Sensibility Analysis - ERT Simulator
Semester Project

Frédéric BERDOZ
Master Student in Computational Science and Engineering
Supervised by Dr Philippe MÜLLHAUPT

CONTENTS

I. Introduction

The EPFL Rocket Team (ERT) is a student association whose objective is the development and production of a Rocket capable of taking part in various competitions, in particular the *Spaceport America Cup*. Rocket science is notoriously known to be one of the hardest engineering fields due to its interdisciplinary character (avionics, fluid mechanics, dynamics, material science, etc.) and the high cost of its experiments. With the rise of computational science in the last few decades, however, it has been possible to run large numbers of simulations in order to decrease the number of physical experiments necessary in the development of aircraft.

The ERT has of course taken advantage of the field and has developed its own fully functional simulator in MATLAB®. The predictions of the simulator are usually within 5% of the physical experiment, which is already quite satisfactory. However, there might be room for improvement and a thorough analysis of the *simulator* itself has never been carried out. The main goal of this project is to assess the different types of inaccuracies that can arise in such a simulator, quantify them, and propose solutions in order to control them.

The core of this analysis will consist of a Sensitivity Analysis (SA), which aims at quantifying the internal dynamics of the underlying mathematical model. In addition to that, this report also aims to summarize the main concepts that have to be taken into consideration when developing a simulator (i.g. numerical integration with MATLAB®).

The code developed in this project can be found on the git repository of the ERT:

https://github.com/EPFLRocketTeam/2018_SP_FLIGHT_SIM-CONTROLLER.git,

in the sub-directory *Sensibility Analysis*.

II. Preamble

A simulator is basically a mathematical model that has inputs and outputs. Its accuracy depends largely on the quality of the assumptions that have been made in order to reduce the physical phenomenon to a mathematical expression. Additional inaccuracies can also stem from the resolution of that said mathematical expression. In order to quantify these concepts, consider the following notation:

- $\mathcal{X}$: Set of inputs (k-dimensional).
- $\mathcal{Y}$: Set of outputs (one-dimensional).
- $\mathbf{X}_e \in \mathcal{X}$: Exact inputs (generally unknown).
- $\mathbf{X} \in \mathcal{X}$: Measured inputs (generally inexact).
- $\mathcal{S} : \mathcal{X} \to \mathcal{Y}$: Physical system representing reality (generally unknown).
- $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$: Mathematical model based on the physical assumptions made using the current knowledge about $\mathcal{S}$.
- $f : \mathcal{X} \to \mathcal{Y}$: Solvable mathematical expression of $\mathcal{F}$, i.e. the simulator.

With this notation, one can easily detect the different sources of inaccuracies. Suppose you want to run a physical experiment $\mathcal{S}$ with exact inputs $\mathbf{X}_e$. Since both $\mathcal{S}$ and $\mathbf{X}_e$ are physical, the real output of the experiment will be $Y_e := \mathcal{S}(\mathbf{X}_e)$, whereas the output of the simulator will be $Y := f(\mathbf{X})$. The following expression illustrates the different approximations that are made in order to relate Y_e and Y:

$$Y_e = \mathcal{S}(\mathbf{X}_e) \overset{\text{(i)}}{\approx} \mathcal{F}(\mathbf{X}_e) \overset{\text{(ii)}}{\approx} f(\mathbf{X}_e) \overset{\text{(iii)}}{\approx} f(\mathbf{X}) = Y. \tag{1}$$

(i) **Conceptual inaccuracies**, i.g. inaccurate modeling of the drag, neglecting viscosity, etc.
(ii) **Numerical inaccuracies**, i.g. numerical integration with large time step, round-off errors, etc.
(iii) **Sensibility inaccuracies**, i.e. how the outputs of the simulator are affected by the inaccuracies of the inputs.

Conceptual inaccuracies are often responsible for a dominant fraction of the total error, but they are also often used as a scapegoat for the two other types of approximation. This is why a better understanding of numerical and sensibility inaccuracies is crucial in order to improve any simulator. This is exactly what this project aims to do.

III. Numerical Stability

This chapter aims to provide a simple numerical stability analysis of the ERT simulator. It is by no mean an extensive analysis, but it presents the fundamental concepts of numerical stability that have to be taken into considerations when developing a simulator.

A. Numerical Stability Theory

Any 3-dimensional classical dynamic problem of a single body boils down to the resolution of the following ordinary differential equation (ODE):

$$\dot{\mathbf{y}}(t) = \mathbf{G}(\mathbf{y}(t), \mathbf{u}(t), t), \qquad \mathbf{y}(t_0) = \mathbf{y}_0. \tag{2}$$

where $\mathbf{y}$ is the state of the system and $\mathbf{u}$ the input (given or tuned by a controller). In addition, when the system is time independent (or weakly time dependent), i.e. when $\mathbf{G}(\mathbf{y}, \mathbf{u}, t) \equiv \mathbf{G}(\mathbf{y}, \mathbf{u})$, and under the the assumption that $\mathbf{G}$ is differentiable at $\mathbf{y}_0$ and $\mathbf{u}_0 := \mathbf{u}(t_0)$, one can linearize Eq. (2) as follows

$$\dot{\mathbf{z}}(t) \approx \mathbf{A}\mathbf{z}(t) + \mathbf{B}\mathbf{v}(t) + \mathbf{c}, \tag{3}$$

where $\mathbf{z}(t) := \mathbf{y}(t) - \mathbf{y}_0$, $\mathbf{v}(t) := \mathbf{u}(t) - \mathbf{u}_0$, $\mathbf{A} := \nabla_{\mathbf{y}}\mathbf{G}(\mathbf{y}_0, \mathbf{u}_0)$, $\mathbf{B} := \nabla_{\mathbf{u}}\mathbf{G}(\mathbf{y}_0, \mathbf{u}_0)$ and $\mathbf{c} := \mathbf{G}(\mathbf{y}_0, \mathbf{u}_0)$. The range of validity of Eq. (3) depends heavily on the linearity of $\mathbf{G}$. In the case of the ERT simulator, $\mathbf{G}$ is not linear since the drag evolves as the square of the velocity. Moreover, the time independence assumption is not verified as the mass of the rocket varies with time. However, since the rate of variation of the mass is relatively low compared to other dynamics, one can still assume that Eq. (3) is valid over a reasonably large time interval (this will help to understand the limitations of the different numerical integrators).

The constant $\mathbf{c}$ and the the modified input $\mathbf{v}$ are case-dependent (in fact, for systems that have a steady state, $\mathbf{c}$ can be taken to be $\mathbf{0}$). Therefore, their influence on the numerical stability is hard to analyse in a general manner. With all these considerations in mind, it is natural to consider the following (decoupled) ODE in order to test the stability of a numerical method:

$$\dot{\mathbf{z}} = \mathbf{\Lambda}\mathbf{z}, \qquad \mathbf{z}(0) = \mathbf{z}_0, \tag{4}$$

with $\mathbf{\Lambda}$ the complex diagonal matrix containing the eigenvalues of $\mathbf{A}$, denoted λ_i. The solutions are then given by $z^{(i)}(t) = e^{\lambda_i t}$. It it clear that the analytical solution z_i is not stable if $\Re\mathrm{e}(\lambda_i) > 0$. Hence, the most relevant setup to analyse the numerical stability is when $\Re\mathrm{e}(\lambda_i) \leq 0$, $\forall i$. In this case, Eq. (4) is referred to as the Dahlquist test equation (DTE). An important property concerning the stability of a numerical method is the stiffness of the ODE. There is no formal definition of this property but in general, one says that an ODE is stiff if it has components that decay much more rapidly than others [1]. Alternatively, one can define the stiffness ratio as follows [2]. Let $\bar{\lambda} = \lambda_j$ with $j = \arg\max\{|\Re\mathrm{e}(\lambda_j)|\}$ and $\underline{\lambda} = \lambda_k$ with $k = \arg\min_i\{|\Re\mathrm{e}(\lambda_i)|\}$, and consider

$$SR := \frac{|\Re\mathrm{e}(\bar{\lambda})|}{|\Re\mathrm{e}(\underline{\lambda})|}. \tag{5}$$

This ratio quantifies the range of time scales that characterize the inner dynamics of the problem. If SR is large, this means that there are components of the system that decays more rapidly than others. This poses a problem for certain numerical methods, in particular explicit, since the time step h will have to be excessively small in order to keep up with high decays even though the exact solution is relatively smooth (since the component that evolves rapidly decays quickly). This will result in unnecessary computations and therefore inefficient (slow) algorithms. In other words, during the numerical integration of stiff ODEs, the size of h is constrained by the stability of the method and not by the desired accuracy. It is also important to note that $\mathbf{v}(t)$ might also have components that decay rapidly, which could also constrain the time step h in the numerical integrator. This is not taken into consideration in the Dahlquist test equation.

Now consider the two simplest numerical schemes for the autonomous ODE

$$\dot{z} = g(t, z), \qquad z(0) = z_0, \tag{6}$$

and the time discretization $t^{(n)} = n \cdot h$:

- **Explicit Euler**: $z^{(n+1)} = z^{(n)} + h \cdot g(t^{(n)}, z^{(n)})$, $z^{(0)} = z_0$,
- **Implicit Euler**: $z^{(n+1)} = z^{(n)} + h \cdot g(t^{(n+1)}, z^{(n+1)})$, $z^{(0)} = z_0$.

Applied to the Dahlquist test equations (4), these numerical schemes can be rewritten as follows:

- *Explicit Euler for the DTE*:

$$z_i^{(n+1)} = z_i^{(n)} + \lambda_i h z_i^{(n)} \quad \Rightarrow \quad z_i^{(n+1)} = (1 + \lambda_i h) z_i^{(n)} \quad \Rightarrow \quad z_i^{(n)} = (1 + \lambda_i h)^n z_{i0}, \tag{7}$$

- *Implicit Euler for the DTE*:

$$z_i^{(n+1)} = z_i^{(n)} + \lambda_i h z_i^{(n+1)} \quad \Rightarrow \quad z_i^{(n+1)} = \frac{1}{1 - \lambda_i h} z_i^{(n)} \quad \Rightarrow \quad z_i^{(n)} = \left(\frac{1}{1 - \lambda_i h}\right)^n z_{i0}. \tag{8}$$

Clearly, from Eq. (7), in order for the explicit Euler scheme to be stable with non zero initial conditions, it must satisfy

$$|1+\lambda_i h| \leq 1. \tag{9}$$

Similarly for the implicit Euler scheme and Eq. (8),

$$\left|\frac{1}{1-\lambda_i h}\right| \leq 1. \tag{10}$$

Inequality (9) constrains the size of h, whereas inequality (10) is always satisfied under the assumptions that $\Re\mathrm{e}(\lambda_i) \leq 0$. In other words, the implicit Euler scheme is stable for any h but not the explicit scheme. More generally, both the implicit and explicit Euler schemes belong to the class of s-stage Runge-Kutta methods, i.e. schemes that can be written as follows when applied to the ODE (6):

$$\begin{cases} z^{(0)} &= z_0 \\ p_i^{(n)} &= g\left(t^{(n)} + c_i h,\, z^{(n)} + h\sum_{j=1}^{s} a_{i,j} p_j^{(n)}\right), \quad i = 1, ..., s \\ z^{(n+1)} &= z^{(n)} + h\sum_{i=1}^{s} b_i p_i^{(n)}. \end{cases} \tag{11}$$

where the coefficients $a_{i,j}, b_i, c_i,\ i,j = 1, ..., s$ characterize the method[1]. Note that the method is explicit only if $a_{i,j} = 0$ for $j \geq i$. An important result is that any s-stage Runge-Kutta method applied to the Dahlquist test equation can be written as follows:

$$z_i^{(n+1)} = R(\lambda_i h) z_i^{(n)}, \tag{12}$$

with $R(z) = \frac{P(z)}{Q(z)}$ a quotient of complex polynomials of degrees at most s, called the stability function. In particular, for the explicit and implicit Euler methods, these stability functions are given by $R(z) = 1 + z$ and $R(z) = \frac{1}{1-z}$, respectively. Moreover, it can be shown that $Q(z) = 1$ for explicit methods, which means that $R(z)$ is simply a polynomial.

Now consider any Runge-Kutta scheme with stability function $R(z)$ and let $\Omega = \{z : |R(z)| \leq 1\}$ (sometimes called the domain of absolute stability). There are different kinds of stability that are often presented in the literature, the most important being

- *A-stability* (for *absolute stability*): If $\{z : \Re\mathrm{e}(z) \leq 0\} \subseteq \Omega$.
- *L-stability*: If A-stable and if $\lim_{|z|\to\infty} R(z) = 0$.

Since explicit methods have a polynomial stability function, they can never be A-stable. Hence, coming back to Eq. (4), the time step h will have to be chosen in a way such that $|R(\lambda_i h)| \leq 1$, $\forall i$. This constraint will be dictated by $\bar{\lambda}$, and if SR is large, h will be excessively small for the numerical approximation of the slowly decaying function $e^{\underline{\lambda} t}$.

It is important to note that even the stiffest ODEs can be solved using an explicit method, but since h will have to be very small, the cost of the computation will increase. Computational resources might not always be a restriction, but in the case of sensitivity analysis, speed is as important as accuracy (if not more). The goal of this section is to gauge the efficiency and accuracy of the different integrator implemented in MATLAB®in other to pick the best one for the problem at hand.

B. Different Integrator

There are infinitely many types of ODE solver for initial value problems, and each of them has its pros and cons. There are libraries that implement some of these in each programming language. Over the years, MATLAB® has natively implemented 8 of them, which are listed below. Since the documentation on these ODE solvers is quite rich, the goal of this section is not to provide an complete presentation of each scheme, but rather a short summary of their most important properties.

- **`ode45`**: This is the most common ODE solver and it should be used as when no additional information is known on the problem. It is 5th order single step[2] explicit Runge-Kutta method, meaning that the local discretization error is of order $\mathcal{O}(h^6)$ and the total error of order $\mathcal{O}(h^5)$. Like most advanced ODE solvers, `ode45` is an adaptive scheme, meaning that the step size h is adjusted at each step in order to achieve a given relative or absolute tolerance. However, since the exact solution is unknown, the true accuracy of the integrator cannot be measured. `ode45` approximates the error by running in parallel a 4th order Runge-Kutta method and by comparing the results obtained with the 5th order scheme. If the difference is significantly lower than the tolerance, the step size h is increased for the next step, and if it is larger, h is decreased. The two methods are optimized so that they minimize the number of function evaluations (see Dormand and Prince [3] for the exact algorithm). Lastly, `ode45` should not be used for stiff ODEs.
- **`ode23`**: This is a 3rd order single step explicit Runge-Kutta method that uses a 2rd order scheme to approximate the local accuracy and adapt the step size. It is in some case computationally more efficient than `ode45` when

[1]For the implicit Euler scheme: $s = a_{1,1} = b_1 = c_1 = 1$, and for the explicit Euler scheme: $s = b_1 = c_1 = 1, a_{1,1} = 0$.
[2]Meaning that the knowledge of $z^{(n)}$ is sufficient to compute $z^{(n+1)}$.

the error tolerance is large, and should only be used for non-stiff ODEs (see Bogacki and Shampine [4] for the exact algorithm).

- **`ode113`**: This is a multi-step and explicit Adams-Bashforth-Moulton predictor-corrector solver, meaning that it uses several previous results to compute the solution at the next time step. Its order can vary from 1 to 13 and it needs less function evaluation than traditional Runge-Kutta methods, which means that it can be useful when the error tolerance is really low. As for `ode45` and `ode23`, it should not be used when the problem is stiff (see Hayes [5] for the exact algorithm).
- **`ode15s`**: This is a variable order (1st to 5th order) multi-step implicit method. It is an implementation of the Numerical Differentiation Formulas (NDF) and it is the general purpose integrator for stiff problems. More details can be found in Shampine and Reichelt [6].
- **`ode23s`**: This is an single step 2nd order implicit Runge-Kutta method that adapts the step size using an 3rd order scheme. It is useful in the case of stiff problems (see Rosenbrock [7] for the exact algorithm).
- **`ode23t`**: This is an single step method that implements the trapezoidal rule (see Hosea and Shampine [8] for the exact algorithm). It is useful when numerical damping[3] is to be avoided, and it can also be used when the problem is moderately stable.
- **`ode23tb`**: This is a slight modification of `ode23t` in which the backward differentiation formula is used (see Hosea and Shampine [8] for the exact algorithm).
- **`ode15i`**: This one is only useful when solving fully implicit ODEs of the form $g(t, y, \dot{y}) = 0$ and will not be treated in this report. It is listed here for the sake of completeness.

C. Benchmark Strategy

First of all, in order to simplify the analysis, the same integrator is used for the five parts of the flight (rail, propulsion, ballistic, drogue parachute, main parachute). The results presented in this report were obtained by sampling at random a set of $N = 50$ inputs $\{\mathbf{X}^{(j)}\}_{1\leq j\leq N}$ uniformly in $[0.95\,\mathbf{x}_0 \quad 1.05\,\mathbf{x_0}]$, with $\mathbf{x}_0$ the current parameters of Bella Lui 2 along with a standard environment, and by running 7 simulations for each $\mathbf{X}^{(j)}$ (one for each ODE solver presented in Section III-B).

In order to identify which phase is computationally more expensive, the time of computation is monitored for each phase. The corresponding results are shown in Figure 1 and in Table I. Concerning the accuracy, since the exact solution is not known, only the relative accuracy between the different solvers can be monitored. One can argue that the most relevant metric for accuracy is the position of the rocket at apogee and at landing. Indeed, if those positions are accurate, it is very likely that other quantities like velocity and acceleration are also accurate throughout the whole flight. The results presented in Figure 2 and in Table II were obtained as follows: For each solver and for each input $\mathbf{X}^{(j)}$, the 3-dimensional position at apogee and landing were stored. The quantity that is presented here is the average distance between the position obtained using a given solver and the 6 other positions obtained using the other solvers. These quantities are obviously correlated but they allow to see if a given solver gives results that are consistently different from the others.

D. Results

Concerning the runtime of the simulations[4], one can see on Figure 1 that the standard integrator `ode45` yields the best results in average. This mean that in most cases, the problem is not stiff (or not stiff enough to render `ode45` inefficient), which also explains why `ode23s` performs relatively poorly. One can also observe on the same figure that the runtime of the simulations is slightly higher at the end. This trend is probably due to the cooling mechanisms of the CPU and should not be considered as part of the ODE solvers performance.

One can see in Table I that the most intensive computations happen consistently in the ballistic and propulsion phases. This is not surprising as the cost of the function evaluation is higher in these phases (drag modelling, large forces, small time scales, etc.).

TABLE I
DISTRIBUTION OF THE COMPUTATIONAL LOAD

ODE solver	**Rail**		**Propulsion**		**Ballistic**		**Droge Descent**		**Main Descent**		**Total**	
	μ [%]	σ [%]	μ [%]	σ [%]	μ [%]	σ [%]	μ [%]	σ [%]	μ [%]	σ [%]	μ [s]	σ [s]
`ode45`	1.2	0.5	31.5	7.3	65.9	7.9	0.5	0.1	0.9	0.2	5.5	1.8
`ode23`	0.7	0.2	31.9	4.4	66.6	4.5	0.3	0.1	0.5	0.1	7.8	1.2
`ode113`	0.7	0.3	29.5	4.1	69.0	4.2	0.3	0.1	0.6	0.2	10.4	3.0
`ode15s`	0.5	0.3	31.3	4.1	67.4	4.1	0.3	0.1	0.5	0.2	11.9	3.6
`ode23s`	0.2	0.0	33.0	3.6	66.3	3.7	0.2	0.1	0.3	0.1	39.5	5.7
`ode23t`	0.7	0.3	33.6	4.2	64.8	4.3	0.4	0.1	0.5	0.2	7.4	1.7
`ode23tb`	0.4	0.1	31.7	3.2	67.0	3.2	0.4	0.2	0.5	0.2	13.5	2.4

[3]Artificial (and steady) drop of the system's total energy due to numerical approximations.

[4]The simulations were executed on a 2GHz Intel Core i5 (dual core) and the error tolerances were left at their default values.

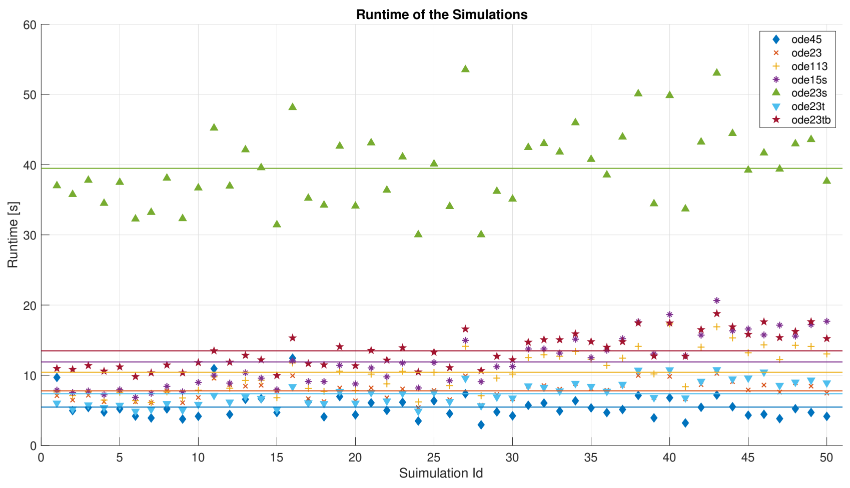

Fig. 1. Total runtime of each integrator.

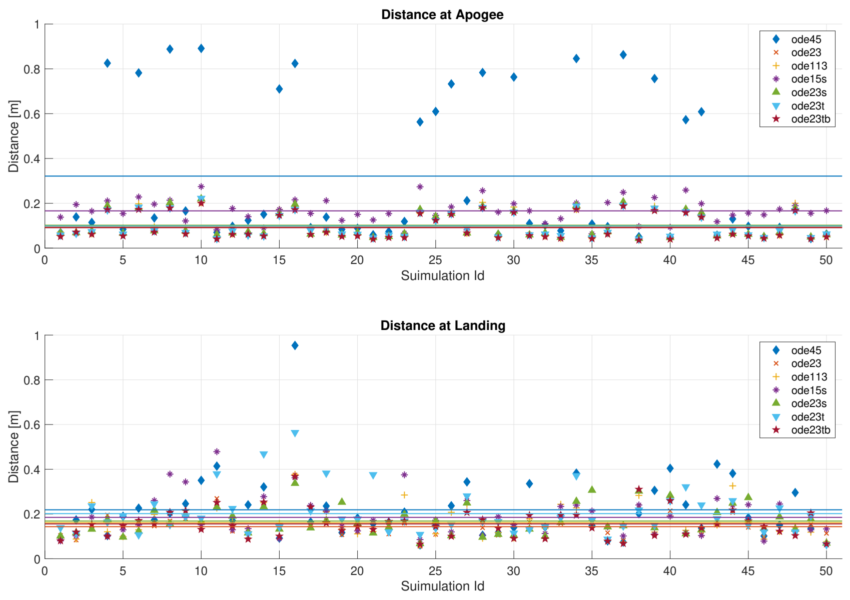

Fig. 2. Average distance of each integrator with respect to the others, at the apogee and at landing.

A perhaps more interesting quantity to analyse is the relative consistency of the different numerical schemes as presented in Figure 2 and Table II. One can observe that at apogee, `ode45` gives a position that is often relatively far from the other schemes. This artefact disappear when looking at the landing position since the distances between them also increase. However, even if the `ode45` method sometimes gives slightly different results, the average distance between the different integrators is always smaller than 1 meter, which is negligible compared to the scale of the problem (kilometers). This means that the numerical inaccuracies mentioned in Section II are most likely negligible.

TABLE II
RELATIVE ACCURACY OF THE ODE SOLVERS

ODE solver	**Apogee**		**Landing**	
	μ [m]	σ [m]	μ [m]	σ [m]
`ode45`	0.32	0.32	0.22	0.14
`ode23`	0.09	0.06	0.14	0.06
`ode113`	0.10	0.06	0.16	0.07
`ode15s`	0.17	0.05	0.18	0.09
`ode23s`	0.10	0.06	0.17	0.07
`ode23t`	0.10	0.05	0.20	0.10
`ode23tb`	0.09	0.05	0.16	0.06

IV. Sensibility Analysis

A. Theory

Sensibility analysis is precisely the study of sensibility inaccuracies and it is generally done by computing *sensitivity indices* (denoted S), i.e. values that quantify the sensibility of the model. Multiple techniques, some heavier than others, can be used to compute such indices. They usually fall into three categories [9]:

- *Screening* methods that evaluate qualitatively the sensitivity of the output w.r.t. each input variable. These are usually done in order to to rank the importance of each input variable (especially when the number of inputs is too large for a local or global sensitivity analysis).
- *Local Sensitivity Analysis* methods that evaluate quantitatively, around a fixed operating point $\mathbf{x}_0$, the sensitivity of the output w.r.t. each input variable x_i, i.e.

$$S_i := \frac{\partial f}{\partial x_i}(\mathbf{x}_0).$$

- *Global Sensitivity Analysis* methods that evaluate quantitatively the variability of the model relatively to the domain of variation of all its inputs. This is a more general approach since it does not depend on a fixed operating point. In addition, it allows to quantify the interactions between the parameters by computing the variability of the model w.r.t. a subset of parameters. The trade-off is that in order to obtain meaningful results, one must carefully tune the domain of variation of the input variables.

There are numerous different methods in each category, and each has its pros and cons. The best choice for the method to use largely depends on the problem at hand. In this report, this choice is governed by the SA map created by Rocquigny et al. [10] (see Figure 3). Since *a priori* the ERT simulator is neither linear nor monotone, and since it has approximately 50 inputs, a screening method will be performed in order to rank qualitatively the impact of each variable (Section IV-E1). After that, a Sobol analysis (Section IV-F1) will be carried out to quantify the global sensitivity of the model in a quantitative manner.

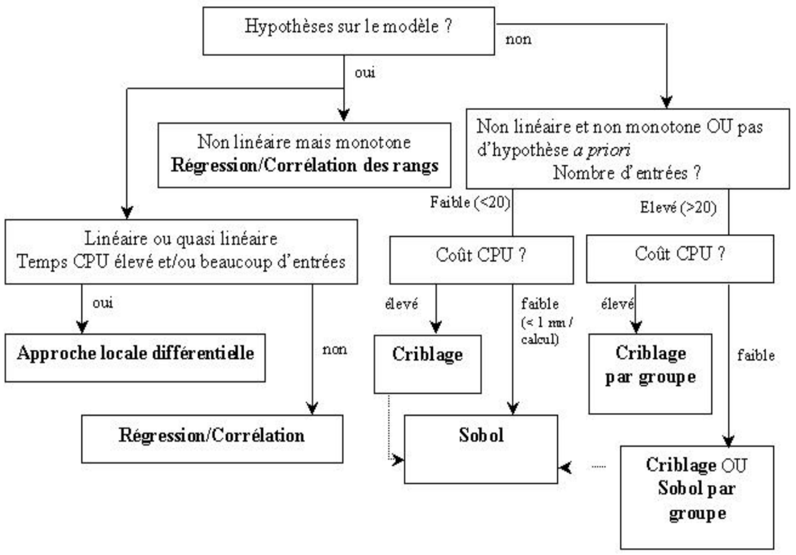
Hypothèses sur le modèle ?
non
oui
Non linéaire mais monotone
Régression/Corrélation des rangs
Non linéaire et non monotone OU pas d'hypothèse a priori
Nombre d'entrées ?
Faible (<20)
Elevé (>20)
Linéaire ou quasi linéaire
Temps CPU élevé et/ou beaucoup d'entrées
Coût CPU ?
Coût CPU ?
oui
élevé
faible (< 1 mn / calcul)
élevé
Approche locale différentielle
non
Criblage
Criblage par groupe
faible
Régression/Corrélation
Sobol
Criblage OU Sobol par groupe

Fig. 3. Decision map of the most appropriate sensitivity analysis depending on the model [10].

B. ERT Simulator Inputs

The ERT simulator has two types of inputs:

1) The parameters of the rocket,
2) The parameters of the environment.

Quantifying the sensibility of the model w.r.t. the parameters of the rocket will provide the ERT with additional design tools. They will be able to see which parameters are critical to the simulation and which ones have no effect. This can be an asset in the design process of the rocket and it could also allow the development of a simpler / faster simulator that has fewer parameters. On the other hand, computing the sensibility of the model with the environmental parameters is crucial as these parameters are often random. A high sensibility would suggest one of two things: either the simulator is not robust or the rocket is physically unstable, and both these conclusions are problematic. The different parameters will obviously evolve with each new version of the rocket and simulator. Some might be added and some might be retrieved. However, this report should provide the tools necessary to perform a new sensibility analysis each time the simulator has drastically changed. At this date, the most recent version of the rocket is Bella Lui 2 and its current sets of parameters are represented in Tables III, IV and V. They are classified as follows:

- Numerical parameters (Num): The true numerical inputs of the simulator, representing physical values. They are sampled from $\mathcal{X}$ using the notation introduced in Section II.
- Meta parameters (Meta): Parameters that give information about the other parameters (i.g. n_s is the size of **d** and **z**).
- Modal parameters (Mod): Parameters that modify the model $\mathcal{F}$ itself (i.g. ID$_{turb}$ that indicates which turbulence model should be used).
- Correction factors (Cor): Artificial variables that allow modifying manually certain intermediate quantities such as the drag coefficient. They act as multiplication factors and have no effect when set to 1.

Only numerical parameters will be considered in the sensibility analysis. However, every time a modal parameter is changed, a new SA should be run. Concerning meta parameters and correction factors, they are of little interest in the SA since they only serve as tools for the user and have no interesting impact on the mathematical model.

C. Practical Considerations

In order to match the standard use of the simulator by the ERT, the following practical considerations will be made for the sensibility analysis:

1) All the correction factors will be set to 1 and will remain constant throughout the SA.
2) All the integer parameters will stay at their original values as they are sometime used as array indices (i.e. $n_{fin} = 3$, $n_{lug} = 2$ and $n_{ab} = 0$). This simplifies greatly the implementation.
3) The mode of propulsion will be set to hybrid (i.e. $\mathbb{1}_{hybr} = 1$) since it is the case on Bella Lui 2 and will not change in the near future.
4) The general shape of the rocket will be fixed to the following standard form:
   - Conic nose,
   - Cylindrical (i.e. with constant diameter) body,
   - Inverted truncated conic engine mount.

   This implies the following: $n_s = 4$, $\mathbf{d} = [0, d_{max}, d_{max}, d_{min}]$ and $\mathbf{z} = [0, z_1, z_2, z_3]$. Moreover, in order to avoid having negative distances when sampling, the following parameters are defined: $d_d := d_{max} - d_{min}$, $z_{12} = z_2 - z_3$ and $z_{23} = z_3 - z_2$.
5) The nosecone will be on the rocket during the whole flight (i.e. $\mathbb{1}_{cone} = 1$) since it is the case on Bella Lui 2.
6) The wind will be modeled as multi-layered with measurements available at $n_l = 6$ different but constant heights $\mathbf{h} = [10, 100, 200, 500, 1000, 2000]$ m. In addition to that, the turbulence intensities $\mathbf{I}_{turb}$ will be set to zero so that the simulator remains deterministic for the SA. The additional randomness of the wind velocities would only add independent but constant noise to the sensitivities of the simulator, which is not wanted. The velocities and azimuths at the different altitude are denoted V_i and a_i, $1 \leq i \leq 6$, respectively.
7) Due to experimental limitations, the thrust function $T(t)$ is only known at time steps **t**. The resulting thrust curve **t**-**T** depends therefore heavily on the motor and fuel models ID$_{mot}$ and ID$_{fuel}$, respectively. For Bella Lui 2, the experimental thrust curve is

$$\mathbf{t} = [0, 0.02, 6.00, 6.35] \text{ s}, \qquad \mathbf{T} = [0, 2700, 1700, 0] \text{ N}.$$

   Thus, in order to stay in a similar regime while also allowing some variability for the SA, the thrust curve will be restrained as follows:

$$\mathbf{t} = t_{bt} \cdot [0, \frac{0.02}{6.35}, \frac{6.00}{6.35}, 1], \qquad \mathbf{T} = [0, T_1, T_2, 0],$$

   where t_{bt} denotes the burning time of the motor.

These considerations, along with the choice of $\mathcal{X}$ (see the next sections) make sure that are the configurations that are sampled are physically feasible (i.e. no negative values for distances, etc.).

TABLE III
ROCKET PARAMETERS

Parameter	Type	MATLAB® input	Symbol	Class	Unit
Number of stages	integer	`stages`	n_s	Meta	-
Diameters of the stages	array of n_s doubles	`diameters`	$\mathbf{d}$	Num	m
Distance from the tip of the rocket to the top of each stage	array of n_s doubles	`stage_z`	$\mathbf{z}$	Num	m
Number of fins	integer	`fin_n`	n_{fin}	Num	-
Distance from the rocket's tip to the fins' leading edge root	double	`fin_xt`	$x_{t,fin}$	Num	m
Span of the fins	double	`fin_s`	s_{fin}	Num	m
Root chord of the fins	double	`fin_cr`	$c_{r,fin}$	Num	m
Tip chord of the fins	double	`fin_ct`	$c_{t,fin}$	Num	m
Axial distance between the fins' leading edge root and tip	double	`fin_xs`	$x_{s,fin}$	Num	m
Fins' thickness	double	`fin_t`	t_{fin}	Num	m
Number of lugs	integer	`lug_n`	n_{lug}	Num	-
Exposed lug surface	double	`lug_S`	S_{lug}	Num	m^2
Rocket's empty mass	double	`rocket_m`	m_{roc}	Num	kg
Rocket's empty inertia	double	`rocket_I`	I_{roc}	Num	$\mathrm{kg\,m}^2$
Rocket's center of mass from rocket's tip	double	`rocket_cm`	$c_{m,roc}$	Num	m
Position of airbrakes from rocket's tip	double	`ab_x`	x_{ab}	Num	m
Number of airbrakes	integer	`ab_n`	n_{ab}	Num	-
Airbrakes' opening angle	double	`ab_phi`	ϕ_{ab}	Num	rad
Payload mass	double	`pl_mass`	m_{pl}	Num	kg
Product between the drag and the area of the main parachute	double	`para_main_SCD`	SCD_{main}	Num	m^2
Product between the drag and the area of the drogue parachute	double	`para_drogue_SCD`	SCD_{dro}	Num	m^2
Main parachute deployment event altitude	double	`para_main_event`	h_{main}	Num	m
Motor ID (Table IV)	string	`motor`	ID$_{mot}$	Meta	-
Boolean indicating if the motor is hybrid	boolean		$\mathbb{1}_{hybr}$	Mod	-
Fuel tank ID (Table IV)	string	`hybr` (Optional)	ID$_{fuel}$	Meta	-
Inter-motor distance	double		l_{int}	Num	m
Boolean indicating if the cone is on the rocket	boolean	`cone_mod`	$\mathbb{1}_{cone}$	Mod	-
Motor thrust multiplication factor	double	`motor_fac`	f_T	Cor	-
Multiplication factor on center of pressure position	double	`cp_fac`	f_{C_p}	Cor	-
Multiplication factor on normal lift coefficient derivative	double	`CNa_fac`	$f_{C_{N\alpha}}$	Cor	-
Multiplication factor on drag coefficient	double	`CD_fac`	f_{C_D}	Cor	-

TABLE IV
MOTOR PARAMETERS

Parameter	Type	MATLAB® input	Symbol	Class	Unit
Motor diameter	double	`moto_dia`	d_{mot}	Num	mm
Motor length	double	`motor_length`	l_{mot}	Num	mm
Motor delay (Not implemented)	string	`motor_delay`	-	-	-
Propellant mass	double	`propel_mass`	m_{prop}	Num	kg
Motor mass	double	`motor_mass`	m_{mot}	Num	kg
Times at which the motor thrust is known	array of doubles	`t`	$\mathbf{t}$	Meta	s
Thrust at times **t**	array of doubles	`T`	$\mathbf{T}$	Num	N

TABLE V
ENVIRONMENT PARAMETERS

Parameter	Type	MATLAB® input	Symbol	Class	Unit
Absolute temperature on the ground	double	`Temperature_Ground`	T_0	Num	K
Pressure on the ground	double	`Pressure_Ground`	p_0	Num	Pa
Humidity level on the ground	double	`Humidity_Ground`	x_0	Num	-
Altitude of the rocket launcher	double	`Start_Altitude`	h_0	Num	m
Latitude of the rocket launcher	double	`Start_Latitude`	-	Meta	°
Longitude of the rocket launcher	double	`Start_Longitude`	-	Meta	°
Rate of change of the temperature w.r.t. the altitude	double	`dTdh`	$\frac{\mathrm{d}T}{\mathrm{d}h}$	Num	$\mathrm{K\,km}^{-1}$
Rail length	double	`Rail_Length`	L_{rail}	Num	m
Rail inclination (from the vertical)	double	`Rail_Angle`	ϕ_{rail}	Num	°
Rail direction (azimuth)	double	`Rail_Azimuth`	α_{rail}	Num	°
Far wind velocity on the ground	double	`V_inf`	V_∞	Num	$\mathrm{m\,s}^{-1}$
Wind direction (azimuth) on the ground	double	`V_Azimuth`	α_{wind}	Num	rad
Turbulence intensity on the ground	double	`Turb_I`	I_{turb}	Num	-
Model of turbulence (none, Gaussian or Von Karman)	string	`Turb_model`	ID$_{turb}$	Mod	-
Boolean indicating if a multi-layered wind model is used	boolean		$\mathbb{1}_{mlw}$	Mod	-
Number of wind measurements (i.e. number of layers)	integer	`multilayerwind`	n_l	Meta	-
Heights of the measurements	array of n_l doubles	(Optional, overrides	$\mathbf{h}$	Meta	m
Far wind velocity at each layer	array of n_l doubles	$V_\infty,\alpha_{wind},I_{turb}$ and sets ID$_{turb}$	$\mathbf{V}_\infty$	Num	$\mathrm{m\,s}^{-1}$
Wind direction (azimuth) at each layer	array of n_l doubles	to Gaussian).	$\boldsymbol{\alpha}_{wind}$	Num	°
Turbulence intensity at each layer	array of n_l doubles		$\mathbf{I}_{turb}$	Num	-

D. *ERT Simulator Outputs*

The ERT Simulator produces a wide range of outputs (positions, velocities, accelerations, forces, coefficients, etc.) and each of those has a unique relation with the input parameters. Hence, one must select the outputs to monitor before performing the sensibility analysis. The three main domains of interest concerning the performance of a rocket are the following: altitude, stability and recovery. Concerning the altitude, the only relevant quantity is the apogee. Stability is harder to quantify but the most common measure is the static margin, i.e. the distance between the center of pressure and the center of mass, divided by the diameter of the rocket. A slightly more meaningful quantity to look at is the product between the static margin and the normal force coefficient C_{N_α}. If this product is low, the rocket is said to be unstable, and if it is too large, it is said to be over-stable. In this report, the quantity that will be analysed is the average of this product over the flight. Finally, for the recovery, the quantity that will be analysed is the landing drift (i.e. the distance between the launch and landing positions). Of course, the choice of the output to analyse can be adapted easily, and the code presented with the report allows to choose a wide range of outputs (12 at this point).

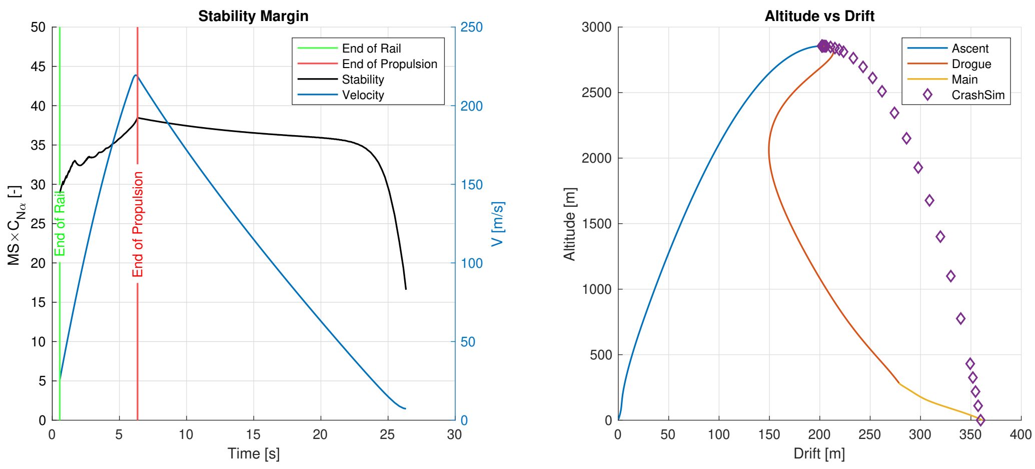

Fig. 4. Some of the ERT simulator outputs for a standard run, namely stability (left) and drift (right).

E. *Screening*

1) Theory: One of the most intuitive and widespread screening method is the *Elementary Effect* method (EE) first introduced by Morris [11] and summarized in [12]. The idea is to increase one and only one input variable and to monitor the change in the outputs. More precisely, one assumes that $\mathcal{X} = [a_1, b_1] \times ... \times [a_k, b_k]$, i.e. that the input domain is a k-dimensional rectangle. Then, the core of this method consists of scaling, shifting and discretizing the k-dimensional input space $\mathcal{X}$ into a p-level unitary grid $\bar{\mathcal{X}}_p = \{0, \frac{1}{(p-1)}, ..., \frac{(p-2)}{(p-1)}, 1\}^k$. After that, each input is incremented One-At-a-Time (OAT) by a step Δ chosen in $\bar{\mathcal{X}}_p$ and the following quantities are computed for an integer r:

$$d_i^{(r)}(\bar{\mathbf{X}}^{(r)}) := \frac{f^*(\bar{\mathbf{X}}^{(r)} + \Delta \mathbf{e}_i) - f^*(\bar{\mathbf{X}}^{(r)})}{\Delta}, \qquad \bar{\mathbf{X}}^{(r)} \overset{iid}{\sim} \textsc{Uniform}\left(\{0, \frac{1}{(p-1)}, ..., 1-\Delta\}^k\right) \tag{13}$$

where $(\mathbf{e}_i)_j = \delta_{ij}$ and $f^* := f \circ \mathbf{g}$ with $\big(\mathbf{g}(\bar{\mathbf{X}})\big)_i := (b_i - a_i)\bar{X}_i + a_i$. Finally, one computes the following measures (derived by Campolongo et al. [13]) in order to rank the importance of each input X_i w.r.t. the output Y:

$$\mu_i := \frac{1}{r} \sum_{l=1}^{r} |d_i^{(r)}(\bar{\mathbf{X}}^{(r)})|. \tag{14}$$

With a naive sampling approach, this method would need $2rk$ function evaluations in order to compute the k indices μ_i (see Eq. (13)). However, a clever sampling trick is presented in [11] where only $r(k+1)$ evaluations of f are needed. These $r(k+1)$ samples are obtained as follows. First, choose p even and set $\Delta = \frac{p}{2(p-1)}$. Secondly, sample independently and uniformly at random the following quantities:

- $\bar{\mathbf{X}}^{(1)}, ..., \bar{\mathbf{X}}^{(r)} \overset{iid}{\sim} \textsc{Uniform}\left(\{0, \frac{1}{(p-1)}, ..., 1-\Delta\}^k\right)$.
- $\mathbf{P}^{(1)}, ..., \mathbf{P}^{(r)} \overset{iid}{\sim} \textsc{Uniform}(\mathcal{P})$ with $\mathcal{P}$ the set of k-dimensional permutation matrices.
- $\mathbf{D}^{(1)}, ..., \mathbf{D}^{(r)} \overset{iid}{\sim} \textsc{Uniform}(\mathcal{D})$ with $\mathcal{D}$ is the set of k-dimensional diagonal matrices in which each diagonal element is either $+1$ or -1.

Then compute

$$\mathbf{B}^{(r)} = \mathbf{J}_{k+1,1}(\bar{\mathbf{X}}^{(r)})^\mathsf{T} + \frac{\Delta}{2}\left[(2\mathbf{B} - \mathbf{J}_{k+1,k})\mathbf{D}^{(r)} + \mathbf{J}_{k+1,k}\right]\mathbf{P}^{(r)},$$

where $\mathbf{J}_{m,n}$ is the matrix of size $m \times n$ filled with 1's and $\mathbf{B}$ is of size $(k+1) \times k$ with components

$$(\mathbf{B})_{ij} = \begin{cases} 1 & \text{if} \quad i > j \\ 0 & \text{if} \quad i \leq j \end{cases}$$

Finally, define the matrix

$$\mathbf{\Lambda} = \left[\begin{array}{cccc} \mathbf{B}^{(1)^\mathsf{T}} & \mathbf{B}^{(2)^\mathsf{T}} & \dots & \mathbf{B}^{(r)^\mathsf{T}} \end{array} \right]$$

whose columns are precisely the $r(k+1)$ samples that need to be evaluated through f.

2) *Results:* For this project, the input domain $\mathcal{X}$ was set to $\pm 10\%$ of the values in the current Bella Lui 2 design. Additionally, the number of repetition was set to $r = 50$. Larger values of r would have increased significantly the computational load since k is rather large in the problem at hand ($k = 53$). The last parameter p was set to $p = 100$, which implies that $\Delta \approx 0.505$. Hence, since the relative width of the input domain is 20%, the relative size of the effective OAT increment is approximately 10%. This is a rather large increment but smaller values failed to give significant results, probably because the number of repetition r is relatively low for such an analysis. Figures 5, 6 and 7 display the results of the EE analysis for the selected outputs. The blue boxes represent the 25 to 75 percentiles, the red lines represent the medians, the black whiskers represent the most extreme data points that are not considered outliers, and the red crosses are the outliers. The inputs are ranked by the average of their elementary effects (i.e. by the quantity defined in Eq. (14)), and the result of the EE analysis for the 9 other outputs that were monitored are presented in Appendix A.

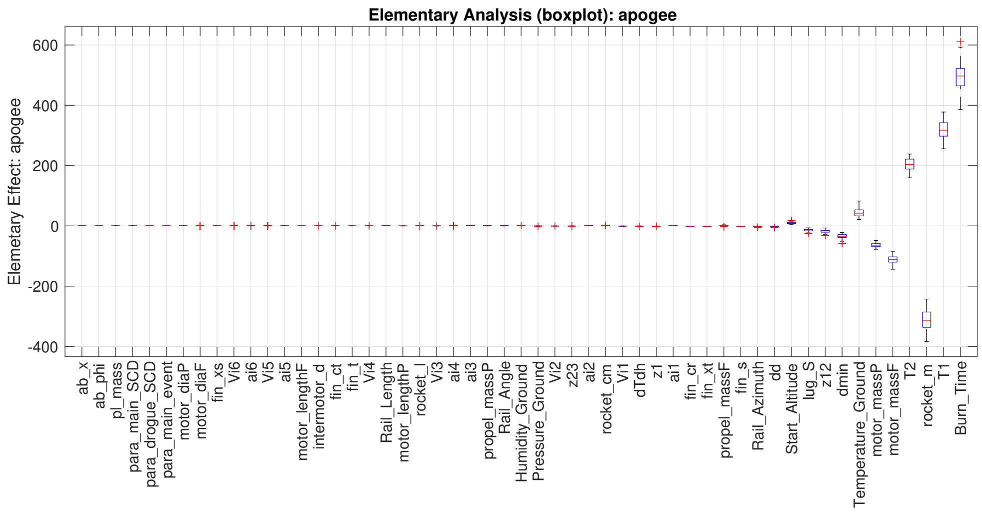

Fig. 5. Elementary Effect analysis for the apogee (in meters), using $p = 100$ and $r = 50$.

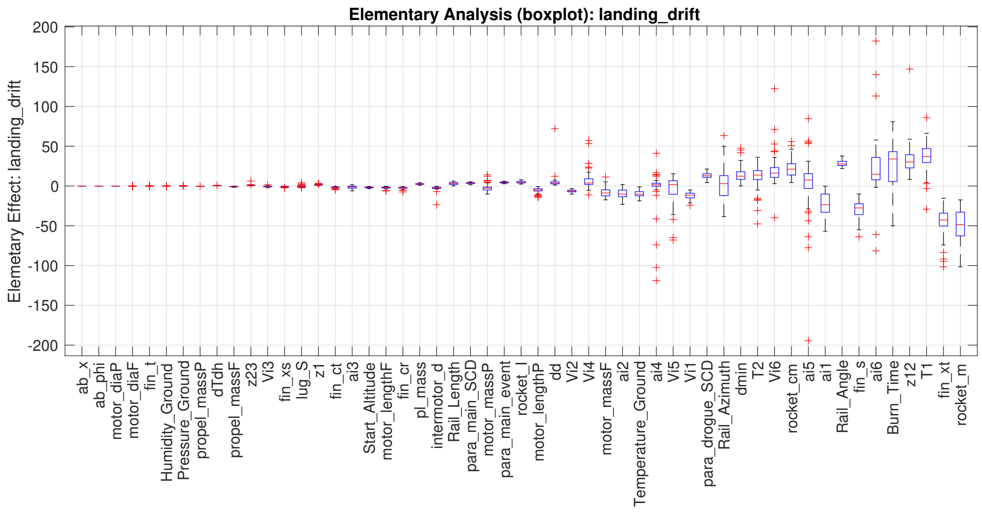

Fig. 6. Elementary Effect analysis for the landing drift (in meters), using $p = 100$ and $r = 50$.

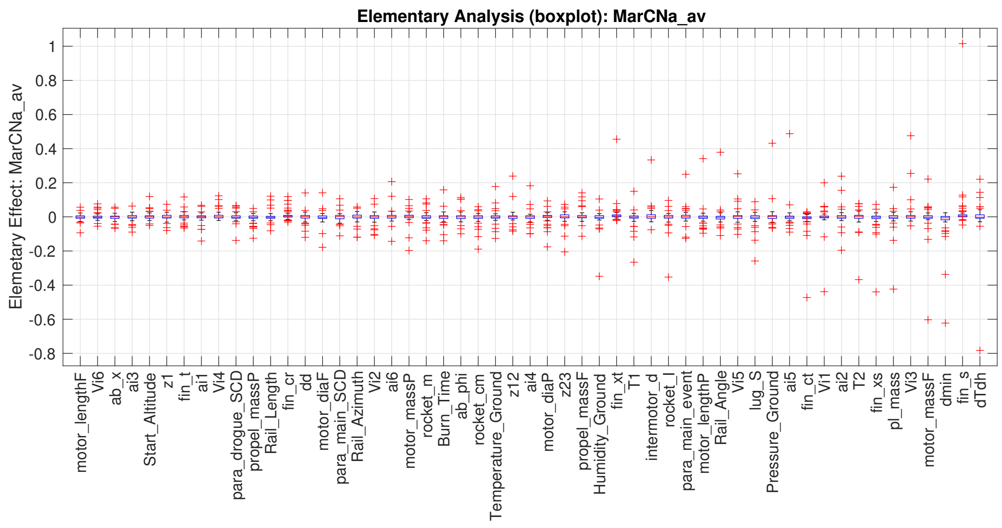

Fig. 7. Elementary Effect analysis for the stability criterion, using $p = 100$ and $r = 50$.

It is tempting to analyse Figures 5, 6 and 7 in a quantitative manner, for example by saying that the first thrust point T_1 is $1.5\times$ more influential on the apogee than the second thrust point T_2 (Fig. 5). Although it is not completely false, this $1.5\times$ factor depends largely on the choice of Δ in the EE analysis. It may be that if the effective increment was 5% instead of 10%, T_1 and T_2 would have the same effect on the apogee, or that T_1 would be $2\times$ more influential. Additionally, it may be less realistic to increase some parameters by 10% than others. For example, the ground temperature T_0 is in Kelvin, meaning that a 10% increase at ambient temperature represents a 30°C jump. This partially explains why the ground temperature ranks surprisingly high both in Figure 5 and 6. One could have set $\mathcal{X}$ differently for each parameter in order to mitigate this effect, but the overall graph would have been harder to read. Figure 7 fails to give any insight into the sensibility of the stability. The output quantity is too robust to the variation of one parameter. Other quantities were monitored but every attempt to quantify the stability of the rocket ended up in a similar plot. This echoes the general difficulty of characterizing the stability of a rocket.

Although the effects of the parameters is overall not surprising with what one would expect, a few exceptions deserve to be mentioned:

- $x_{t,fin}$ ranks surprisingly high in the sensibility of the landing drift. It is hard to imagine why an 10% increase in the position of the fin would have the same effect than a 10% increase in the mass of the rocket. Besides, one can observe on Figure 5 that $x_{t,fin}$ has little to no effect on the apogee.
- Most of the outliers in the landing drift are related to the wind parameters (a_i and V_i). After investigation, it was found that the wind model is prone to produce unrealistically high (or low) velocities near the apogee of the flight. This is due the to Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) used to interpolate the velocities between the data points. Figure 8 illustrates a few versions of the wind model obtained by varying each data point at random by at most 30% (a value that is typically used for $\mathbf{I}_{turb}$). A way to solve this problem would be to ensure that a data point is known near the expected apogee. A different interpolation technique, with less extreme border effects, could also be used.

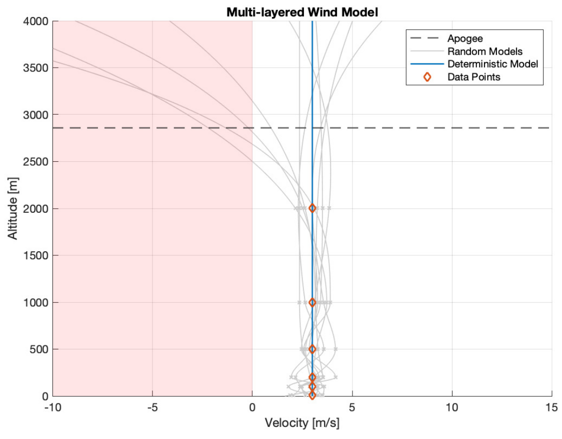

Fig. 8. Current Multi-layered wind model. The red data-points represent the base model used for the SA.

F. Sobol Analysis

1) Theory: As opposed to the *Elementary Effect* analysis presented by Morris [11], the sensitivity analysis developed by Sobol is a variance-based method. In other words, instead of looking at the average effect of one input parameter on the output, one look at the variance of the output with respect to the induced variance of the inputs. The quantities used to measures this are called Sobol indices (also called importance measure), and were introduced by Sobol [14]. They can be of k different orders for a model that has k inputs (i.e. they can quantify the joint effect of a subset of the parameters, which is not possible using the *Elementary Effect* method). The core of their derivation is presented below (see Jacques [15] for more details). Consider the model

$$Y = f(\mathbf{X}) \equiv f(X_1, ..., X_k),$$

and assume that the input variables are independent[5]. Now let $[k] := \{1, ..., k\}$ and consider the following notation: $\forall \phi \subseteq [k]$, $\mathbf{X}_\phi := \{X_i\}_{i\in\phi}$ (in particular, $\mathbf{X}_\emptyset = \emptyset$ and $\mathbf{X}_{[k]} = \mathbf{X}$). With this in mind, one can decompose f into the sum of 2^k functions (*Sobol decomposition*):

$$Y = f(\mathbf{X}) = \sum_{\phi\subseteq[k]} f_\phi(X_\phi)$$

where $f_\phi(\mathbf{X}_\phi)$ is defined recursively as follows:

$$\begin{aligned} f_\emptyset &= \mathbb{E}[Y], \\ f_i(X_i) &= \mathbb{E}[Y|X_i] - f_\emptyset, \quad \forall i \in [k], \\ &\vdots \\ f_{\phi\cup\{i\}} &= \mathbb{E}[Y|\mathbf{X}_{\phi\cup\{i\}}] - \sum_{\phi'\subset\phi\cup\{i\}} f_{\phi'}(\mathbf{X}_{\phi'}), \qquad \forall\phi \subset [k], \forall i \in [k] \setminus \phi. \end{aligned}$$

This allows to write the variance of the model as

$$V := \mathbb{V}[Y] = \sum_{\phi\subseteq[k]} V_\phi, \tag{15}$$

where

$$\begin{aligned} V_\emptyset &= \mathbb{V}[\mathbb{E}[Y]] = 0, \\ V_i &= \mathbb{V}[\mathbb{E}[Y|X_i]], \quad \forall i \in [k], \\ &\vdots \\ V_{\phi\cup\{i\}} &= \mathbb{V}[\mathbb{E}[Y|\mathbf{X}_{\phi\cup\{i\}}]] - \sum_{\phi'\subset\phi\cup\{i\}} V_{\phi'}, \qquad \forall\phi \subset [k], \forall i \in [k] \setminus \phi. \end{aligned}$$

Finally, the Sobol indices of order $|\phi|$ are defined a follows:

$$S_\phi := \frac{V_\phi}{V}, \qquad \forall\phi \subseteq [k]. \tag{16}$$

In particular, the 1st and 2nd order Sobol indices can be expressed as

$$\begin{aligned} S_i &= \frac{V_i}{V} = \frac{\mathbb{V}[\mathbb{E}[Y|X_i]]}{V}, && \forall i \in [k] \\ S_{ij} &= \frac{\mathbb{V}[\mathbb{E}[Y|X_i, X_j]] - V_i - V_j}{V} = \frac{\mathbb{V}[\mathbb{E}[Y|X_i, X_j]]}{V} - S_i - S_j, && \forall i, j \in [k], \quad i \neq j. \end{aligned}$$

An important property of these indices is that they sum up to one (which can be easily seen from Eq. (15)). Furthermore, one can define the total sensitivity index of variable X_i as follows:

$$S_{T_i} := \sum_{\substack{\phi\subseteq[k] \\ i\in\phi}} S_\phi. \tag{17}$$

It is important to note that there are 2^k distinct Sobol indices. Therefore, it is often unrealistic to compute all of them. A more practical approach is to compute the first order indices S_i and/or the total indices S_{T_i} for each input variable. Then, one can regroup all the higher order indices into the quantity $S_{H_i} := S_{T_i} - S_i$. Intuitively, the quantity S_i represents the fraction of the model variability that is solely due to the variability of X_i, and S_{T_i} represents the fraction of the model variability that is due to the variability of any subset of input containing X_i.

[5]This is a very restrictive assumption that is not satisfied in general (environmental parameters are obviously dependent, the burning time depends on the quantity of fuel, etc.). However, these dependence are often hard to identify and this is why all the parameters are considered independent in this report. The reader must analyse the results knowingly. To solve this problem, one could create a non-uniform probability density function on $\mathcal{X}$ in order to incorporate the dependence between the parameters when sampling. However, this should be done carefully since inducing wrong dependence would affect the results in an unknown manner.

There exist multiple techniques in order to estimate S_i and S_{T_i}, but they all boil down to the estimation of $V_i, V_{\sim i}$ and V (denoted $\hat{V}_i, \hat{V}_{\sim i}$ and $\hat{V}$, respectively). Once known, the estimators are then given by

$$\hat{S}_i = \frac{\hat{V}_i}{\hat{V}}, \qquad \hat{S}_{T_i} = 1 - \frac{\hat{V}_{\sim i}}{\hat{V}}. \tag{18}$$

Consider two independent sets of N samples $\{\mathbf{X}^j\}_{1 \leq j \leq N}$ and $\{\mathbf{Z}^j\}_{1 \leq j \leq N}$, obtained using a random or quasi-random scheme (see below for more details). The most common estimators are presented below (see Kucherenko & Song [16]).

- **Sobol' original estimators**: In the original derivation, Sobol estimated $V_i, V_{\sim i}$ and V as follows:

$$\hat{f}_\emptyset = \frac{1}{N} \sum_{j=1}^{N} f(\mathbf{X}^j), \tag{19}$$

$$\hat{V} = \frac{1}{N} \sum_{j=1}^{N} f(\mathbf{X}^j)^2 - f_\emptyset^2, \tag{20}$$

$$\hat{V}_i = \frac{1}{N} \sum_{j=1}^{N} f(\mathbf{X}^j) f(\mathbf{Z}^j + (X_i^j - Z_i^j)\mathbf{e}_i) - \hat{f}_\emptyset^2, \tag{21}$$

$$\hat{V}_{\sim i} = \frac{1}{N} \sum_{j=1}^{N} f(\mathbf{X}^j) f(\mathbf{X}^j + (Z_i^j - X_i^j)\mathbf{e}_i) - \hat{f}_\emptyset^2. \tag{22}$$

  The total number of function evaluation needed for these estimators is $N(2k+1)$, with N the number of sample (dictating the accuracy of the Monte Carlo integration) and k the number of input parameters.

- **Kurechenko's estimators**: In an attempt to improve the convergence, Kurechenko reused Eq. (19) and (20) but slightly improved the estimatiors for V_i and $V_{\sim i}$ as follows:

$$\hat{V}_i = \frac{1}{N} \sum_{j=1}^{N} f(\mathbf{X}^j) \left[f(\mathbf{Z}^j + (X_i^j - Z_i^j)\mathbf{e}_i) - f(\mathbf{Z}^j) \right], \tag{23}$$

$$\hat{V}_{\sim i} = \frac{1}{N} \sum_{j=1}^{N} f(\mathbf{X}^j) \left[f(\mathbf{X}^j + (Z_i^j - X_i^j)\mathbf{e}_i) - f(\mathbf{Z}^j) \right]. \tag{24}$$

  The number of function evaluations required for the Kurechenko's estimators is $N(2k+2)$.

- **Myshetzskay's estimators**: Sobol and Myshetzskay noticed that the term $f(\mathbf{X}^j)$ in Kurechenko's estimators can be replaced by $f(\mathbf{X}^j) - f_\emptyset$, yielding

$$\hat{V}_i = \frac{1}{N} \sum_{j=1}^{N} \left[f(\mathbf{X}^j) - f_\emptyset \right] \left[f(\mathbf{Z}^j + (X_i^j - Z_i^j)\mathbf{e}_i) - f(\mathbf{Z}^j) \right], \tag{25}$$

$$\hat{V}_{\sim i} = \frac{1}{N} \sum_{j=1}^{N} \left[f(\mathbf{X}^j) - f_\emptyset \right] \left[f(\mathbf{X}^j + (Z_i^j - X_i^j)\mathbf{e}_i) - f(\mathbf{Z}^j) \right]. \tag{26}$$

  The number of function evaluations required for this estimator is the same as the one for Kurechenko's estimators, i.e. $N(2k+2)$.

- **Double loop reordering**: A radically different approach consist of estimating $V_i = \mathbb{V}[\mathbb{E}[Y|X_i]]$ directly. With this method, for each X_i, the sample set $\mathbf{X}^{(j)}$ is sorted so that the $X_i^{(j)}$'s are in ascending order (i.e. $X_i^{(1)} \leq ... \leq X_i^{(N)}$). Then, the sample set is divided into $M = N/N_{bin}$ equally populated bins (N and N_{bin} must be chosen so that M is an integer, and as a rule of thumb, M should be chosen so that $M \approx \sqrt{N}$.). For each bin, $\mathbb{E}[Y|X_i \in \text{ bin } b]$, $1 \leq b \leq M$, is approximated by

$$\mu_i^b \approx \frac{1}{N_{bin}} \sum_{j=b}^{b+N_{bin}-1} f(\mathbf{X}^{(j)}).$$

  Finally, the estimator for $V_i = \mathbb{V}[\mathbb{E}[X|X_i]]$ is given by $\frac{1}{M}\sum_{b=1}^{M}(\mu_i^b)^2 - f_\emptyset^2$. The main advantage with this scheme is that the number of function evaluation is independent of the number of input parameters. This allows to increase N in order to have more accurate estimators. The main drawback of this technique is that it cannot be used to compute total sensitivity indices.

There are numerous sampling schemes when it comes to the approximation of integrals (in this case expected values and (co)variances) with averages, but the most common ones are the following:

- *Standard Monte Carlo* (MC): Sample independently and uniformly at random N points in the input space $\mathcal{X}$.

- *Latin Hypercube* (LHC): Divide each one of the k marginal input spaces into N equiprobable intervals, which create a k-dimensional grid and divide the input space into N^k equiprobable sub-spaces. Then select N of those sub-spaces, making sure that their supports are disjoint on any of the k axis. Finally, sample uniformly at random 1 point in each of those sub-spaces.
- *Sobol' Sequences* (LP$_\tau$): These are quasi-random sequences that sample well the input space. They are slightly less intuitive but work well in many situations. See Sobol (1967) [17] for the full algorithm.

The standard Monte Carlo method, although unbiased, exhibits relatively bad convergence behavior (standard error of order $\mathcal{O}(\frac{1}{\sqrt{N}})$) compared to the LP$_\tau$ ($\mathcal{O}(\frac{\log(N)^k}{N})$), which is considered to be one of the best method when k is relatively small (see Homma and Saltelli [18]). However, the Monte Carlo approach has the advantage of being simple to implement and has a convergence behavior that is relatively independent of k. A trade-off between these two scheme is the LHC sampling scheme (see McKay *et al.* [19]). In this report however, the Sobol' sequences will be used since convergence appears to be a major issues in the problem at hand.

2) Results: For computational reasons, convergence was impossible to reach using the standard Sobol' estimators or the improved Kurechenko/Myshetzskay formulas. Indeed, after running $16+$ hours simulations[6], the standard error of the simulation were one orders of magnitudes too large. Knowing that the convergence of Monte Carlo integration is of order $\mathcal{O}(1/\sqrt{N})$, this means that the number of simulations required for the desired accuracy should have been at least 100 times larger (i.e. $N \approx 10'000$). This would have taken approximately 2 months with the hardware at hand. To bypass this limitation, the double loop reordering technique was used with[7] $N = 2^{12} = 4094$. However, as explained above, the total sensibility indices could not be computed using this scheme.

The final results are presented in Figures 9, 10 and 11 (see Appendix B for the Sobol analysis of the 9 other outputs that were monitored). The rocket parameters are plotted in red, the environment parameters in green and the rail parameters in blue. The domain of variability was set to $\pm 5\%$ of their original values.

The correct way of estimating the errors would have been to run several independent copies of the experiment and then to compute the standard error of the estimators. However, this would have multiplied the number of simulations by at least one order of magnitude, which was not feasible. Therefore, instead of providing an estimate of the error, the black bars in Figures 9, 10 and 11 represent the sensitivity indices obtained using a different sampling scheme, LHC in this case. One can see that the results vary slightly depending on the sampling scheme, but that the order of importance is overall preserved.

A multitude of conclusions can be drawn from Figures 9, 10 and 11. First of all, as for the EE analysis, the stability criterion does not produce interesting results, regardless of the sampling scheme. More generally, the graphs are in accordance with the ones obtained as part of EE analysis. The sensibility indices that are below 0.02 are probably smaller than the accuracy, and hence should be analysed with care. Overall, the rocket parameters (in red) seem to play a more important role than the environment parameters (green). However, it would certainly be beneficial to improve to wind model if one wants to improve the accuracy of the landing prediction. Additionally, one can see that the drogue parachute parameter has some influence on the apogee when using the Sobol' sequence, but not when using LHC sampling. Since the wind model does not generate vertical winds, this effect should be 0, which is an indication that the accuracy of the estimator is relatively poor in some cases. One can also observe that the thrust points (especially T_1) have a significant impact on both the landing drift and the apogee. Hence, the relatively simplistic thrust model (linear interpolation between two sample points) is prone to induce large sensitivity inaccuracies. A more precise thrust curve would certainly be beneficial for the overall accuracy of the simulator.

[6]Precisely: $N = 100$, $k = 53$, 5.5 seconds per simulation.
[7]The power of two yields better results when using the Sobol' sequence for the sampling scheme.

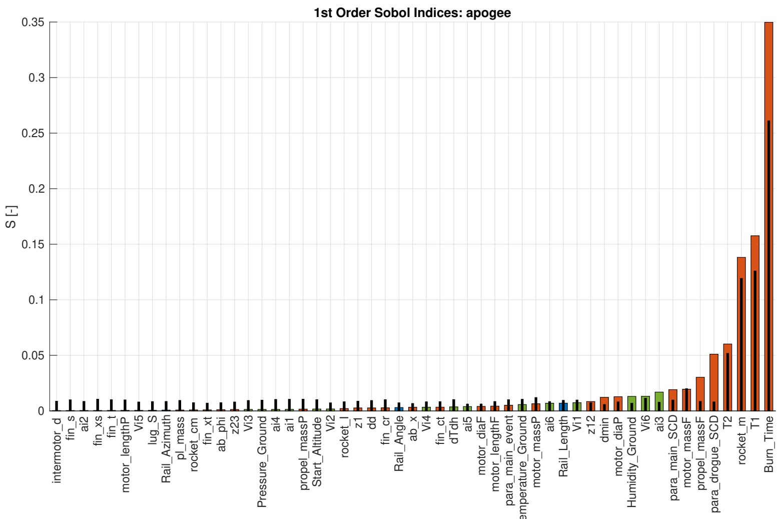

Fig. 9. First order Sobol indices for the apogee, using DLR with 4096 samples from a LP_τ Sobol' sequence (colors) or LHC (black).

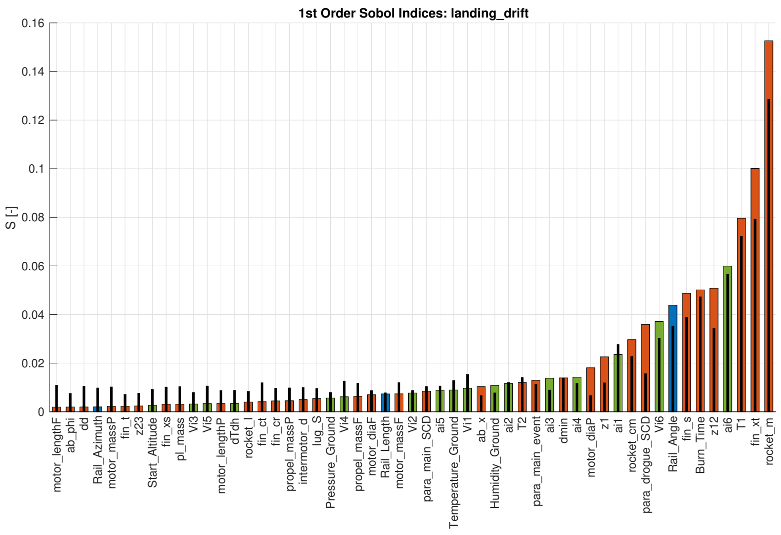

Fig. 10. First order Sobol indices for the landing drift, using DLR with 4096 samples from a LP_τ Sobol' sequence (colors) or LHC (black).

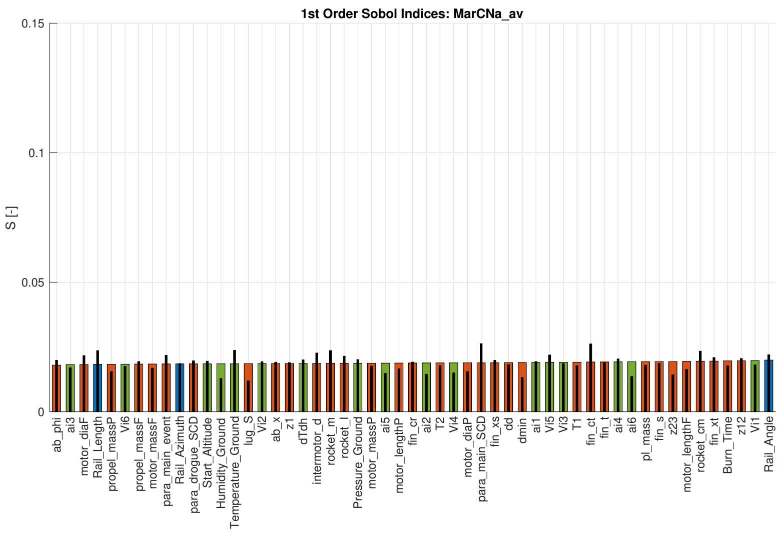

Fig. 11. First order Sobol indices for the stability criterion, using DLR with 4096 samples from a LP_τ Sobol' sequence (colors) or LHC (black).

V. Conclusion

This report, along with the code, should provide the ERT all the tools and references needed to analyse thoroughly their next simulator. Although the results presented here were largely limited by the computational power at hand, all the theory necessary to perform a more ambitious analysis should be written in this report. Performing a sensitivity analysis is rather tedious as each parameter needs to be treated separately. Additionally, the required number of simulations generally grows exponentially with the number of inputs. A few tricks like DLR allow to bypass the limitation but at the cost of a less insightful analysis.

Concerning the actual results, it was firstly shown that the numerical inaccuracies lead to errors in the rocket's position that are below 1 meter. Knowing that the flight path is several kilometers long, this kind of error seems tolerable. The computational load of one simulation was also monitored using different numerical ODE solver, and it seems that the standard `ode45` is the fastest with the default error tolerance. Since speed is more important than accuracy (to an extent) when performing a SA, the `ode45` solver was used. Stiffness in the ODE was not observed directly, but it might be responsible for the slightly lower accuracy of `ode45`. If the ERT is in a situation were the accuracy is of utmost importance, it might be safer to run the simulation of interest with two different ODE solver, `ode45` and a stiff solver (i.g. `ode23s`).

Although highly dependent of the choice of $\mathcal{X}$, the sensitivity analysis yielded results that were overall in accordance with what one would expect. The elementary effect and the variance-based analysis had similar outcoumes. Although it looks like the EE analysis is more insightful since it gives a sense of direction to the effect of one parameter (i.e. if increasing the said parameter increases or decreases the corresponding output), one must remember to interpret the EE analysis in a qualitative manner rather than quantitatively. The Sobol analysis is here to complement this shortcoming and allows a quantitative analysis.

The next steps would be to use the standard formulas presented in Section IV-F1 in order to compute the total sensibility indices and higher order Sobol indices. This would also allow to compare the indices obtained using radically different approaches. Moreover, one could try to take into consideration the dependence between the parameters by tuning the input domain $\mathcal{X}$. This would be particularly relevant if the design team knows exactly the minimum and maximum value of each parameter and how they relate to each other. Tuning $\mathcal{X}$ could also be done by incorporating additional information about the environment, like the relation between the different thermodynamic properties or a better wind model. Here is a non-exhaustive list of additional possible next steps:

- Perform a SA using other modal parameters (different wind model, motor type, etc. See Table III, IV and V). Then observe is the sensitivities of the models are in accordance with what one would expect (for example for the parameter $c_{t,fin}$ with respect to which the landing drift seems highly sensitive).
- Perform a localised SA (by fixing all the parameters that are not of interest). This would reduce the total number of parameter and allow for a greater accuracy in the sensitivity indices. However, when doing this, one must keep in mind that any change in the parameter that were fixed could shadow the sensibility of the parameter of interests. This is partly why all the parameter were considered in this project.
- Perform an standard error analysis (important). As mentioned previously, the simplest way of doing this is by running several independents copies of the SA of interest, and then by looking at the standard deviation of the estimators. Although rudimentary, this type of error analysis is sufficient in most applications.

References

[1] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem.* USA: John Wiley & Sons, Inc., 1991.

[2] Wikipedia contributors, "Stiff equation — Wikipedia, the free encyclopedia," 2020, [Online; accessed 19-November-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stiff_equation&oldid=965373116

[3] J. Dormand and P. Prince, "A family of embedded runge-kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19 – 26, 1980. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0771050X80900133

[4] P. Bogacki and L. Shampine, "A 3(2) pair of runge - kutta formulas," *Applied Mathematics Letters*, vol. 2, no. 4, pp. 321 – 325, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0893965989900797

[5] A. Hayes, "The adams-bashforth-moulton integration methods generalized to an adaptive grid," 04 2011.

[6] L. SHAMPINE and M. REICHELT, "The MATLAB ODE Suite," *SIAM Journal on Scientific Computing*, 1997. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01333731

[7] H. H. Rosenbrock, "Some general implicit processes for the numerical solution of differential equations," *The Computer Journal*, vol. 5, no. 4, pp. 329–330, 01 1963. [Online]. Available: https://doi.org/10.1093/comjnl/5.4.329

[8] M. Hosea and L. Shampine, "Analysis and implementation of tr-bdf2," *Applied Numerical Mathematics*, vol. 20, no. 1, pp. 21 – 37, 1996, method of Lines for Time-Dependent Problems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0168927495001158

[9] A. Saltelli, K. Chan, and E. M. Scott, *Sensitivity analysis.* John Wiley & Sons, Ltd., 2000.

[10] E. d. Rocquigny, *Uncertainty in industrial practice: a guide to quantitative uncertainty management.* Wiley, 2009.

[11] M. D. Morris, "Factorial sampling plans for preliminary computational experiments," *Technometrics*, vol. 33, no. 2, pp. 161–174, 1991. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00401706.1991.10484804

[12] Wikipedia contributors, "Elementary effects method — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Elementary_effects_method&oldid=838340298, 2018, [Online; accessed 28-October-2020].

[13] F. Campolongo, J. Cariboni, and A. Saltelli, "An effective screening design for sensitivity analysis of large models," *Environmental Modelling & Software*, vol. 22, no. 10, pp. 1509 – 1518, 2007, modelling, computer-assisted simulations, and mapping of dangerous phenomena for hazard assessment. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364815206002805

[14] I. Sobol, "Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates," *Mathematics and Computers in Simulation*, vol. 55, no. 1, pp. 271 – 280, 2001, the Second IMACS Seminar on Monte Carlo Methods. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378475400002706

[15] J. Jacques, "Contributions à l'analyse de sensibilité et à l'analyse discriminante généralisée," Theses, Université Joseph-Fourier - Grenoble I, Dec. 2005. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00011169

[16] S. Kucherenko and S. Song, "Different numerical estimators for main effect global sensitivity indices," *Reliability Engineering & System Safety*, vol. 165, pp. 222 – 238, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832016301065

[17] I. Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86 – 112, 1967. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0041555367901449

[18] T. Homma and A. Saltelli, "Use of sobol's quasirandom sequence generator for integration of modified uncertainty importance measure," *Journal of Nuclear Science and Technology - J NUCL SCI TECHNOL*, vol. 32, pp. 1164–1173, 11 1995.

[19] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979. [Online]. Available: http://www.jstor.org/stable/1268522

APPENDIX

A. *Auxiliary Results: Elementary Effect Analysis*

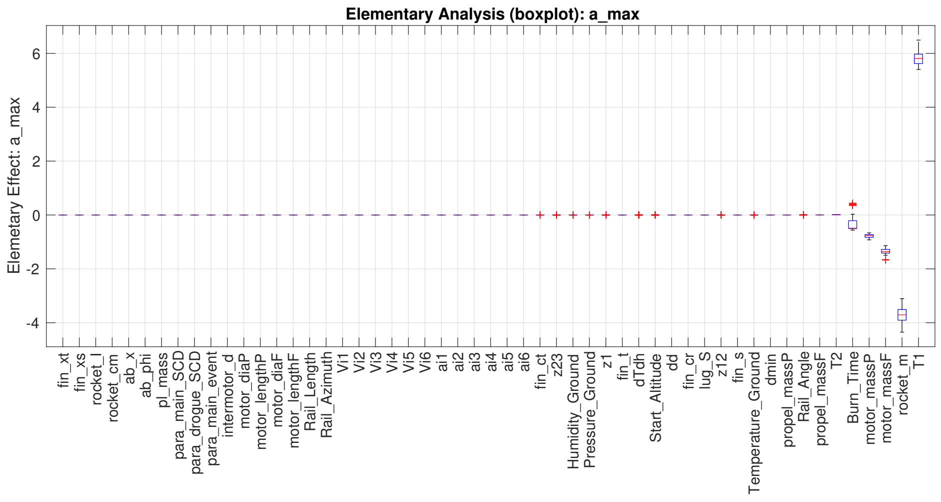

Fig. 12. EE analysis for the maximum acceleration during the flight. Unit: $[\mathrm{m\,s^{-2}}]$.

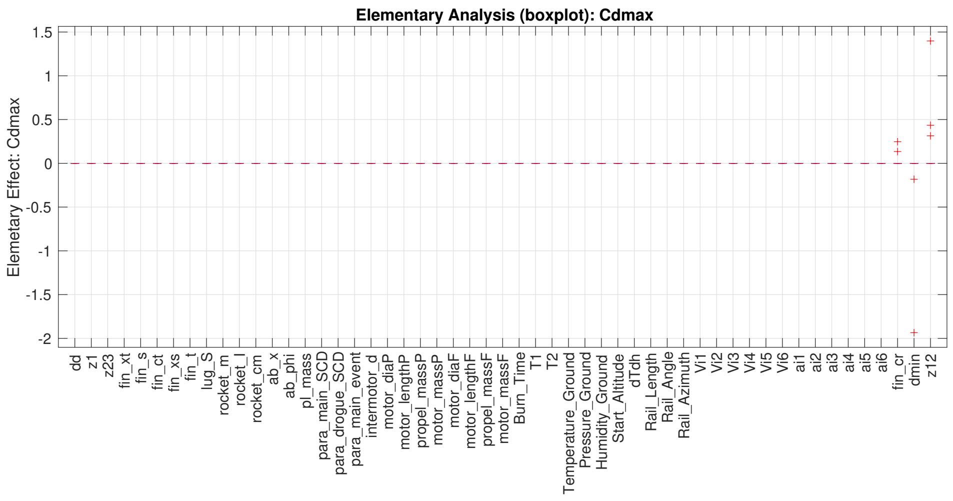

Fig. 13. EE analysis for the maximum drag coefficient during the flight. Unit: [-].

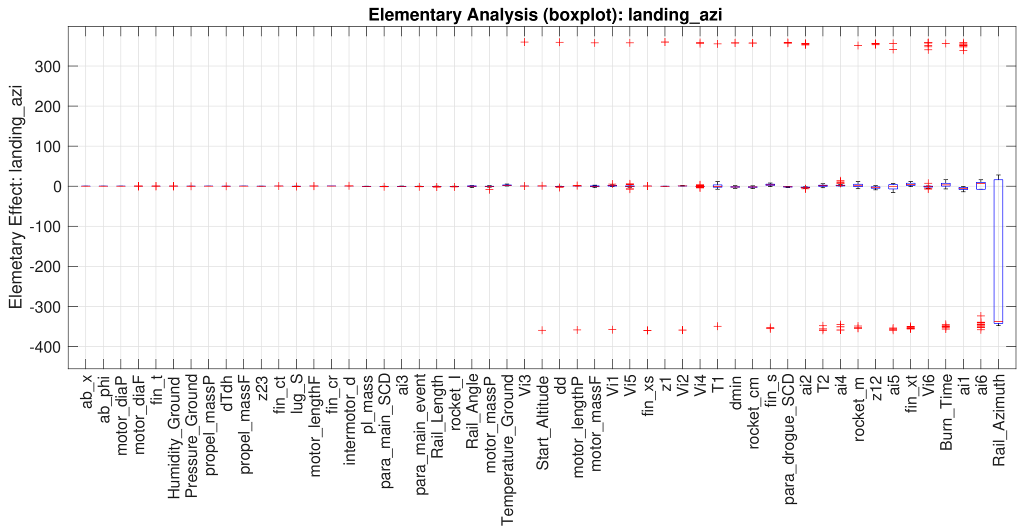

Fig. 14. EE analysis for the landing azimuth. Unit: [°].

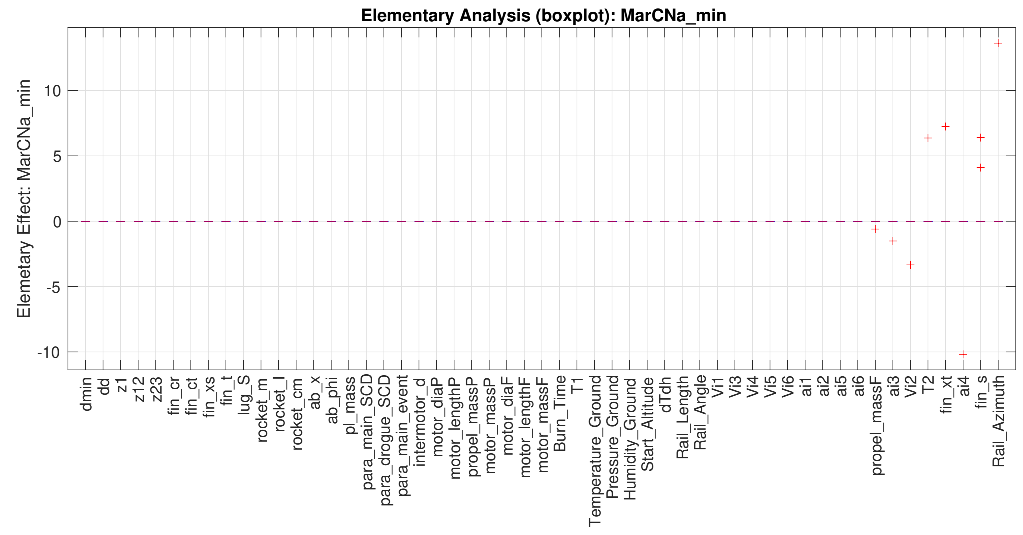

Fig. 15. EE analysis for the minimum of the product between the static margin and the normal drag coefficient during the flight. Unit: [-].

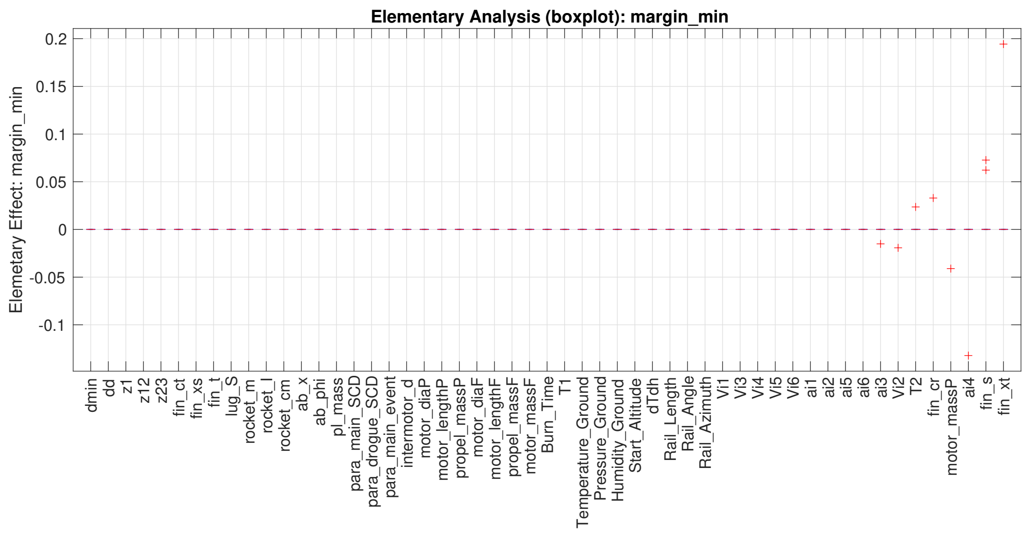

Fig. 16. EE analysis for the minimum static margin over during the flight. Unit: [-].

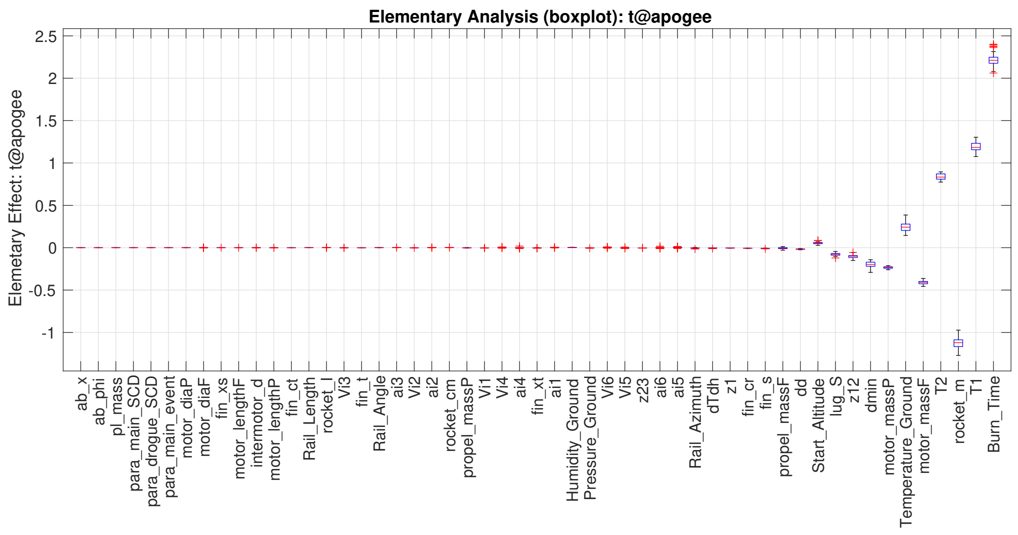

Fig. 17. EE analysis for the time between take-off and apogee. Unit: [s].

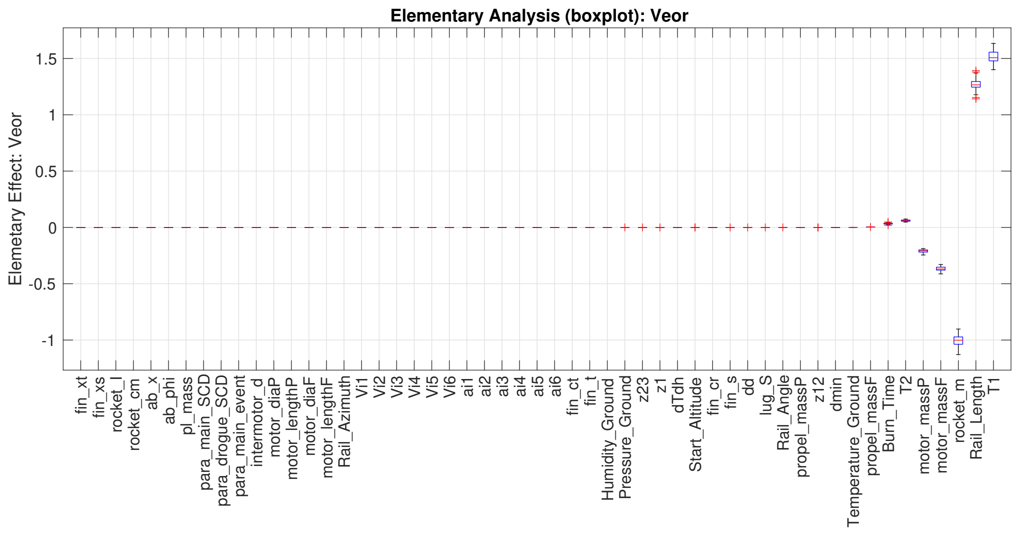

Fig. 18. EE analysis for the rocket velocity at the end of of the rail. Unit: [$\mathrm{m\,s^{-1}}$].

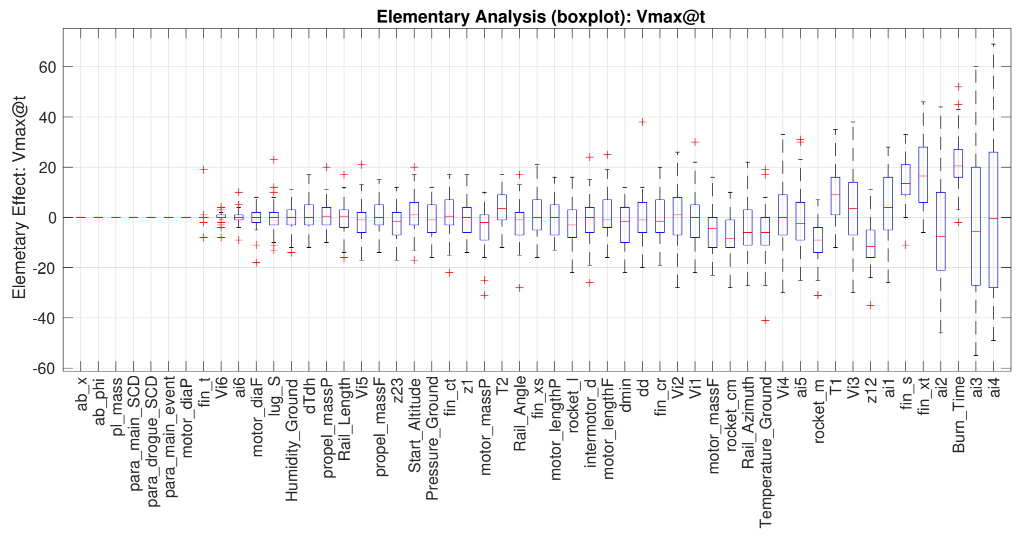

Fig. 19. EE analysis for the time at which the rocket has the maximum velocity. Unit: [s].

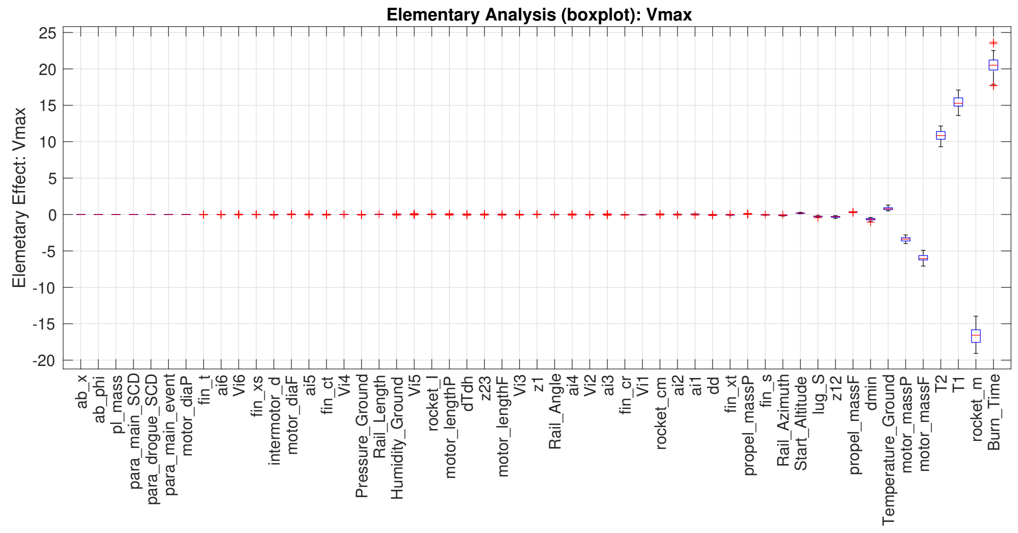

Fig. 20. EE analysis for the rocket's maximum velocity during the flight. Unit: [$\mathrm{m\,s^{-1}}$].

B. *Auxiliary Results: Sobol Analysis*

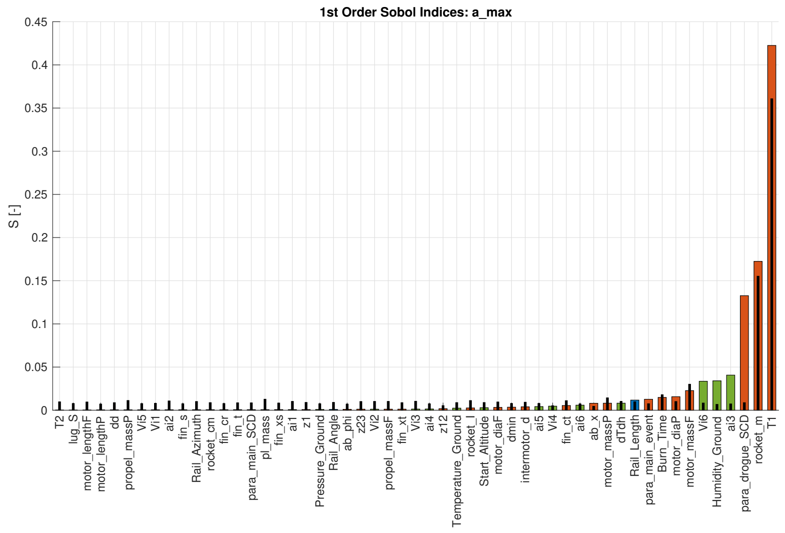

Fig. 21. Sobol analysis for the maximum acceleration during the flight.

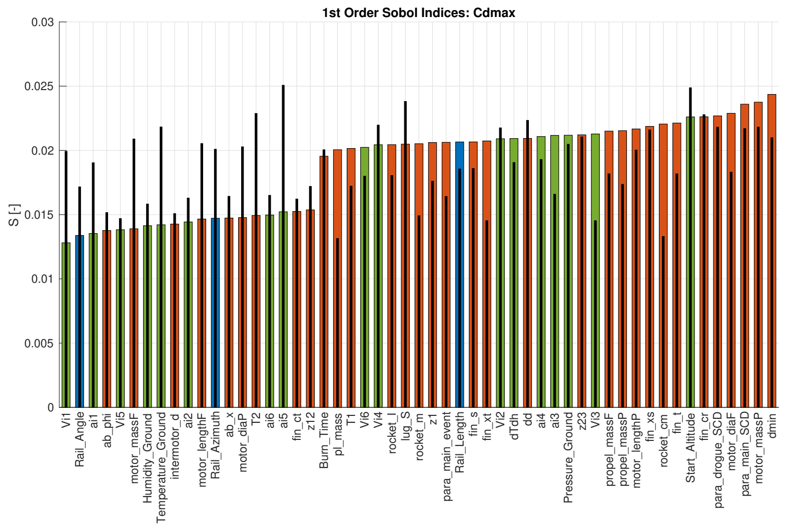

Fig. 22. Sobol analysis for the maximum drag coefficient during the flight.

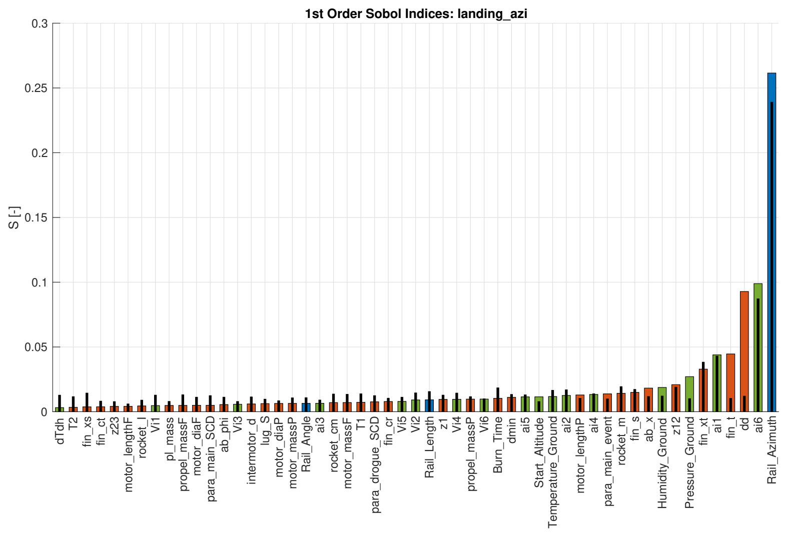

Fig. 23. Sobol analysis for the landing azimuth.

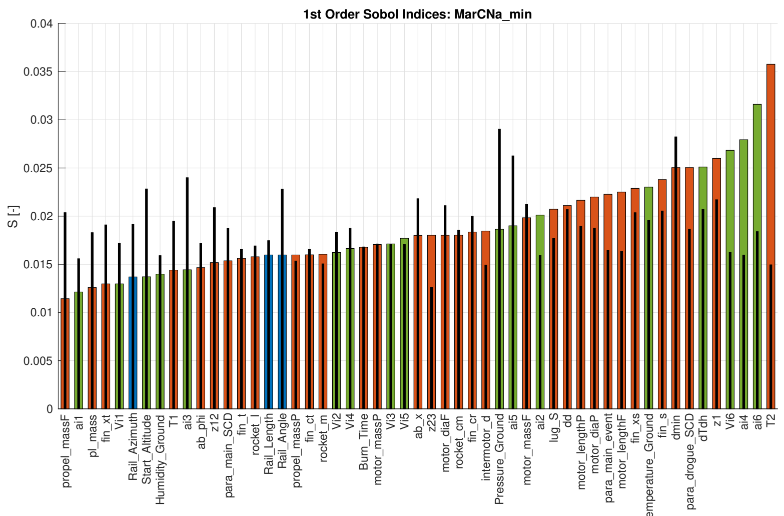

Fig. 24. Sobol analysis for the minimum of the product between the static margin and the normal drag coefficient during the flight.

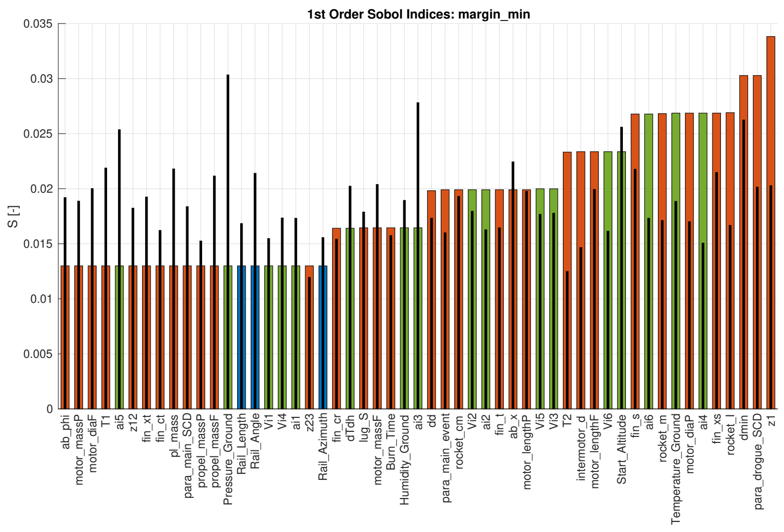

Fig. 25. Sobol analysis for the minimum static margin over during the flight.

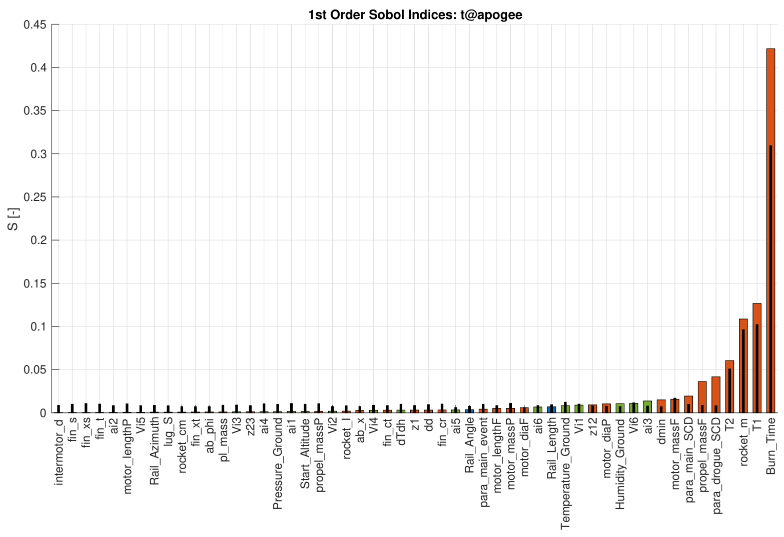

Fig. 26. Sobol analysis for the time between take-off and apogee.

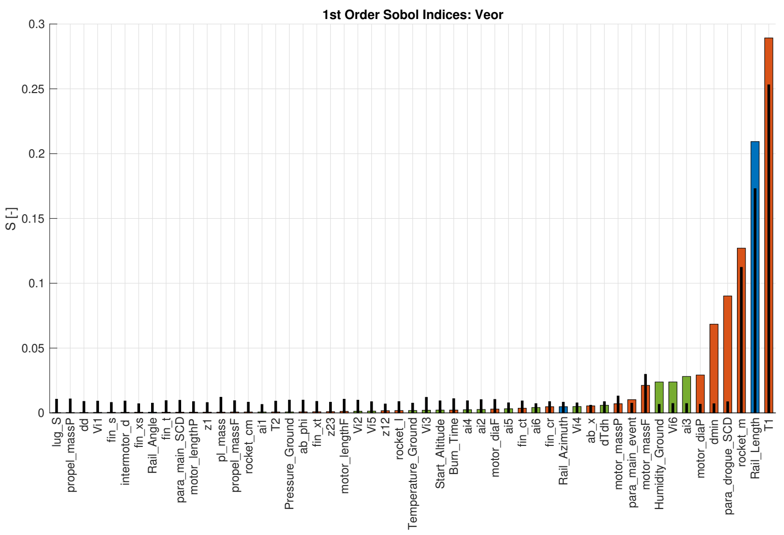

Fig. 27. Sobol analysis for the rocket velocity at the end of of the rail.

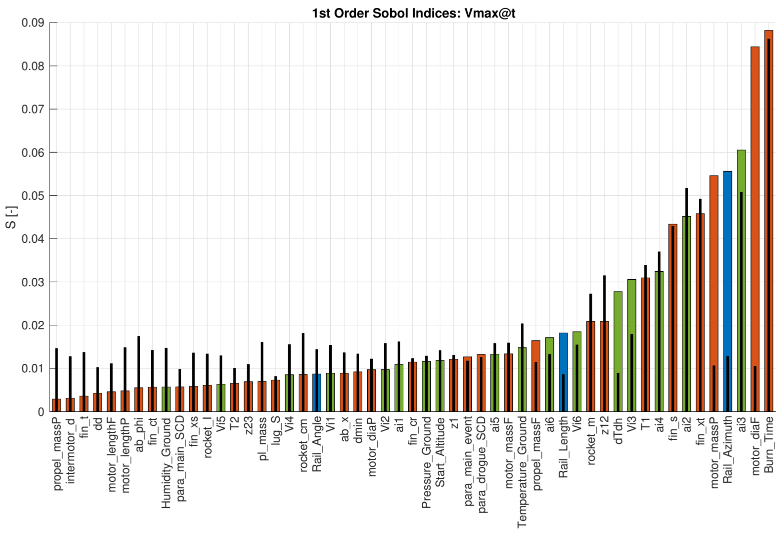

Fig. 28. Sobol analysis for the time at which the rocket has the maximum velocity.

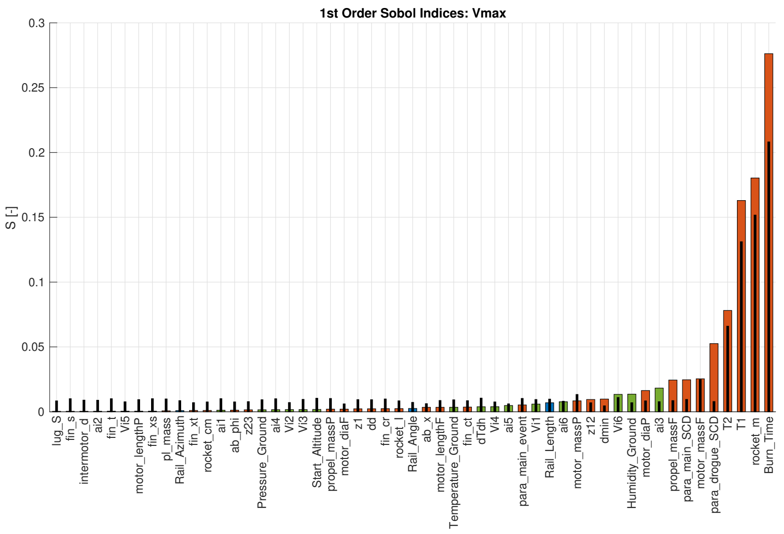

Fig. 29. Sobol analysis for the rocket's maximum velocity during the flight.