

Technical Note: EOCFI – ASGARD Comparison

Contents

1. Change Log	2
2. Introduction	3
2.1 General Overview	3
2.2 Important considerations	3
2.3 External References	4
3. Function Selection Process	5
4. Comparisons	6
4.1 xl_change_cart_cs	7
4.1.1 Data generation	7
4.1.2 Results	9
4.2 Planet Ephemerises (sun, moon and others)	11
4.2.1 Data generation	11
4.2.2 Results	11
4.3 xl_geod_distance	12
4.3.1 Data generation	12
4.3.2 Results	13
4.4 xp_target_inter	15
4.4.1 Important Considerations/Notes	15
4.4.2 Data generation	17
4.4.3 Results (focus on YSM's implementation)	19
4.4.4 Results (focus on intersection algorithm)	20
4.5 xo_osv_compute (interpolation)	21
4.5.1 Data generation	21
4.5.2 Results	22
4.6 xo_osv_compute (Single, Double and Precise mode)	24
4.7 xo_osv_compute (OSF initialisation)	27
4.7.1 Data generation	27
4.7.2 Results	29
5. List of Provided Items.	32
6. Results Summary	33
7. Conclusions	34

1. Change Log

Changes	Issue	Date
Initial writing of Document.	1	18/12/2024
Correction of slight mistakes + correction of retained drag coefficient + inclusion of additional files (data package to reproduce test data)	1.1	23/12/2024

2. Introduction

2.1 General Overview

The new ASGARD Image processing chain aims to replace certain functionalities formerly assumed by EOCFI library. This document's aims to assess the difference between EOCFI's and Asgard results given by functions computing the same, or if not possible as similar as possible, values.

It is important to note that, considering:

- the time allowed for this study,
- the general lack of information regarding EOCFI's implementation of said functions,
- the fact the EOCFI's code has been provided in a form that only allow for straight execution, not deep code diving/analysis,

this study's focus is only on the production and presentation of results yielded. The close validation or root cause analysis of said differences is, at this moment, out of the scope of this document. However, if a difference in implementation is obvious and explains discrepancies observed during comparison, it will be brought forward.

This document is also accompanied by several scripts and tools (along with a quick usage guide) used to produce the observed results. General details on those items are available at section 5. Specific details are presented in each of the function's comparison sections (whole section 4).

As different sources will be discussed, and in the interest of clarity, a colour scheme is used. Throughout this document, data produced, or scripts to be executed by **EOCFI will be blue coded**. Likewise, data or script related to **ASGARD will be green coded**. Occasionally, purely **EXTERNAL/THIRD PARTY** data/sources can be presented, **those will be red coded**.

Section 3 presents the selection process of EOCFI's functions that are part of this document's scope. This process takes into consideration the technical notes provided by ESA regarding the accuracy of EOCFI's functions. All relevant external documents/sources can be found in section 2.3. Section 4 constitutes the core of this document and presents the differences observed between functions/features. A comprehensive summary of results can be found in section 6 for quick reference.

2.2 Important considerations

It is important to note that ASGARD is not necessarily intended to provide a "one to one" equivalent to EOCFI features. Certain aspects and usages are not yet implemented or simply not required by ASGARD. The design and architecture are not the same. Certain functionalities require specific setup in ASGARD. This is, in part, the reason why several scripts have been created to support this study.

The provided version of EOCFI is 4.27. All comparisons and relevant documents refer to this specific version of the library.

ASGARD is built upon several other products and libraries. In the frame of this study, one library of particular interest is Orekit, an orbital mechanics library developed and maintained by CS GROUP/SOPRA-STERIA in Java. This library is completely independent from ASGARD and follows its own development and release cycles. The specific version on which ASGARD is built is 11.3.2 (for reference current latest release version is 12.2). As will appear shortly, comparison directly with Orekit have been made. Those comparison do not impact the relevance of results as it directly calls upon Orekit for specific computation of interest in this study.

2.3 External References

All documents or other references are listed below:

Internal ID (in this document)	Title	ESA ID
[REF01]	Accuracy of EO CFI Software functions in the Lib and Orbit libraries, Issue 2 Rev 1	PE-TN-ESA-GS-404
[REF02]	Accuracy of EO CFI Software functions in the Pointing library, Issue 1 Rev 1	PE-TN-ESA-GS-470
[REF03]	ASGARD quick start page : https://geolib.pages.eopf.copernicus.eu/asgard/quickstart.html	N/A
[REF04]	Orekit v11.3.2 javadoc page: https://www.orekit.org/site-orekit-11.3.2/apidocs/	N/A
[REF05]	Standards of Fundamental Astronomy, http://www.iausofa.org/	[RD09] in PE-TN-ESA-GS-404
[REF06]	Earth Observation Mission CFI Software. EO_LIB Software User Manual, Issue 4.27	EO-MA-DMS-GS-0003
[REF07]	Earth Observation Mission CFI Software. CONVENTIONS DOCUMENT, Issue 4.27	EO-MA-DMS-GS-0001

Table 1 : Reference Documents

Side Note:

For convenience, specific references can be made “on the spot” where deemed relevant. Those references/quotations will generally take the form of a hyperlink to a specific publicly consultable page.

3. Function Selection Process

As briefly mentioned in 2.2, this study is intended to replicate, as close as possible, the 2 technical notes provided by ESA, namely PE-TN-ESA-GS-404 and PE-TN-ESA-GS-470. However, considering that:

- Not all EOCFI functions have their equivalent in ASGARD
- Some of ASGARD features/functionalities are still in development.

The entirety of function cannot be tested within ASGARD. Hence, the scope of this study has been dressed as follows:

- Based on [RD03] page, all EOCFI directly quoted have been listed.
- Among those all functions that are quoted in either [RD01] or [RD02] have been kept.
- Among those, all functions that are not marked as accuracy relevant (flag “Accuracy Determination” marked as “Required”) have been left out of the scope.
- Within [RD01] and [RD02], even though certain functions have been marked as accuracy relevant, their results have not been presented. The section being empty or To Be Written, no context is available (specific data file used, treatment applied to data, etc...). As such they also have been left out of the scope of this study.

Applying those steps yields the following list of functions to be considered in this study:

EOCFI sub Library	Function
LIB	xl_change_cart_cs
	xl_sun
	xl_moon
	xl_planet
	xl_geod_distance
ORB	xo_osv_compute (Interpolation)
	xo_osv_compute (Kep Elem Prop, double Mode)
	xo_osv_compute (Kep Elem Prop, single Mode)
	xo_osv_compute (Kep Elem Prop, init OSF)
	xo_osv_compute (precise propagation)
POINT	xp_target_inter

Table 2 : List of Functions considered

4. Comparisons

For flexibility and to produce results in a timely manner, specific files and routines have been either altered (generally for EOCFI's case) or created (generally for ASGARD/Orekit's case) for each function to be tested.

As general example, comparison can be made by:

- Processing (parsing) a specific file within EOCFI,
- used as input data for a function (in a specific EOCFI altered source file),
- the EOCFI input data (if relevant) and associated results printed out in a CSV file
- This file is then parsed within ASGARD/OREKIT and used as input data to launch an equivalent function / process.
- Another CSV file is then produced containing ASGARD/OREKIT's results.
- The 2 files (EOCFI's and ASGARD/OREKIT) are then generally ingested by a Scilab script managing plots.

This process can be different depending on each function, as the context (datawise) needed to execute a specific function can be tedious setup. In any case, the whole process will be presented in general terms for each function.

In the interest of clarity, and completing the colour scheme, in the following sections:

- Files will be highlighted in **bold**, example: **explorer_lib_run_c.c** (file in the LIB sub library of EOCFI's functions)
- Routines/functions will be highlighted in *italic*, example: *getSSOIncl* (specific routine in **OSFData.java**, part of Orekit mock-up).

4.1 *xl_change_cart_cs*

4.1.1 Data generation

The results presented in [RD01] section 2.3, represents conversion of coordinates from the Earth Fixed and True of Date (ToD) reference frames also as the conversion from EF to Geocentric Mean 2000 (GM2000).

Side Notes:

The “Earth Fixed” (EF) reference frame is the given name within EOCFI’s context. ASGARD equivalent frame of reference is generally mentioned as “ECEF” (Earth Centered Earth Fixed). Orekit’s equivalent frame official name is “ITRF” (<https://www.orekit.org/static/architecture/frames.html>).

No ambiguity for the “ToD” frame, the same name refers to the same frame independent of context.

The “GM2000” frame is the name present in EOCFI’s context. ASGARD refers to this specific frame as “J2000”, Orekit’s official name is “EME2000”.

The inputs positions fed into *xl_change_cart_cs* are not presented in [REF01]. Only the resulting difference between EOCFI and the **SOFA library** are. Furthermore, results are presented in function of latitude. One can assume that the positions coordinate system has been changed from cartesian to spherical.

To reflect this process, routine *GenPosXL_CHANGE_CART_CS* has been created in **explorer_lib_run_c.c**. The detail of this routine is the following:

- This routine generates at specific dates a range of spherical positions, which altitude has been arbitrarily set to 700km.
- The dates correspond to the context presented in [RD01] section 2.3. However, only the months are specified (March, June, September and December). The 15th at 12:00:00 (TAI) has been arbitrarily chosen for the specific day (caution: these dates are hard coded).
- The correlation data necessary to instantiate dates [REF06] are provided by several calls to *computeEOCFITimeCorrelationData*. Alternatively, user can directly launch *genTimeRefData* to verify the time correlation data corresponding to *GenPosXL_CHANGE_CART_CS*.
- Those positions have then been converted into cartesian and then fed to *xl_change_cart*.
- The inputs positions (cartesian input positions along with corresponding latitudes and dates) and the output positions have been written into **EF_GM2000_valid_data.csv** and **EF_ToD_valid_data.csv** and can be consulted at ESA’s discretion.

Within ASGARD, functions to operate the change in frames have to be properly initialized. This process is context dependant (IERS data, leap second history, etc...). In order to streamline and bypass this process a specific Orekit routine has been implemented (*Process_xl_change_cart_cs*, for details).

This routine does the following:

- Parsing of files **EF_GM2000_valid_data.csv** and **EF_ToD_valid_data.csv**
- Retrieves the initial input positions (EF)
- Transforms those positions into EME2000 and ToD.
- Computes the difference between the 2 generated positions.
- The results are written into **diff_eocfi_orekit_EF_EME2K.csv** and **diff_eocfi_orekit_EF_ToD.csv**

- Those results are then processed by **Exploit_xl_change_cart_cs.sce**.

4.1.2 Results

The results are presented hereunder by date and type of transformation computed:

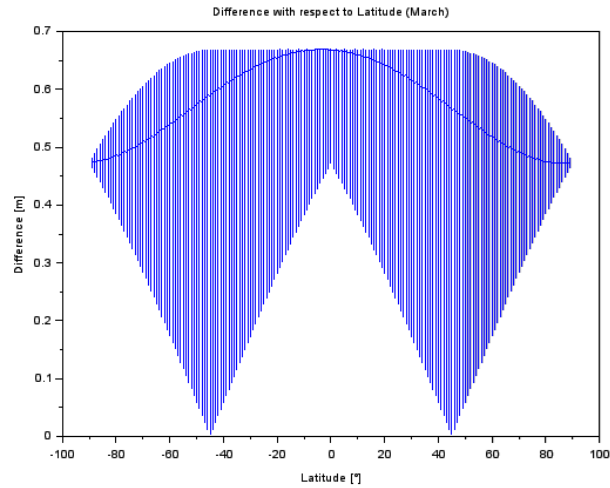


Figure 1: Absolute difference in position (EF to GM2000, March)

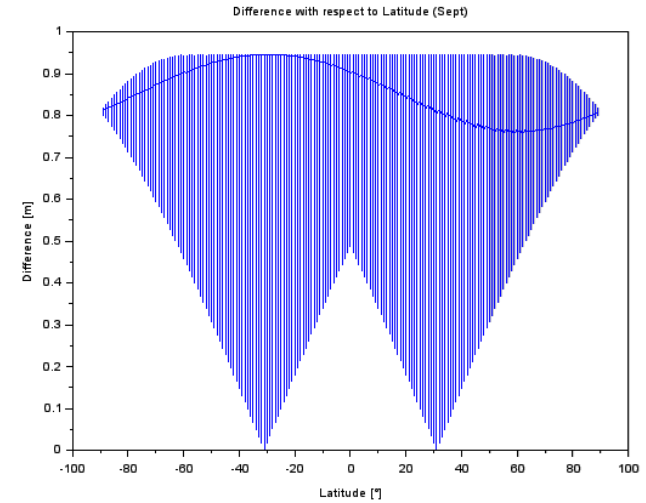


Figure 3: Absolute difference in position (EF to GM2000, September)

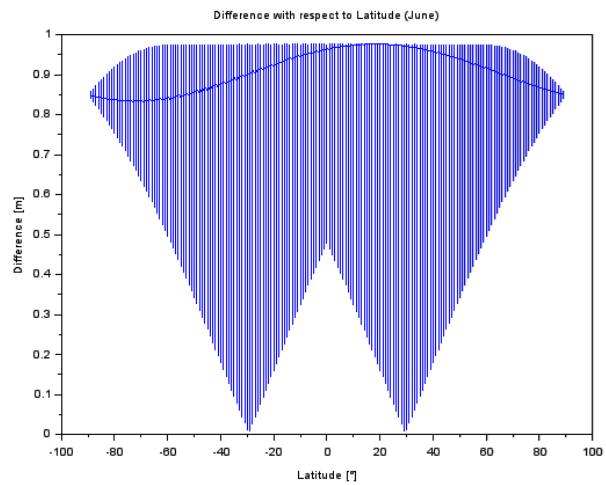


Figure 2: Absolute difference in position (EF to GM2000, June)

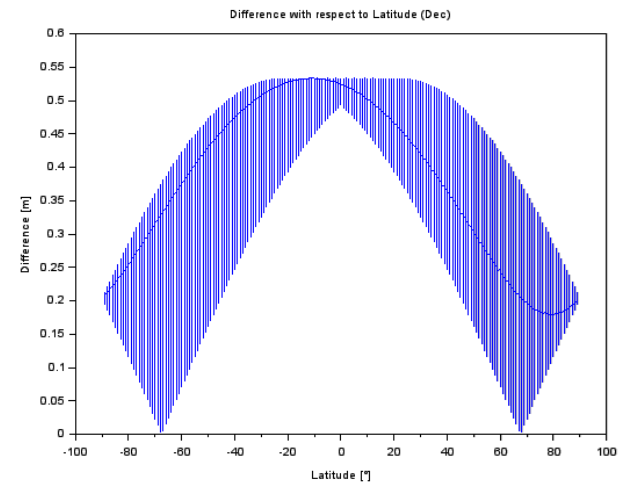


Figure 4: Absolute difference in position (EF to GM2000, December)

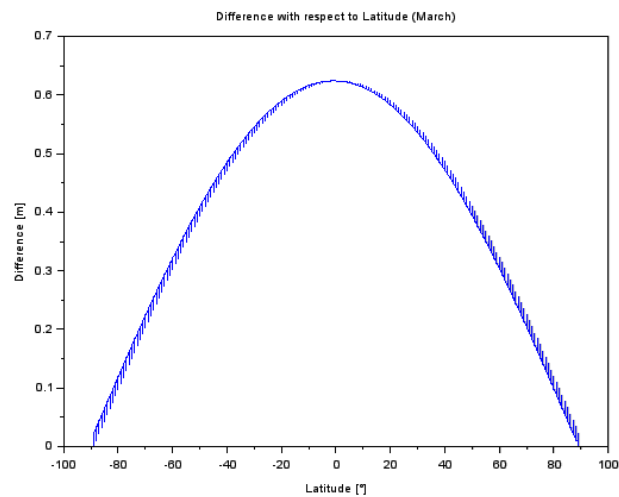


Figure 5: Absolute difference in position (EF to ToD, March)

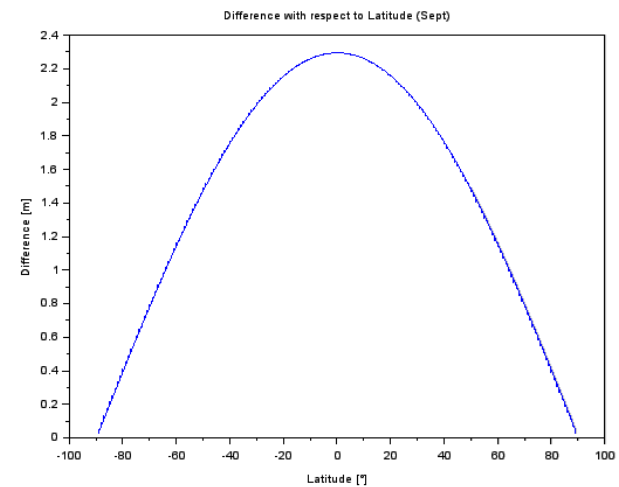


Figure 7: Absolute difference in position (EF to ToD, September)

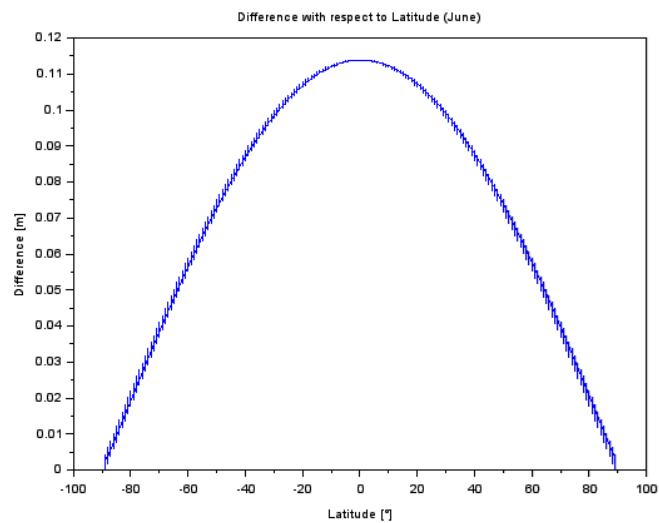


Figure 6: Absolute difference in position (EF to ToD, June)

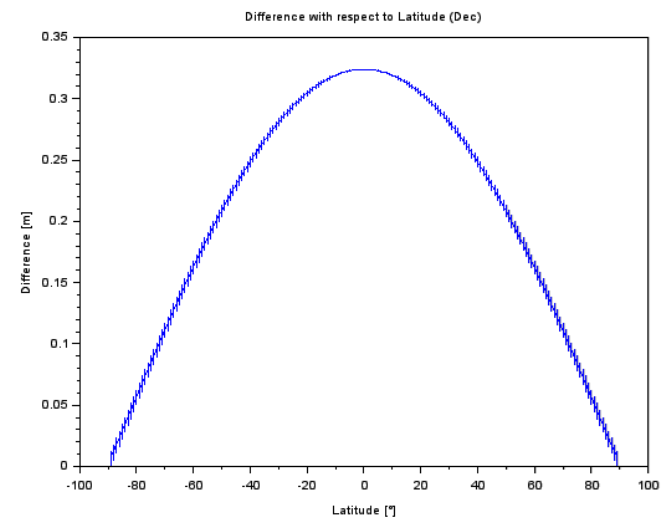


Figure 8: Absolute difference in position (EF to ToD, December)

4.2 Planet Ephemerises (sun, moon and others).

4.2.1 Data generation

EOCFI implements a specific algorithm to generate planetary ephemerises (van Flandern, T. C.; Pulkkinen, K. F., “Low-precision formulae for planetary positions “) and quotes directly its precision which is 1 arcminute (0.017 deg).

The script [GenPlanetEphemJ2000](#) has been created in [explorer_lib_run.c.c](#) to generate planetary ephemerises. The process goes as follows:

- Based on script inputs a planet is selected, either Sun's, Moon's or Jupiter (arbitrary choice for testing purposes).
- At an arbitrary date 1999-04-30T21:36:00.000 TAI (hard coded in routine) a single position is generated, either by calling [xl_sun](#), [xl_moon](#) or [xl_planet](#) with relevant inputs.
- This position is then written into [Sun_pos_valid_data.csv](#) (or [Moon_pos_valid_data.csv](#) or [Jup_pos_valid_data.csv](#)).
- The file is then read by [processBody](#).
- The same position is then computed by Orekit and compared to EOCFI.

Side Note:

Discrepancies are to be expected here, Orekit position is based on interpolation of JPL Ephemerides (DE430). Those ephemerides are part of Orekit context data which can be tailored. To better represent an actual ASGARD use case, the same context as the one present in ASGARD is used here.

4.2.2 Results

Comparisons are compounded in the following table:

Sun	0.013581950335946402 (deg)
	0.8149170201404857 (arcmin)
Moon	0.012193708924065344 (deg)
	0.7316225354292882 (arcmin)
Jupiter	0.01058393132625334 (deg)
	0.6350358795624996 (arcmin)

Table 3: Difference in angle for Planetary bodies.

4.3 *xl_geod_distance*

4.3.1 *Data generation*

According to [REF01], the reference taken for EOCFI precision evaluation on the geodetic distance is the GeographicLib library (<https://geographiclib.sourceforge.io/index.html>). The library is used to generate a pair of points, P1 and P2, in such a way that:

- given a specific azimuth/direction
- and a distance relative to P1

P2 is generated. It so happens that this library is available in several programming languages and particularly in Java & Python. Those languages are of interest as:

- Java is Orekit's core language.
- Python is Asgard's core language.

Hence, comparison with respect to either of the two can be implemented easily. The routine *genGeodDist-GGL* reproduces the step mentioned hereabove taking the same range of values of [REF01] 2.4. The results are written in **Run_GEOGRAPHIC_LIB.csv**. This file is then used as inputs (geographic coordinates of P1 and P2 points) for both **EOCFI** and **ASGARD**.

On EOCFI's side, the routine *GenGeodDistEOCFI* parses the aforementioned file, and evaluates the distance between P1 and P2 through *xl_geod_distance*. Results are then saved in **Run_EOCFI_xl_geod_dist.csv** (along with GeographicLib's inputs relating to P1 and P2).

The same is done in ASGARD through the routine *test_GeodDist*, it also computes the difference between EOCFI's run and ASGARD's. Results are saved in **Run_Asgard_xl_geod_dist.csv**. This last file is then fed into Scilab script **Exploit_xl_geod_dist.sce**.

4.3.2 Results

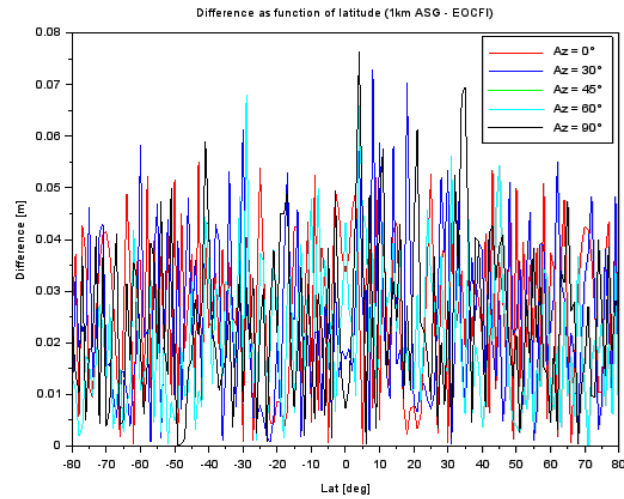


Figure 9: Geod distance ASGARD vs EOCFI (1km)

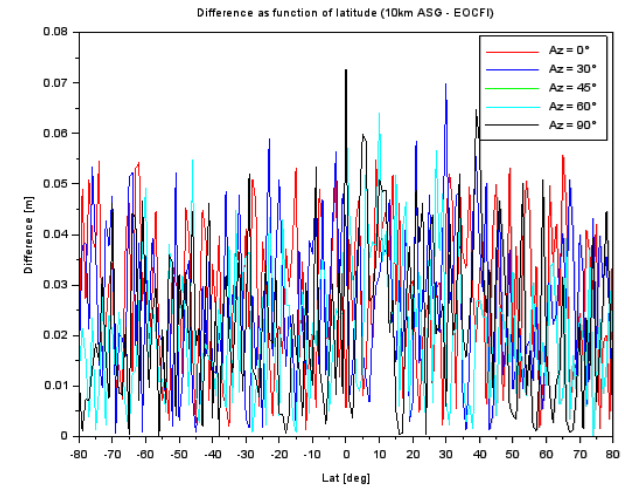


Figure 11: Geod distance ASGARD vs EOCFI (10km)

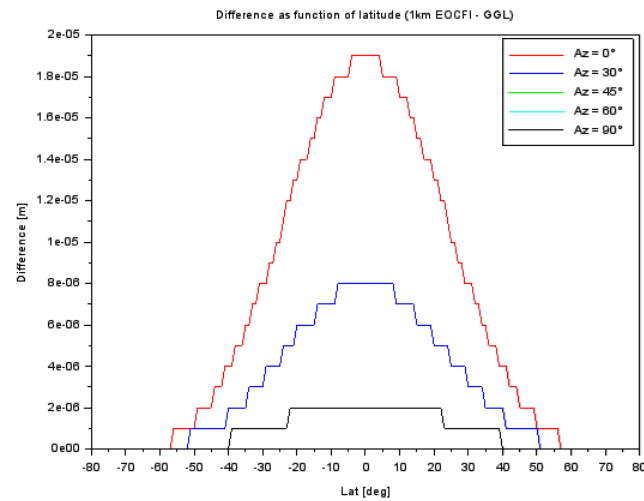


Figure 10: Geod distance EOCFI vs GeographicLib (1km)

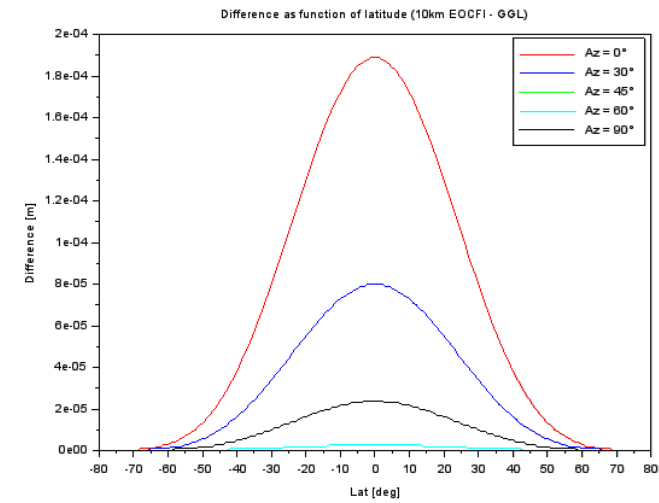


Figure 12: Geod distance EOCFI vs GeographicLib (10km)

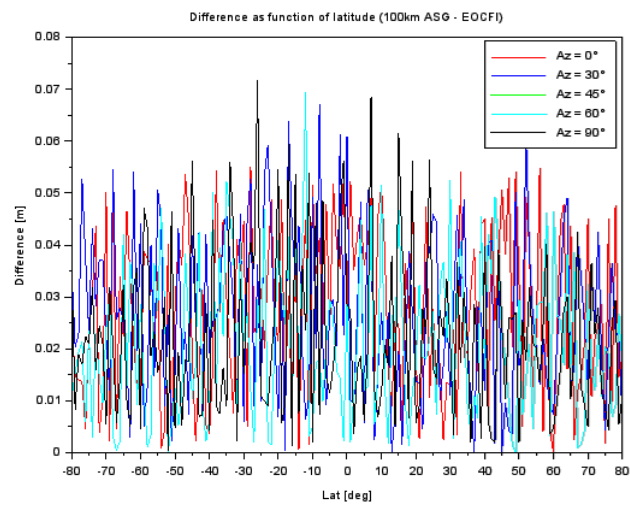


Figure 13: Geod distance ASGARD vs EOCFI (100km)

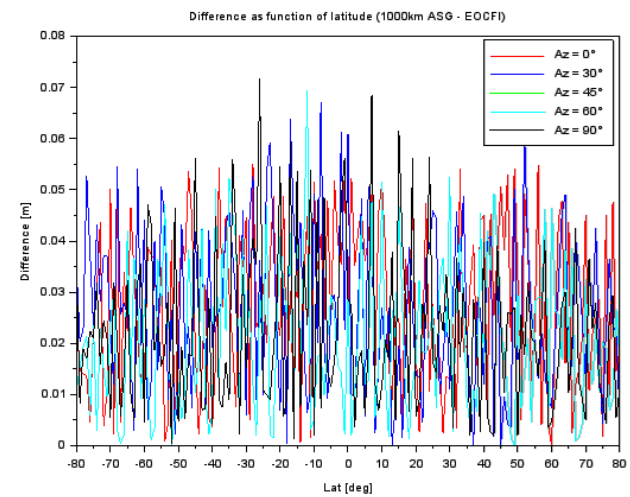


Figure 15: Geod distance ASGARD vs EOCFI (1000km)

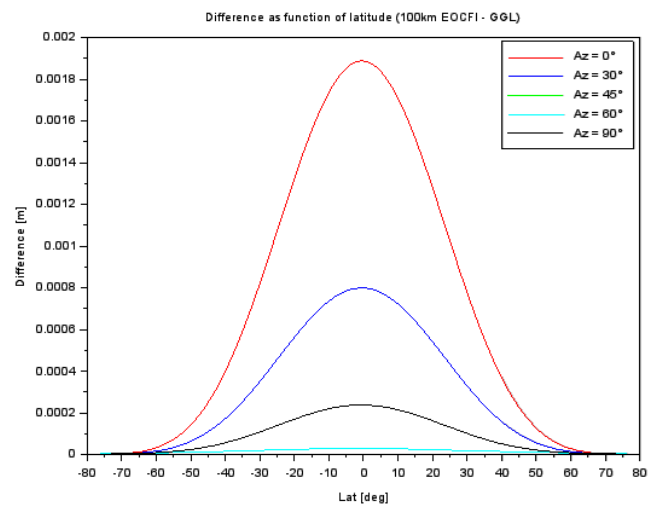


Figure 14: Geod distance EOCFI vs GeographicLib (100km)

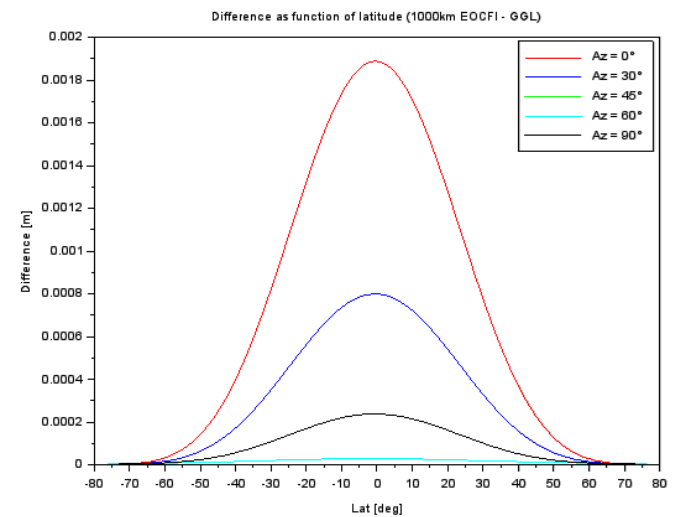


Figure 16: Geod distance EOCFI vs GeographicLib (1000km)

Side Note:

If ASGARD precision is deemed not sufficient, considering that the Geographic library is available in Python, it can be used as an “off the shelf” component and integrated into ASGARD. However, one must also consider the legal/licensing aspect of such an integration. This is not the focus of this study and thus no comment is offered on that point.

4.4 *xp_target_inter*

4.4.1 *Important Considerations/Notes*

During testing, it has appeared that heavy limitations are present in generating a data set similar to what is presented in [REF02] 2.2.1. In the aforementioned section, several configurations have been tested. Notably, the satellite’s Line of Sight has been tilted to Nadir, 27.5deg off-Nadir and 55deg off-Nadir positions.

The details of this tilt LoS are not present in [REF02]. The shift might have been applied along the satellite’s motion, to observe ahead of the subsatellite’s point (Nadir) or trailing behind. The axis around which this shift has been applied is also not clear. It might have been applied perpendicular to the satellite’s motion, observing leftwards or rightwards. In the following discussions, it is assumed that this tilt has been applied forwards along the satellite’s motion.

In addition, the 2 datasets compared in [REF02] 2.2.1 differ in the manner with which the attitude has been represented:

- In one case: the attitude is directly injected via a quaternion file. There is no mention on whether 1 file per configuration exists or if only 1 file (assuming pure Nadir pointing) has been postprocessed to include the desired tilts.
- In the other case: the attitude is the results of the implementation of the “Ideal Yaw Steering Mode” attitude law. Again, no details on the implementation of a Nadir shift are presented.

Side Note:

Upon further EOCCI’s code inspection, it appears that an angle shift can be applied directly at *xp_target_inter*’s call. However, as stated above, it is not clear what combination of elevation and azimuth parameters corresponds to whichever tilt is desired.

Regardless, ASGARD equivalent function, *direct_loc*, does not have such parameters (elevation and azimuth).

The ability to configure/manipulate the LoS, whether it be the product of a direct input (attitude/ quaternion file) or a simulated law (Ideal YSM), is central to generating a similar dataset that is relevant to any comparison with respect to the results presented in [REF02].

Several difficulties are present to generate a similar dataset:

- ASGARD processing is still in active development. As such functions available are limited and have been prioritized depending on usage scenarios. It is not clear at this point whether the processing chain nominal use case are designed to produce data without relying on NAV_ATT files (satellite’s pointing). Hence, an equivalent of a YSM is in development yet (linked to 3rd point hereunder).
- ASGARD in its current state does NOT easily allow manipulations of the satellite’s attitude. However, several tools are present to apply shifts and tilts for testing purposes but only for low level functions.
- The logic implemented to compute a LoS intersection differs with its EOCCI’s counterpart. In ASGARD, this intersection resulting from the LoS is tied to the sensor’s modelisation. The implementation of YSM (and the tools needed to alter the resulting pointing) cannot be paired with the current state of the sensor’s model.

In addition to the points mentioned above, ASGARD's equivalent ideal Yaw Steering Mode is implemented differently with respect to EOCFI. This is due to the underlying Orekit's library and addressed here.

Firstly, to avoid any confusion, the term "Yaw Steering" can be ambiguous depending on which context it is found. Orekit's equivalent pointing is referred as Yaw Compensation (<https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/attitudes/YawCompensation.html>). Yaw Steering in Orekit, refers to a specific pointing with constraints linked to the Sun's angle relative to the satellite's orbit (<https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/attitudes/YawSteering.html>). Its usage is not related to a payload/mission/sensor issue but is related to a way of providing optimum lighting conditions to a solar array (platform issue, not mission).

Based on the description of EOCFI's ideal YSM ([REF07]), ASGARD's implementation combines 2 pre-existing and fully validated attitude laws within Orekit:

- the Nadir Pointing attitude law
- the Yaw Compensation law. This particular law is built on top of the Nadir Pointing law and cannot be implemented "on its own".

The difference lies in the way that Orekit's represent a Nadir Pointing attitude. Its implementation goes as follows:

- the Z axis (which is assumed to be the axis on which instruments are mounted), is pointed towards the Earth (Nadir point, not geocentric). Whereas in EOCFI's this axis is pointed towards the opposite (Zenith).
- the X axis is colinear and has the same direction as the satellite's velocity in Inertial Frame (as opposed to the Earth Fixed frame, or EF/ECEF in EOCFI's implementation).
- Y completes the right-hand coordinates system.

This can be summarized in the following figure:

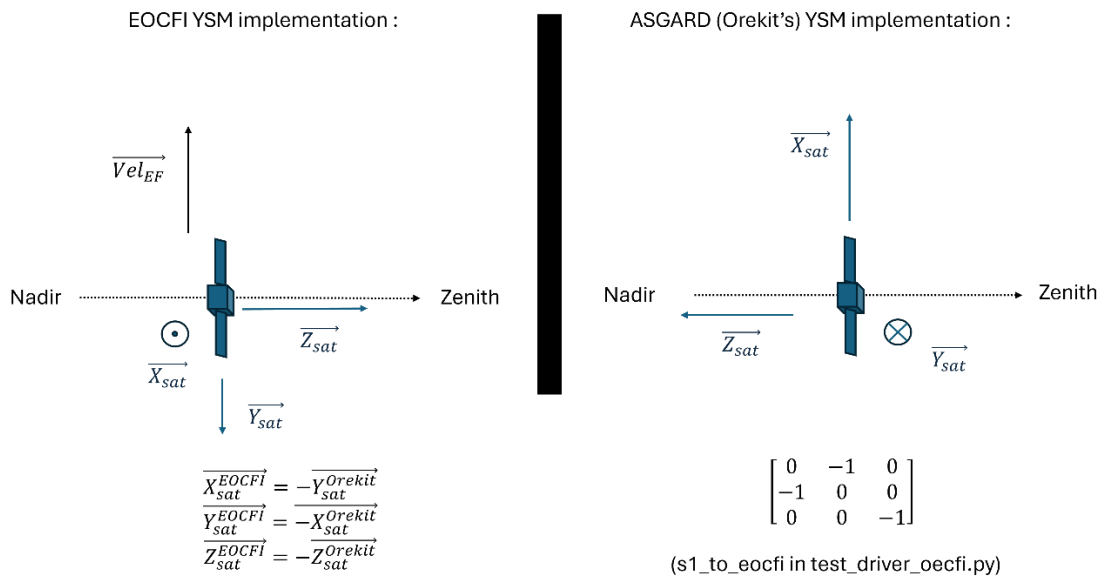


Figure 17: YSM implementation EOCFI vs Orekit/ASGARD

4.4.2 Data generation

4.4.2.1 Replication of [REF02] test case.

Considering that the comparison logic adopted in [REF02] that is:

- Generating a point on ground resulting from the LoS originating from a file (assumed to be a NAV_ATT file).
- Generating a point on ground resulting from a LoS originating from the ideal YSM pointing modelisation.
- Comparison between the two

and the technical limitations mentioned in 4.4.1, there is no straightforward way to manipulate the attitude during a call to the equivalent to `xl_target_inter` (which is for ASGARD the `direct_loc` routine linked to the sensor model).

Hence the only way to represent the same observation is by generating the desired attitude instead of simply relying on an external source. This is the purpose of `GenOrbitAndQuatFor_XP_TARGET_INTER` (external Orekit mock-up tool). This script generates quaternions implementing the equivalent to the Ideal YSM (Nadir Pointing + Yaw Compensation) attitude law plus the same offset as [REF02]. Those quaternions are then directly injected in ASGARD (via `test_xp_target_inter`, and specifically `launchLocFiles`).

The equivalent is done in `launchLocYSM`, but no quaternions are injected, they are directly computed within ASGARD.

The differences between the 2 methods are presented in section 4.4.3

The main limitation is that, even by tinkering with ASGARD models and forcing the `direct_loc` to take into account a special attitude, the origin of this attitude is the same (Orekit). The differences are expected to be low. Comparing the two is of limited value and only brings confidence in the correct implementation of Orekit's equivalent to YSM in ASGARD.

Another approach can be taken here which aims to compare only the 2 different implementations of the intersection algorithm (`direct_loc` for ASGARD and `xl_target_inter` for EOCCI). This is the approach discussed in section 4.4.2.2 and results are presented in section 4.4.4).

4.4.2.2 Focus on intersection.

This section presents an alternative method that focuses on comparing the equivalent of (`xl_target_inter`) within ASGARD (`direct_loc`). This algorithm has been isolated from the sensors modelisation. The only parameters being:

- The satellite's position from which the intersection is computed.
- The LoS to be taken into account.
- The shape on which the LoS will be projected upon.

The aim is to have a set of data (data listed above in addition to the position of intersection point) generated by EOCCI, and to feed this set of data to the isolated `direct_loc` algorithm and compare the resulting intersection position.

The routine `CustomIntersection` generates the data set following the steps below:

- Taking an arbitrary position. To present the data in the same manner as in [REF02], these positions are generated by transforming geodetic positions into cartesian. The Longitude being set to 0, the Latitude varying across all possible values, the altitude arbitrarily set to 700km.
- The spacecraft attitude is instantiated to represent an Ideal YSM attitude law. This is done by making use of `xp_sat_nominal_att_init`.

- Then *xp_target_inter* is run as well as *xp_target_extra_vector* in order to retrieve the intersection position as well as the LoS vector. The different LoS values (Nadir, 27.5° offset, 55° offset) is directly injected here.

This data set is then injected directly into ComparisonTools (via *EOCFI_Data_xp_target_inter.csv* file) and the isolated algorithm is run. The resulting intersection position is then compared to EOCFI's and the difference are outputted to *diff_EOCFI_RUGGED.csv* file.

The results are presented in section 4.4.4

4.4.3 Results (focus on YSM's implementation)

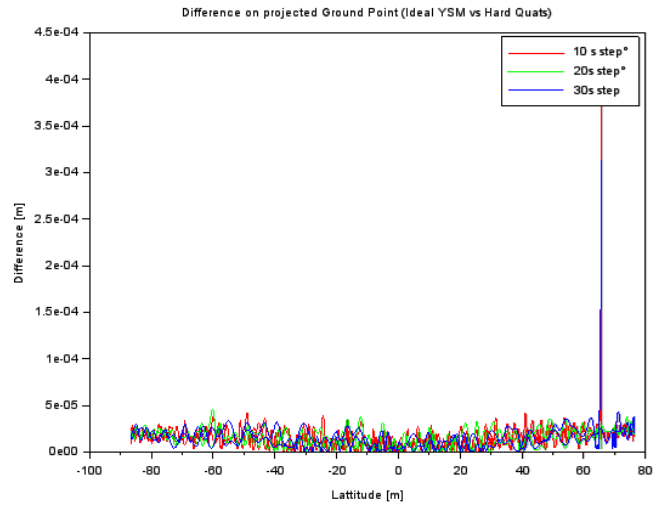


Figure 18: Difference on projected point (Nadir)

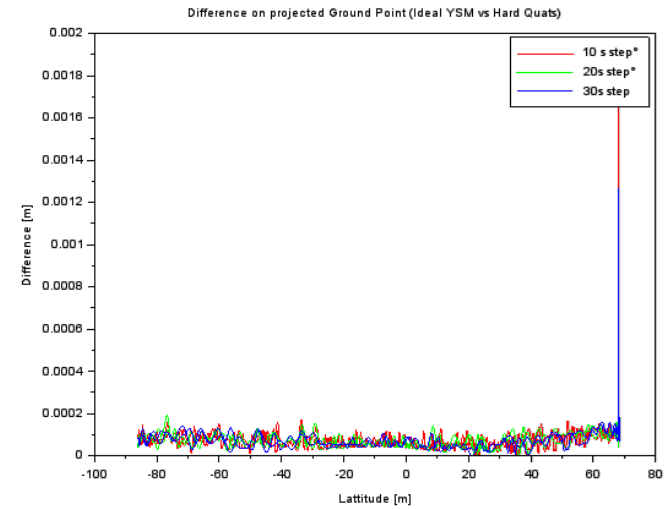


Figure 20: Difference on projected point (55° offset)

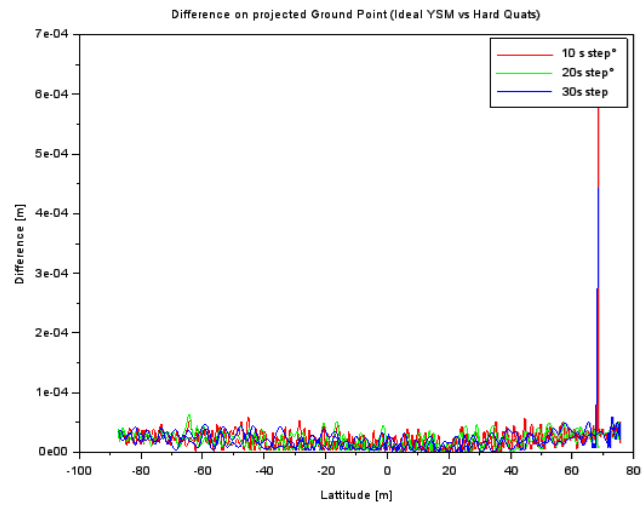


Figure 19: Difference on projected point (27.5° offset)

4.4.4 Results (focus on intersection algorithm)

Hereunder all results are compounded in a single plot presented hereunder:

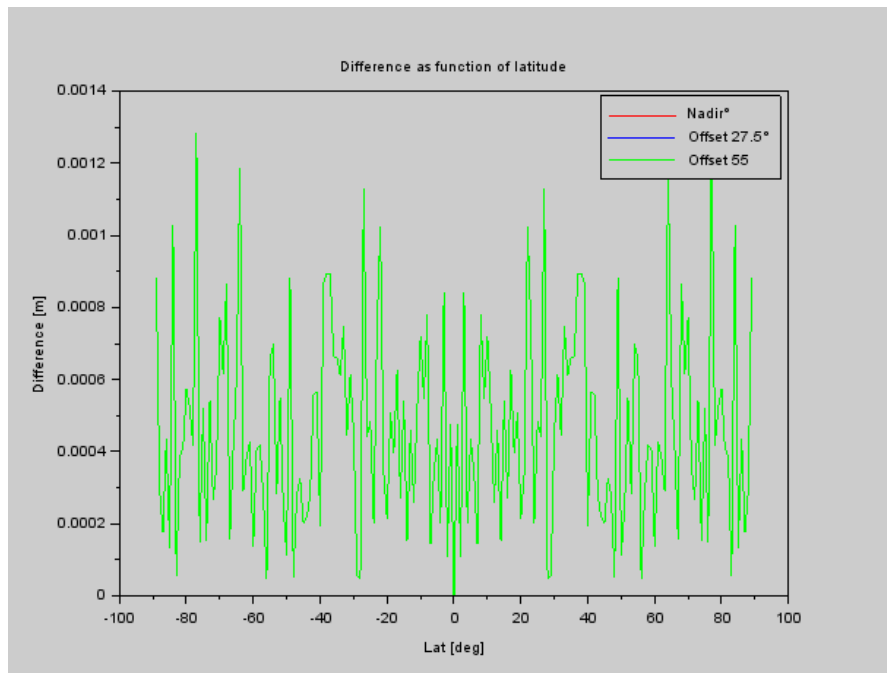


Figure 21: Difference in meter between EOCFI's and ASGARD intersection algorithms.

The offset with respect to Nadir does not seem to generate any variation. Upon further inspection it seems like the dedicated EOCFI's script to generate data [CustomIntersection](#) is at fault here (investigation pending). However, the LoS used within EOCFI is the same as the one passed on to Rugged. The result is still comparable.

4.5 xo_osv_compute (interpolation)

4.5.1 Data generation

The methodology to generate test data is the same as presented in [REF01]. A POD file, with initial step of 10s, have been subsampled to 30 and 60s. The positions resulting from ASGARD interpolation have then been directly compared to the reference (POD initial data). No special treatment was required here. Data generation can be reviewed in *test_Interpolator* routine.

4.5.2 Results

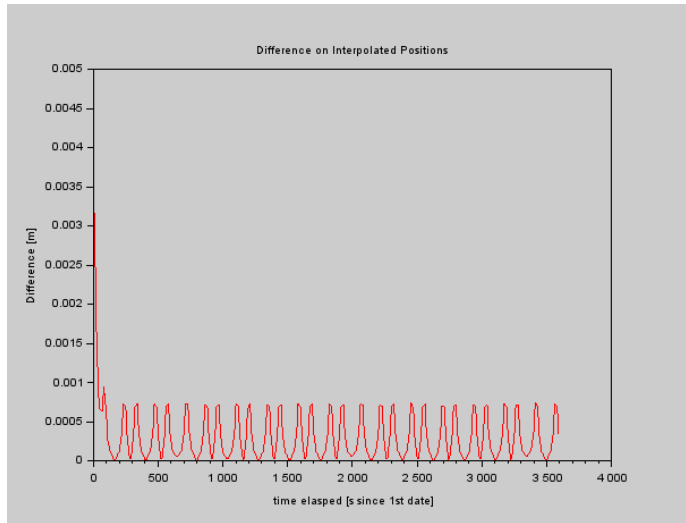


Figure 22: Difference on Interpolated positions vs Reference (30s step).

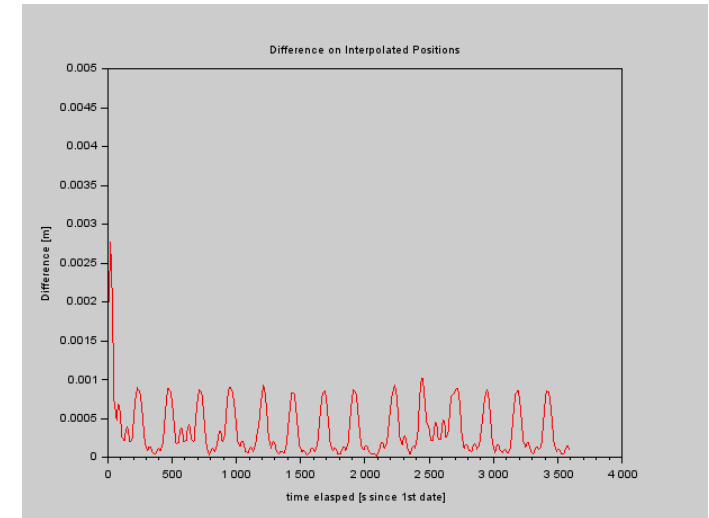


Figure 24: Difference on Interpolated positions vs Reference (60s step).

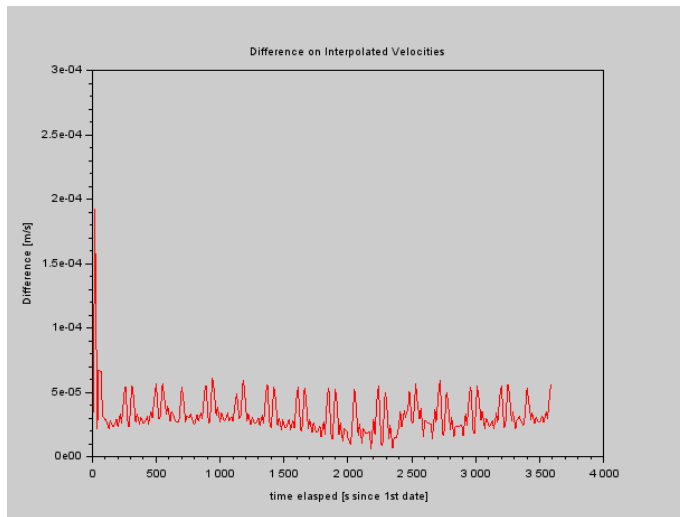


Figure 23: Difference on Interpolated velocities vs Reference (30s step).

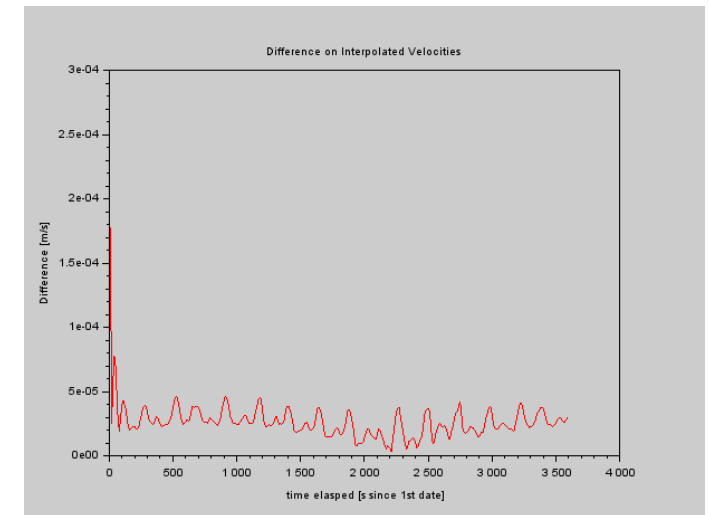


Figure 25: Difference on Interpolated velocities vs Reference (60s step).

	Position (30s, m)	Velocities (30s, m)	Position (60s, m/s)	Velocities (60s, m/s)
Max	0.0031629	0.0001926	0.0027641	0.0001772
Mean	0.0003165	0.0000325	0.0003724	0.0000262
RMS	0.0004645	0.0000365	0.0005111	0.0000293

Table 4 : Statistics of Interpolation

The fact that 60s values have better accuracy is unexpected here. Upon a first, and admittedly coarse check due to time constraints, no error (swaps between 30 and 60s values) has been detected at this point. This might warrant further investigation.

4.6 *xo_osv_compute (Single, Double and Precise mode)*

As mentioned in 4.4.1, ASgard in its current state, is not able to perform several. At the moment the priority has been set on the full implementation of computation relying on the user providing NAV_ATT files.

In particular, it does not allow at the moment, any propagation to be performed. As of yet, only interpolation from NAV_ATT file is fully implemented. Hence no data can be generated for comparison against EOCCI's propagations.

However, ASgard being built upon Orekit, which comes with several propagators and associated models. If deemed necessary, a propagator based on Orekit features (propagator and associated perturbation effects models) can be "put together" to fulfil this feature.

As [REF01] is only a comparison document, and not a specification/ detailed implementation model, the information presented can only hint at the effects / forces (and their associated models) taken into account in EOCCI's propagator. This limits the ability to select within Orekit a similar set of models and associated parameters to approach as much as possible EOCCI's features.

From [REF01] it is hinted that propagation takes into account several effects:

- Atmospheric drag, although as mentioned no information is present on which model is implemented (Isotropic drag, specific implementation, associated atmospheric model, etc...).
- Drag effects can be influenced by space weather effect (Solar activity for example). Space weather is mentioned in [REF01].
- The Earth attractive potential seemed to be based on a harmonics model (Tesseral and Zonal effects are mentioned).

Considering the lack of information available, several implementations have been tested. The results (positions) have been compared to a POD file in order to select the closest propagator (and associated models).

Several implementations have been attempted, mostly based on ease of implementation:

- Simple Keplerian Propagator, literally the simplest propagation model possible.
- Eckstein-Hechler Propagator (<https://www.orekit.org/site-orekit-11.3.2/architecture/propagation.html>, <https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/propagation/analytical/EcksteinHechlerPropagator.html>). This model is already partially used in ASgard (albeit not for proper propagation as mentioned previously). Adaptations from this propagation model should be minimal.
- Draper Semianalytical Satellite Theory propagator (referred as DSST, <https://www.orekit.org/site-orekit-11.3.2/architecture/propagation.html>, <https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/propagation/semianalytical/dsst/DSSTPropagator.html>). Mainly as the implementation of atmospheric drag and harmonic attractive potential is easy. For this particular propagation, additional perturbation forces have been attempted (addition of Sun's and Moon's attractive forces). However, results were degraded, thus those perturbation have been dropped.
- Finally, a full NumericalPropagator (<https://www.orekit.org/site-orekit-11.3.2/architecture/propagation.html>, <https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/propagation/numerical/NumericalPropagator.html>) has been tested.

It is no surprise that the closest results compared to POD data has been observed with the use of the Numerical Propagator. In the interest of brevity, only the results given by best matching model are presented here. However, data needed to generate differences with respect to other models can be reconstructed (see additional files).

The models associated with the propagator are the following:

- A HolmesFeatherstoneAttractionModel for Earth's Potential (<https://www.orekit.org/site-orekit-11.3.2/architecture/forces.html>, and <https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/forces/gravity/HolmesFeatherstoneAttractionModel.html>). More details to follow.
- For drag modelisation, the IsotropicDrag model (<https://www.orekit.org/site-orekit-11.3.2/architecture/forces.html> and <https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/forces/drag/IsotropicDrag.html>) has been selected. More details to follow.
- Atmosphere is modeled by the NRLMSISE00 model (<https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/models/earth/atmosphere/NRLMSISE00.html>).

The instantiation of those model requires several other data. In particular:

- the Earth Potential model is based on the EIGEN-6S model which require/contains numerous coefficients to be instantiated. Those coefficients have been added to the ASGARD context data folder (**eigen-6s.gfc** file).
- The Atmospheric model require Space Weather data. Normally, in an operation context, prediction of space weather data (solar activity) is consumed daily (information provided by the NOAA typically) and influences the orbit determination process and can also be used to tune propagation models. Orekit's provide generic data for simulation purposes. This same data has been added to ASGARD context folder (**SpaceWeather-All-v1.2.txt** file).

The number of coefficients to take into account for Earth Potential, along with the parameters linked to the drag model have been tuned as follows:

- Concerning the coefficients of the Earth harmonics Potential, all possible combinations have been tested up to an order of 20. The combination leading to minimal discrepancies has been retained (17 for Tesseral, 15 for Zonal).
- Concerning atmospheric drag: this model is as simple as can be. Since no details were available to properly model the individual surfaces inducing drag, a simple Isotropic Drag model has been selected (<https://www.orekit.org/site-orekit-11.3.2/apidocs/org/orekit/forces/drag/IsotropicDrag.html>). This is akin to a sphere associated to a drag coefficient. This modelisation present mainly 2 benefits:
 - It is the simplest in terms of implementation, only 2 parameters are needed to model drag.
 - As it represents a sphere, the cross section is constant and is independent from the attitude during propagation (simplifying further implementation).

As briefly mentioned, the inputs are:

- The cross section generating drag. This cross section has been evaluated (based on Sentinel's 3 dimensions) to an initial surface of 7m². Several variations on this value were tested. The value resulting in the least deviation is 1m².
The associated drag coefficient. Again, several values (ranging from 0, hence no drag at all, to 3) have been tested (value retained 0.39).

The atmospheric density needed to compute the actual drag force is directly derived from the Atmospheric model. This is influenced by with space weather data.

Side Note:

The value minimized is the final position present in the POD file. Another method could have been to minimize the RMS instead of last value (process resembling what is done during typical orbit determination). However, a single value has been used, as it seemed more akin to what is done for the Double Mode (information on a single point used to "calibrate" propagation). The last value has been chosen as it appeared that it would take into account all effects, cumulated over the entirety of the propagation.

The difference in position during the whole POD file can be observed in the following plot.

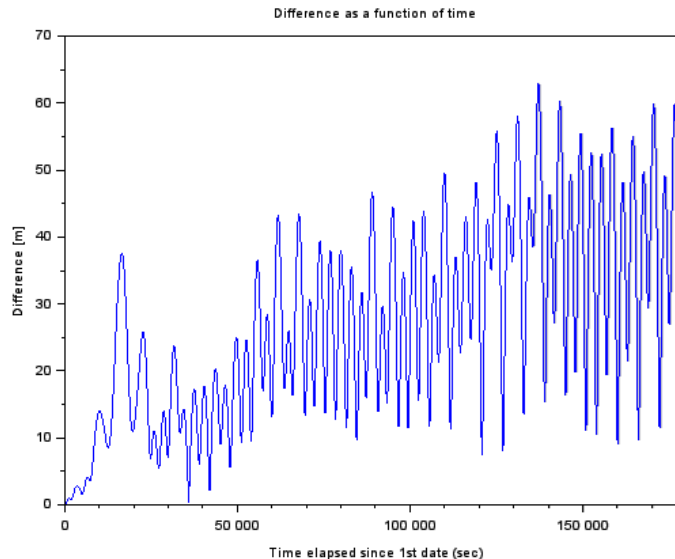


Figure 26: POD position vs reconstructed numerical propagation

It is to be noted here that this propagation model has been initialised precisely at the reference point (no error on 1st step). Any error the initial position (and velocity) will naturally be carried along the propagation process. Here only the resulting discrepancies due to the dynamics during propagation are observed. Errors on the initial point depend on the orbit model initialization and is discussed in 4.7.

These results can be compared to what is done in [REF01] on Single and Double Mode.

- Single Mode is similar in the fact that the Numerical Propagator is initialized only using the first OSV provided in POD file (once it has been tuned).
- Double Mode initialization is similar to the tuning process described. In which information is extracted from another OSV (here at the end of propagation) to minimize the position difference.

Concerning the Precise Mode, it seems that this propagation model is the most complex/tuned implemented in EOCFI. As the proposed propagation model in this section has been built purely in a speculative manner (based on hints given by [REF01]), efforts have been focused on comparing it with respect to the POD file (which is in the order of a couple of meters off with respect to EOCFI's Precise Propagator according to [REF01]).

A more precise model would be possible; however, it would also require detailed information on which model are used within EOCFI for a meaningful comparison.

4.7 *xo_osv_compute (OSF initialisation)*

4.7.1 Data generation

Since there is, at this stage, no propagation capabilities within ASGAR, the methods used to initialize a propagation model are naturally not implemented. However, an OSF based initialization process is demonstrated in the provided Orekit based mock-up.

Judging from explanations given in [REF01], it appears that OSF purpose is to set the 1st point to be propagated in the event that the POD is not available. The fact that results in [REF01] are adjusted (difference in time injected in the propagated OSVs) based on POD data appears contradictory. If POD are indeed available, it seems like an initialisation directly from the 1st POD would yield more accurate results.

Regardless, the approach considered here is twofold:

- Try to make an “educated guess” on the satellite’s 1st position derived solely from information contained in the OSF. Hence a “coarse” orbit/OSV is evaluated. This is consistent with the assumption that no POD is available.
- To approach what is done in essence in [REF01], this coarse orbit can be adjusted with additional, but limited, information coming from POD data.

The current implementation is represented in the following diagram

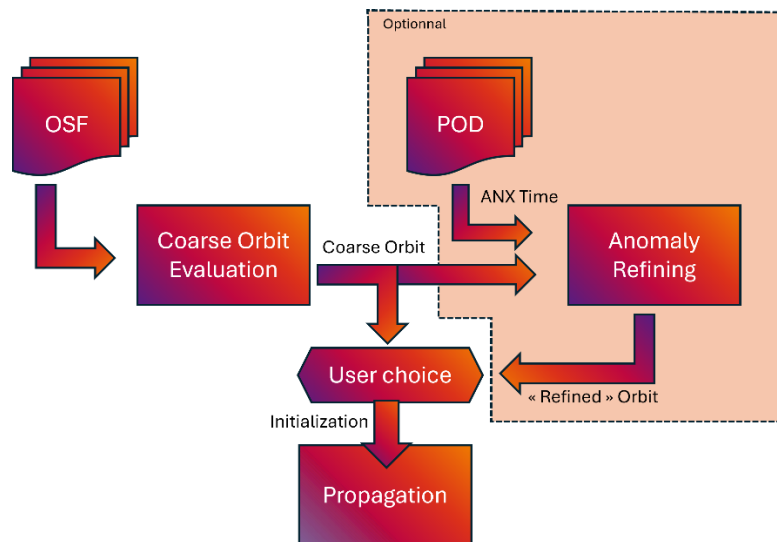


Figure 27: OSF Initialization process

Based on information solely contained in OSF, the orbit’s cycle information can be derived to approximate the period, and hence the orbit semi-major axis. Then, 2 major assumptions are made:

- The orbit is near circular (low eccentricity, here the value taken is 1e-3). Hence the eccentricity is arbitrary assigned.
- The orbit is considered Sun-Synchronous. This, along with the eccentricity and derived semi-major axis, allows for an approximation of the orbit’s inclination.

The other parameters remaining for a full Keplerian orbit model are:

- The perigee’s argument (pa) Since the orbit is near circular, this is not well defined for values close to 0. However, the value taken for eccentricity does not cause issues. It is arbitrarily fixed to 0.
- The Right Ascension of Ascending Node (raan). This is derived from the Mean Local Solar Time (information part of the OSF). It also is adjusted in order for the satellite’s projected point on ground to match the ANX longitude value present in OSF.
- The satellite’s Anomaly which is akin to its angular position on its orbit (referred as Position On Orbit or POO from now on). At this stage it is arbitrarily set to 0. This, along with null pa, geometrically places the satellite at its ANX point along its orbit.

Based on user's choice the Anomaly can be further refined. However here is where the approach differs from what is presented in [REF01]. Instead of adjusting the dates based on the average time difference at ANX passages, here only the actual ANX passage (the closest to the considered OSF dataset) is taken into account. The anomaly/POO is then iteratively adjusted to match the ANX passage date interpolated from POD.

Hence a single initialization point/state is generated. This can be compared to EOCFI's initialization. The EOCFI's data has been generated based on the same POD and OSF (POD selected around the OSF last dataset) and several OSV has been computed. However, as mentioned in 4.6, all points other than the 1st are the product of the propagation model, and thus the dynamics, used in EOCFI. Only the 1st point is of significance for initialization.

The results, and their associated details are the subject of the following section.

4.7.2 Results

The results presented in this section are both of qualitative (images originating from simulations) and quantitative nature (hard difference in position figures). Images based on positions computed either by ASGARD/Orekit mock-up and EOCFI have been used in 3D simulations in order to give a macroscopic sense of the quality of the initialization process.

The EOCFI's results are the product of the [genOrbFromOSF_S3](#) script which in turn:

- Initializes the orbit via [xo_orbit_id_init](#)
- Propagates the orbit via [xo_osv_compute_run](#)

However, it has come to attention that the orbit generated by EOCFI seem to degenerate over time as shown hereunder. The image presented here represents the reference orbit (pure POD data) and the states computes by EOCFI at different stage of propagation roughly at start, 1/3, half and toward end of simulation time spanning the whole duration of POD data, which is approximately 2 days.

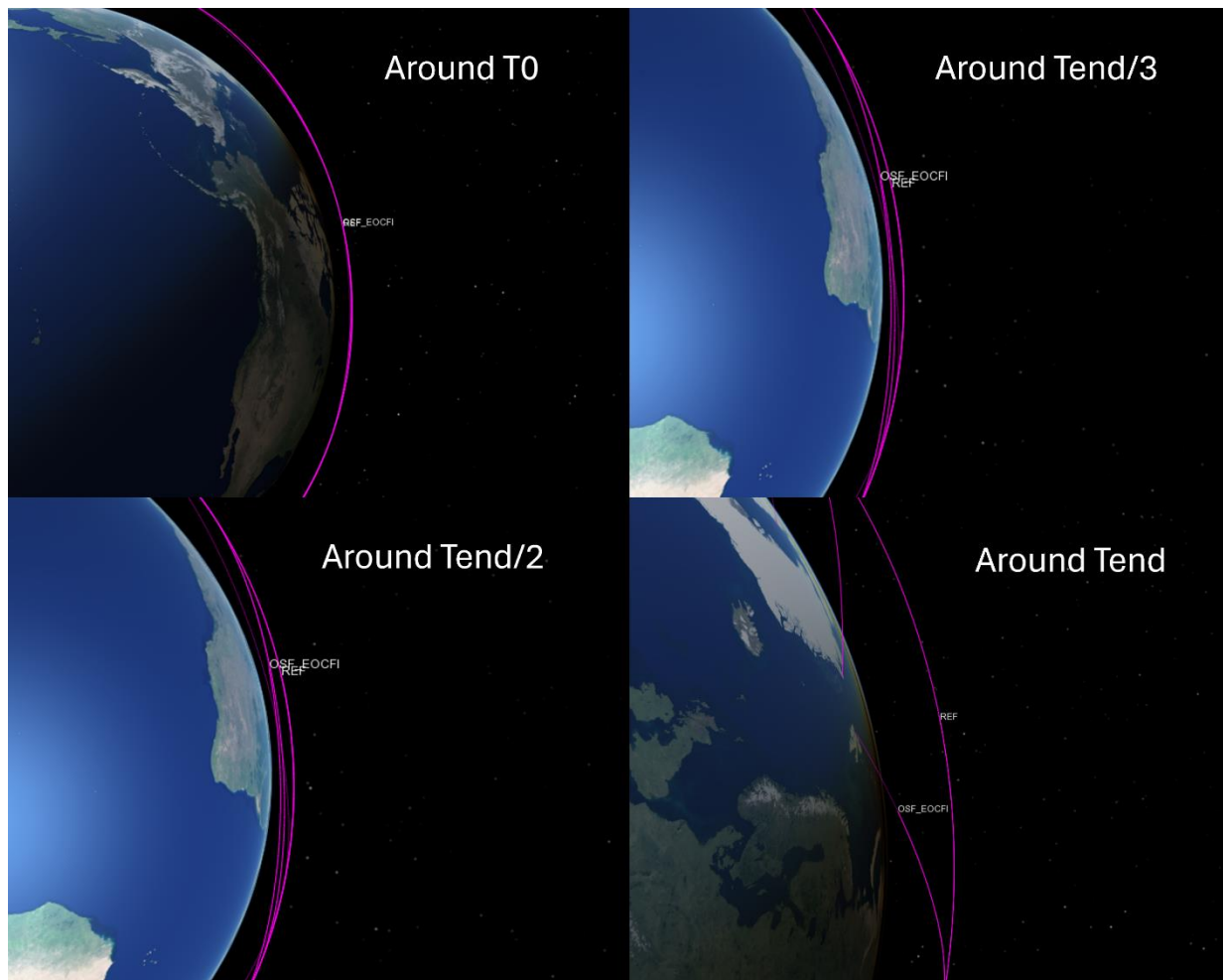


Figure 28: EOCFI's degenerating orbit (from OSF initialization)

As previously stated, the data used by EOCFI has been computed using [xo_orbit_id_init](#) and particularly setting the parameter `orbit_file_mode` to `XO_ORBIT_INIT_OSF_MODE`. Upon further investigation of the provided EOCFI validation and example files (searching for `XO_ORBIT_INIT_OSF_MODE`) it appears that this mode is never used to trigger a propagation. This seems to indicate that this method of initialization is representative of a degraded mode and limited to short term orbit representation.

Independently from the actual dynamics/propagation, the EOCFI's initialization also appear to be further away from POD Data than what is presented in [REF01].

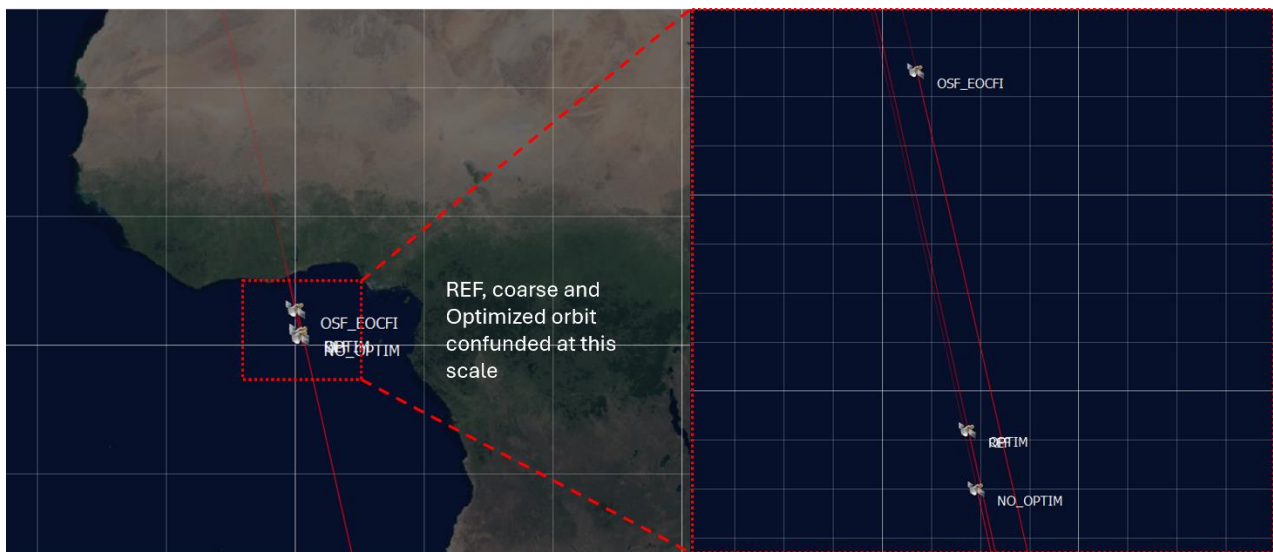


Figure 29: 2D projection on ground of EOCFI vs Orekit OSF initialization.

This is also observed from a full 3D view in our simulations.

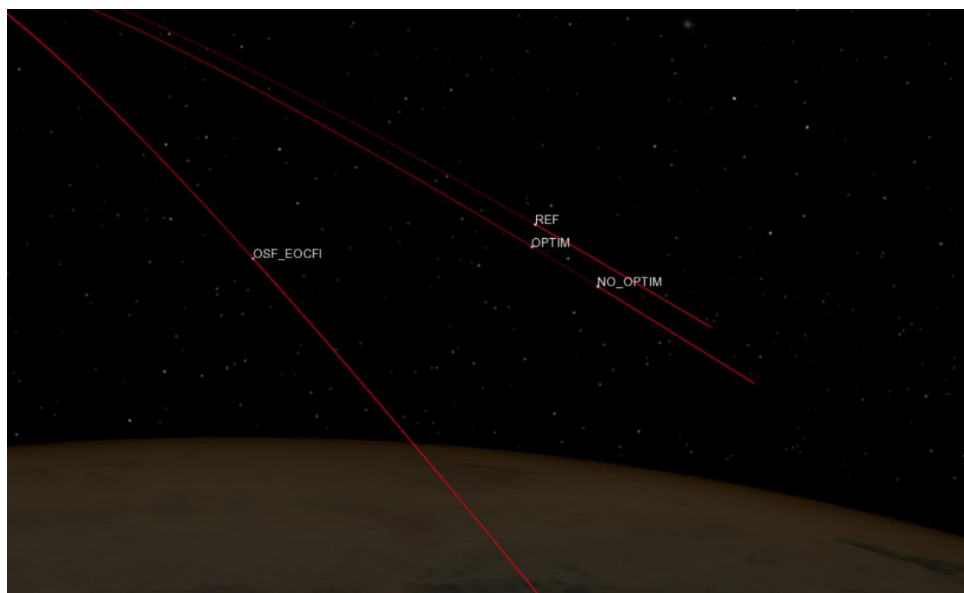


Figure 30: full 3D view at first initialization point.

These discrepancies might be explained by the time manipulation and injection mentioned in [REF01] however due to time constraints these results have not been analysed in depth. Admittedly the implementation of OSF initialization in the provided EOCCI script ([genOrbFromOSF_S3](#)) might contain some errors.

Quantitatively the different methods of initialization (along with their sources) are compounded in the following table:

Source	diff on 1st point in m
Orekit OSF init (no optimization / POD data injection)	approx. 38 600
Orekit OSF init (optimized)	approx. 7900
EOCCI OSF init	approx 9170

Table 5 : Difference in meters on 1st point

5. List of Provided Items.

This section is under redaction and pending changes as denoted by the highlighting

Along with this document, several tools in the form of script are provided at one's discretion. The list along with a brief description of its function is provided hereunder.

Item	Description / Content
COMPARISON.7z	Archive containing all relevant data necessary to reproduce data presented in this document*
README.txt	File containing the description and instruction to reproduce data.

Table 6: list of provided files and tools

*Please note that it is necessary to have a valid ASGARD installation (which installation process can be found in ternal git repository) and a valid EOCF C library install (which steps are recalled within README.txt).

6. Results Summary

Function	Difference	Worst Case
xl_change_cart_cs (EF to GM2000)	max < 1m	mid June
xl_change_cart_cs (EF to ToD)	max < 2.4m	mid Sept
Planetary ephemerises	0.013581950335946402 (deg) 0.8149170201404857 (arcmin)	Sun
xl_geod_distance	Max < 0.08m	All cases have similar discrepancies
xp_target_inter	Max < 2e-4m (YSM implementation) Max < 1.3e-3 m (intersection algorithm)	55° offset Nadir/ 77° latitude
xo_compute_osv (interpolation)	Max diff on Positions <= 0.0031629 Max diff on Velocities <=0.0001926	30s interpolation step
xo_compute_osv (Propagation)	Max diff on Positions < 70m	Along a 2 day period.
xo_compute_osv (OSF initialization)	Diff on initial position approx. 38.6 km	Case w/o Anomaly optimisation

7. Conclusions

The main difficulties during this study have been twofold:

- Due to ASGARD specifications scope, not all features/functions are implemented.
- The lack of information regarding EOCFI modelisation of computation.

The former point being mitigated by Orekit and its easily implementable algorithms. Efforts have been made to provide an “as close as possible” feature to cover ASGARD lack of coverage. However, the ease of portability of the provided mock-up directly into ASGARD has not been the focus of this study. This can warrant its own investigation.

The latter point being especially impacting on the proposed model to cover propagation needs particularly to approach a precise model as close as possible. A best effort to approximate a full-fledged propagator has been made. It is unrealistic to expect that an “educated guess” (plus trial and error) method as employed during this study will lead to an accuracy of the order of just a couple of meters. Especially in a timely manner. Further details on model would facilitate the implementation of a propagation model offering the same accuracy as EOCFI’s.

As a remainder, this study is to present in a context as significant as possible the results provided by both sources. Whether the accuracy of those results is satisfactory is dependent on the use cases and what accuracy is needed to generate the product in the whole process.