JOSCAR/JDRAGON: TOOLS FOR MANEUVER STRATEGY COMPUTATION DEVELOPED IN JAVA AND USING PATRIUS

Ivan Sumelzo Martinez [(1)]
Pierre Labourdette [(2)]
(1) CNES, 18, Av. Edouard Belin, 31401 Toulouse Cedex 9, France, Email: ivan.sumelzomartinez@cnes.fr
(2) CNES, 18, Av. Edouard Belin, 31401 Toulouse Cedex 9, France, Email: pierre.labourdette@cnes.fr

ABSTRACT

JOSCAR/JDRAGON are new tools of maneuver strategy computation, developed internally in CNES (Centre National d'Etudes Spatiales, French Space Agency) at the Orbital Maneuvers Office (DCT/SB/MO). Both tools have been rewritten in JAVA even if they are always based on the same basic principles of the initial OSCAR/DRAGON Fortran versions, which were intensively used for the Automated Transfer Vehicle (ATV) and still today, for the operational design of the LEOP, phasing and rendezvous scenarios for GALILEO missions.
This paper describes the methods implemented as well as the software functionalities, pointing out the differences between JAVA versus FORTRAN version, the first one taking advantage of the new functionalities of CNES PATRIUS library as well as almost 20 years usage feedback.

Index Terms— Maneuver strategy computation, rendezvous, optimization, PATRIUS, JAVA

1. INTRODUCTION

Since many years ago, CNES has been involved in missions requiring phasing maneuvers computation. In that frame, OSCAR/DRAGON tools [1] were developed for the ATV project in 1997, in order to perform mission analysis studies and conduct End-to-End simulations. This was possible after a straight collaboration with Russian specialists of KIAM (Keldysh Institute of Applied Mathematics) and MCC-M (Mission Control Center at Moscow), who had wide experience in rendezvous missions thanks to Saliut-Mir-ISS program.

Expert tools developed at the Orbital Maneuvers office (some of them used in operational contexts as GALILEO, ROSETTA, ATV), are essentially coded in Fortran. However, thanks to the decision some years ago to use Java technology, existing flight dynamics tools and libraries are being rewritten within the framework of the SIRIUS project [2]. While the development of the basic software layout (PATRIUS) and the operational tools (FDS) already started some time ago, the redevelopment of the analysis tools was recently set up. In that frame, it was decided to take DRAGON/OSCAR tools as "pilots", given the numerous computations as well as the considerable data management conducted. As expected, DRAGON/OSCAR porting to Java has been useful to point out the implications that using Java-based software can have in the very particular frame of orbital study tools.

Concerning the tools functionalities, JDRAGON is capable of computing a near-optimal mission plan, using initial conditions for target and chaser spacecraft, an amount of maneuvers to be optimized respecting some constraints of application as well as certain rendezvous conditions. It is based on a robust and fast method, which requires calling a numerical propagator iteratively. For this purpose, JPSIMU has been also developed based on its PSIMU predecessor, which is the heart of numerous CNES flight dynamics tools (as in ATV-CC or GALILEO FDS ones).

At a higher level, JOSCAR, which uses JDRAGON as a kernel, allows to perform End-to-End Monte-Carlo simulations, necessary for mission analysis purposes, allowing testing the robustness of the computed strategies.

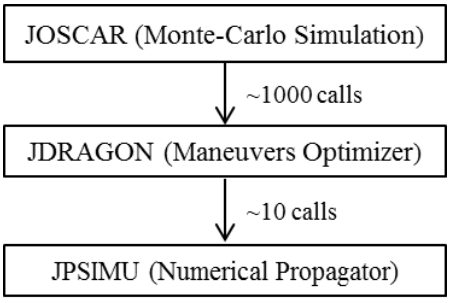


Figure 1: Tools dependency

Above the new design of these tools, thanks to the Java object approach, their validation is also a big challenge. Thematic validations have been conducted with no easy comparisons with the Fortran version, given the differences in their corresponding flight dynamics libraries. Special efforts have been put into code performance, bearing in mind an expected penalty in computation cost of a factor two approximately, with respect to Fortran language. Hence, it has been also important to look for optimal tools settings, aiming at having both fast computations and satisfactory

accurate results. Concerning the quality of the code, Eclipse [3] environment analysis tools have been used in order to be compliant with CNES coding standard rules.

At last, in order to deal with the considerable input/output data generated, a Graphical User Interface has been developed using GENIUS (a higher level CNES JAVA toolkit based on Swing) which allows using these tools in a more friendly way on many different Operating systems from Windows to Linux.

2. DEVELOPMENT ENVIRONMENT

The development environment has been based on the SIRIUS project workbench. Some of the important components are:

- *Java Development Kit*: containing the Java execution environment as well as the basic development resources.
- *Eclipse*: the Integrated Development Environment. It regroups useful plug-ins for software development such as *M2eclipse*, *Checkstyle*, *FindBugs* or *PMD*, among others.
- *Maven*: useful to build and manage Java projects.

This set up has been efficient, helpful and user-friendly, which has increased the productivity of the development.

2.1 PATRIUS

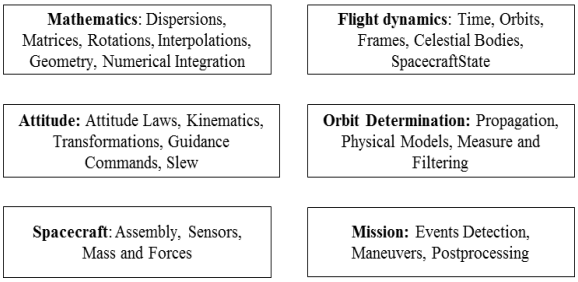


Figure 2: PATRIUS library features overview

The reference low level library used for mathematical and flight dynamics functions has been PATRIUS (*PATrimoine de base siRIUS*), which is based on *Orekit* [4] and *CommonsMath* [5], as well as other supplementary libraries. Its development started in 2011 and today, it is considered as a powerful library with many features, fully tested and validated, ready to be used in next generation FDS development as well as in mission analysis tools and internal studies.

2.2 GENIUS

For standardization purposes, each developed tool has been devised to provide 3 modes of utilization:

- 1. *Subroutine*[1] *mode*: containing the pure tool computations, possible to be called from other classes.
- 2. *Batch mode*: capable of reading an input data file (".xml") and calling the subroutine mode.
- 3. *GUI mode*: a Graphical User Interface (GUI) which permits to create specific scenarios and launch the computation (via the batch mode).

The GUI mode implementation has been possible thanks to GENIUS (*GENeration of Interface for Users of Scientific S/W*), a higher level toolkit, fully written in JAVA and based on Swing. It was recently developed internally in CNES and it was conceived as an easy-to-use toolkit, oriented towards users not necessarily familiarized with low level language.

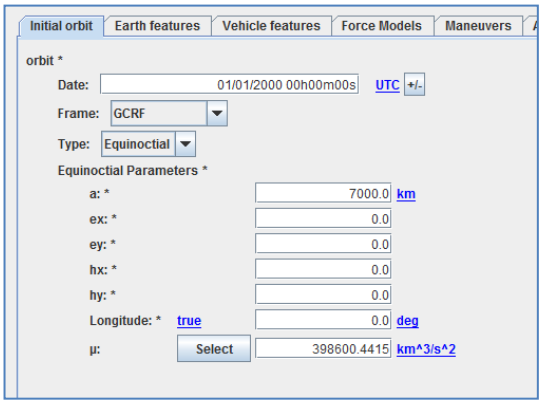


Figure 3: Example of GENIUS GUI appearance

Thereby, GENIUS permits to create GUIs in the context of general scientific applications. The main advantages are:

- Units management
- Performing conditional display
- Simplified approach, in particular about events management (setting actions before/after a certain event is reached).
- Read/write for files directly integrated. Configuration file containing GUI data (".xml") is generated automatically.
- Process management compatible in all OS (thanks to JAVA)

[1] Linked to Fortran principles

3. JPSIMU: THE NUMERICAL PROPAGATOR

Similar to DRAGON/OSCAR tools which call PSIMU numerical propagator iteratively, JDRAGON/JOSCAR required their equivalent PSIMU JAVA version. Even though PATRIUS provided sufficient high level classes for propagation purposes, it was decided to develop a specific tool, JPSIMU, independent of JDRAGON/JOSCAR, which might be useful for other tools in the coming future.

Inspired by its FORTRAN predecessor version, JPSIMU is a numerical orbit extrapolator around the Earth, whose main functionalities are:

- Taking into account different forces:
  - Earth potential
  - Third body perturbations
  - Aerodynamics forces
  - Solar Radiation Pressure
  - Ocean/ Terrestrial Tides
- Customizing the vehicle features
- Defining a maneuver sequence
- Defining an attitude sequence
- Using different numerical integrators
- Identifying orbital events

However, due to the change of flight dynamics libraries (PATRIUS for JPSIMU, BIBMS for its predecessor); some differences arise when comparing both versions.

For instance, while PSIMU reference first order integrator is Cowell, JPSIMU implements a Dormand-Prince of 8th order with variable step-size (available in PATRIUS). It is characterized by a good performance for a large variety of orbits, especially for highly eccentric ones. Furthermore, it treats in a more accurate way the forces discontinuities (maneuvers, eclipses) and contains an interpolation function which allows dense output data and good precision at event detections, without interfering in the propagation results.

3.1 Propagation modes

Aside from the three modes of utilization (section 2.2); JPSIMU can propagate orbits in two different manners, depending on the propagation purposes. First, the slave mode, which is the fastest, propagates the initial orbit until a certain final date, returning a single final spacecraft state (position, velocity and date). It could be the interest, for example, when performing several calls from JDRAGON. The second possibility, using which is called the master mode, registers the spacecraft states sequentially every given time step. This is the case, for example, when using JPSIMU GUI mode, which allows generating an ephemeris file with up to 80 output variables, depending on user needs.

3.2 Event detectors

The use of event detectors along the propagation is one of the most powerful new functionalities with respect the Fortran version. Each event of interest is translated into a continuous function *g*, which is a function of the spacecraft state. The event detection is equivalent to find the roots of this function. At each integration step, these functions are evaluated, monitoring the sign changes. When a sign change is detected, an iterative process is triggered in order to detect the event (finding the function's root) with the required precision.

Although PATRIUS library provides a large variety of event detectors already coded, each user can define its own ones, respecting the EventDetector interface. In JPSIMU they are constantly used along the propagation with different purposes:

- To stop the propagation
- To switch between attitude laws
- To define the beginning/end of maneuvers (either impulsive or spread maneuvers)
- To identify orbital events of interest as: ascending/descending nodes, station visibilities, entering/exiting an eclipse or arguments of latitude, among others.

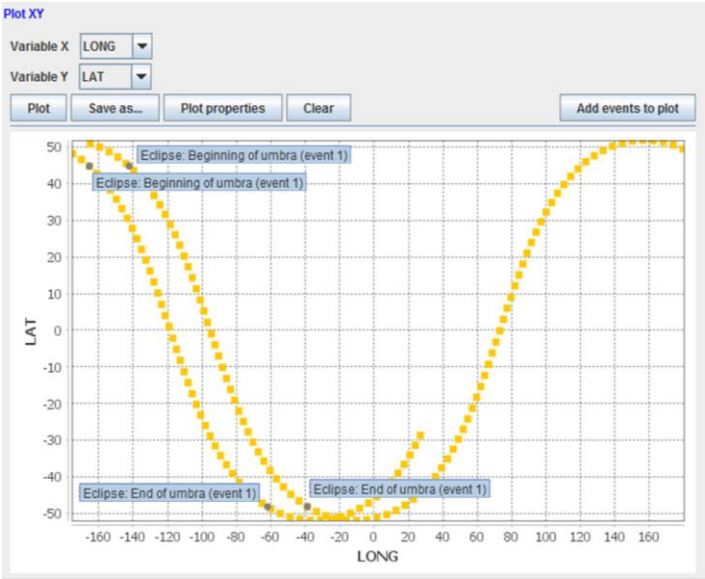


Figure 4: JPSIMU GUI plot panel example: Detecting eclipses events on International Space Station ground track (longitude, latitude in degrees)

3.3 Frames configuration management

Finally, the last substantial new feature is the possibility to manage the frames configuration used for the propagation. In other words, it permits to vary the level of accuracy considered (vs CPU time) when performing frames transformations along the propagation.

Table 1: Possible customizable frames corrections in PATRIUS

From	To	Phenomena	Corrections
GCRF[2]	CIRF	Earth rotation axis around Ecliptic pole	Precession, Nutation
CIRF[3]	TIRF	Diurnal motion	UT1-UTC difference
TIRF[4]	ITRF[5]	Earth rotation axis wrt Earth's crust	Tides, libration, S' effects, EOPs

PATRIUS frames configuration used by default is IERS2010 convention, which takes into account all corrections presented in Table 1. Managing the frames configuration is important since it has a direct impact on both precision and propagation's performance:

Table 2: CPU times comparison for different frames configurations and max deviations in position and velocity (wrt IERS2010) for a 30 days LEO propagation

# Test	Corrections	dPmax (m)	dVmax (m/s)	CPU time (s)
1	All (IERS2010)	-	-	4.24
2	All except EOPs	1.46E1	1.68E-02	4.02
3	Only Prec-Nut	1.46E1	1.68E-02	2.88
4	No corrections	4.69E2	5.42E-01	1.97

Finally, it is important to point out that this customization is not possible in PSIMU Fortran version, which takes into account only precession-nutation corrections (the most important ones in terms of accuracy). As shown in Table 2, this configuration (Test 3) provides good trade-off between accuracy and performance and thereby, it might be the optimal setting to use in JDRAGON/JOSCAR tools.

3.4 Validation

Once the tool was fully developed, it was required an exhaustive validation, since JPSIMU was expected to become the core of JDRAGON/JOSCAR, as well as other future tools.

First comparisons were done with respect to the Fortran version. Nevertheless, the results obtained were not totally satisfactory for all test cases. Some differences arose due to the different flight dynamics libraries used. The main sources inducing results discrepancies were the frames configuration management, the first order integrator used (Cowell vs Dormand-Prince) and the force models implementation.

It is for that reason that an alternative strategy was chosen: JPSIMU is based on the *NumericalPropagator* class provided by PATRIUS. This class was strongly validated against ZOOM (the precise orbit restitution tool at CNES). Thus, since the numerical propagator class had already been fully tested for a large variety of scenarios, it was decided to set PATRIUS as the reference for the validation. The validation process was organized as follows:

1. *Thematic validation*: only the classes containing the pure computations of JPSIMU were validated. Firstly, a set of ephemeris files were generated using pure PATRIUS. Then, they were taken as reference in order to validate the ephemeris generated by the JPSIMU subroutine mode. A total of 42 tests were performed, divided in seven different topics: initial orbit, earth features/ frames configuration, vehicle, force models, maneuvers scenario, attitude laws and integrator.
2. *Batch/GUI mode validation*: it was oriented towards the validation of the GUI and the configuration data file (".xml") generation. The 42 thematic test cases were reproduced using the Graphical User Interface, expecting to obtain exactly the same ephemeris generated during thematic validation.

This validation was performed using the *JUNIT* environment which, among others advantages, permits to easily run unitary tests, performing ephemeris files comparison in the case of JPSIMU. The battery of tests conducted had coverage of more than 80% in the code source. Even if the tool evolves, the battery of tests can always be launched, taking less than 5 minutes. On top of that, special attention was paid on the quality of the code. In order to do that, *FindBugs* and *CheckStyle* tools were used. They help programmers to write Java code (or readapt in this case) compliant with certain coding standards. These tools were set in order to verify CNES coding standard rules, which were provided by the Quality Office.

[2] Geocentric Celestial Reference Frame (inertial)
[3] Celestial Intermediate Reference Frame (pseudo-inertial)
[4] Terrestrial Intermediate Reference Frame (non-inertial)
[5] International Terrestrial Reference Frame (non-inertial)

4. JDRAGON: COMPUTING MANEUVERS

Based on its predecessor version, JDRAGON is a maneuver strategy computation tool used for transfers and rendezvous problems. It optimizes a fixed number of maneuvers that permit to reach a non-cooperative target, within a fixed phasing duration and fulfill certain relative rendezvous conditions. For this purpose, the same DRAGON algorithm has been implemented, since it has largely demonstrated its capabilities during ATV and GALILEO missions, being fast, robust and compliant with operational constraints. Nevertheless, the fact of porting DRAGON to JAVA has been also profited to provide new functionalities, such as the possibility to optimize more than six unknowns.

4.1 Problem statement

Equations of motion including rendezvous maneuvers can be stated as:

Setting:

- $E_{ch}(t^0) \coloneqq (a, e, i, \omega, \Omega, \nu)_{ch}^0$: chaser state vector at initial date
- $E_{ch}(t^f) \coloneqq (a, e, i, \omega, \Omega, \nu)_{ch}^f$: chaser state vector at rendezvous
- $E_{ta}(t^f) \coloneqq (a, e, i, \omega, \Omega, \nu)_{ta}^f$: target state vector at rendezvous
- V_{aim}: relative targeted state vector at rendezvous with respect to target.
- $\boldsymbol{T}$ operator: real function of transfer, which takes into account all forces and simulates the "real world"

Then, the rendezvous final condition can be expressed as:

$$E_{ta}(t^f) + V_{aim} = E_{ch}(t^f) = T(\overrightarrow{\Delta V_j}, \varphi_j, N_m, E_{ch}(t^0))_{j=1,N_m}$$

Where:

- N_m is the number of maneuvers to optimize
- $\overrightarrow{\Delta V_j}$ is the value of the maneuver j, with $\Delta V_j = \sqrt{{R_j}^2 + {T_j}^2 + {N_j}^2}$, being R_j, T_j and N_j the radial, tangential and out-of-plane ΔV_j components correspondingly
- φ_j is the location of the maneuver j, with $\varphi_j = 2\pi({N_{ch}}^j - 1) + \alpha_j$, being ${N_{ch}}^j$ the orbit number and α_j the argument of latitude where it is performed

The problem {P1} to calculate rendezvous maneuvers can be stated as a general optimization problem:

$$\text{Find } \boldsymbol{x} \coloneqq \{N_m, \overrightarrow{\Delta V_j}, \varphi_j\}_{j=1,N_m}$$
$$\text{To minimize } J(\boldsymbol{x}) \coloneqq \sum_{j=1}^{N_m} |\overrightarrow{\Delta V_j}|$$

And subjected to the following constraints:

- C_0, the rendezvous condition:
  $E_{ta}(t^f) + V_{aim} = T(\overrightarrow{\Delta V_j}, \varphi_j, N_m, E_{ch}(t^0))_{j=1,N_m}$
- $C_1(\varphi_j)$:
  - $\varphi_j \in ({\varphi_j}^{min}, {\varphi_j}^{max})$: allowable location to perform maneuver
  - $\varphi_{j+1} - \varphi_j \in [\Delta{\varphi_j}^{min}, \Delta{\varphi_j}^{max}]$: limitations for angular distance between two burns
- $C_2(\overrightarrow{\Delta V_j})$:
  - $\Delta V_j \in [\Delta V_j^{min}, \Delta V_j^{max}]$: limitations for maneuver modules
  - $R_j \in [R_j^{min}, R_j^{max}]$: limitations for radial maneuver components
  - $T_j \in [T_j^{min}, T_j^{max}]$: limitations for tangential maneuver components
  - $N_j \in [N_j^{min}, N_j^{max}]$: limitations for out-of-plane maneuver components

4.2 Solution approach

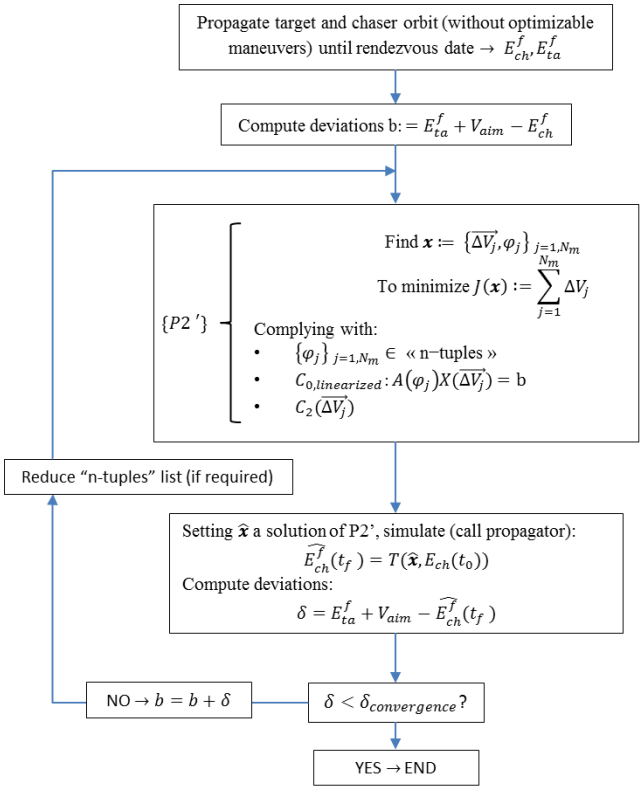
The problem {P1} is non-linear with non-linear and with non-convex constraints. Given its complexity, it is simplified to a near-optimal problem {P2}, considering the following assumptions:

- The number of maneuvers to optimize N_m is fixed. In addition, for each maneuver, the type and number of components to optimize $N_{c_j} \in \{1,2,3\}$ is given.
- The research domain of maneuvers locations $\varphi_j \in ({\varphi_j}^{min}, {\varphi_j}^{max})$ is discretized using a given $\Delta\varphi_j$. A preprocessing is conducted in order to create a list of feasible maneuver locations combinations $\{\varphi_j\}_{j=1,N_m}$ complying with $C_1(\varphi_j)$, called "n-tuples" (see annexes for more details).
- **T** operator is simplified by a linearized operator **L**, derived from the Gauss equations. Thus, the final chaser state vector is computed as:

$$E_{ch}(t^f) \approx L\left(\overrightarrow{\Delta V_j}, \varphi_j, E_{ch}(t^0)\right)_{j=1,N_m} =$$
$$= E_{ch}(t^0) + A(\varphi_j)X(\overrightarrow{\Delta V_j})$$

- Trim maneuvers (fixed in location and value) can be added to take into account operational constraints and scenario robustness.

Then, problem {P2} can be solved by a standard iterative process:



Problem {P2'} requires solving a linear system $AX = b$ iteratively, $A \in \mathbb{R}^{n_e \, x \, m}, X \in \mathbb{R}^{m}, b \in \mathbb{R}^{n_e}$, for all available "n-tuples", consisting of n_e equations and m unknowns. The number of equations will depend on the number of orbital parameters to reach demanded by the user:

Table 3: JDRAGON problem types as a function of the orbital parameters[6] to reach.

Problem type	Parameters	# Equations (n_e)
Transfer 2D	a, e_x, e_y	3
Rendezvous 2D	a, e_x, e_y, τ	4
Transfer 3D	a, e_x, e_y, i, Ω	5
Rendezvous 3D	$a, e_x, e_y, \tau, i, \Omega$	6

On the other hand, the number of unknowns m will depend on the maneuver scenario defined by the user:

$$m = \sum_{j=1}^{N_m} N_{c_j}$$

Therefore, the linear system to solve will be:

Linear System type	# Solutions
Undetermined ($n_e > m$)	0
Determined ($n_e = m$)	1
Overdetermined ($n_e < m$)	∞

For the undetermined linear system case, a least squares standard approach has been implemented which minimizes the sum of squared residuals, $\|AX - b\|$ and whose closed-form solution is $\widehat{\boldsymbol{X}} = (A^t A)^{-1} A^t b$. Then, for the determined linear system, a unique solution exists $\widehat{\boldsymbol{X}} = (A)^{-1} B$.

Finally, for the overdetermined linear system case two options have been implemented, letting the user to choose among them:

	Pseudo-inverse	**Minimization of a Sum of Norms**
Minimization criterion	$\sum_{j=1}^{N_m} \left\|\overrightarrow{\Delta V_j}\right\|^2$	$\sum_{j=1}^{N_m} \left\|\overrightarrow{\Delta V_j}\right\|$
Resolution	Analytical $\widehat{\boldsymbol{X}} = A^t (A A^t)^{-1} b$	Iterative
Performance	Fast	Slow (~x25 wrt pseudo-inverse)

Both methods present its advantages and disadvantages. From a fuel minimization point of view, the minimization of a sum of delta-Vs will be always more optimal. The minimization of a sum of Euclidean norms is a non-differentiable and non-linear optimization problem. Nevertheless, given its properties, it can be transformed into a differentiable linear problem with quadratic constraints and thus, it can be solved by a Sequential Quadratic Programming method (SQP). However, from the computational cost point of view, the pseudo-inverse is largely faster, since its solution is retrieved analytically. The tests performed with both methods show that, while the pseudo-inversion permits to manage yet a typical DRAGON case with 1 million of "n-tuples", the minimization of a sum of norms should be used for, at much, fifty thousand combinations, in order to obtain solutions in similar times. This new approach opens new optimization possibilities but forces the user to better precise the maneuver locations.

[6] Considering near-circular, circular orbital parameters are used, being τ: along-track distance

4.3 Example I: ATV3 – Johannes Kepler

Figure 5: ATV approaching ISS for docking (Credit: ESA/NASA)

The Automated Transfer Vehicle was an unmanned space transport vehicle whose mission was to contribute to the logistic servicing of the ISS. It was a European Space Agency (ESA) funded program: the spacecraft was designed and built by Airbus Defense and Space and operated from Toulouse, by CNES, at the ATV Control Center (ATV-CC). A total of five missions were conducted achieving a 100% success: first ATV- 1 *Jules Verne* mission was sent in March 2008, while last ATV-5 Georges Lemaitre launched end of July 2014 [6], [7].

From now on, we will focus now on the phasing phase of the ATV-3 *Johannes Kepler*, which covers the orbital flight from the insertion point IP (260km x 260 km, 51.6 °) until the interface point at the vicinity of the ISS ($S_{-1/2}$, 39km behind and 5km below ISS). The maneuver strategy to study has been:

- A *Transfer to the Phasing orbit* cycle (TP): 2 maneuvers to optimize (tangential, out-of-plane components)
- 2 corrective *Mid-Course* (MC) cycles of 2 maneuvers each (ΔV = 4x1.5 = 6m/s)
- 3 *Transfer to ISS Vicinity* (TIV) cycles of 2 maneuvers each: TV1 to optimize (tangential components), TV2 fixed (ΔV = 2x6 = 12 m/s) and TV3 fixed (ΔV = 2x3= 6 m/s)
- 1 *Transfer to the Interface* orbit (TIF) cycle consisting of 1 maneuver (ΔV = 1.2m/s)

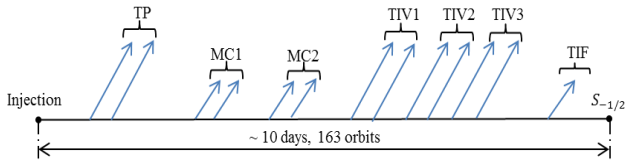


Figure 6: ATV-3 simplified phasing strategy

This problem is a rendezvous in 3D (Table 3), with 6 unknowns, 4 for the TP cycle and 2 for the TIV1 cycle. The aim has been to compare the different methods, bearing in mind the results found by the reference DRAGON Fortran version.

The computed solution (Table 3) differs 0.2% in terms of DV with respect to Fortran version, which is acceptable bearing in mind the different numerical propagators used.

Table 4: Results obtained with JDRAGON

Maneuver strategy to optimize	TN-TN-T-T
#JPSIMU calls	8
Total DV (m/s)	55.27
CPU time (s), including ephemeris generation	1min 54s

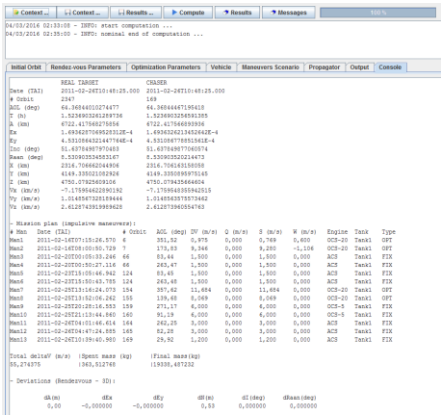


Figure 7: GUI JDRAGON - output frame appearance

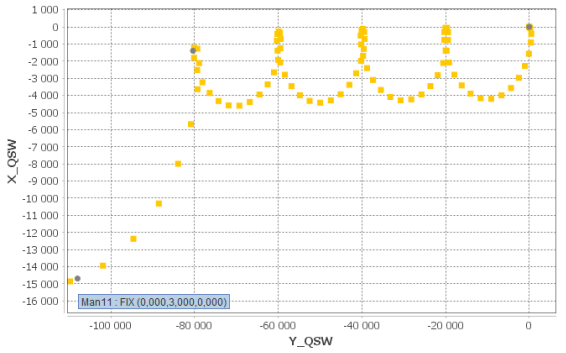


Figure 8: TIV3 and TIF phase containing the last 3 maneuvers in target QSW local frame (JDRAGON)

On top of that, alternative strategies have been studied trying to optimize the two out-plane components of the TIV cycle.

Table 5: Alternative strategies results (TN-TN-TN-TN)

Method	Pseudo-Inverse	Min sum norms
"n-tuples"	194481	28561
#JPSIMU calls	7	7
Total DV (m/s)	55.04	55.03
CPU time	2min	3min 11s

As expected, solutions are very little improved since Ariane-5 was in charge of injecting the ATV at the right plane (taking into account Ω drift until RDV date) so additional out-of-plane maneuvers do not play an important role.

4.4 Example II: Mango-Picard phasing

The IRIDES experiment (Iterative Reduction of Inspection Distance with Embedded System) consisted in approaching the PRISMA Mango spacecraft to the Picard non-cooperative spacecraft ([8], [9]), in order to demonstrate rendezvous and inspection technology. Originally, Mango took part in the Swedish PRISMA mission, aiming at demonstrating strategies and technologies for formation flying and rendezvous. On the other hand, Picard was a French satellite that became non-operational in April 2014, after taking more than one million images of the Sun, among other objectives. After Mango finished its nominal and extended mission phases, and since a considerable amount of fuel was remaining, it was decided to set the IRIDES experiment as its new objective.

Table 6: Orbital parameters derived from public TLE at 28/01/2013 (00h 20min)

Osculating parameters	Mango	Picard
Sma (km)	7138.4	7110.5
Ecc	0.004	0.001
Inc (deg)	98.28	98.24
Pa (deg)	-191.34	80.87
Raan (deg)	202.64	214.93

From now on, we will focus on the study of the phase prior to inspection, the phasing phase. Observing the orbital parameters, it is possible to notice that the phasing problem difficulty relies mainly on the large difference in the right ascension of the ascending node $\Delta\Omega = \Omega_{picard} - \Omega_{mango} \approx 12.2\text{deg}$. Moreover, due to J2 effects, this difference increases as the time goes by since $\dot{\Omega}_{picard} > \dot{\Omega}_{mango}$.

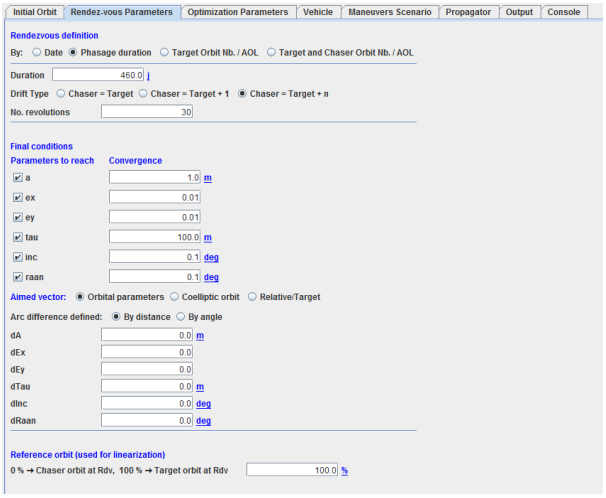


Figure 9: Rendezvous parameters (phasing duration, orbital parameters to reach, convergence thresholds and aimed vector are given)

For the study performed, we have imposed a duration of 460 days and a drift number of 30 (relative number of completed orbits performed by the chaser with respect the target).

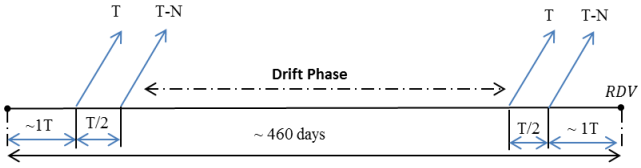


Figure 10: Maneuver strategy studied

The optimal strategy is to vary the semi-major axis of Mango, in order to have a favorable relative Ω drift. Thus, the maneuver strategy has been composed of two cycles. Each cycle formed by two maneuvers separated by 180deg, the first cycle for starting the drifting phase and the second one for stopping it. Moreover, all maneuver locations are researched within an interval of 360deg. Besides the classical strategy presented in Figure 10, we have performed parallel studies optimizing two additional out-of plane components (TN-TN-TN-TN), obtaining the following results:

Table 7: Method comparisons results

Method	Linear inversion	Pseudo-Inverse	Minimization of Norms
Strategy	T-TN-T-TN	TN-TN-TN-TN	TN-TN-TN-TN
DV (m/s)	51.67	51.54	50.86
CPU time	5min 39s	7min 50s	10min 36s
"N-tuples"	95319	95319	23919
#JPSIMU calls	4	6	4

This study has been conducted without considering the real propulsive properties of Mango, which would have forced to break down the 4 maneuvers into many equivalent small maneuvers. However, this example has been useful to show how, for rendezvous problems with important in-plane and out-plane maneuvers coupling, the new methods implemented improved the classical approach (0.3% and 1.6% reduction in terms of delta-V using pseudo-inverse and minimization of norms correspondingly).

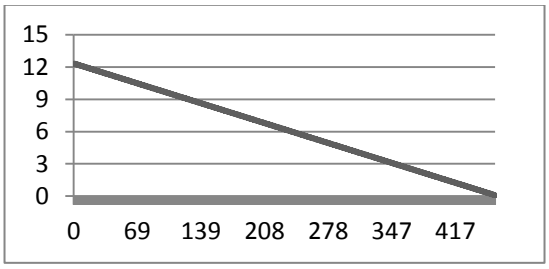


Figure 11: Time (days) VS ΔΩ (deg)

5. JOSCAR: END-TO-END SIMULATIONS

The aim of JOSCAR is to account for variables dispersions that could be experienced all along the flight. Based on a Monte-Carlo analysis, it permits to simulate the maneuvers update process performed on-ground all along the mission by computing end-to-end simulations with the Control Center in the loop. Consequently, it is a time consuming process, since multiple calls to JPSIMU and JDRAGON are required. It is important to understand how JOSCAR switches continuously between:

- *The "real world":* it is composed of the real parameters (force models, engines performance, chaser/target orbits and vehicle features). For each run of the Monte-Carlo simulation, they are generated by a dispersion of the nominal parameters.
- *The "predicted world":* the lack of knowledge of the real world. When performing maneuver computations the input data considered is sometimes uncertain: chaser/target restituted orbits, engines calibrations, atmosphere forecasts or vehicle features, among other parameters.

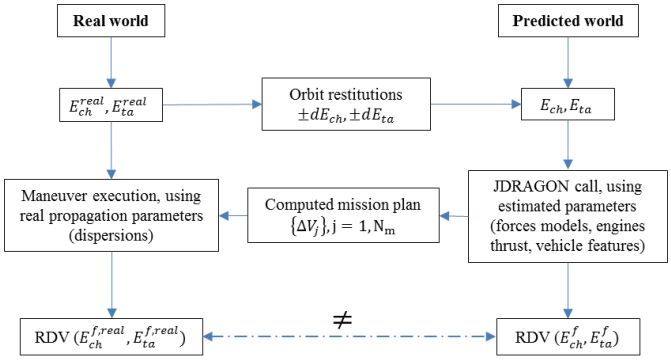


Figure 12: Example of JOSCAR simulation composed of three events: chaser orbit restitution, target orbit restitution and a maneuver strategy computation (calling JDRAGON)

Based on its Fortran predecessor, JOSCAR has been split into two modules: Genoscar and Exoscar.

5.1 Genoscar

This first module offers the possibility to define the End-to-End scenario to be simulated as well as the variables to disperse along the Monte-Carlo analysis. The scenario is defined by a sequential number of events, each of them positioned either by means of absolute parameters (date or number of orbit and argument of latitude) or by relative parameters with respect to other events. The possible events to define are:

- *Chaser orbit restitution:* dispersions can be Gaussian or uniform and defined by different orbital parameters (keplerian, circular, QSW[7] or TNW[8]). In addition, Gaussian dispersion can be defined either by the covariance matrix or by the correlation matrix and standard deviation vector.
- *Target localization*: similar to chaser orbit restitution with the difference that the dispersion can be also defined at rendezvous date.
- *Thrusters calibration*: usually placed after a maneuver has been performed, permits to estimate the engine thrust.
- *Maneuver computation*: a maneuver scenario is defined. When the event is reached, a JDRAGON computation is performed.
- *Change rendezvous point*: this event allows changing the active targeted rendezvous point. This feature was used, for instance, during ATV mission, where the sequence from insertion to parking point at ISS was followed by the sequence to reach the rendezvous interface.
- *Sequence of maneuvers*: a sequence of fixed maneuvers is triggered.
- *Atmospheric forecast*: permits to have an estimation of the atmospheric conditions (usually function of the solar activity).

In addition, other GUI panels permits to define the engines features (thrust, dispersions), different aimed target points, the atmospheric models settings (multiplicative factors, ballistic coefficients dispersion) as well as to customize the Monte-Carlo settings. Once the scenario is defined, Genoscar is launched and a file containing all random variables is generated, which will be used then by the following module, Exoscar.

5.2 Exoscar

This second module allows defining a set of data required for the simulation propagation as the chaser/target vehicle features, the force models or the attitude sequences. Contrary to the Fortran version, it has been decided to make the difference between the propagation models (forces, vehicle features, integrator) used in maneuver computations (at JDRAGON level) and the models used for "real word"

[7] QSW local frame : radial, tangential and out-of-plane directions

[8] TNW local frame: tangential, out-of-plane and radial directions

propagation. This feature could be interesting for instance, when simulating on-board maneuver computations, which usually make use of simplified models. This module is yet under development but, as the predecessor version, will permit to define a set of actions concerning the required output files.

6. CONCLUSIONS

This paper has presented JDRAGON/JOSCAR tools, written in JAVA. Based in their predecessor Fortran versions DRAGON/OSCAR, they are used for maneuver strategy computation and mission analysis purposes. Additionally, it has been necessary to develop JPSIMU, a numerical propagator that is required for JDRAGON/JOSCAR algorithm computations. Their development has been possible thanks to the SIRIUS project workbench and particularly PATRIUS, a powerful library that provides many flight dynamics low level functions. The development of JDRAGON/JOSCAR tools using JAVA language has shown significant advantages:

- *Faster development*: JAVA language is easier to write, compile and debug than other programming languages. Moreover, JAVA uses automatic memory allocation and garbage collection which improves the programmer productivity.
- *Object-oriented*: Development is reduced to create objects, manipulate them and make them work together. This has resulted in modular programs and reusable codes. For instance, many JPSIMU widgets (classes used for GUI) have been reused in JDRAGON/JOSCAR GUIs.
- *Platform-independent*: It is sure to obtain the same results when running tools either on Windows or Linux.
- *Robustness*: Early checking for possible errors. Unlike Fortran language, where many errors would first show up during execution time, JAVA compilers are able to detect many problems in advance.
- *Graphical User Interfaces*: By using GENIUS, the development of the GUI has been easier, thanks to its simplified approach.
- *New features*: After 20 years usage feedback, some functionalities from the predecessor tools have been erased, while new improvements have been implemented.

On the other hand, special attention has been put into the possible disadvantages that using JAVA language could have with respect to Fortran:

- *Validation:* JPSIMU has been largely validated against PATRIUS in order to dispose of a reliable numerical propagator. When comparing with PSIMU Fortran version some differences arose due to the different flight dynamics libraries used (mainly because of force models implementation, frames management and integrators used). However, these differences are not significant when running JDRAGON/JOSCAR tools, since they are capable of computing same delta-V budgets for varied types of missions such as ATV and GALILEO (with a precision better than 1mm/s when studying same n-tuples scenarios and same force models).
- *Performance:* Computation time relies mainly on numerical propagation performed at JSPIMU level. It is highly dependent on frames management settings or integrator tolerances. When comparing with Fortran version and for a same level of precision, it is possible to notice that JAVA is approximately twice slower. At JDRAGON level, no degradation is observed, JDRAGON/DRAGON computational times ratios are similar to those of JPSIMU/PSIMU. Typical problems as ATV or GALILEO problems last between one and two minutes, which continue to be acceptable.

Finally, beyond JDRAGON/JOSCAR tools, it is important to point out that there are other ongoing mission analysis tools redevelopments in JAVA such as CRASH (guided reentry), DOORS (controlled deorbit) and ELECTRA (risk reentry) [10].

7. ACKNOWLEDGEMENTS

The authors gratefully acknowledge Jean-François Goester (DCT/SB/MO) for his valuable technical support and paper review, Richard Epenoy (DCT/SB/MO), for his helpful assistance in optimization techniques and Vincent Ruch (DCT/SB/SP), for his time and help concerning PATRIUS issues.

8. REFERENCES

[1] P.Labourdette, A. Gaudel-Vacaresse, D.Carbonne *"Oscar/Dragon: Tools for maneuver strategy computation"*, 5th International Conference on Astrodynamics Tools and Techniques, ESTEC/ESA, The Netherlands, 29 may – 1 June 2012.

[2] Denis, C. and Tanguy, Y. "*The SIRIUS Flight Dynamics Library for the next 25 years*". 5th International Conference on Astrodynamics Tools and Techniques, ESTEC/ESA, The Netherlands, 29 may – 1 June 2012.

[3] Eclipse website: https://eclipse.org/ (February 2016)

[4] Orekit website: https://www.orekit.org/ (February 2016)

[5] Commons Math website (February 2016): https://commons.apache.org/proper/commons-math/

[6] Martinez-Alcalde, S., Labourdette, P., Goester, J.F., Delattre S., De Pasquale, E., "*Lessons learned from the phasing strategy design in the ATV program*", 25th International Symposium on Space Flight Dynamics, 19-23 October, 2015. Munich, Germany.

[7] Martinez, Santiago and al., "*High Reactivity Maneuver Design in ATV Missions*", 24th International Symposium on Space Flight Dynamics, 2014. Laurel, Maryland, US.

[8] Karlsson, T. and al., *"Prisma Irides: performance at the end of the drift phase & planned rendezvous experiments*", 9th International ESA Conference on Guidance, Navigation & Control Systems, 2-6 June 2014. Porto, Portugal.

[9] Karlsson, T. and al., *"Irides new rendezvous objectives for the Prisma mission"*, 64th International Astronautical Congress. Beijing, China.

[10] Pavero, P. and al., *"Space dynamic software ELECTRA"*, 6th International Conference on Astrodynamics Tools and Techniques, Darmstadt, Germany, 14-17 Mars 2016.

8. ANNEX: N-TUPLES DETERMINATION

Bearing in mind the discretization of the research maneuver locations domain, the aim now is to determine the whole sequence of feasible maneuver locations combinations, which from now on will be called the "n-tuples": $\left((\varphi_{j,k})_{j=1,N_m}\right)_{k=1,Nt}$ respecting $C_1(\varphi_{j,k})\ \forall\ k = 1, N_t$

Where N_m now is the total number of maneuvers (fixed or to optimize) and N_t is the total number of φ_j combinations complying with $C_1(\varphi_j)$.

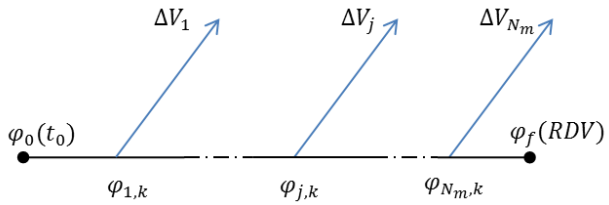


Figure 13: N-tuples kth combination

The simplest method to obtain the n-tuples (but not faster) would be to test, for all possible combinations of φ_j, if $C_1(\varphi_j)$ is fulfilled and if this is the case, save the combination. Since for the DRAGON general case it is possible to have more than one million combinations, it has been decided to implement an algorithm more efficient and therefore, less computationally expensive. The method implemented has been the following:

1) *Verifying input data given by the user*: Each maneuver is defined by the following parameters:

- The allowable[9] maneuver location ${L_{allowable}}^j := \{\varphi_j \mid \varphi_j \in [{\varphi_j}^{min}, {\varphi_j}^{max}]\}$
- The research step: $\Delta\varphi_j$, with $\Delta\varphi_j \geq 0$
- The limitations to respect in angular distance with respect the following maneuver $\{{\Delta\varphi_j}^{max}, {\Delta\varphi_j}^{min}\}$, with ${\Delta\varphi_j}^{max} \geq {\Delta\varphi_j}^{min} > 0$

2) *Erasing not feasible locations*: Due to the fact that not all $\varphi_j \in L_{allowable}(j)$ verify $C_1(\varphi_j)$, a pre-processing method is conducted in order to compute ${L_{feasible}}^j$:

$${L_{feasible}}^j := \{\varphi_j \in {L_{allowable}}^j \mid \exists\, \varphi_{j+1} \in {L_{feasible}}^{j+1}, {\Delta\varphi_j}^{max} \geq \varphi_{j+1} - \varphi_j \geq {\Delta\varphi_j}^{min}\}$$

With ${L_{feasible}}^{N_m} = {L_{allowable}}^{N_m}$. This can be graphically deduced as:

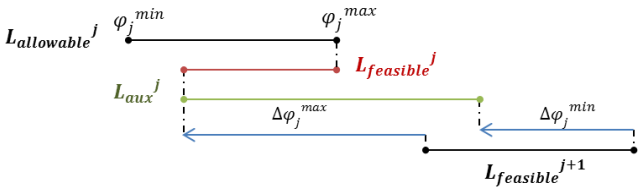


Figure 14: Graphical deduction of ${L_{feasible}}^j$

And it can be formalized as ${L_{feasible}}^j = {L_{aux}}^j \cap {L_{allowable}}^j$, with ${L_{aux}}^j$:

[9] Note that for fixed maneuvers : ${\varphi_j}^{min} = {\varphi_j}^{max}$ and $\Delta\varphi_j = 0$

$$L_{aux}{}^{j} \coloneqq \{\varphi_j \mid \varphi_j \in [\min(L_{feasible}{}^{j+1}) - \Delta\varphi_j{}^{max}, \max(L_{feasible}{}^{j+1}) - \Delta\varphi_j{}^{min}]\}$$

3) *Discretizing the feasible locations:* Taking into account $\Delta\varphi_j$, a discretized version of $L_{feasible}{}^{j}$ is derived:

$$\overline{L_{feasible}{}^{j}} \coloneqq \{\varphi_j \in L_{feasible}{}^{j} \mid \varphi_j = \min(L_{feasible}{}^{j}) + i\Delta\varphi_j, \; i = 0, \dots, i_{max}{}^{j}, i_{max}{}^{j} \in \mathbb{N}_0\}$$

It is important to bear in mind that, by definition, all elements φ_j belonging to $\overline{L_{feasible}{}^{j}}$ verify $C_1(\varphi_j)$ for at least one φ_{j+1}.

4) *Determining feasible indexes*: Now that the list $\overline{L_{feasible}{}^{j}}$ is discrete, all feasible φ_j locations are automatically defined by a certain index $i \in K^j$:

$$K^j \coloneqq \{i \in \mathbb{N}_0 \mid \min(L_{feasible}{}^{j}) + i\Delta\varphi_j \in \overline{L_{feasible}{}^{j}}\}$$

This set can be translated into a sequence of i, from $i = 0 = \min(K^j)$ to $i = i_{max}{}^{j} = \max(K^j)$:

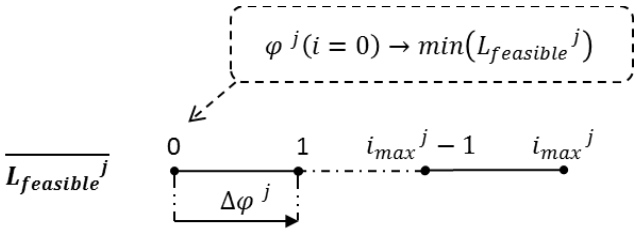


Figure 15: Maneuver j possible locations

The aim now is to determine, for each index $i \in K^j$, which subset $K_i{}^{j} \subseteq K^{j+1}$ of indexes $i' \in K^{j+1}$ verifies $C_1(\varphi_j(i))$. By definition of K^j this subset always exists and it is not empty. Furthermore, given the properties of C_1, this subset is continuous on $\mathbb{N}_0$, so it can be defined by two indexes:

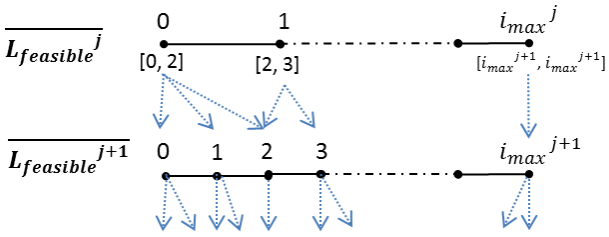


Figure 16: Example of feasible indexes representation

5) *N-tuples*: Finally, the last step is to compute all possible indexes combinations $\left((index_{j,k})_{j=1,N_m}\right)_{k=1,Nt}$. This has been done thanks to the implementation of a recursive algorithm, not detailed here, which efficiently combines the $K_i{}^{j}$ subsets. For a given list of indexes $\{i_1, i_2, \dots i_{N_m}\}$ the corresponding list of maneuvers locations $\{\varphi_1, \varphi_2, \dots \varphi_{N_m}\}$ is computed as $\varphi_j = \min(L_{feasible}{}^{j}) + i\Delta\varphi_j, j = 1, N_m$. Note that finally in terms of CPU memory, the n-tuplets could be saved as integers (and not doubles) which, for the general case of more than one million combinations, would be desirable.