight the large differences in language and method used for propagation between the Python libraries and reference tools.

The C++ propagation code of *Tudatpy* gives it a noticeably strong advantage in computation speed over *poliastro*, which still suffers from unoptimized code for a few perturbation models. *Tudatpy* also beats *GMAT* in most situations, only slowing down for longer propagations. Slight differences between *GMAT (p)* and *GMAT (T)*—the two configurations of GMAT matching *poliastro* and *Tudatpy* respectively—can also be noticed, especially when using the `complete` model where they differ the most.

Finally, a striking difference within *CelestLab* can be noticed depending on the propagation model used. Indeed, `keplerian` and `earthZonal` are based on analytical computations performed in FORTRAN, achieving the best or second best times. However, the `complete` model of *CelestLab* is based on its STELA propagator, which is designed to be efficient for long-term propagations and only becomes a faster when propagating for much longer durations.

Overall, *Tudatpy* stands out as the fastest Python library of the two, with *CelestLab* being the only reference tool consistently faster in `keplerian` and `earthZonal` models.

C. Ease of use

An important criterion of this study to compare the selected Python libraries is their ease of use. This considers how fast they can be installed and run for the first time, their customizability, their available features, and how accessible they are for various levels

Force model	Propagation	poliastro	Tudatpy	CelestLab	GMAT (p)	GMAT (T)
keplerian	10 min	2.3 s	8 ms	2 ms	0.58 s	0.58 s
	1 hour	2.3 s	41 ms	3 ms	0.59 s	0.59 s
	1 day	2.8 s	0.98 s	3 ms	0.93 s	0.95 s
earthZonal	10 min	2.7 s	9 ms	19 ms	0.58 s	0.58 s
	1 hour	2.7 s	47 ms	19 ms	0.59 s	0.59 s
	1 day	3.5 s	1.10 s	39 ms	0.94 s	0.96 s
complete	10 min	5.7 s	18 ms	31.8 s	1.0 s	0.60 s
	1 hour	6.2 s	0.10 s	32.2 s	1.1 s	0.66 s
	1 day	20.7 s	2.5 s	44.3 s	2.4 s	2.5 s

Table 9. Computation speed of the propagation step only for all libraries and reference tools. *GMAT (p)* and *GMAT (T)* represent different configurations of *GMAT* to be compared with *poliastro* and *Tudatpy* respectively.

of experience in orbital mechanics and numerical propagation. It is important to note that the evaluation of most of these aspects is subjective by nature and is likely to change in the future with new versions.

During testing, *poliastro* has proven fast to implement in basic scenarios and easy to expand with other features thanks to its detailed documentation. However, its feature set can be a limiting factor for more advanced use cases. On the other hand, *Tudatpy* has proven less accessible, with more setup steps required even in the most basic scenario. However, these steps also help make it more modular and easy to customize using a large amount of available features.

Overall, both libraries can be recommended based on their ease of use for different use cases. For experienced users seeking a more mature and comprehensive library to be implemented in research or advanced applications, *Tudatpy* is the recommended option. For less experienced users, or more educational purposes requiring a fast implementation and plotting features, *poliastro* stands out as ideal.

Finally, for a user that does not plan to implement a library in a Python workflow but is only looking for a standalone solution for orbit propagation, the two reference tools can also be valid options. Indeed, *CelestLab* has proven to be the fastest and easiest code-based solution, whereas GMAT remains a powerful and very customizable option with a GUI that makes it accessible to a wider audience.

VII. Conclusion

Starting from an overview of Python libraries able to replace other tools in the space mission design workflow of the CREME satellite mission development, a comparison of their main features was established and two prime candidates, *poliastro* and *Tudatpy*, were selected. While *Tudatpy* achieved a slight advantage in numerical measurement comparisons, both libraries proved acceptable considering the CREME mission requirements. However, it proved able to perform the propagation step much faster than *poliastro* as well as *GMAT* in most situations. When considering their ease of use, *poliastro* was quicker to implement whereas *Tudatpy* is more appropriate for experienced users with more feature requirements. For these reasons, *Tudatpy* stands out as the better option for advanced applications, while *poliastro* is more suited to educational purposes and rapid implementation.

Given that the CREME space mission design takes place in a research context with an advanced workflow, *Tudatpy* is the recommended solution.

Further research following this study could investigate Orekit and JVM-based libraries in more detail as well as update the computation speed and features analysis with new releases of *Tudatpy* and *poliastro* slated to include optimization improvements.

Acknowledgments

Special thanks go to developers of poliastro and Tudatpy for their reactivity addressing questions and bug reports, namely Juan Luis Cano Rodríguez and Geoffrey Garrett, Dominic Dirkx, Jérémie Gaffarel. Thanks are also due to Jérôme Puech for his mentorship during this research internship.

References

[1] ONERA, "Space weather: a nanosatellite to measure radiation," https://www.onera.fr/en/news/space-weather-a-nanosatellite-to-measure-radiation, 2021. Accessed: 2022-06-05.

[2] Vallado, D., Crawford, P., Hujsak, R., and Kelso, T., "Revisiting Spacetrack Report #3," *Collection of Technical Papers - AIAA/AAS Astrodynamics Specialist Conference, 2006*, Vol. 3, 2006, pp. 1–91. doi: 10.2514/6.2006-6753.

[3] Cano Rodríguez, J. L., Martínez Garrido, J., et al., "poliastro," , 2022. doi: 10.5281/zenodo.6533238, URL https://www.poliastro.space.

[4] "TU Delft Astrodynamics Toolbox in Python "Tudatpy"," , 2022. URL https://github.com/tudat-team/tudatpy.

[5] Rhodes, B., "Skyfield: Generate high precision research-grade positions for stars, planets, moons, and Earth satellites," , 2020. URL https://rhodesmill.org/skyfield.

[6] "Orbit Predictor," Satellogic, 2020. URL https://github.com/satellogic/orbit-predictor.

[7] "Orbitdeterminator," , 2022. URL https://github.com/aerospaceresearch/orbitdeterminator.

[8] "Orekit Python Wrapper," Swedish Space Corporation, 2022. URL https://gitlab.orekit.org/orekit-labs/python-wrapper.

[9] "orbdetpy," , 2022. URL https://github.com/ut-astria/orbdetpy.

[10] "TU Delft Astrodynamics Toolbox," , 2022. URL https://github.com/tudat-team/tudat.

[11] "Orekit," CS GROUP, 2022. URL https://www.orekit.org/.

[12] Gonzalez, A., "Astrodynamics with Python," , 2022. URL https://github.com/alfonsogonzalez/AWP.

[13] CNES, "CelestLab," , 2022. URL https://logiciels.cnes.fr/en/node/67.

[14] NASA, "General Mission Analysis Tool," , 2020. URL https://sourceforge.net/projects/gmat.

[15] ESI-Group, "Scilab," , 2021. URL https://www.scilab.org/.

[16] MathWorks, "MATLAB," , 2022. URL https://www.mathworks.com/products/matlab.html.

[17] Brouwer, D., "Solution of the problem of artificial satellite theory without drag," *The Astronomical Journal*, Vol. 64, No. 1, 1959.

[18] Lyddane, R. H., "Small eccentricities or inclinations in the Brouwer theory of artificial satellite motion," *The Astronomical Journal*, Vol. 68, No. 8, 1963.

[19] CNES, "CelestLabX," , 2022. URL https://logiciels.cnes.fr/en/node/70.

[20] Légifrance, "LOI n° 2008-518 du 3 juin 2008 relative aux opérations spatiales," , 2022. URL https://www.legifrance.gouv.fr/loda/id/JORFTEXT000018931380/.

[21] Le Fevre, C., Morand, V., Fraysse, H., Deleflie, F., Mercier, P., Lamy, A., and Cazaux, C., "Long Term Orbit Propagation Techniques Developed in the Frame of the French Space Act," *A Safer Space for Safer World*, ESA Special Publication, Vol. 699, edited by L. Ouwehand, 2012, p. 89.

[22] Barnes, N. C., Derbes, W. C., Player, C. J., and Diedrich, B. L., "Sunjammer: a solar sail demonstration," *Advances in Solar Sailing*, Springer, 2014, pp. 115–126.

[23] Luzum, B., Capitaine, N., Fienga, A., Folkner, W., Fukushima, T., Hilton, J., Hohenkerk, C., Krasinsky, G., Petit, G., Pitjeva, E., et al., "The IAU 2009 system of astronomical constants: the report of the IAU working group on numerical standards for Fundamental Astronomy," *Celestial Mechanics and Dynamical Astronomy*, Vol. 110, No. 4, 2011, pp. 293–304.

[24] Jacchia, L. G., "Thermospheric temperature, density, and composition: New models," *SAO special report*, Vol. 375, 1977.

[25] Fecher, T., Pail, R., and Gruber, T., "GOCO05c: a new combined gravity field model based on full normal equations and regionally varying weighting," *Surveys in geophysics*, Vol. 38, No. 3, 2017, pp. 571–590.

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

[27] Lemoine, F. G., Kenyon, S. C., Factor, J. K., Trimmer, R. G., Pavlis, N. K., Chinn, D. S., Cox, C. M., Klosko, S. M., Luthcke, S. B., Torrence, M. H., et al., "The development of the joint NASA GSFC and the National Imagery and Mapping Agency (NIMA) geopotential model EGM96," , 1998.

[28] Jacchia, L. G., "Revised static models of the thermosphere and exosphere with empirical temperature profiles," *SAO special report*, Vol. 332, 1971.

[29] Hedin, A. E., “Extension of the MSIS thermosphere model into the middle and lower atmosphere,” *Journal of Geophysical Research: Space Physics*, Vol. 96, No. A2, 1991, pp. 1159–1172.

[30] Brouwer, D., and Clemence, G., “Methods of celestial mechanics. 1961,” *URL http://adsabs. harvard. edu/abs/1961mcm.. book..... B*, 1946.

[31] Hairer, E., Nørsett, S. P., and Wanner, G., *Solving ordinary differential equations. 1, Nonstiff problems*, Springer-Vlg, 1993.

[32] Prince, P. J., and Dormand, J. R., “High order embedded Runge-Kutta formulae,” *Journal of computational and applied mathematics*, Vol. 7, No. 1, 1981, pp. 67–75.

Appendix

The following pages contain numerous plots of differences between ECI or Keplerian states computed using the Python libraries and the reference tools, for both CREME and Sunjammer, the three force models detailed in Table 5, and for three durations: 10 minutes, 1 hour, and 1 day.

All the code and raw results used in this study can be found in the following public repository: Nosudrum/orbit-propagation-python-study.

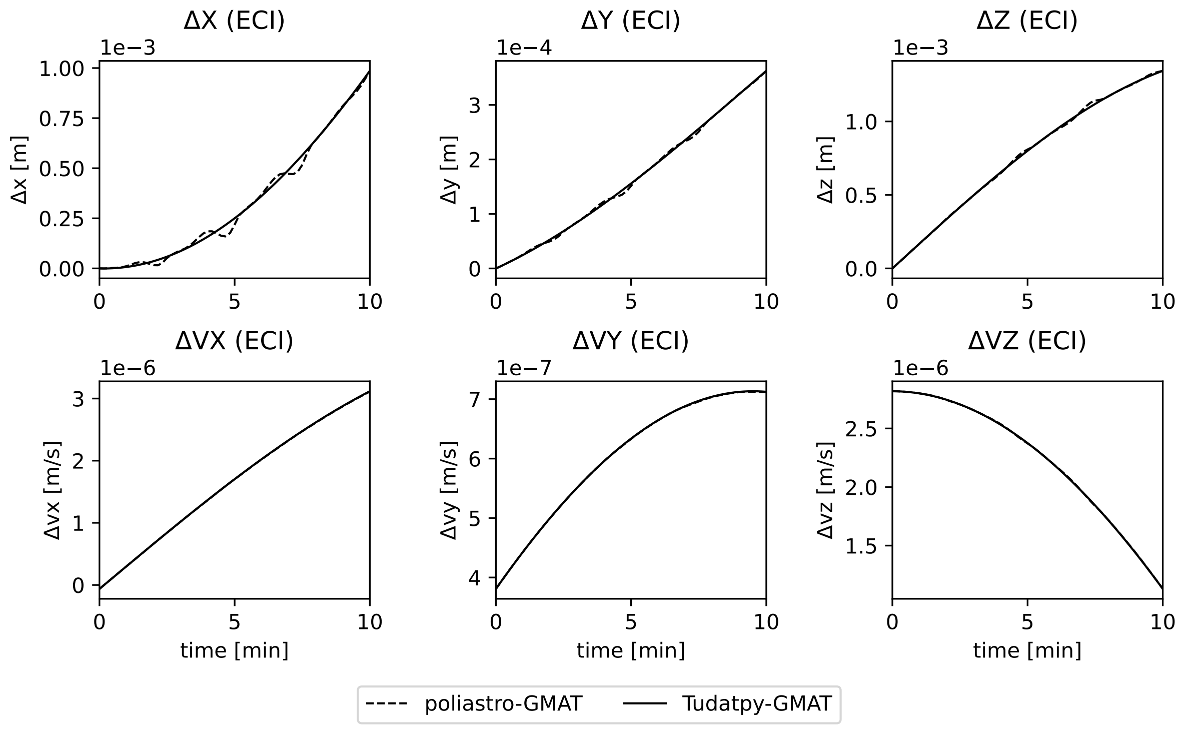

Figure 3. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 10 minutes with the `keplerian` model.

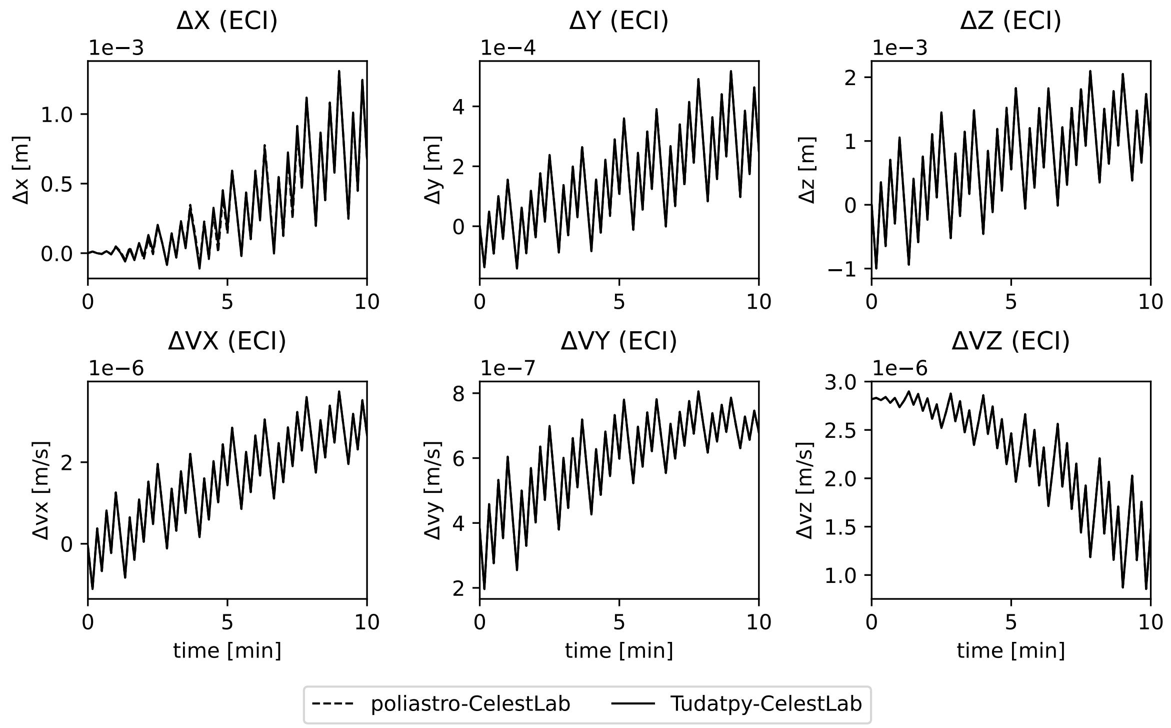

Figure 4. Difference between the CREME ECI state computed with Python libraries and with *CelestLab* over 10 minutes with the `keplerian` model.

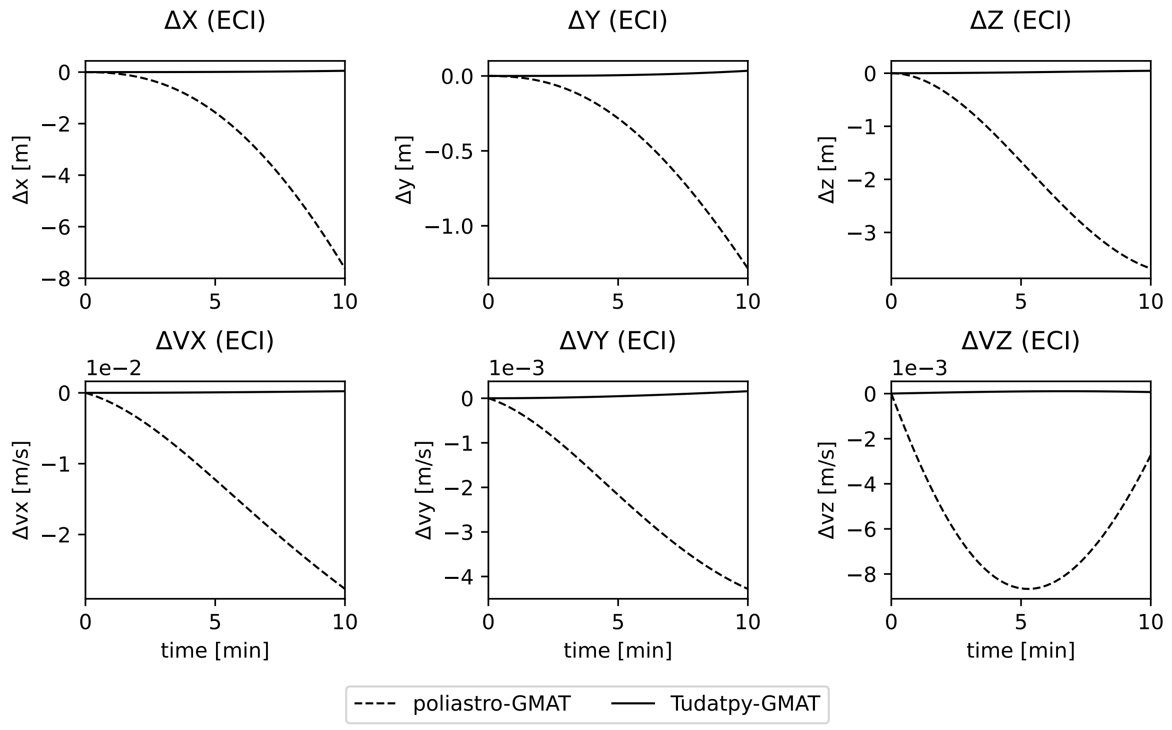

Figure 5. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 10 minutes with the `earthZonal` model.

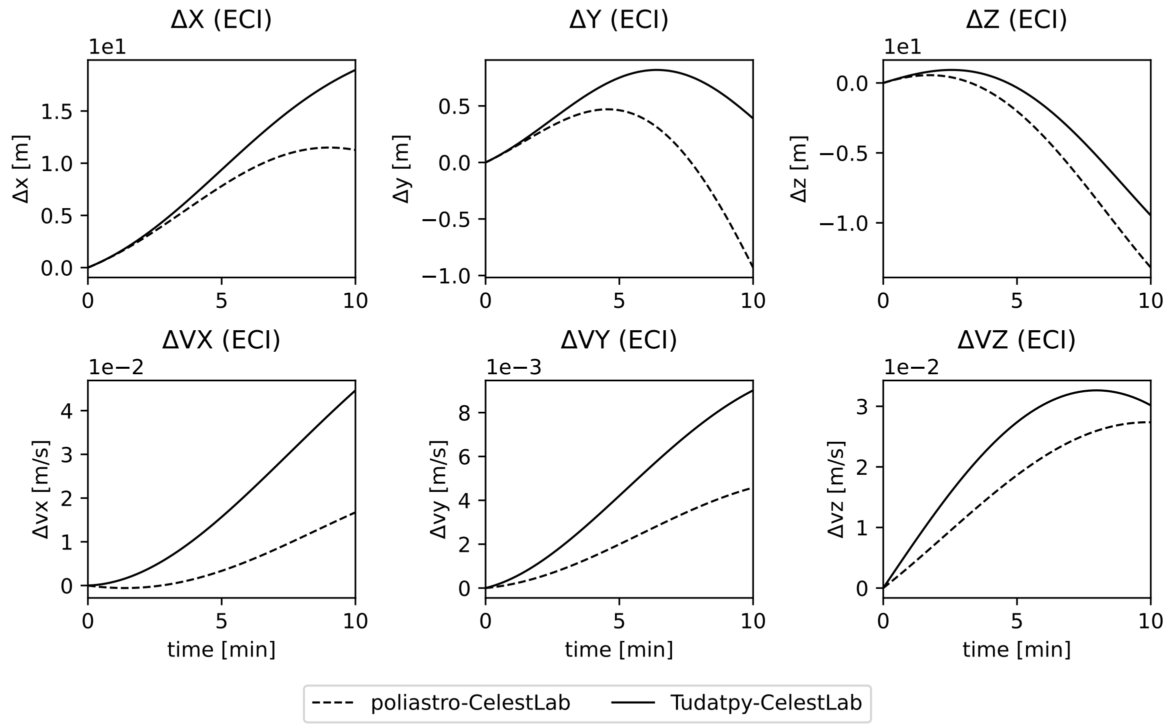

Figure 6. Difference between the CREME ECI state computed with Python libraries and with *CelestLab* over 10 minutes with the `earthZonal` model.

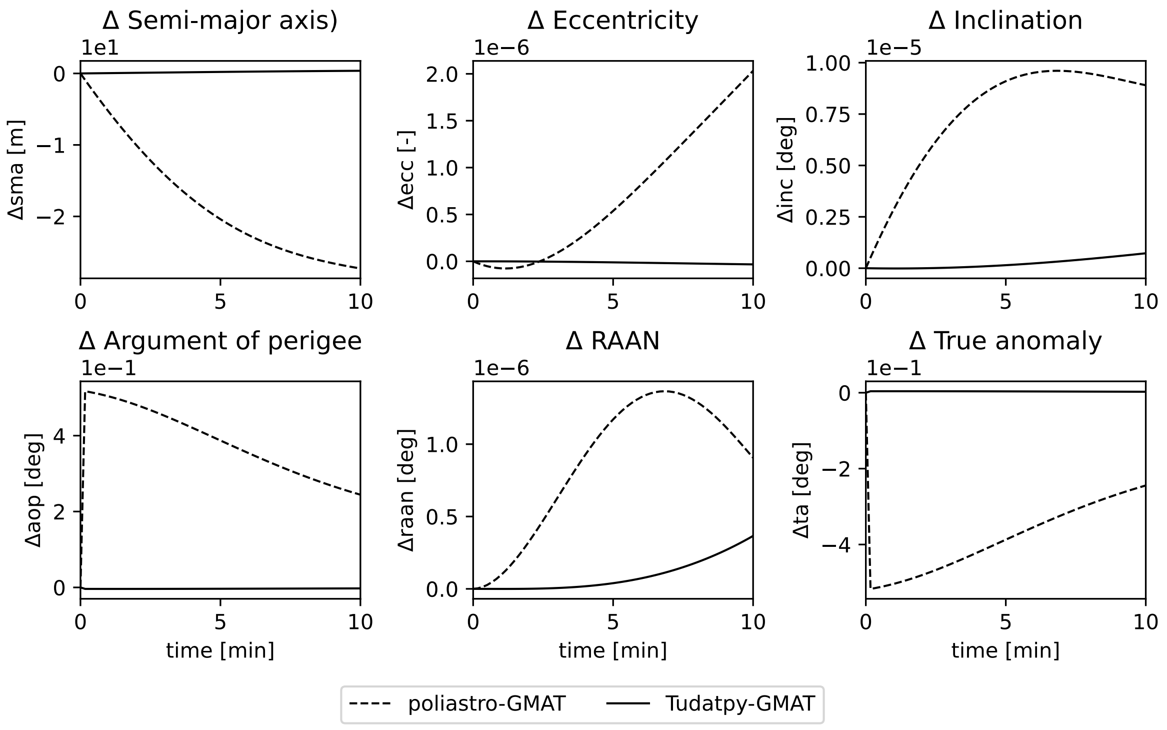

Figure 7. Difference between the CREME Keplerian state computed with Python libraries and with *GMAT* over 10 minutes with the `earthZonal` model.

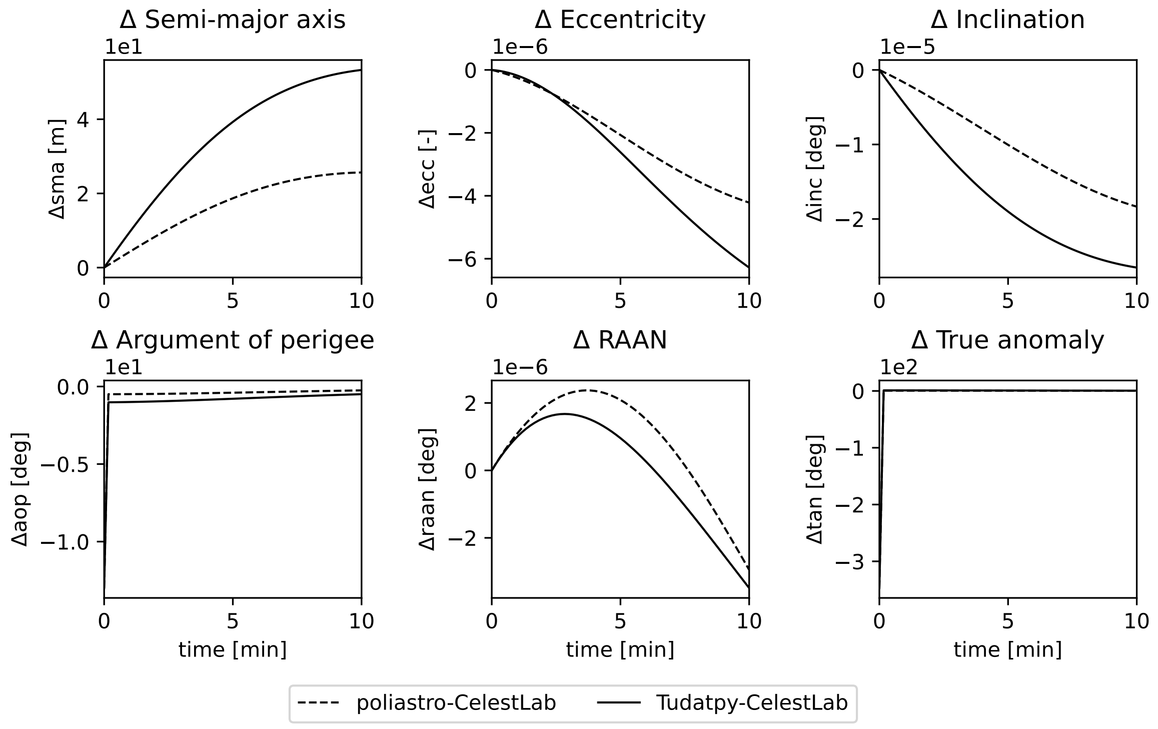

Figure 8. Difference between the CREME Keplerian state computed with Python libraries and with *CelestLab* over 10 minutes with the `earthZonal` model.

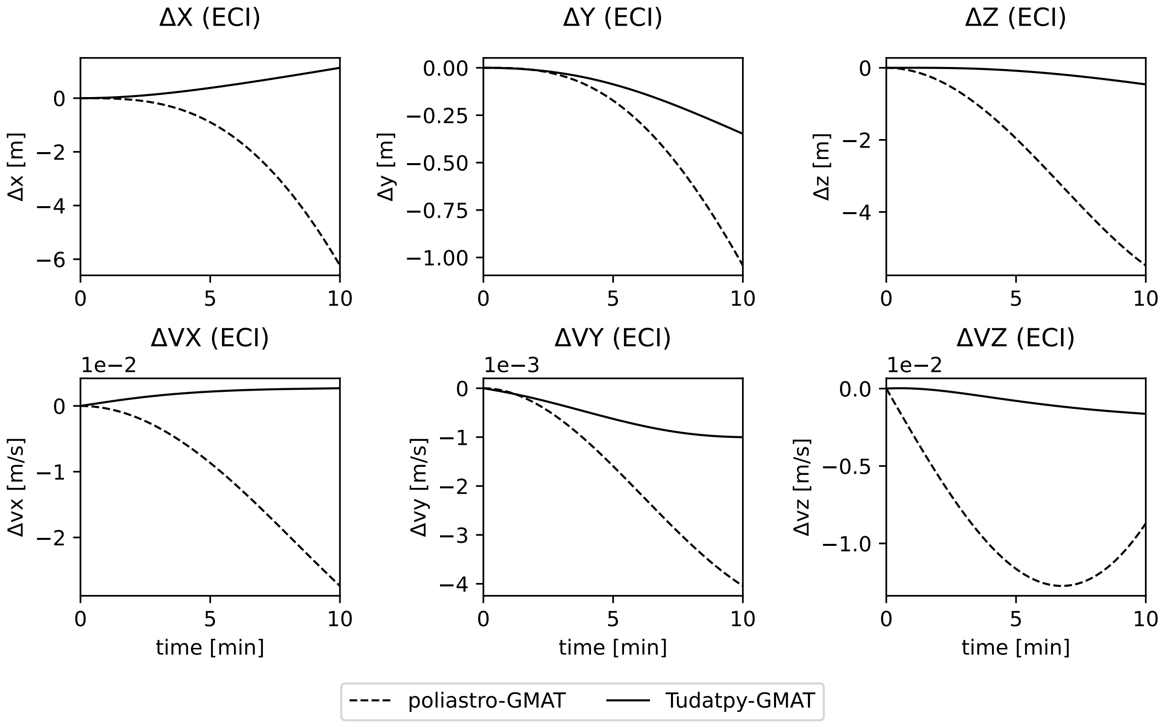

Figure 9. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 10 minutes with the `complete` model.

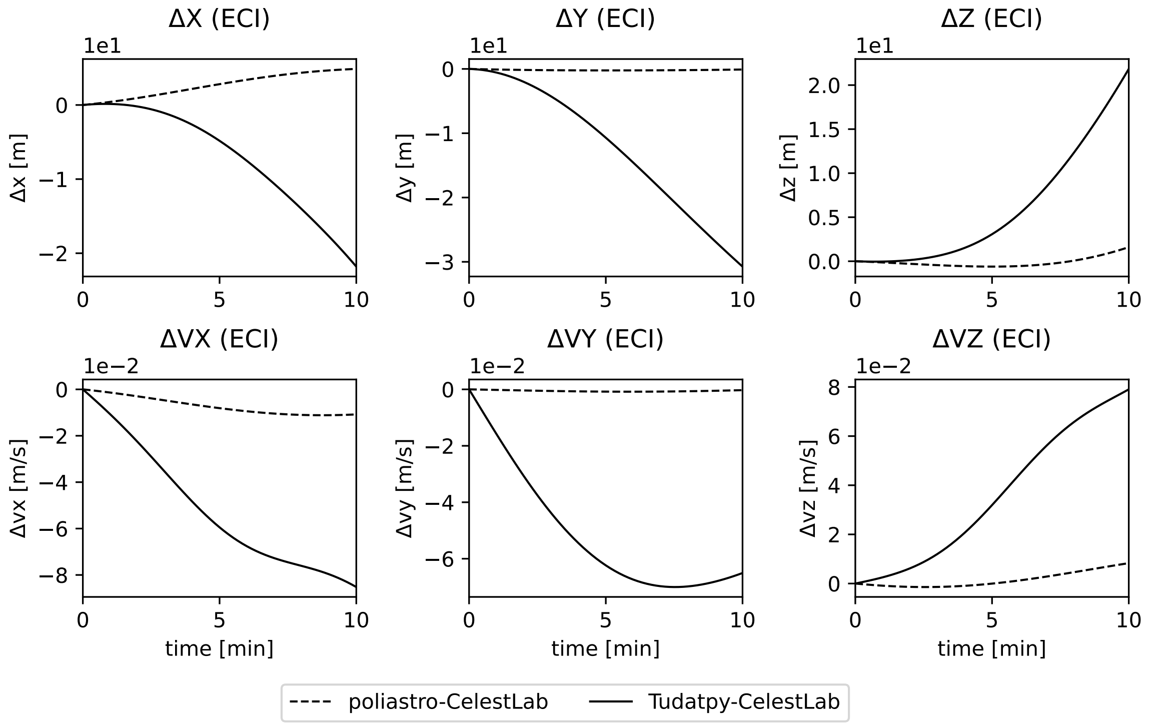

Figure 10. Difference between the CREME ECI state computed with Python libraries and with *CelestLab* over 10 minutes with the `complete` model.

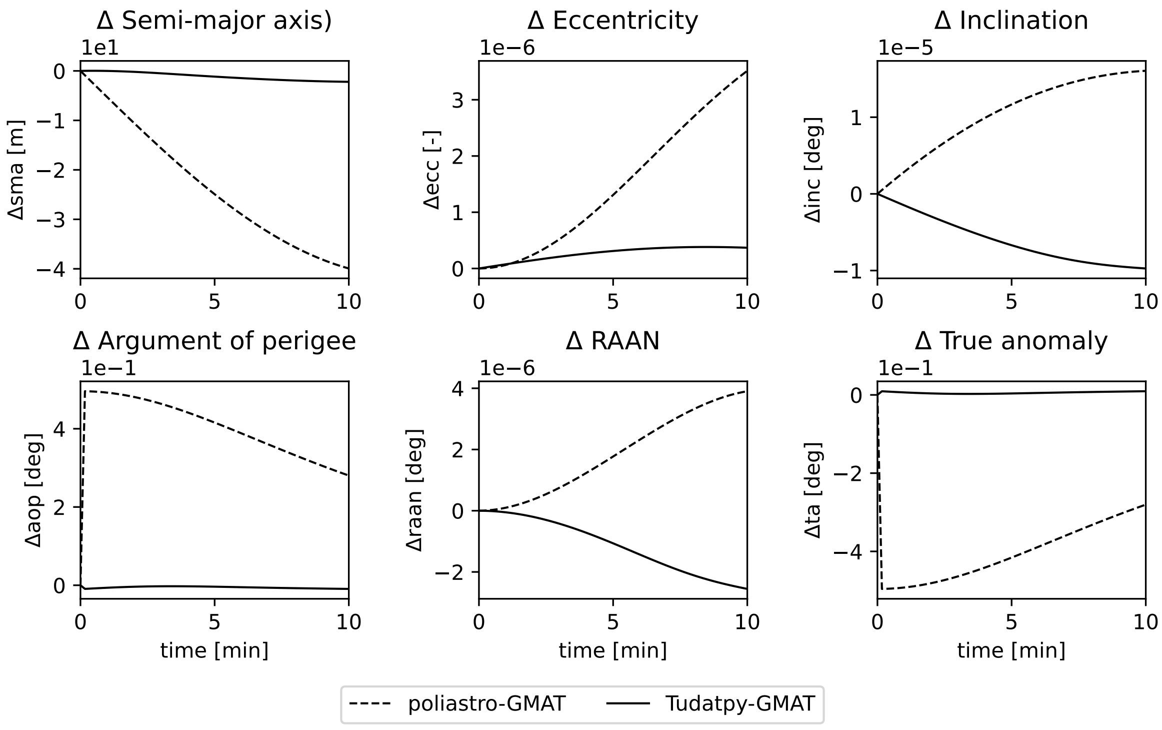

Figure 11. Difference between the CREME Keplerian state computed with Python libraries and with *GMAT* over 10 minutes with the `complete` model.

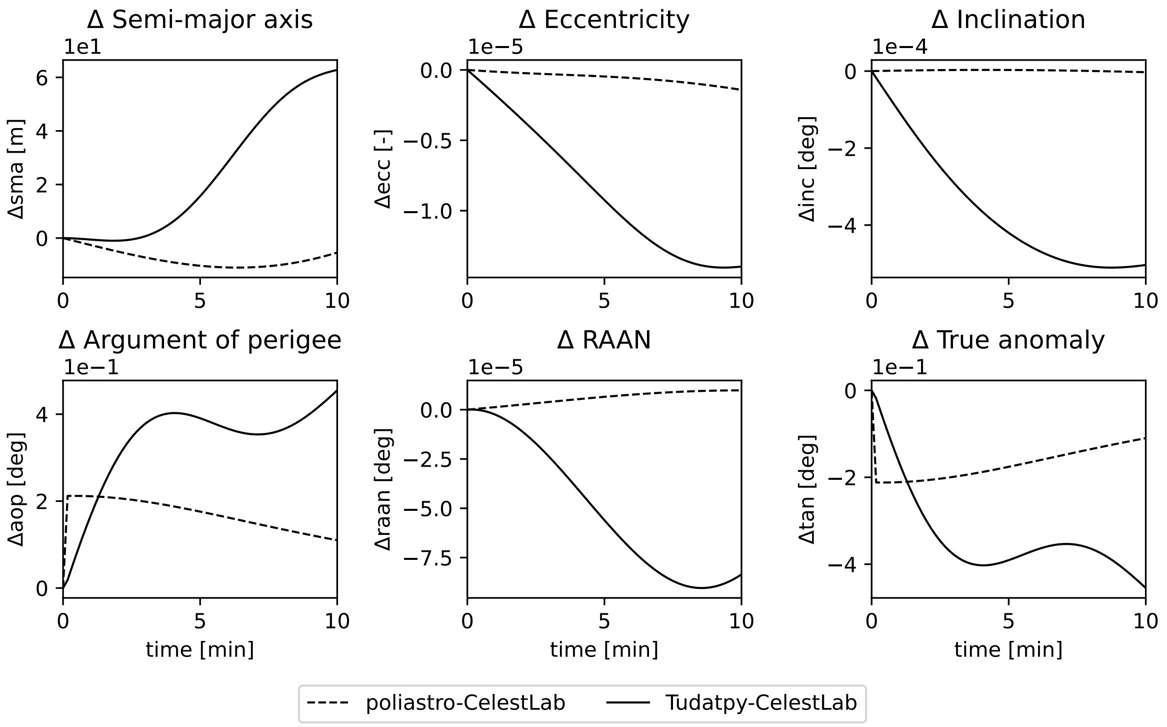

Figure 12. Difference between the CREME Keplerian state computed with Python libraries and with *CelestLab* over 10 minutes with the `complete` model.

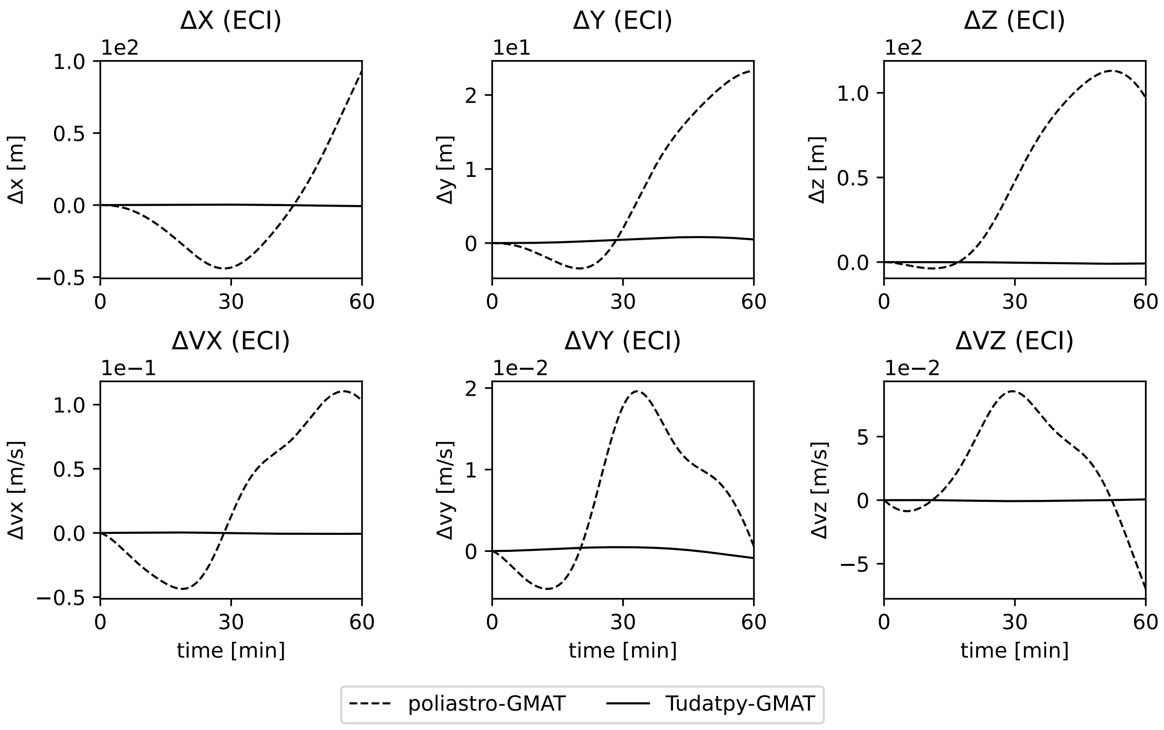

Figure 13. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 1 hour with the `earthZonal` model.

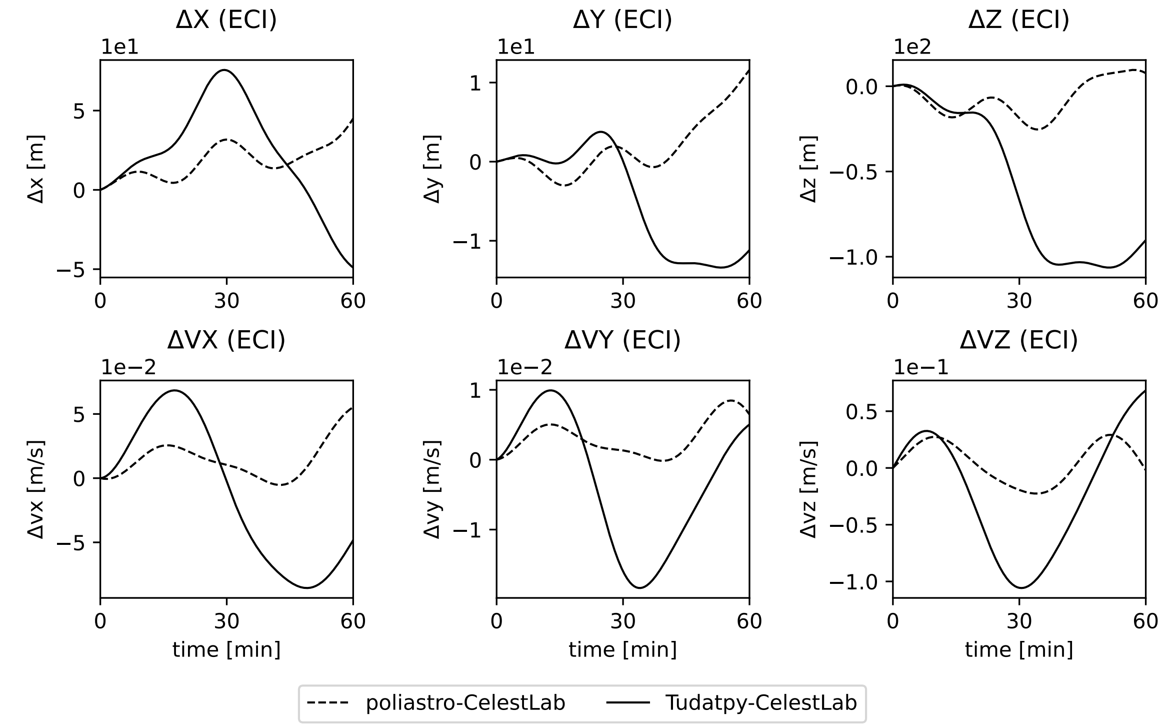

Figure 14. Difference between the CREME ECI state computed with Python libraries and with *CelestLab* over 1 hour with the `earthZonal` model.

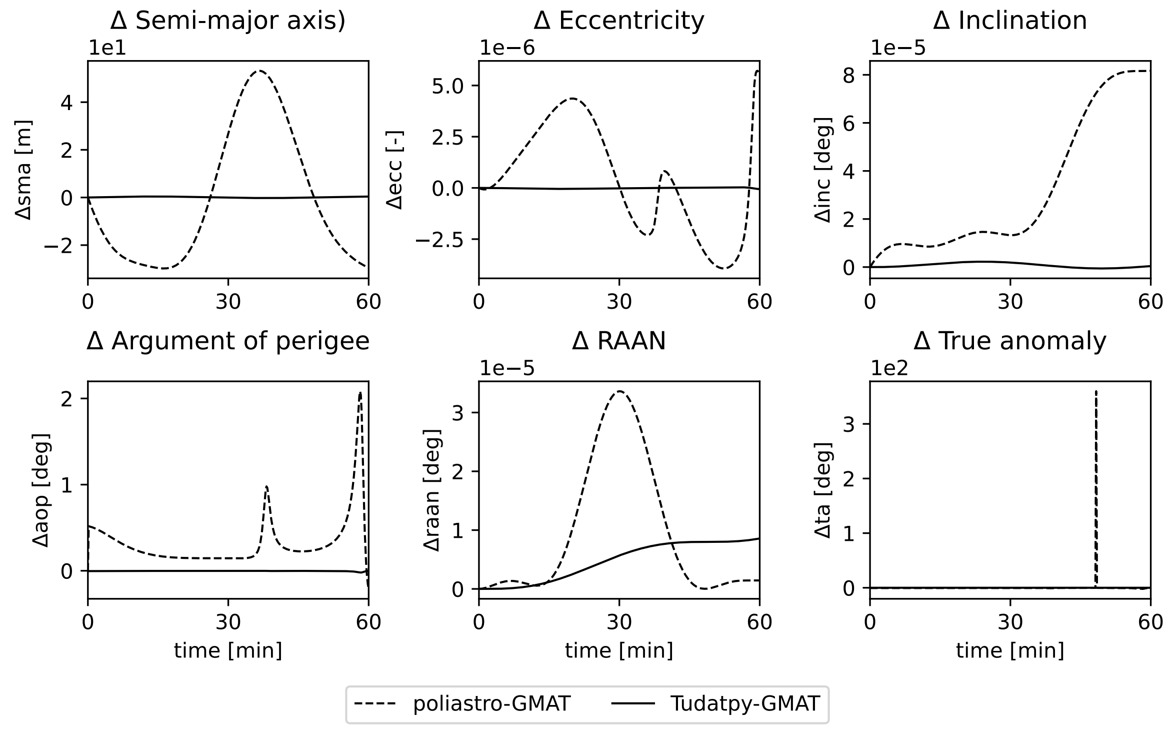

Figure 15. Difference between the CREME Keplerian state computed with Python libraries and with *GMAT* over 1 hour with the `earthZonal` model.

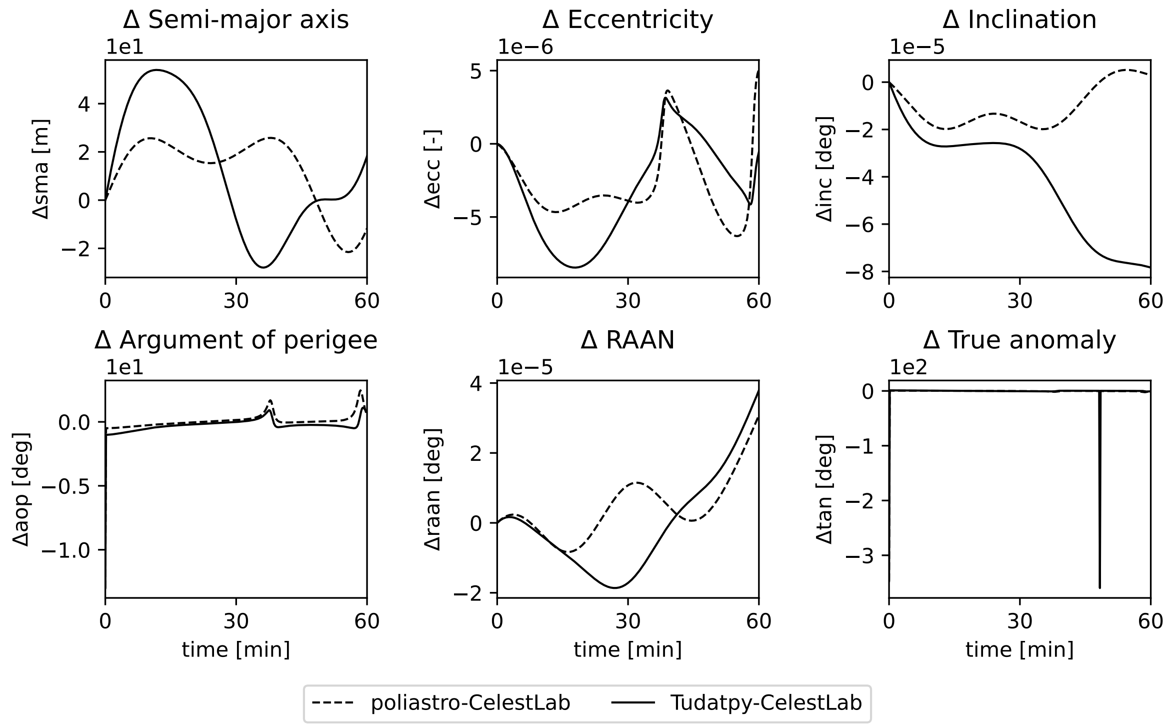

Figure 16. Difference between the CREME Keplerian state computed with Python libraries and with *CelestLab* over 1 hour with the `earthZonal` model.

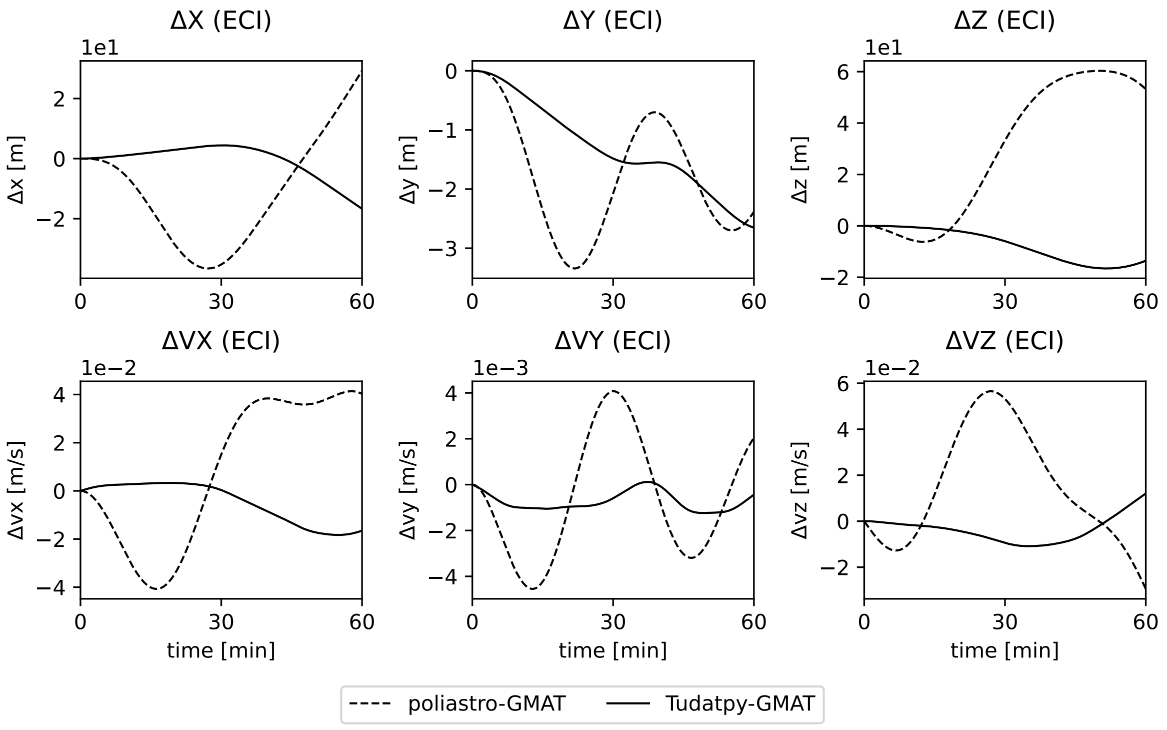

Figure 17. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 1 hour with the `complete` model.

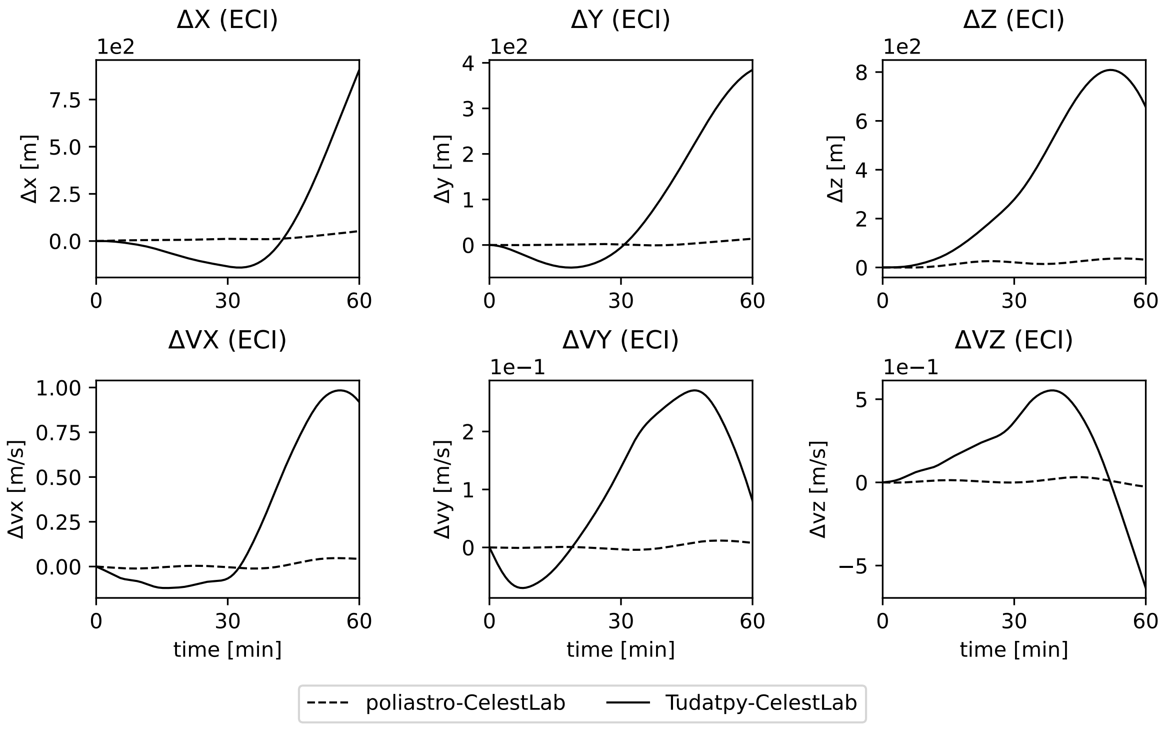

Figure 18. Difference between the CREME ECI state computed with Python libraries and with *CelestLab* over 1 hour with the `complete` model.

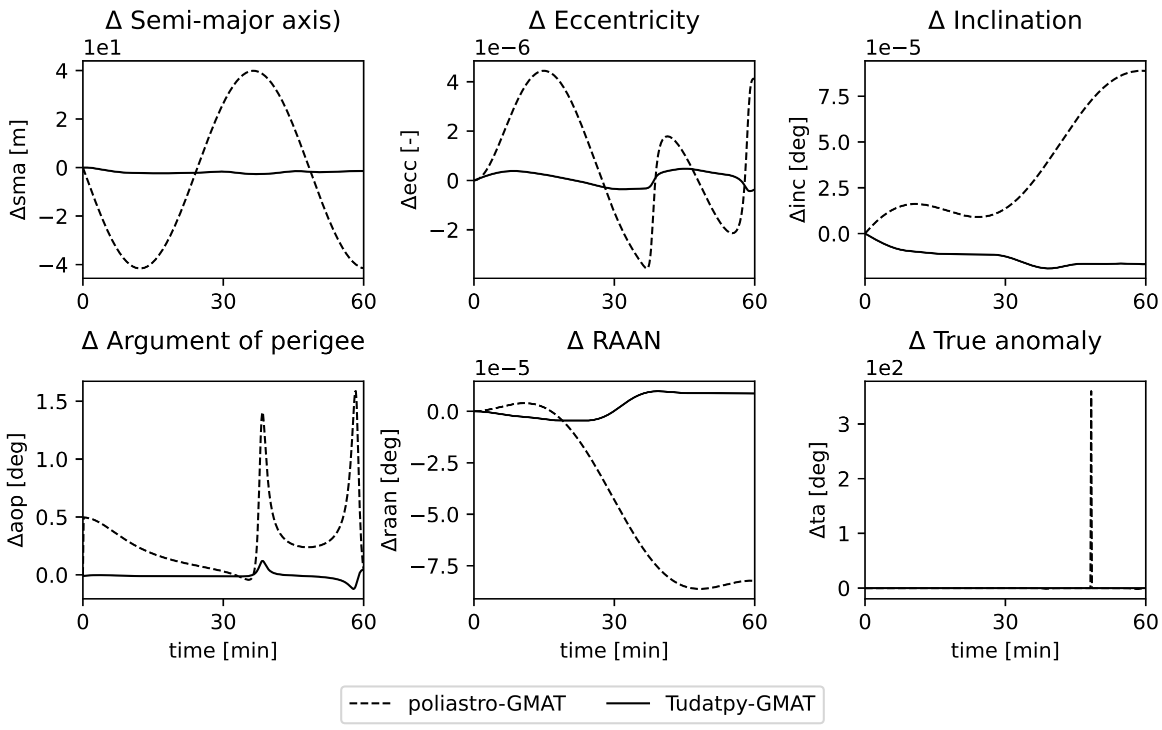

Figure 19. Difference between the CREME Keplerian state computed with Python libraries and with *GMAT* over 1 hour with the `complete` model.

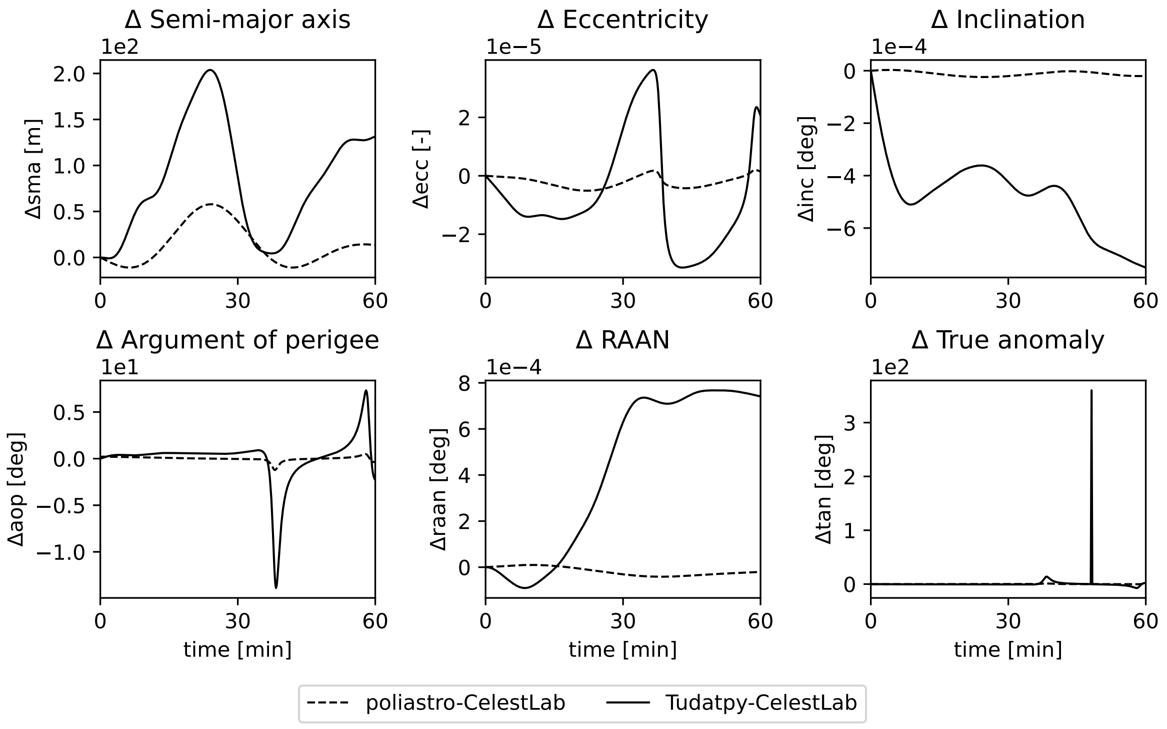

Figure 20. Difference between the CREME Keplerian state computed with Python libraries and with *CelestLab* over 1 hour with the `complete` model.

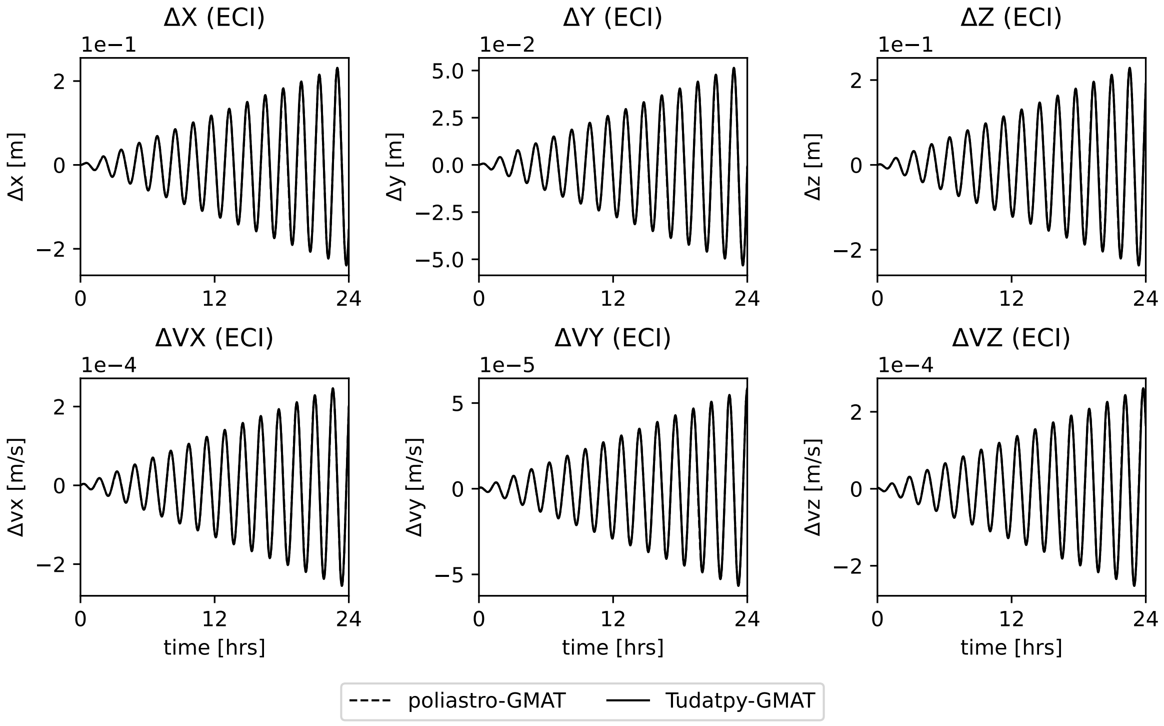

Figure 21. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 1 day with the `keplerian` model.

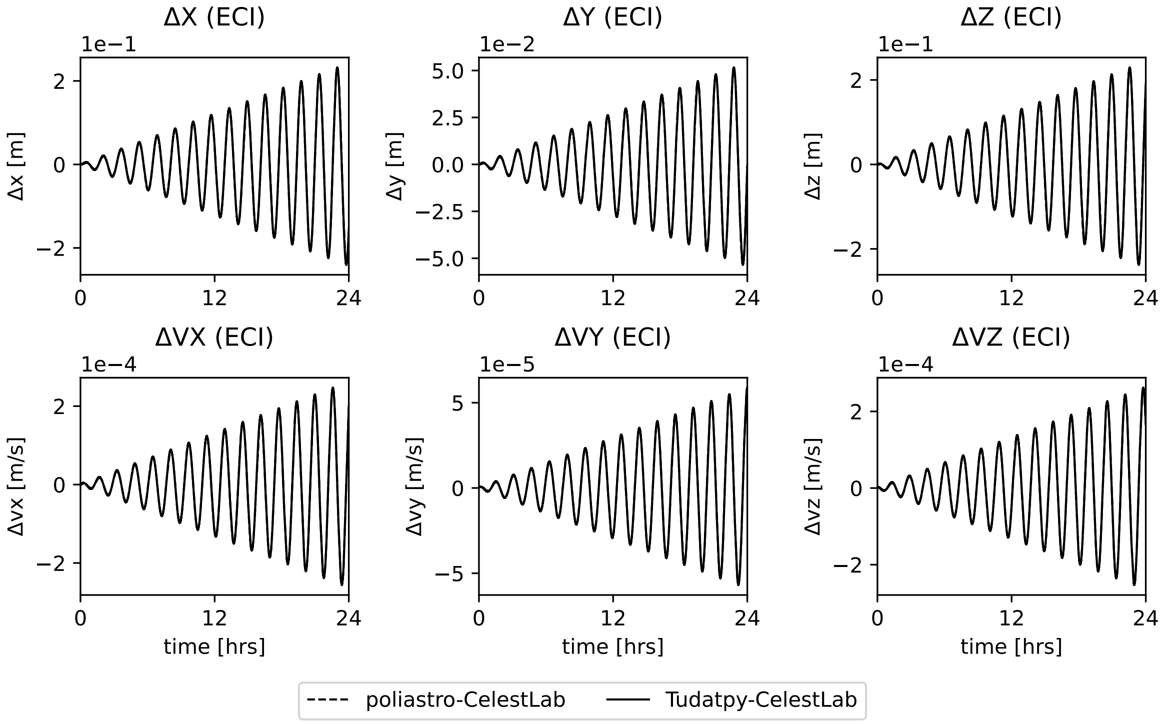

Figure 22. Difference between the CREME ECI state computed with Python libraries and with *CelestLab* over 1 day with the `keplerian` model.

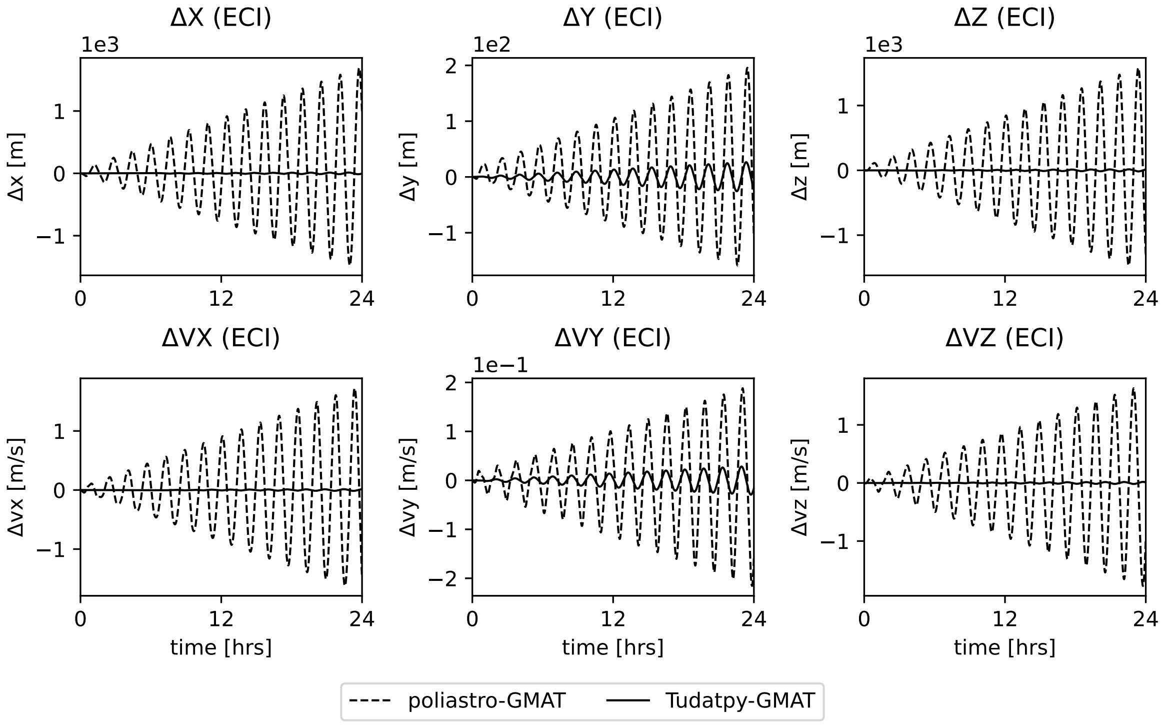

Figure 23. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 1 day with the `earthZonal` model.

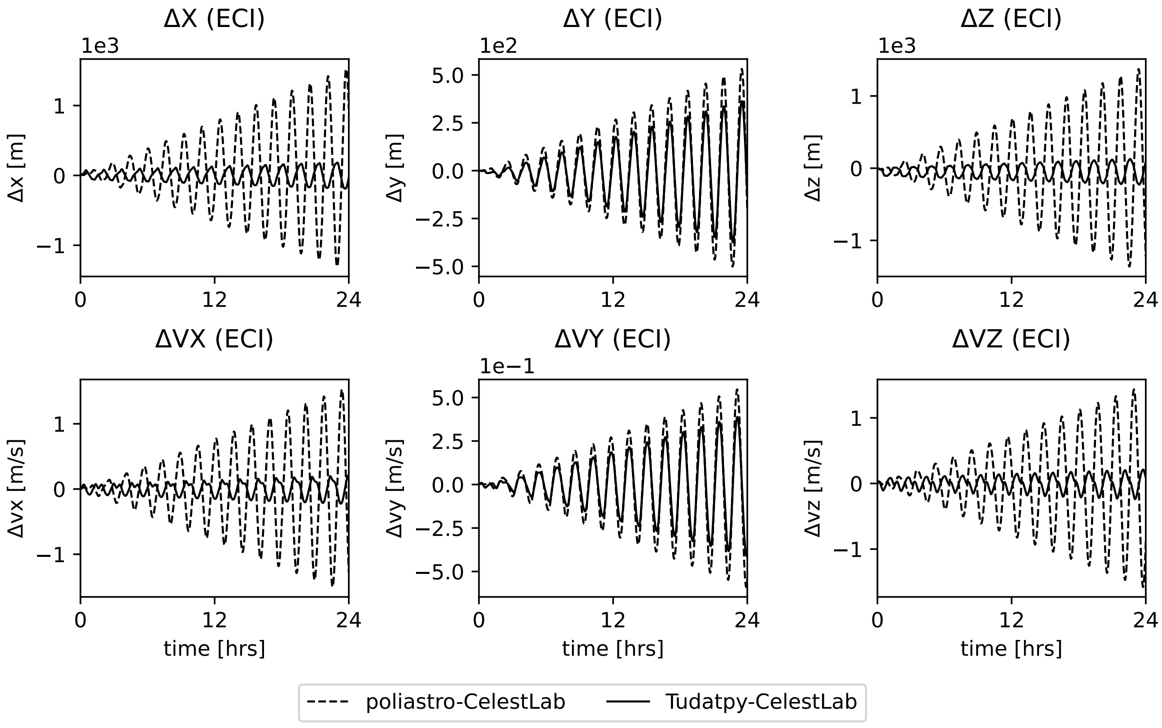

Figure 24. Difference between the CREME ECI state computed with Python libraries and with *CelestLab* over 1 day with the `earthZonal` model.

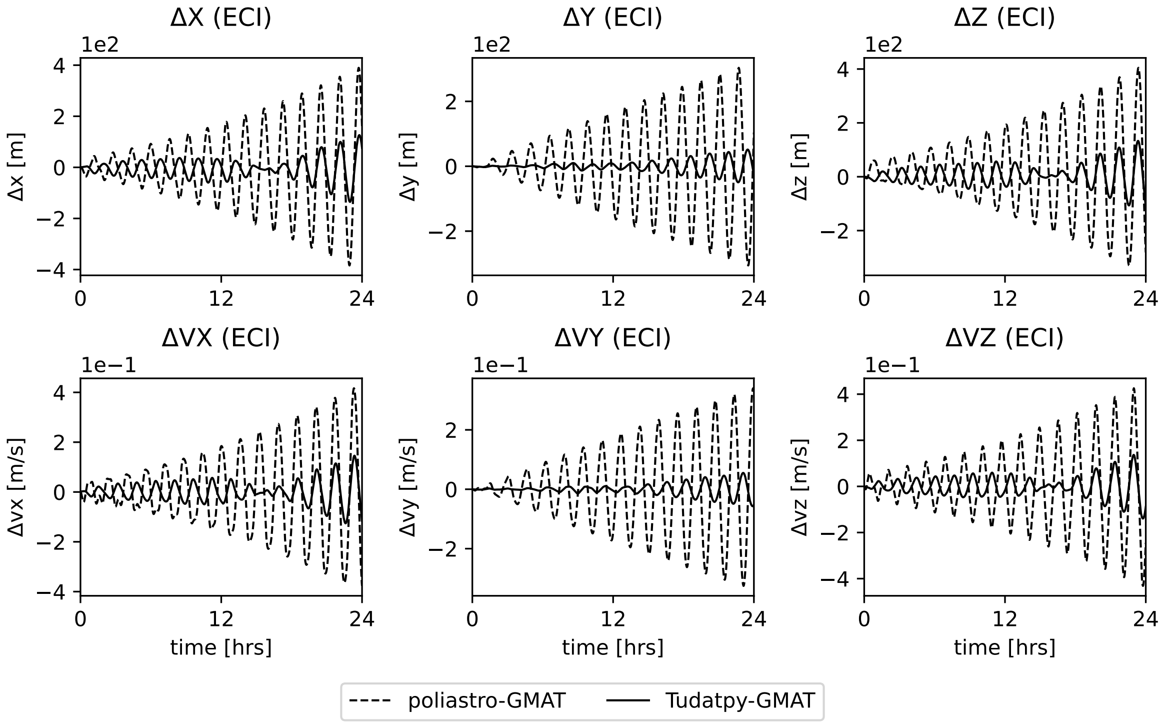

Figure 25. Difference between the CREME ECI state computed with Python libraries and with *GMAT* over 1 day with the `complete` model.

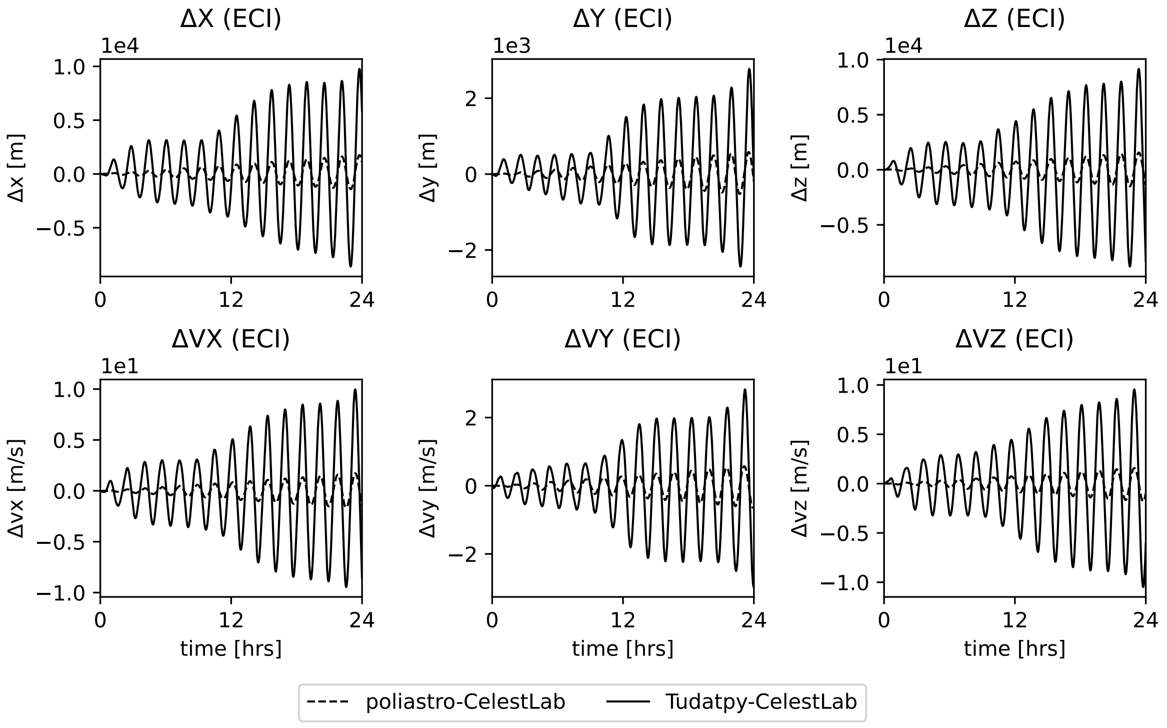

Figure 26. Difference between the CREME ECI state computed with Python libraries and with *CelestLab* over 1 day with the `complete` model.

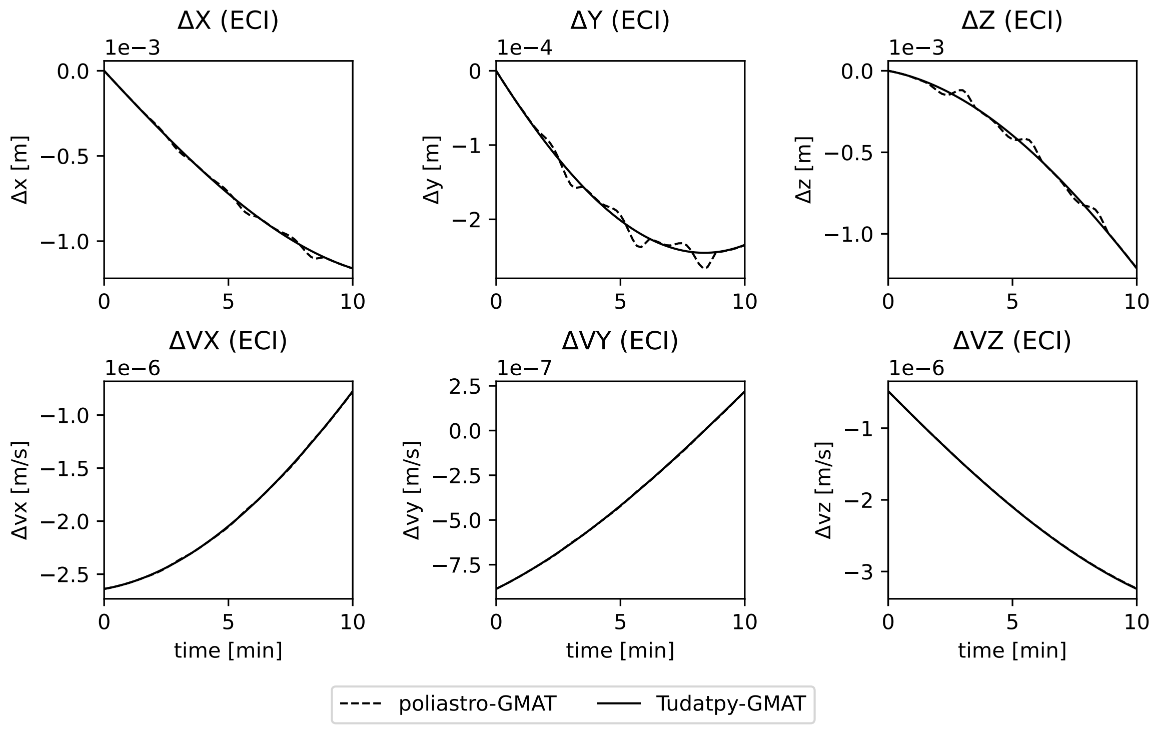

Figure 27. Difference between the Sunjammer ECI state computed with Python libraries and with *GMAT* over 10 minutes with the `keplerian` model.

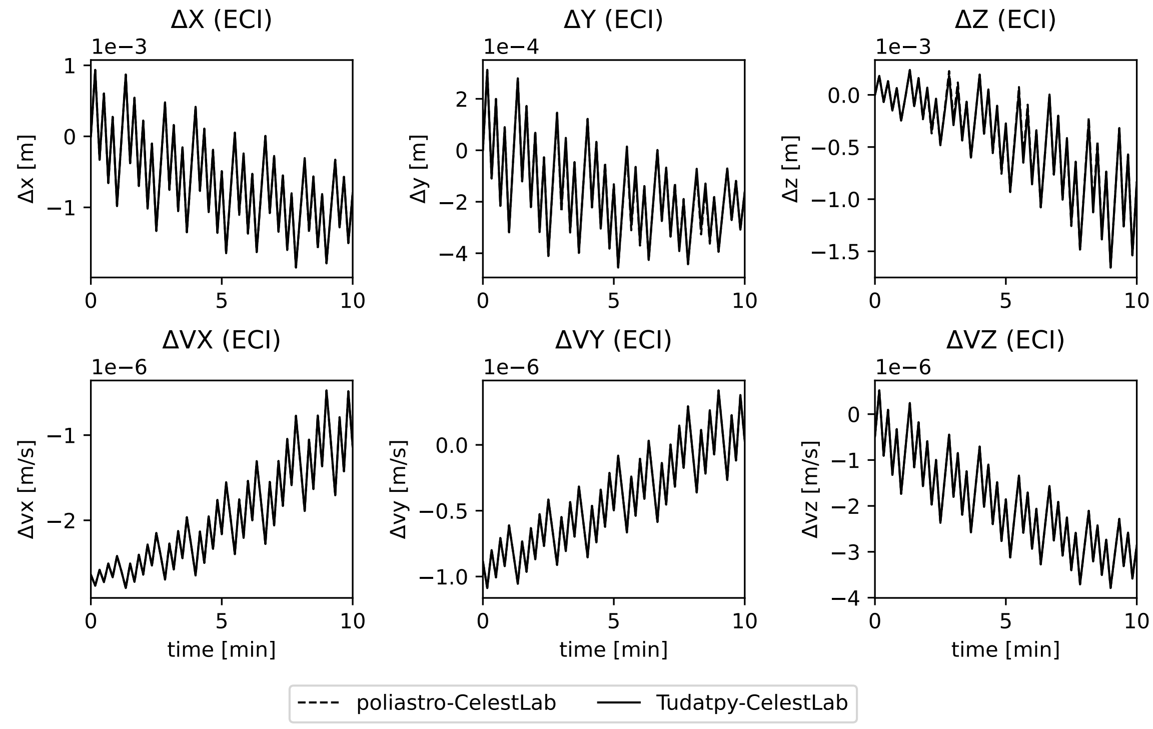

Figure 28. Difference between the Sunjammer ECI state computed with Python libraries and with *CelestLab* over 10 minutes with the `keplerian` model.

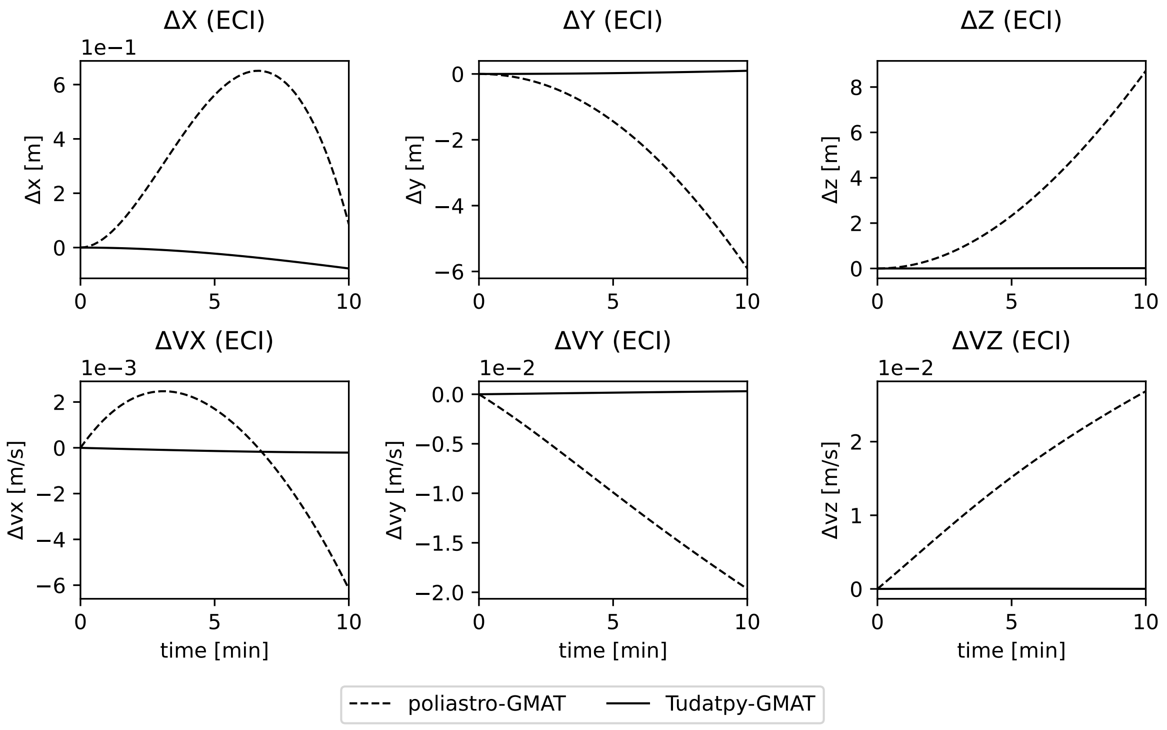

Figure 29. Difference between the Sunjammer ECI state computed with Python libraries and with *GMAT* over 10 minutes with the `earthZonal` model.

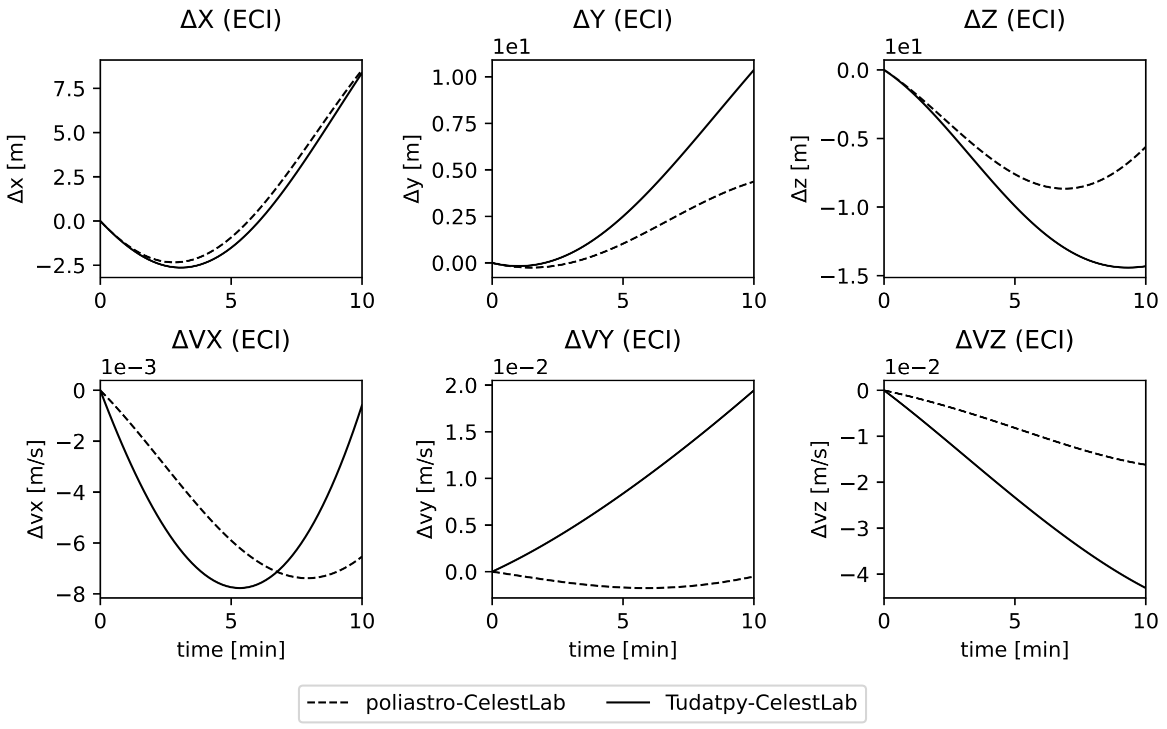

Figure 30. Difference between the Sunjammer ECI state computed with Python libraries and with *CelestLab* over 10 minutes with the `earthZonal` model.

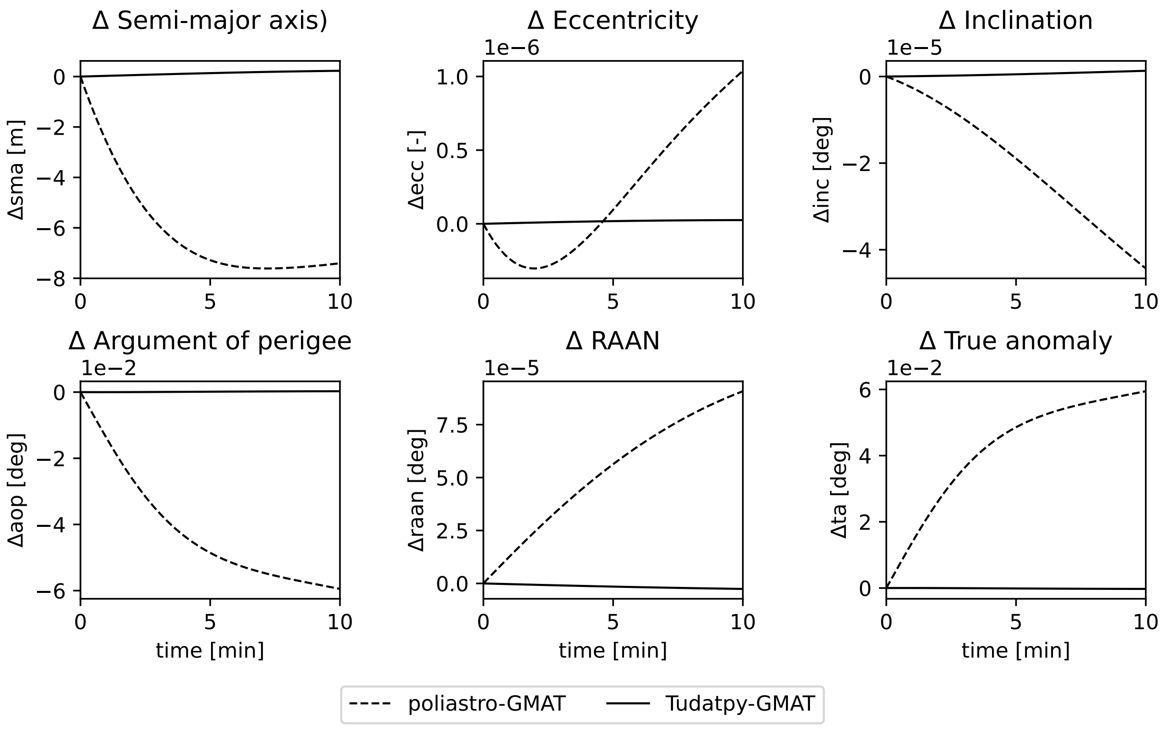

Figure 31. Difference between the Sunjammer Keplerian state computed with Python libraries and with *GMAT* over 10 minutes with the `earthZonal` model.

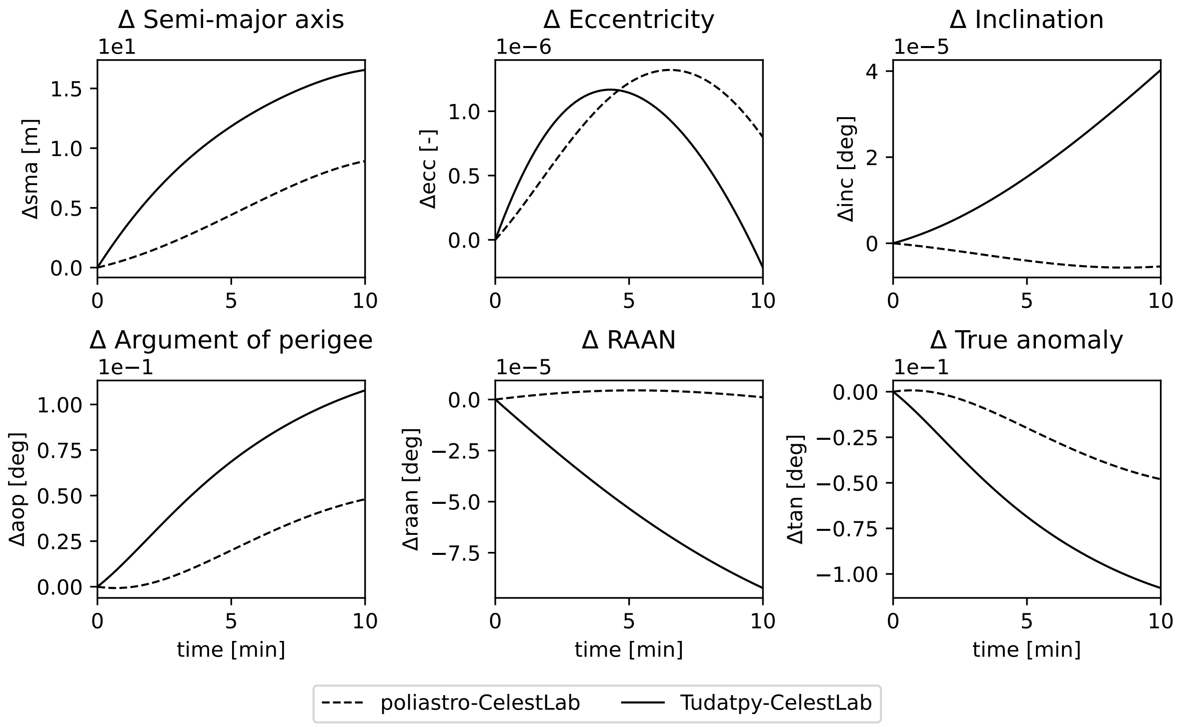

Figure 32. Difference between the Sunjammer Keplerian state computed with Python libraries and with *CelestLab* over 10 minutes with the `earthZonal` model.

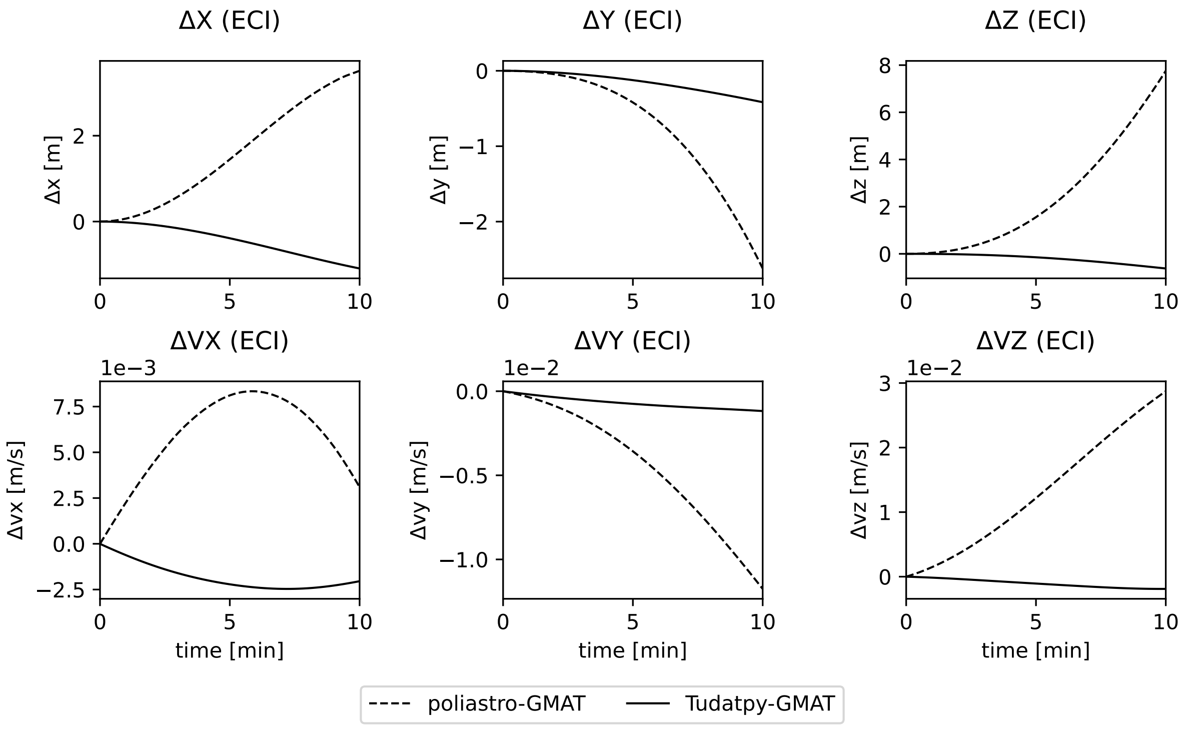

Figure 33. Difference between the Sunjammer ECI state computed with Python libraries and with *GMAT* over 10 minutes with the `complete` model.

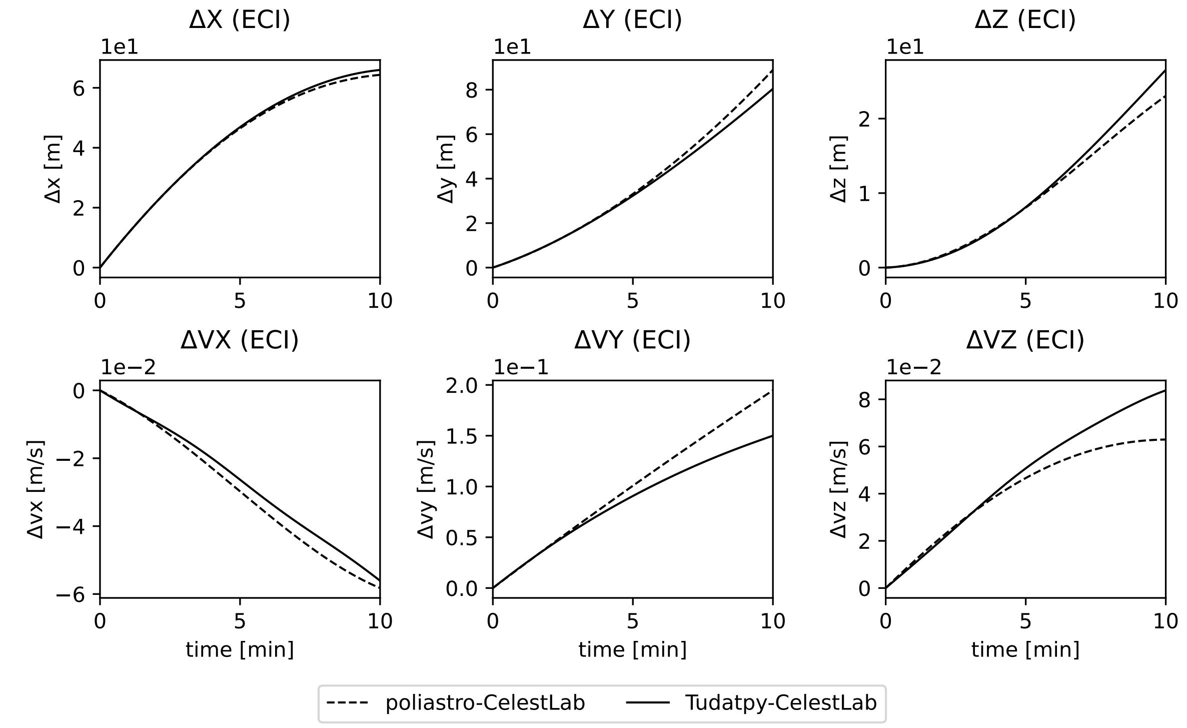

Figure 34. Difference between the Sunjammer ECI state computed with Python libraries and with *CelestLab* over 10 minutes with the `complete` model.

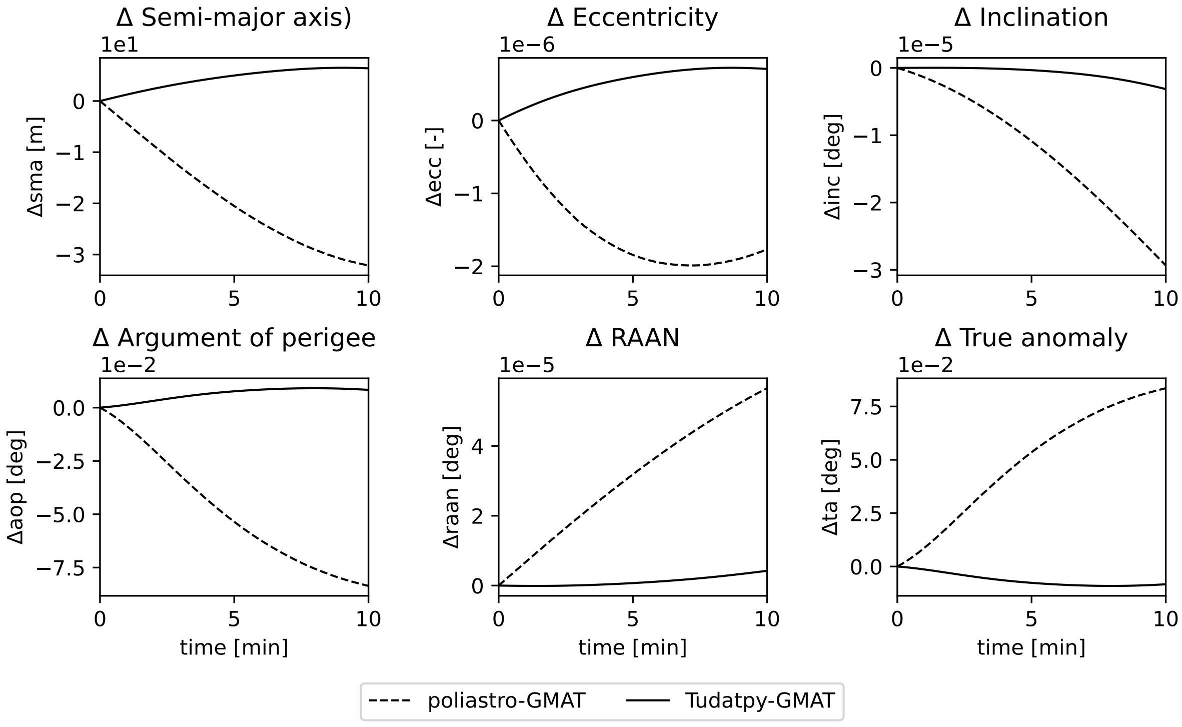

Figure 35. Difference between the Sunjammer Keplerian state computed with Python libraries and with *GMAT* over 10 minutes with the `complete` model.

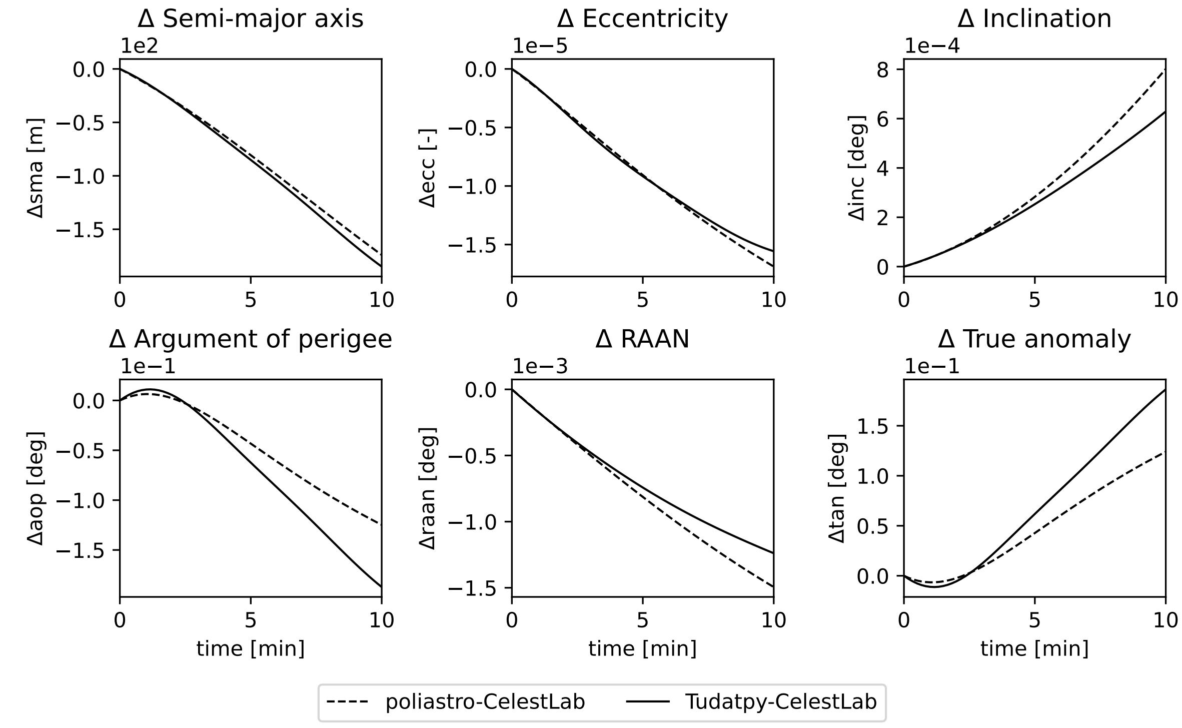

Figure 36. Difference between the Sunjammer Keplerian state computed with Python libraries and with *CelestLab* over 10 minutes with the `complete` model.

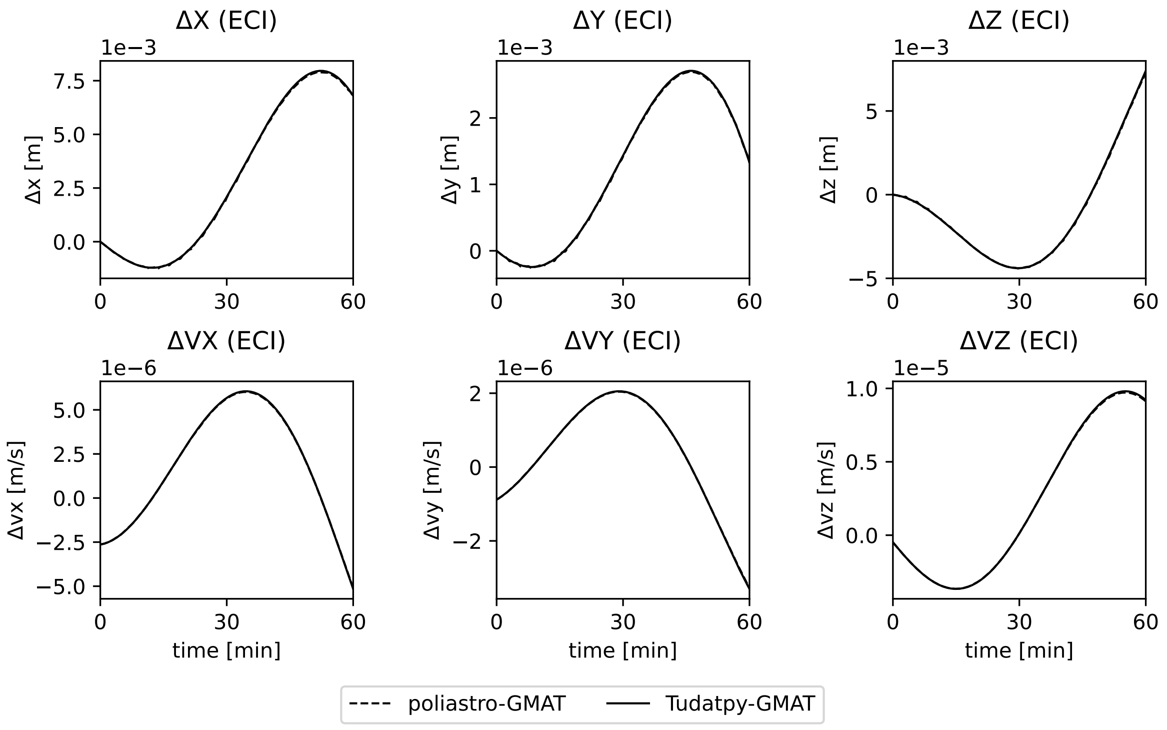

Figure 37. Difference between the Sunjammer ECI state computed with Python libraries and with *GMAT* over 1 hour with the `keplerian` model.

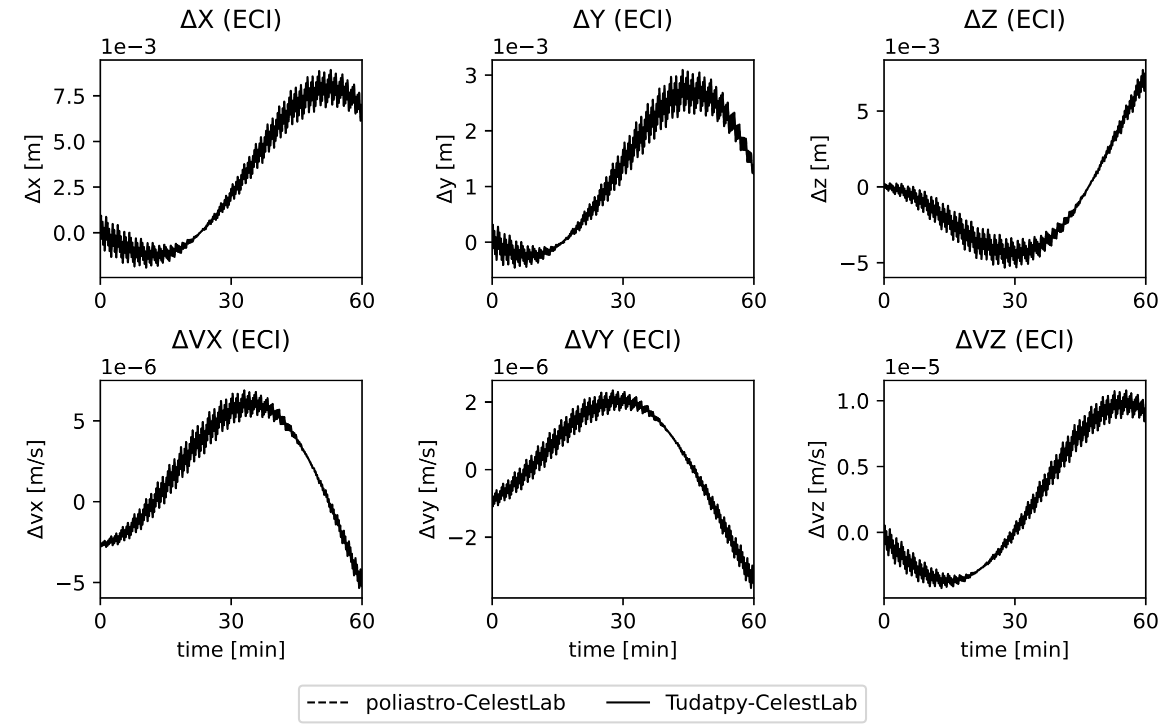

Figure 38. Difference between the Sunjammer ECI state computed with Python libraries and with *CelestLab* over 1 hour with the `keplerian` model.

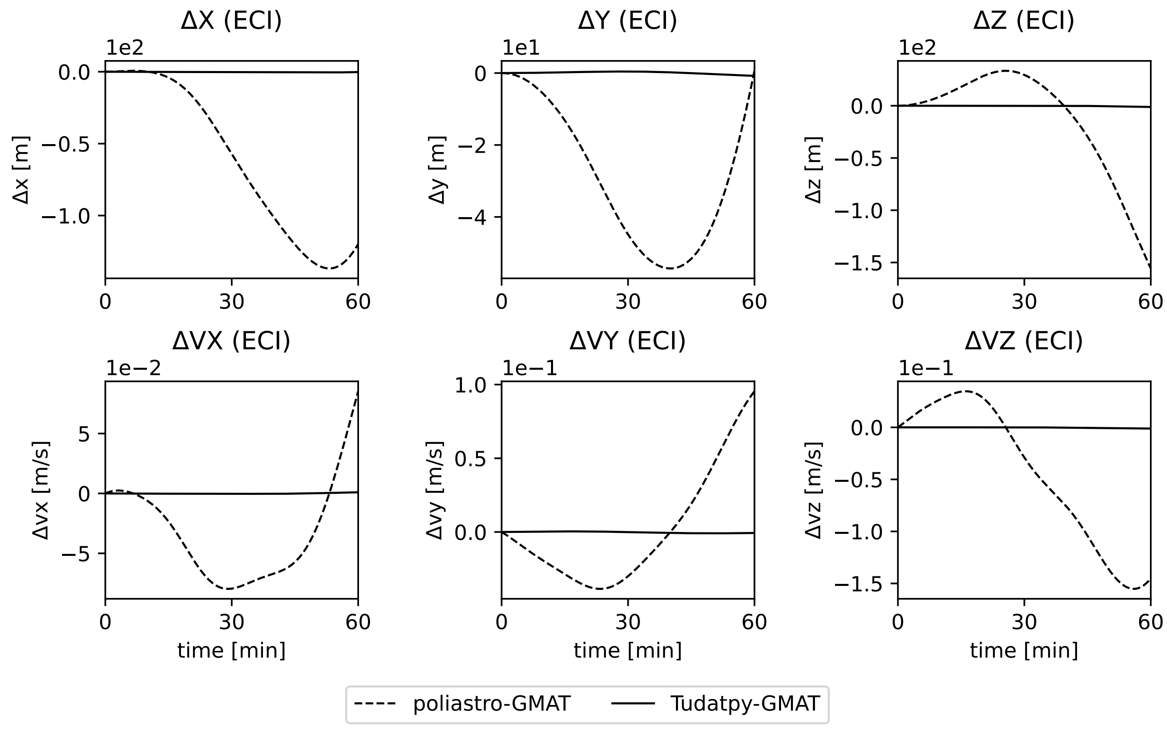

Figure 39. Difference between the Sunjammer ECI state computed with Python libraries and with *GMAT* over 1 hour with the `earthZonal` model.

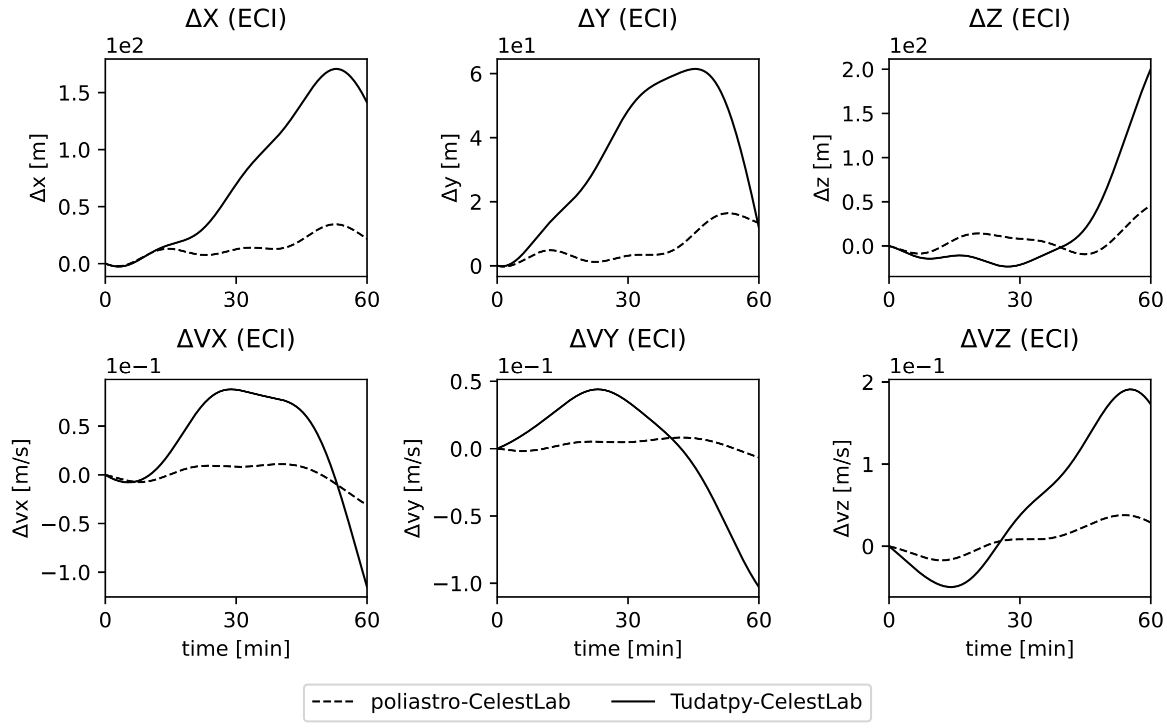

Figure 40. Difference between the Sunjammer ECI state computed with Python libraries and with *CelestLab* over 1 hour with the `earthZonal` model.

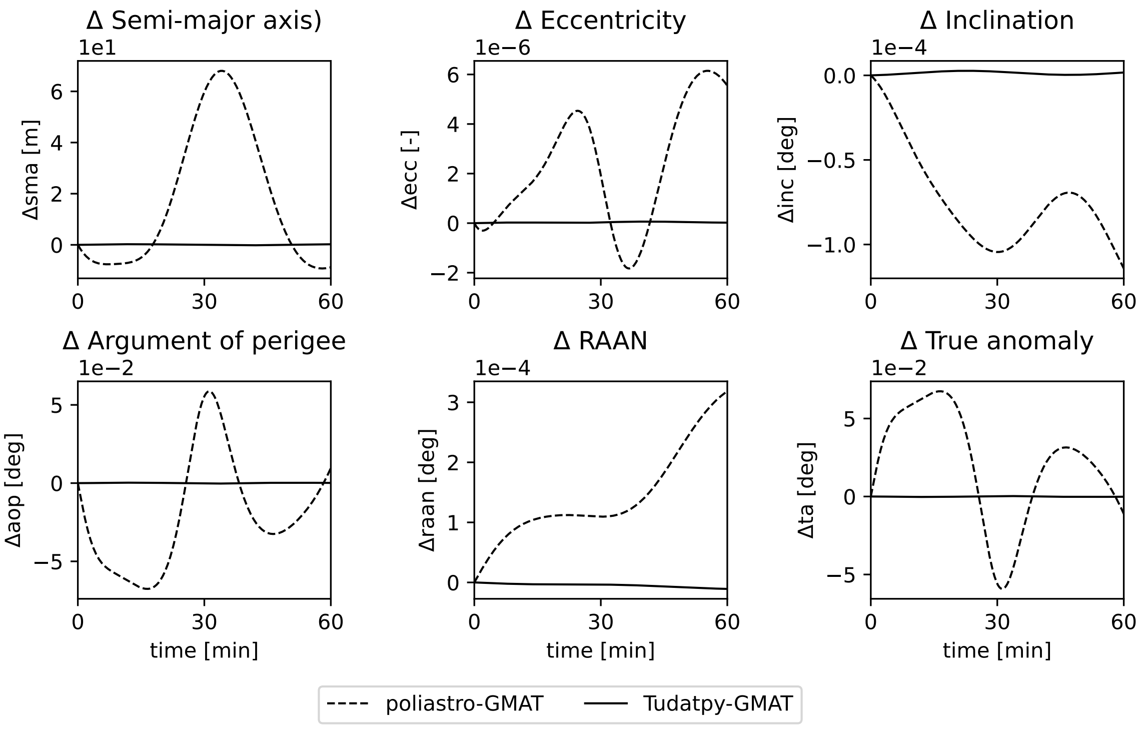

Figure 41. Difference between the Sunjammer Keplerian state computed with Python libraries and with *GMAT* over 1 hour with the `earthZonal` model.

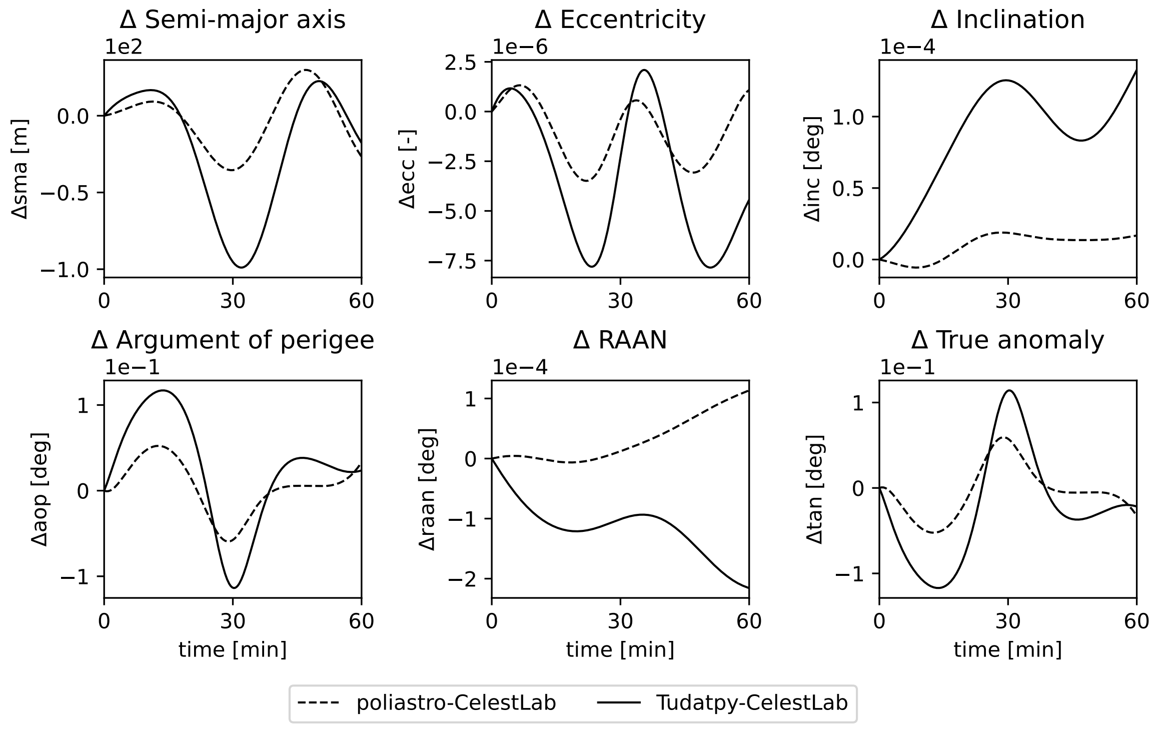

Figure 42. Difference between the Sunjammer Keplerian state computed with Python libraries and with *CelestLab* over 1 hour with the `earthZonal` model.

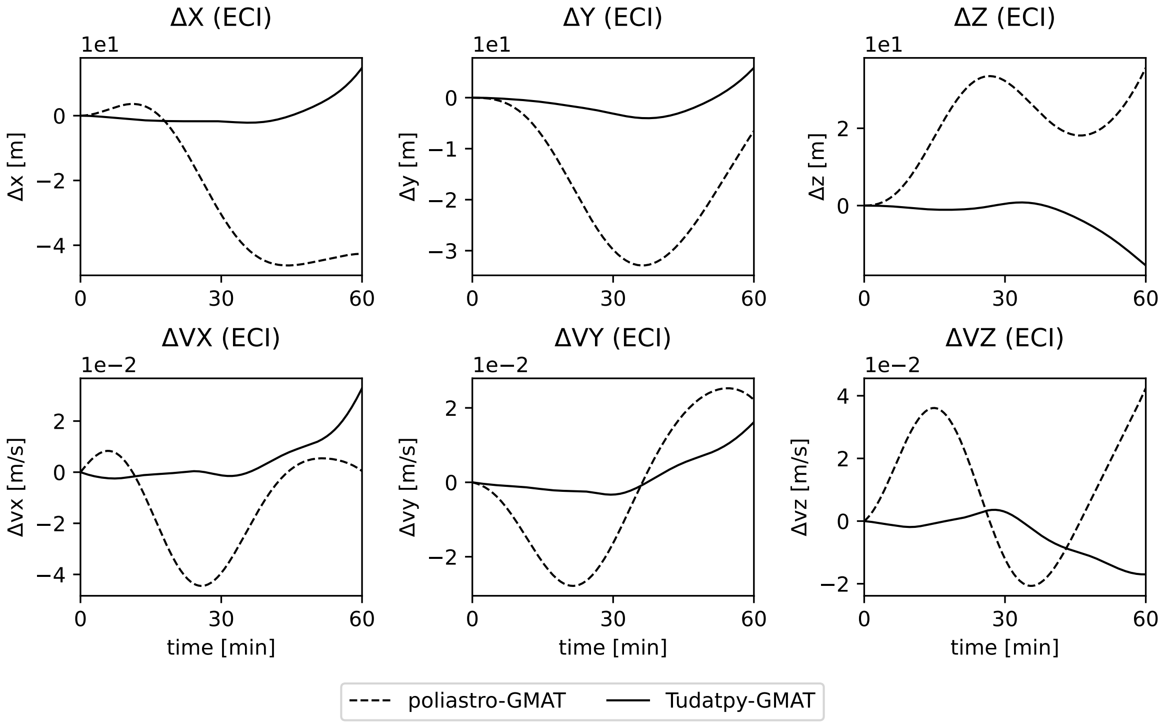

Figure 43. Difference between the Sunjammer ECI state computed with Python libraries and with *GMAT* over 1 hour with the `complete` model.

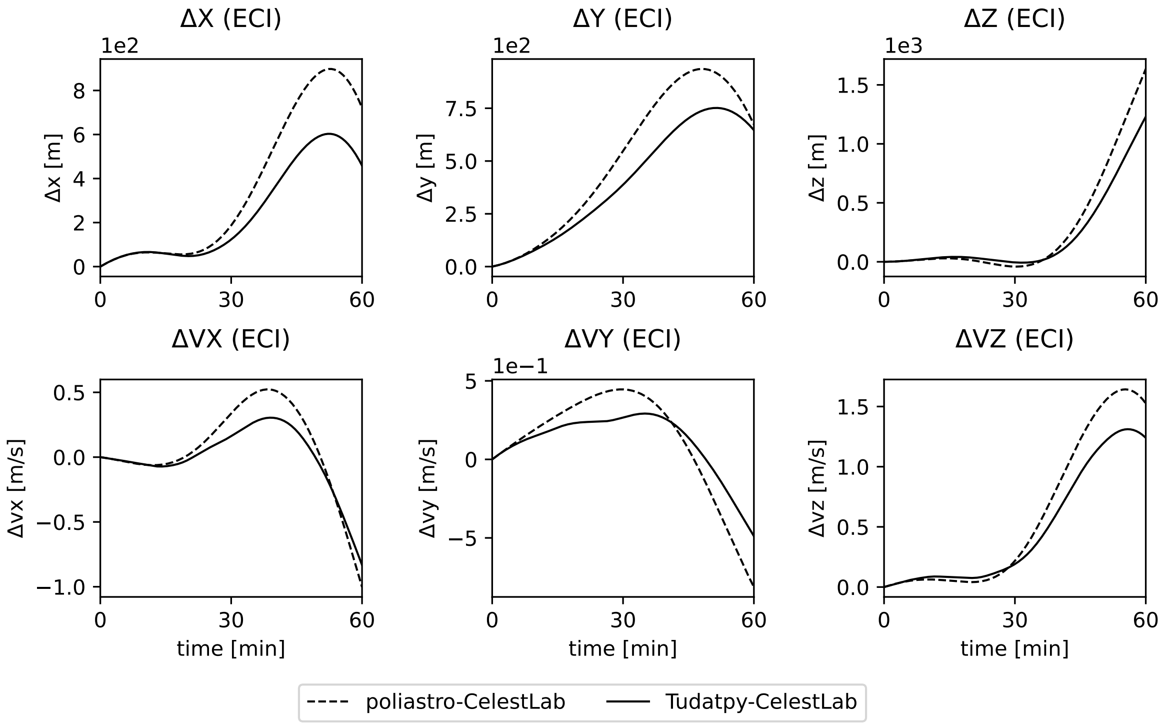

Figure 44. Difference between the Sunjammer ECI state computed with Python libraries and with *CelestLab* over 1 hour with the `complete` model.

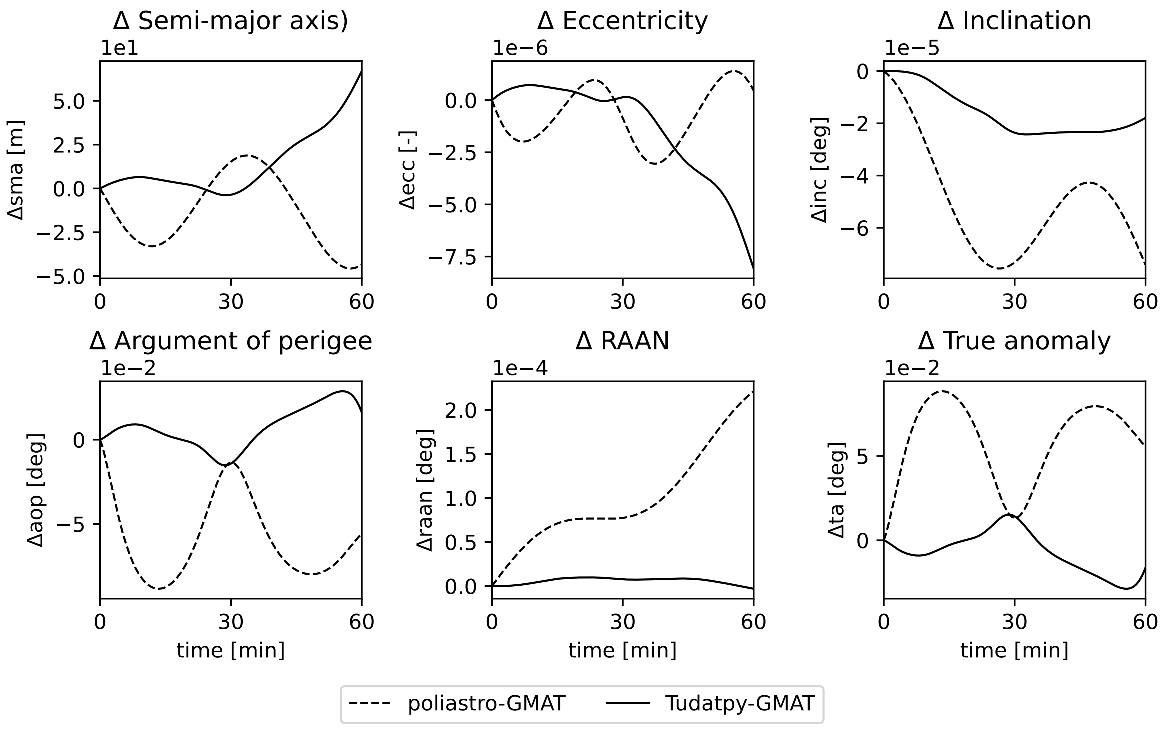

Figure 45. Difference between the Sunjammer Keplerian state computed with Python libraries and with *GMAT* over 1 hour with the `complete` model.

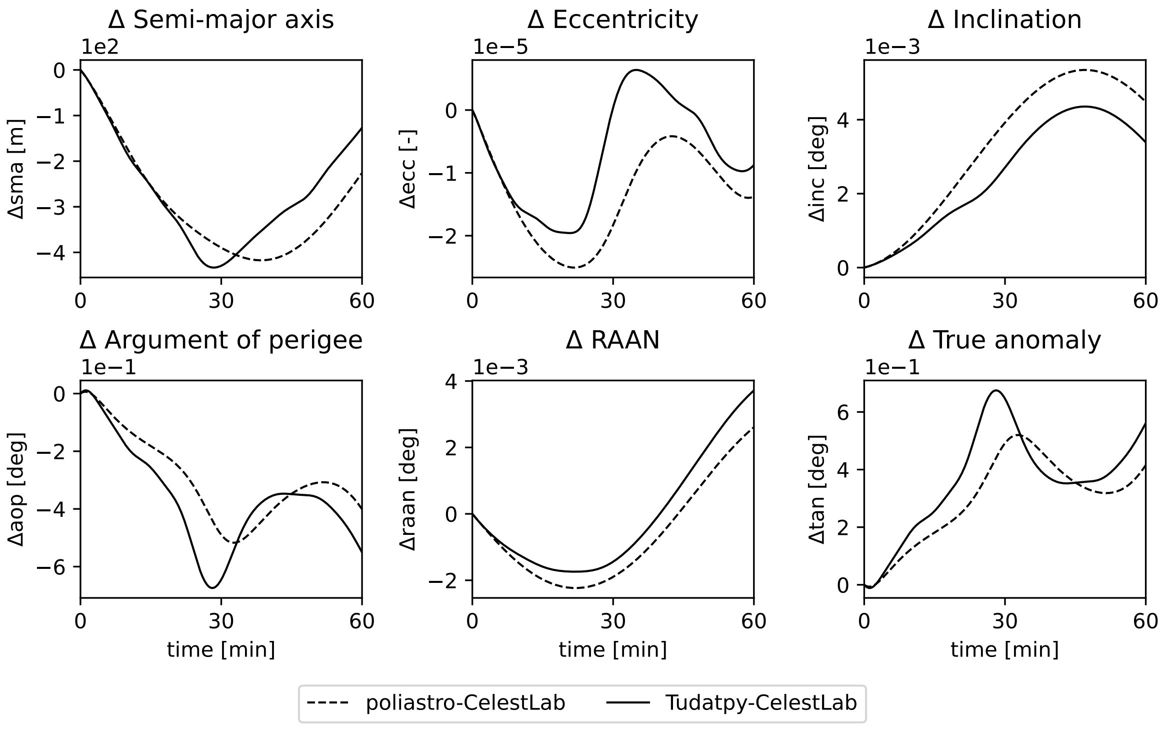

Figure 46. Difference between the Sunjammer Keplerian state computed with Python libraries and with *CelestLab* over 1 hour with the `complete` model.