MAORI – A New Flight Dynamics and Geodesy Library

Jaime Fernández [(1)], Carlos Fernández [(1)], Javier Berzosa [(1)]

[(1)] GMV AD., Isaac Newton 11, 28760 Tres Cantos, Spain
Email: jfernandez@gmv.com, cfernandez@gmv.com, jberzosa@gmv.com

ABSTRACT

GMV has recently developed a new Flight Dynamics (FD) and geodesy library called ***MAORI***, written in modern C++17 and Python 3 from scratch. This library supports several projects led by GMV, including the operational provision of Precise Orbit Determination (POD) products of the Copernicus Sentinel satellites in the frame of the Copernicus POD (CPOD) Service, the simulation of tracking data for Galileo 2nd Generation (G2G) System Test Bed, or the maintenance of a catalogue of orbit debris.

The architecture of ***MAORI*** has been designed following four basic **principles**: i) to develop a library, rather than a collection of programs, ii) to split data from algorithms, to enhance the reusability of data among different algorithms, iii) to incorporate a clear data model that represents the physical meaning (e.g., satellites, stations, instruments, observations, etc.) and the relationships of the data and iv) to allow multiple use cases: C++ binaries, Python´s wrappers, HMI, micro-services, etc.

On top of these principles, there are two main **requirements** that drove the development of ***MAORI***: i) to achieve the state-of-the-art accuracy, in terms of modelling and estimation, and ii) to improve the performance (i.e., processing time and memory usage) as compared to similar SW, like NAPEOS. These two requirements, together with the innovative design, were addressed with an agile methodology, where capabilities were added incrementally, paying special attention to achieving high performance and accuracy. The design of each algorithm, and the data model was optimized to save processing time and memory. For instance, the use of buffers, or ad-hoc mechanisms to make use of multi-threading wherever was needed.

Finally, the architecture of the library, and each algorithm, takes advantage of the capabilities of C++, like OOP, polymorphism, templates, Standard Template Library (STL), etc.

Currently, the library is capable of supporting the most relevant FD capabilities: orbit propagation (complex numerical propagator, analytical propagators…), orbit determination (based on GNSS, radar, optical, passive ranging, Satellite Laser Ranging…), manoeuvre handling, event calculation, attitude handling, observation simulation, etc. The development of the library continues in several fronts: POD and geodesy, flight dynamics (including interplanetary), space traffic management, mission analysis and simulation.

This contribution will present the key aspects of the design of ***MAORI*** that have contributed to its success, focusing on the notion of library, the data model, its performance, and accuracy. The use of ***MAORI*** to develop ***FocusPOD***, an operational POD application for the Copernicus POD Service, will be shown as an example of operational use. The roadmap of ***MAORI*** will also be summarized.

I. INTRODUCTION

MAORI (**Multipurpose Advance Orbit Restitution Infrastructure**) is a Flight Dynamics (FD) and geodesy library written in modern C++17 and Python 3, developed by GMV from scratch as part of an internal research and development (R&D) activity. The library has been designed and implemented with the goal to support several projects that require astrodynamics capabilities (handling of ephemerides, propagation, events calculation…) and parameter estimation (orbit determination, geodesy…). It combines the great technical knowledge in the field acquired by GMV over the years with different projects, and modern technologies that are more compatible with today's programming paradigms and IT specifications. The projects that ***MAORI*** currently supports include:

- Copernicus POD Service: operational provision of orbital and auxiliary precise products to ESA and EUMETSAT for the Copernicus Sentinel satellites [2]. Seamless operational transition from NAPEOS to a solution based on ***MAORI*** in January 2023, keeping the same state-of-the-art accuracy standards and improving the timeliness of the near-real time products thanks to the new software. Both GNSS (GPS+GAL) and SLR are routinely processed as part of the service.
- Tracking simulator for Galileo 2nd Generation (G2G) System Test Bed: the new G2G system test bed will count with a tool developed in Python, and using ***MAORI***, to generate realistic tracking of GNSS observations from GNSS sensor ground stations, SLR data from ground telescopes, and ISL between the future Galileo satellites.
- Quality control for LEO-PNT: the consortium led by GMV in LEO-PNT, the new constellation of European GNSS emitters in LEO regime, will also make use of a solution based on ***MAORI*** to perform offline quality control in post-processing of the broadcast ephemerides.
- Support to Space Situational Awareness (SSA) activities: COTS developed by GMV based on ***MAORI*** are also routinely used in other projects to perform cataloguing activities, association (track-to-orbit, orbit-to-orbit, track-to-track) and

covariance analysis.

II. Design Principles

The library has been designed following four basic principles, depicted in Fig. 1:

- The focus is to develop a **library**, rather than a collection of programs. This means that all the functionality in the library is as generic as possible, without focusing on one particular use-case, and allowing a great level of user-interaction to decide how to build their applications. This is key to fulfil the requirement to support a variety of projects that require different architectures and technologies, always keeping flexibility and generality in mind during the development. The programs can then be easily built using ***MAORI*** as a dependency with clear interfaces, both for entry-points (inputs & configuration) and results (outputs).
- The **data** shall be kept clearly separated from **algorithms**. This principle means that modules within the library should work with clearly documented data that is not part of the algorithm itself; the algorithm should only concern with the calculations. This is a must when trying to fulfil the previous design principle, as being flexible means that the algorithms do not take the responsibility to acquire the data, organise it, or post-process it in any hard-coded way. Instead, the algorithms work with data that is already organised.
- Data shall be organised in a clear, relational **data model** that represents reality as faithfully as possible. This means, e.g., that the data model includes physical entities such as satellites, stations, instruments, oscillators, or antennas. This principle comes naturally from the previous one, since the algorithms are required to work with previously organised, common data; it is the library's duty to ingest, organise, and circulate this data around the different algorithms as the user requires. Moreover, this also helps greatly when supporting different projects as the captured reality is the same, even though the particular data structures for a given project may be particular. More information on the data model is provided later on.
- Multiple **use cases** shall be supported. This principle means that the library's flexibility is to be exploited by allowing to build applications in many different ways: as a stand-alone end-to-end C++ program, as a collection of instructions that can be chained by a script, as a service exposed in an HTTPS server via an API, or by making use of the Python bindings in a Jupyter Notebook, to name a few examples. More information on bindings is provided in a subsequent dedicated section.

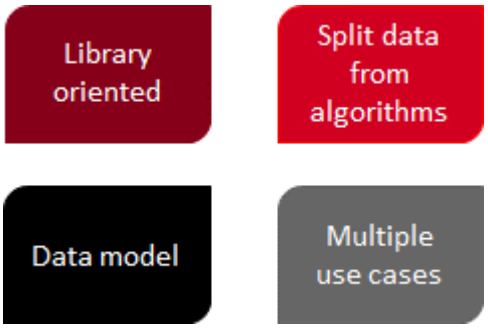

Fig. 1. ***MAORI*** *design principles.*

A. Library oriented

MAORI is a library, not a collection of programs. However, the library already contains high-level functionalities, like complex orbital propagation, measurement reconstruction or parameter estimation.
The design of the library tries to be as general as possible to allow using the same general algorithms for multiple purposes.
The library contains the following high-level elements:

- **Clocks**: It contains mostly data classes to store clock biases in support of GNSS/DORIS processing. It also contains the logical entities related to clock processing (such as the oscillator).
- **Contributions**: It contains the algorithms for the calculation of dynamic perturbations used in numerical propagation, and the related geophysical models (such as MSISE, EGM…) and their associated data classes for storage of model data.
- **Covariance**: It contains data classes to support the storage and operations with covariance matrices.
- **Events**: It contains the algorithms for the calculation of events, including the root finding algorithm, and the definition of event functions grouped per category (satellite-related, station-related…).
- **Estimation**: It contains the algorithms required to estimate parameters, in particular the Batch Least Squares estimator and Extended Kalman Filter. Some auxiliary data classes are also used in support of the estimation.
- **Frames**: It contains the algorithms required to perform frame transformation, mostly focusing on inertial to Earth-fixed conversions based on SOFA external library.
- **IOD**: It contains algorithms which perform Initial Orbit Determination (IOD).
- **Logging**: Auxiliary module with some basic functionality related to logging based on spdlog external library.
- **Math**: It contains some generic mathematical algorithms such as interpolation, Legendre polynomial functions, etc.
- **Models**: It contains common algorithms for geophysical models considered by different elements of the library, such as geopotentials, tides,

ocean loading, troposphere, and ionosphere, among others.
- **Observations**: It contains both the data classes that support the storage of observations in the library (such as the logical entities of Instrument, Link, ObsType…) and the algorithms to work with them, including pre-processing and reconstruction.
- **Orbit**: It contains the data classes that store satellites, orbits and state vectors in generic representations, as well as some algorithms such as the calculation of planetary ephemerides (based on calceph external library).
- **Orientation**: It contains the data classes that support attitude and pointing as a wrapper to the external class `Eigen::quaternion`, as well as the attitude simulation algorithm.
- **Parameters**: It contains classes that support auxiliary functionality related to parameters that can be estimated by the estimation module.
- **Persistence**: It contains the algorithms required to read data from external entities (files, DBs…) into the library, and also to export data from the library to an external medium.
- **Propagation**: It contains the algorithms to perform propagation of a state vector. It includes different types of propagators, such as numerical, Keplerian, SGP4, etc.
- **Scenario**: It contains the Scenario, which is the data class that represents the data model. It is mainly used to organise access to the central information and to enforce the relationships between logical entities in the data model.
- **Station**: It contains data classes to support both stations as logical entities and coordinates.
- **Time**: It contains data classes such as the Epoch and the algorithms required to work with time (parse epochs, time conversions…).
- **Utils**: Generic utility classes mostly related to C++ (streams utils, string utils…). It also contains the exception types defined for the library.

B. Data vs. algorithms

With ***MAORI***, it was decided that **data** will have a central role. The goal of supporting multiple applications requires to adopt the strategy that considers the most demanding in terms of data, which are geodesy, space traffic management and mega-constellations.
To allow re-using the same algorithms, or data, for different applications requires to split them by design. Data was organized within a data model (see next sub-section), and algorithms were designed to make use of this data model using ad-hoc APIs and restricting the changes over the central data.

C. Data model

The notion of data is key in the ***MAORI*** design. This is so because the philosophy of the library revolves around how the data is used, what physical reality it represents, how it relates to other entities, and what volume it may acquire in terms of performance and ease of use. For these reasons, the data model in ***MAORI*** has been carefully designed to:
1. Represent the physical reality of the data.
2. Provide simplified access to this data to the different algorithms and users.
3. Scale adequately in terms of number of elements (e.g. keeping in mind high-rate observations, or mega-constellations).

The adopted solution tries to mimic a relational DB, in the sense that each component (e.g. a satellite, an instrument) is related to each other using a unique mechanism, which is based on keys (identifiers). Programmatically, there are different versions tailored for specific needs that offer a common interface across the library: the basic one is implemented using a C++ `std::map` container, which uses as key a unique identifier (whose implementation depends on the entity being stored), and an index representing the last accessed element, based on the principle that data is typically used in an orderly fashion. Other specialization is available for data which require interpolation based on `std::vector` of pairs to benefit from contiguous memory allocation, ideal for intensive access. These containers are referred to as **tables** in the library. All specializations of tables allow keeping large volumes of data, and provide efficient search algorithms, which are enhanced keeping the location of the last accessed element, which is useful for time-series.

The data model is represented by a class called **Scenario**, which contains all the different tables and handles the access to the data by providing pointers and taking care of the life cycle of the objects in memory. The Scenario is not a singleton by design (i.e., several Scenarios with independent data can coexist). This has been done so to ensure an easy means of **parallelization**, albeit not ideal in terms of data duplicity.

Within the library, conceptually, we can distinguish two types of tables:
- **Logical Entities**, which represent physical (i.e., real) entities, like a satellite, or a station.
- **Physical Data**, which represent elements like orbits, clocks, or attitude.

The main reason of this separation is that while both are located on tables, the logical entities do not have the concept of interpolation, while there is with the physical data. For instance, it is possible to interpolate clocks, orbits, attitude, etc., but it is hardly seen how a table of satellites could be interpolated.
The Logical Entities are composed by: Satellite, Station, Instrument, Oscillator and Antenna. It would be possible to even consider Oscillator and Antenna as Instrument, but the separation is done because Oscillator and

Antennas do not generate observables, while Instruments do. Fig. 2 shows the relationship between the Logical Entities.

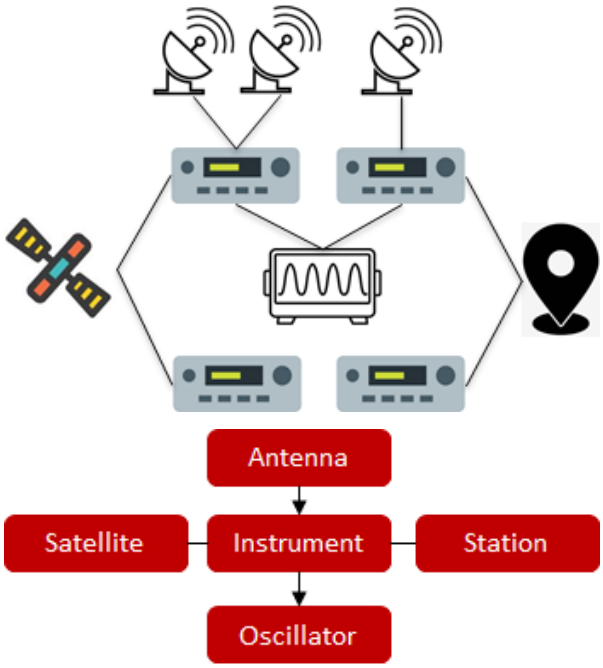

Fig. 2. *Relationship between Logical Entities.*

It can be seen that:

- There could be several *instruments* on each *satellite* or *station*. For instance, a GNSS, DORIS and S-band receivers.
- The *instruments* could have [0, 1 …] *antennas*, to cover cases like:
  - None: e.g., an accelerometer does not use an antenna.
  - One: e.g., a ground station on S-band with one antenna.
  - Several: e.g., an on-board GNSS receiver connected to two antennas on opposite sides of the satellite.
- *Oscillators* can provide the signal to several instruments, which is typically the case on-board and on ground stations. For example, in Sentinel-3 or Sentinel-6 mission, there is an Ultra Stable Oscillator (USO) that provides a signal to the GNSS and DORIS receivers.

The **Physical Data** can have relationships with the Logical Entities (e.g. orbits), or not (e.g. the EOPs). Fig. 3 shows some of the relationship between the Logical Entities (in red) and some of the most usual Physical Data (in grey).

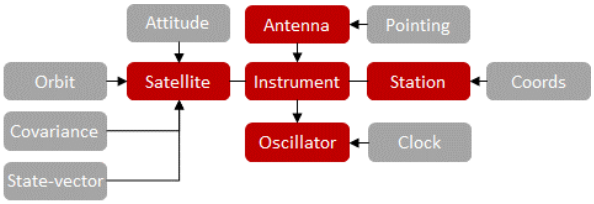

Fig. 3. *Relationship between some Logical Entities and Physical Data.*

It can be seen:

- A *satellite* can have [0, 1 …] **Orbits**, **Attitude**, **Covariance** and **State-Vectors**. It is clarified that even if Orbits are composed of state-vectors, a separation is done to support specific applications where single state-vectors are needed. Having several orbits associated to a single satellite (and the same happens with the other physical data), is useful, for instance, to perform comparisons between them.
- An antenna can have [0, 1 …] **Pointing** describing its orientation in time. Typically, it will be just one.
- A station can have [0, 1 …] **Coordinates** describing its evolution in time. Typically, it will be just one.
- Finally, an oscillator will have [0, 1 …] **Clocks**. This is useful to perform clock comparisons, but also to support advance applications where different clocks realizations are done of the same oscillator, depending on the GNSS signals used.

Although many more elements are present in the library, the Observations layer is showed in Fig. 4 as an example.

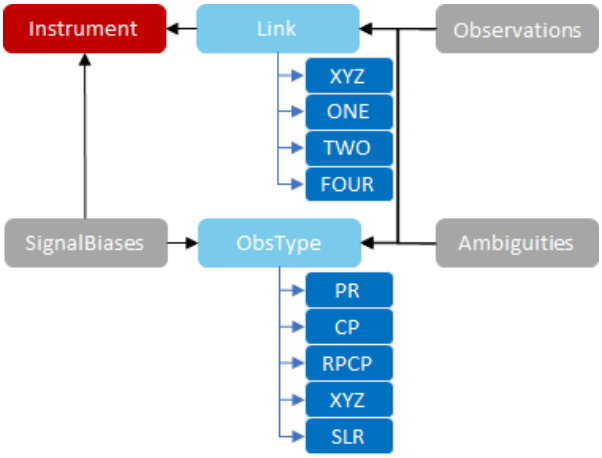

Fig. 4. *Relationship between Logical Entities and Observations organisation.*

It can be seen that:

- A **Link** is basically the concept that established the relationship between the different instruments that create an observable. Link is a virtual class, with several derived classes that can be:

    - XYZ, which in this case just refers to the centre of mass of a satellite, or the location of a particular instrument on-board (e.g., the GNSS antenna).
    - One way and two-way links, which include the instruments involved on the satellite/s and the ground station/s.
    - Four ways, on which there can be up to four instruments involved (e.g., GNSS double differences).
- The **ObsType** is the virtual class that represents the type of observable. Its derived classes can be:
    - GNSS Pseudo-range, Carrier-phase.
    - XYZ.
    - SLR.
    - DORIS.
    - Optical.
    - Radar.
    - Passive ranging.
- **Observations** and **Ambiguities** are related with the **Link** (to get information about the instruments involved), and **ObsType** (to know the type of observation). The main difference between both is that **Observations** are typically epoch-based, while (*GNSS phase)* **Ambiguities** are interval-based (per pass).
- **SignalBiases** represents the instrumental delays, and therefore, it is associated to the **Instrument**.

The central data model is key to the design of ***MAORI*** and has helped achieved the challenges posed by the demanding requirements of the library.

D. Use cases

The design of MAORI aims to support different use cases, understood as different ways of generating applications or using the library. Following are just some examples:

- C++ binaries: to construct a C++ program that performs specific tasks as configured. Examples are a propagator, an orbit determination, an orbit comparison tool, etc.
- Python binding: to construct Python scripts, which perform specific tasks by calling to the C++ library. Indeed, the same C++ binaries described in the previous bullet can also be constructed with Python scripts (not compiled), with minimum computational penalty meanwhile there is not much data transfer between the C++ and Python layers.
- Service / micro-services: to construct applications using a service or micro-services principles. They could be constructed using any of the above-mentioned architectures.
- Service / client architecture.

III. HIGH LEVEL REQUIREMENTS

On top of the mentioned design principles, there are two main requirements that drove the development of ***MAORI***:

1. To achieve the state-of-the-art **accuracy**, in terms of modelling and estimation.
2. To improve the **performance** (i.e., processing runtime and memory/CPU usage) as compared to similar SW suites.

These two requirements, together with the innovative design, were addressed with an **agile methodology**, where capabilities were added incrementally, paying special attention to achieving high performance and accuracy in each of the steps of the way. The design of each algorithm, and the data model was optimized to save processing time and memory. For instance, by using buffers of previously computed calculations, or ad-hoc mechanisms to exploit multi-threading wherever was needed.

An example of this process of development is the inclusion of a relational data model in the library based on the C++ Standard Template Library (STL): in a first attempt, several implementations were added (e.g. based on an indexed `vector`, based on a `map` using IDs as key, based on `linked lists`…). When the different options were implemented, they were tested in a benchmark to select which was the option that satisfied better the different use cases in terms of addition, retrieval, and erasure, considering various orders of magnitude of expected elements (e.g. number of potential satellites vs. number of potential observations). Out of this process, a final solution based on the C++ `std::map` with an indexed access was chosen. Later on, in order to improve access performance, the `std::vector` was implemented too while keeping a common, transparent interfaces across the library.

The added value of the agile methodology together with the careful implementation and benchmarking of all low-level elements allowed that once the more complex algorithms were available (e.g. to carry out a full end-to-end orbit determination, including observation decoding, preprocessing, initial solution computation and iterative batch estimation, with multiple propagations and measurement reconstructions), there was low risk of not meeting these two requirements in terms of accuracy and performance. Available results of some of the algorithms are included in the following subsections.

IV. TECHNOLOGIES

MAORI core library is written in C++ following the **C++17** standards; it makes extensive use of the C++ Standard Template Library (**STL**) and open-source dependencies for all duties that are not part of the

business logic of the application (i.e. astrodynamics and geodesy). The list of **dependencies** is summarized in Tab. 1.

Table 1. External dependencies of ***MAORI***

Name	Description
SOFA	Standard models used in fundamental astronomy, based on IERS Conventions.
SGP4	SGP4 propagator for TLEs.
IERS2010	Set of routines that extend the IERS 2010 conventions.
Calceph	Access the binary planetary ephemeris files, such INPOPxx, JPL DExxx and SPICE ephemeris files.
NRLMSISE-00	NRLMSIS 00 Atmosphere Model.
Nequick-G	Algorithm to correct for ionospheric delays based on an adaptation of the three-dimensional NeQuick electron density model.
Eigen	Template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.
YAMLCpp	A YAML parser and emitter.
FMT	Modern formatting library.
SPDLog	Fast C++ logging library.
Xerces-c	Validating XML parser.
bxzstr	Access to compressed streams.
Thread-pool	Thread Pool for parallel processing.
Pybind11	Python binding library.
Pagmo	Optimization algorithms.

In order to exploit the flexibility granted by Python and the powerful libraries available in that language (e.g., for data visualisation or to apply machine learning), ***MAORI*** exposes a set of **Python bindings** to the most relevant functionality of the library so that it can be transparently used from Python. This version is called ***pymaori***, although it conceptually represents the same library as the C++ core version.

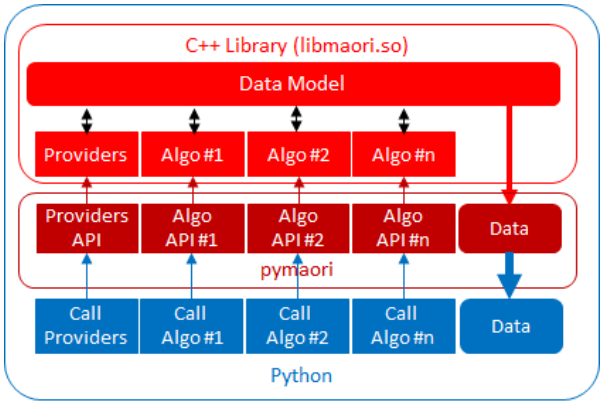

Fig. 5. *Data flow between C++ (library layer), Python binding (pymaori) and Python app.*

The interaction between the C++ layer and the Python layer is shown in Fig. 5, where also the exploit of the data model is shown. In this diagram, the data is read using the providers and ingested into data model (within the C++ classes), where it is organised, then the driving algorithm (in Python) calls the required algorithms to carry out the business logic. Finally, the obtained results (data) can be handled in C++ or in Python directly to exploit it (e.g. in Grafana, or in a notebook).

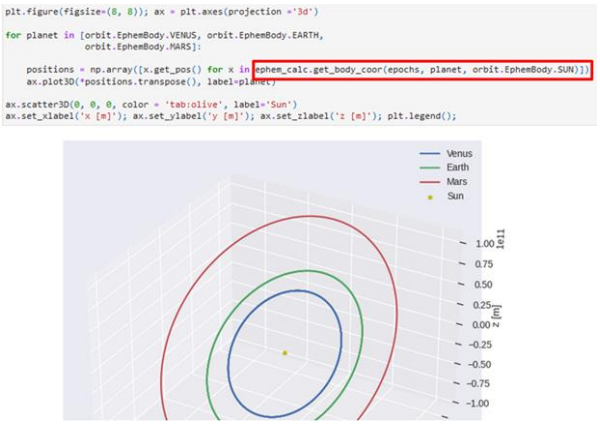

Fig. 6. *Example of Python code making use of MAORI bindings (marked in red).*

In order to show the look and feel of this concept, Fig. 6 shows an example of the seamless usage of ***MAORI*** from a Python environment, in this case a Jupyter Notebook exploiting visualisation in 3D.

V. CAPABILITIES

The capabilities of ***MAORI*** include, inter alia, the following elements:

- Implementation of the most common **international formats** in FD, POD, and geodesy, including observation and navigation RINEX v3 and v4, SP3c/d, Clock RINEX, SINEX, ESA EOF, Geopotentials, EOPs, leap seconds, RSGA (solar activity), atmospheric gravity, among others.
- **Frame of reference** transformations making use of the latest IERS conventions (2010), focusing on transformations between Earth-fixed (ITRF) and inertial (GCRF). Intermediate frames and different IERS conventions are used for specific algorithms, such as TEME for TLE propagation.
- Handling of different **time scales** including TAI, UTC, UT1 and various GNSS time systems such as GPS and Galileo reference time. Internally, ***MAORI*** uses TAI.
- Handling of different satellites' **attitude**, including interpolation of real attitude data or simulation following a number of available theoretical laws (including the GNSS satellites). More attitude laws can be readily added by extending the current basic pointing laws, which include geocentric and

geodetic modes based on different ellipsoids, Sun-pointing, orbital pointing…

- Most common **interpolation** algorithms for orbits, attitude, clock biases, EOPs…
- **Event calculation**, including visibility from sensors at orbiting platforms and stations, third body eclipses…
- **Merging** algorithms for orbits and clocks, and geophysical data such as the EOPs or solar activity data which allow to easily combine different inputs while keeping the expected accuracy (e.g., offering capabilities to easily overwrite predicted data with computed values, or to align clock biases to a common reference).
- Numerical **integrators**, including fixed-step algorithms (such as the explicit Runge-Kutta 8, or the multi-step Adams-Bashforth-Moulton 8) and variable-step algorithms (such as the Runge-Kutta-Fehlberg 7/8).
- State of the art **orbit modelling**, considering the most common perturbations in geodesy required for POD applications such as geopotential (including time-varying terms), solid and ocean tides, atmospheric gravity, relativity, solar radiation pressure (both with fixed area and using a macro-model for the satellite), atmospheric drag (both with fixed area and macro-model, based on MSISE 2000), Earth's albedo and infra-red, empirical accelerations (Cycle-Per-Revolution, CPR), both long and impulsive manoeuvres, CODE empirical accelerations (ECOM-2 model), power thrust, etc. The full list is given in Tab 2.
- Availability of **analytical propagators** based on Keplerian motion and Two-Line Element (TLE) SGP4 propagation theory.
- Capability to handle **linear covariance** propagation.
- Handling of the most common **tracking techniques**, including multi-GNSS (GPS, Galileo), SLR, DORIS, among others including trajectory data (XYZ) as tracking measurements.
- State of the art **observation reconstruction** modelling, including clock biases, relativity, time of flight effects, instrumental delays (both as code and phase biases), phase wind-up, ambiguities, phase centre offset and variations (PCO, PCV), ionosphere (based on the Galileo Nequick model), troposphere (based on Mendes-Pavlis, Saastamoinen or Niell models), ground station tide effects (including pole tides, permanent tides, ocean loading, solid tides and ITRF20 seasonal geocenter motion modelling effects).
- **Parameter estimation** algorithms based on Weighted Least Squares and Extended Kalman Filter (EKF), allowing to estimate orbit parameters, clocks and biases (epoch-wise, using the snapshot approach), ground station coordinates, and geophysical model parameters such as tropospheric zenith delay and associated mapping functions. Additionally, single-receiver ambiguity fixing algorithms to resolve integer carrier phase ambiguities are also implemented.

MAORI is continuously evolving, adding new capabilities as required by internal and external needs.

Table 2. Geophysical models available in ***MAORI***

Category	Models
Frames	IERS 96 and 2010 conventions Linear Mean Pole Local frame converter (topocentric, QSW, TNW)
Gravity	COST-G, EIGEN, EGM Support to time-varying terms. Ocean tides (OTIERS): FES2014 Solid tides (STIERS): IERS 2010 Atmospheric gravity (AOD1B) Seasonal geocenter motion: ITRF20
Third body	JPL ephem. DE405, DE421
Density	MSISE00
Space weather	NOAA (RSGA), GFZ (kp_index)
Radiation	Solar Radiation Pressure (SRP) Earth´s albedo & infra-red Antenna Power Thrust
Empiricals	Constant-Per-Revolution (CPR) ECOM/ECOM2
Ionosphere	Nequick
Troposphere	Mendes-Pavlis, Niell, Saastamoinen
Satellite	Fixed area, macro-model Theoretical or real attitude laws Impulsive and long manoeuvres
Station	North-East-Up or ad-hoc pointing Post-Seismic Deformations ITRF20 seasonal geocenter motion
Biases	Absolute and Differential signal biases

VI. ACCURACY AND PERFORMANCE

In order to showcase the achievable results of the library in terms of accuracy, results from the Copernicus POD Service (with stringent accuracy requirements of cm-level) are shown in Fig 7. It shows the daily 3D RMS of the differences between a reprocessing of Sentinel-1A precise orbits (computed with GPS data) with ***FocusPOD*** (SW developed using ***MAORI***) and a combined solution generated as a weighted mean of several independent external solutions. The agreement between both solutions is below the 1-cm level.

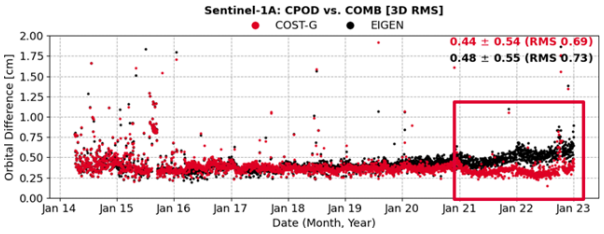

*Fig. 7. Accuracy of reprocessed POD solution of Sentinel-1A based on **MAORI** [2].*

In terms of performance, the algorithms have been compared with legacy SW (NAPEOS) to measure the runtime, providing results in Fig. 8. These results show an improvement of a factor ~2 with respect to the legacy SW in terms of runtime, depending on the scenario.

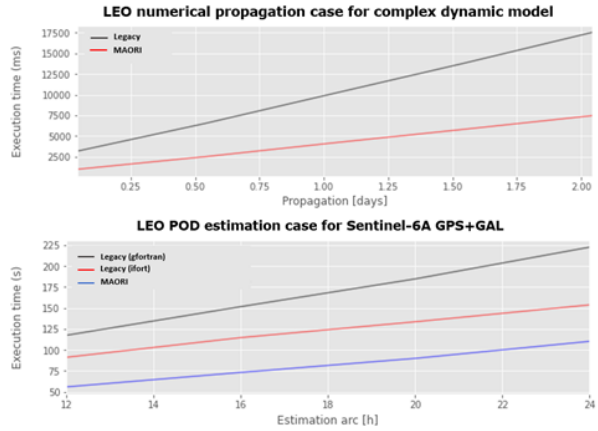

*Fig. 8. Runtime performance of **MAORI** vs legacy SW.*

One of the main design considerations when keeping in mind accuracy and performance, considering the modern paradigm of HW architecture, is **parallelization**. The library is designed to run in parallel out of the box thanks to the explained notion of the Scenario not being a singleton entity. Since all algorithms work with data stored in the scenario, working with independent scenarios ensures that the processes can run in parallel with no risk of concurrency issues. However, this is not ideal from the point of view of data usage, as it requires loading all data in each of the scenarios (EOPs, geopotential coefficients, solar activity…). In order to enhance this, the library offers the notion of Light Weight Scenario (LWS), which is a clone of the scenario that copies the data in the models' tables and that is usually common to all use-cases. The LWS are meant to be short-lived, as they are created to run a specific task and then write their results in the master Scenario, as depicted in Fig. 9.

In addition to this, internally some algorithms make use of a thread-pool to run certain tasks in parallel (e.g., multi-satellite propagation, or event calculation). This however requires the user to be aware of data usage and to handle locks explicitly for the tables involved (for instance, to persist orbit data after a propagation), which may be cumbersome and therefore is recommended only for advanced users.

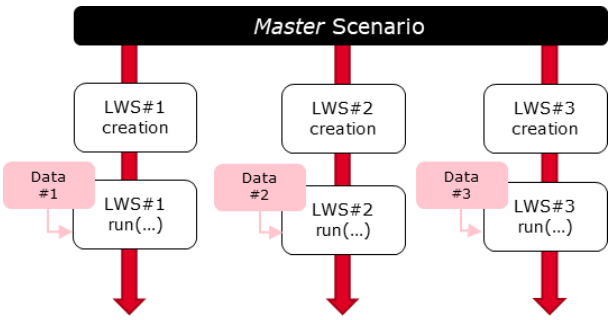

Fig. 9. Depiction of Light Weight Scenario running in parallel.

VII. Way-forward and conclusions

GMV has developed a new FD and Geodesy library, written in C++ and Python, to support a large variety of projects, including FD, POD, SST, Simulation, etc. In terms of capabilities, it already contains much of the basic functionality of FD, but more is continuously added. In terms of performance, the new design and implementation has improved them compared to the legacy software.

MAORI roadmap includes the implementation of manoeuvre planning and optimization, mission planning, interplanetary support (frames, propagation and manoeuvre planning), to complete the GNSS, SLR and DORIS processing, and to include VLBI, to support geodesy applications. ***MAORI*** is also being updated with SST applications (e.g., close approach), and simulation (e.g., orbit, clock and measurement simulation).

VIII. References

[1] Fernández Martín, C., Berzosa Molina, J., Bao Cheng, L., Muñoz de la Torre, M. Á., Fernández Usón, M., Lara Espinosa, S., Terradillos Estévez, E., Fernández Sánchez, J., Peter, H., Féménias, P., and Nogueira Loddo, C.: "***FocusPOD***, the new POD SW used at CPOD Service", EGU General Assembly 2023, Vienna, Austria, 24–28 Apr 2023, EGU23-1908, https://doi.org/10.5194/egusphere-egu23-1908.

[2] Berzosa, J., Fernández Martín, C., Matuszak, M., Fernández Usón, M., Fernández Sánchez, J., Nogueira Loddo, C., Femenias, P., Peter, H., and Meyer, U.: "Reprocessing of Copernicus Sentinel POD solutions with COST-G geopotential models", EGU General Assembly 2023, Vienna, Austria, 24–28 Apr 2023, EGU23-16137, https://doi.org/10.5194/egusphere-egu23-16137