SPARTAN: A Novel Pseudospectral Algorithm for Entry, Descent, and Landing Analysis

Marco Sagliano, Stephan Theil, Vincenzo D'Onofrio, Michiel Bergsma

Abstract In the last decades the theoretical development of more and more refined direct methods, together with a new generation of CPUs, led to a significant improvement of numerical approaches for solving optimal-control problems. One of the most promising class of methods is based on Pseudospectral Optimal Control. These methods not only provide an efficient algorithm to solve optimal-control problems, but also define a theoretical framework for linking the discrete numerical solution to the analytical one in virtue of the covector-mapping theorem. However, several aspects in their implementation can be refined. In this framework SPARTAN, the first European tool based on flipped Radau pseudospectral methods, has been developed. The tool, and the method behind it include two novel aspects. First, the discretized problem is automatically scaled with a novel technique, called Projected-Rows Jacobian Normalization. This avoids ill-conditioned problems, which could lead to non-reliable solutions. Second, the structure of the Jacobian matrix is exploited, and the dual-number theory is used for its computation. This yields faster and more accurate solutions, since the associated Jacobian matrix computed in this way is exact. Two concrete examples show the validity of the proposed approach, and the quality of the results obtained with SPARTAN.

1 Introduction

In the last decades the theoretical development of more refined direct methods, together with a new generation of CPUs, led to a significant improvement of numerical approaches for solving optimal-control problems. One of the most promising class of methods is based on Pseudospectral (PS) Optimal Control, originally formulated by Ross et al. [13]. These methods transform the original infinite-dimensional problem, that is, the continuous Bolza problem, into a finite-dimensional, discrete

Marco Sagliano
German Aerospace Center, Robert Hooke Str. 7, Bremen, Germany e-mail: marco.sagliano@dlr.de

Nonlinear Programming (NLP) problem, which can be solved with one of the well-known off-the-shelf solvers, like Snopt [7] and Ipopt [17]. The discrete solution can be later converted into a continuous form by using Lagrange polynomials. Several tools implementing PS methods, have been developed, among others DIDO [4]. However, even if performing excellent, this tool requires ad-hoc manual scaling for the problems analyzed, which turns into a large time-consuming process when done by hand, and to the risk of numerical issues if ignored. Alternatively, Rao et al. [12] proposed a self-scaling method, based on the so-called Jacobian Rows Normalization (JRN) scheme. However, one can see that this scaling method, although properly working, is not optimal, as requires multiple computations of the Jacobian matrix associated with the problem, and at the same time does not offer the largest reduction of the condition number, here taken as measure of the numerical conditioning of the problem to be solved. In SPARTAN (Shefex-3 Pseudospectral Algorithm for Reentry Trajectory ANalysis) a second self-scaling method, based on the so-called Projected-Jacobian Rows Normalization (PJRN) [14] is implemented.

Another important aspect, related with the solution of the finite NLP problem, is the computation of the Jacobian. In fact, one can see that the exploitation of the Jacobian structure leads to a sum of three contributions, which can be computed exactly, and provides computational advantages. The overall result is a state-of-the-art pseudospectral method, which is a valid choice for performing preliminary analyses of entry, descent, and landing scenarios, and that can be easily used to rapidly prototype a solution for complex, nonlinear problems, as it will be shown in this paper. This work is structured as follows: in Sec. 2 the general optimal-control problem we deal with is briefly introduced, while pseudspectral methods, and specifically the flipped-radau PS method, are described in Sec. 3. The proposed improvements on PS methods are fully described in Secs. 4 and 5. Specifically, in Sec. 4 the projected jacobian rows normalization (PJRN) is introduced, while the systematic hybrid Jacobian computation, together with the dual number theory, is explained in Sec. 5. Numerical results obtained for the Space Shuttle entry guidance problem, and the JAXA's Trojan mission-based asteroid descent and landing are shown in Sec. 6. Finally, in Sec. 7 some conclusions are drawn.

2 Optimal Control Problem

There are several approaches for the generation of reference trajectories. Some methods exploit the structure of the specific problems we deal with. Often, they require simplifications to make the problem mathematically tractable, and therefore generate solutions valid under given hypotheses. A different approach, which is gaining popularity, and is helped by the development of the computational capabilities of modern CPUs is the representation of the trajectory generation problem as an optimal-control problem. This means we are looking for solutions minimizing (or maximizing) a given criterion, and satisfying at the same time several constraints, which can be differential (e.g., the equations of motion of a spacecraft) and / or algebraic (e.g., the maximum heat-flux that a vehicle can tolerate). The standard form for representing optimal-control problems is the so-called Bolza problem. Given a state vector $\mathbf{x}(t) \in \mathbb{R}^{n_s}$, a control vector $\mathbf{u}(t) \in \mathbb{R}^{n_c}$, the scalar functions $\Phi(t, \mathbf{x}, \mathbf{u})$ and $\Psi(\mathbf{x}, \mathbf{u})$, and the vector $\mathbf{g}(t, \mathbf{x}, \mathbf{u}) \in \mathbb{R}^{n_p}$ we can formulate the problem as follows.

Minimize (maximize) the cost function J

$$J = \Phi\left[t_f, \mathbf{x}\left(t_f\right), \mathbf{u}\left(t_f\right)\right] + \int_{t_0}^{t_f} \Psi\left[\mathbf{x}(t), \mathbf{u}(t)\right] dt \tag{1}$$

subject to the differential equations

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{u}) \tag{2}$$

and to the path constraints

$$\mathbf{g}_L \leq \mathbf{g}(\mathbf{x}, \mathbf{u}) \leq \mathbf{g}_U \tag{3}$$

The first term in the cost function (1) takes the name of *Mayer* term, and represents punctual constraints (e.g., the minimization of a distance according to a given metric), while the argument of the integral is called the *Lagrange* term and is used to maximize or minimize variables over the entire mission (e.g., the heat load obtained by integrating the heat-flux over time). Moreover, since we deal with physical systems, the problem has usually bounded states and controls, that is, $\mathbf{x}(t)$ and $\mathbf{u}(t)$ are compact in $\mathbb{R}^{n_s}$ and $\mathbb{R}^{n_c}$, respectively.

$$\mathbf{x}_L \leq \mathbf{x}(t) \leq \mathbf{x}_U \tag{4}$$

$$\mathbf{u}_L \leq \mathbf{u}(t) \leq \mathbf{u}_U \tag{5}$$

Equations (1)-(5) represent a generic continuous optimal control problem. In the next section we will see how this type of OCP can be transcribed by using Pseudospectral methods.

3 Pseudospectral Methods

Numerical methods for solving OCPs are divided in two major classes, namely, indirect methods and direct methods. Indirect methods are based on the Pontryagin Maximum Principle, which leads to a multiple-point boundary-value problem. Direct methods, instead, consist in the proper discretization of the OCP, (or *transcription*), having as a result a finite-dimensional, nonlinear programming (NLP) problem. PS Methods represent a particular area of interest in the frame of the wider class of direct methods. In detail, SPARTAN, an optimal-control package developed by the German Aerospace Center[16, 14, 9, 3] uses the global flipped Radau Pseudospectral Method (fRPM), based on the flipped Legendre-Radau polynomials [6, 15]. This choice allows for a straightforward definition of the initial conditions of the problem. Moreover, the following properties are valid:

- ”Spectral” (i.e., quasi-exponential) convergence in the case of a smooth problem
- Runge phenomenon is avoided
- Straightforward implementation
- Sparse structure of the associated NLP problem
- Mapping between the discrete costates of the associated NLP and the continuous costates of the Optimal Control Problem (except for LPM) in virtue of the Pseudospectral Covector Mapping Theorem [8].

In addition, the fRPM shows a smooth convergence of the costates. This is not always the case when other PS methods are employed [6]. Therefore, it is useful to have a look at the fRPM and how it can be conveniently employed to solve OCPs, focusing on the transcription process which defines the corresponding NLP. This does not only involve the choice of the discrete nodes, but also determines the discrete differential and integral operators needed to solve the differential and integral parts of the associated OCP. Therefore, the *transcription* is a more general process than the *discretization*. The minimum fundamental steps of a *transcription* are the following:

- domain discretization
- discrete to continuous conversion of states and / or controls
- characterization of differential and integral operators

The first step is the domain discretization. In the frame of the fRPM, the time domain discretization in n nodes uses the roots of the flipped Legendre-Radau polynomial, defined as the combination of the Legendre polynomial of order n and $n-1$ with coefficient equal to 1 and -1 respectively.

$$R_n(\tau) = L_n(\tau) - L_{n-1}(\tau) \;\; \tau \in [-1, 1] \tag{6}$$

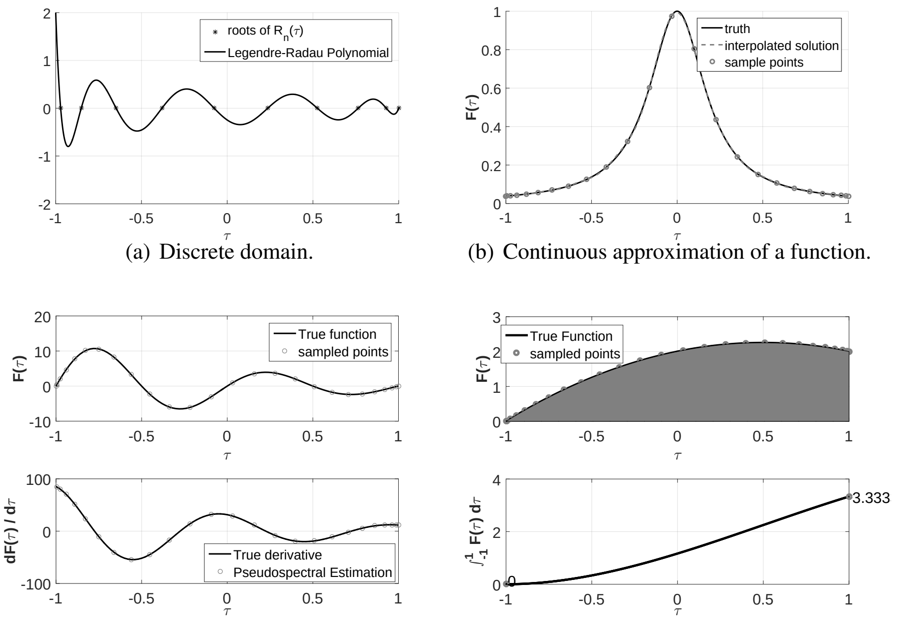
An example of roots associated with the Legendre-Radau polynomial of order 10 is depicted in Fig. 1(a), together with the corresponding polynomial. [1] This discrete representation of the domain is useful to reconstruct continuous representations of the functions $x(t)$ as:

$$x(t) \cong \sum_{i=0}^{n} X_i P(t), \qquad P(t) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{t-t_k}{t_i-t_k} \tag{7}$$

An example of this approximation is depicted in Fig. 1(b), where the function $1/(1+25\tau^2)$ is reconstructed by using 25 fRPM nodes[2].

[1] Note that the $R_n(-1)$ is not a root of the underlying polynomial, therefore it is not a collocation point, although it is required for the evaluation of the polynomial. This choice is motivated by the fact that over the left-open, right-closed interval $(-1,+1]$ these polynomials are orthogonal.

[2] Note that the approximation becomes more accurate when the number of nodes is increased. This is the opposite behavior observed when uniform distributions of nodes, which suffer from the aforementioned *Runge Phenomenon*, are employed.



(a) Discrete domain.
(b) Continuous approximation of a function.
(c) Differential operator example.
(d) Integral operator example.

Fig. 1 Transcription steps: domain discretization (a), continuous reconstruction of functions (b), definition of differential (c) and integral (d) operators.

Once that the domain has been discretized, and the discrete-to-continuous conversion of states has been defined, the corresponding differential operator needs to be derived. This is required for the proper representation of the left-hand side of Eq. (2). The differential operator will be in the form

$$\dot{\mathbf{X}}_i \cong \mathbf{D} \cdot \mathbf{X}_i, \ \ , i = 1, \ldots n \tag{8}$$

and the dynamics defined in Eq. (2) will be replaced by

$$\mathbf{D} \cdot \mathbf{X} = \frac{t_f - t_0}{2} \mathbf{f}(t, \mathbf{X}, \mathbf{U}) \tag{9}$$

where t_0 and t_f are the initial and final time, and the term $\frac{t_f - t_0}{2}$ is a scale factor related to the transformation between the physical time domain t, and the pseudospectral time domain $\tau \in (-1, 1]$, given by the following affine transformations.

$$t = \tfrac{t_f - t_0}{2} \tau + \tfrac{t_f + t_0}{2}, \qquad \tau = \tfrac{2}{t_f - t_0} t - \tfrac{t_f + t_0}{t_f - t_0} \tag{10}$$

In the case of the fRPM the matrix $\mathbf{D}$ has dimensions $[n \times (n+1)]$. Again, this is due to the fact that the states are defined for $n+1$ discrete points, while the controls

$\mathbf{U}$ and the derivatives of the states $\mathbf{f}(t,\mathbf{X},\mathbf{U})$ are defined in the n collocation points. This means that the initial state $\mathbf{X}_0$ is an input and not an output of the optimization, and it is thus assumed to be known. If we look at Eq. (7), and taking the derivative w.r.t. time, we get

$$\dot{\mathbf{x}}(t) \cong \frac{d}{dt}\sum_{i=0}^{n}\mathbf{X_i}P(t) = \sum_{i=0}^{n}\mathbf{X_i}\frac{d}{dt}P_i(t) \tag{11}$$

as the nodal points are time-independent. The derivatives in Eq.(11) can be efficiently computed with the Barycentric Lagrange Interpolation [10]. An example of the differential operator is depicted in Fig. 1(c), where $\mathbf{D}$ is used to approximate the derivative of a continuous function $F(\tau) = Ae^{-\tau}\sin(\omega\tau)$ sampled in 25 collocation nodes. It can be seen that the polynomial approximation fits the analytical derivative very well. In addition to the differential operator, we need an integral operator. This operator is required as the cost function in Eq. (1) may contain the Lagrange term, which needs a proper discretization. In that case the Gauss quadrature formula is used [1]. This approach consists of replacing the continuous integral with the discrete sum given by:

$$\int_{t_0}^{t_f}\Psi[t,\mathbf{x}(t),\mathbf{u}(t)]\,dt = \frac{t_f-t_0}{2}\sum_{i=1}^{n}w_i\Psi[\mathbf{X}_i,\mathbf{U}_i] \tag{12}$$

It can be shown that Eq. (12) yields exact results for polynomials of order at most equal to $2n-1$. Once again, the presence of the term $\frac{t_f-t_0}{2}$ is a consequence of the mapping between pseudospectral and physical time domains described in the relationships (10). For the fRPM the weights w_i can be computed as

$$w = flip(\tilde{w}) \tag{13}$$

$$\tilde{w_i} = \begin{cases} \frac{1}{(1-\tau_j)^2\dot{P}_{n-1}^2}, & j \in [2,...n] \\ \frac{2}{n}, & j = 1 \end{cases} \tag{14}$$

where the operator *flip* simply multiplies the input by a factor equal to -1, and sorts the results in increasing order. To give a practical example the integral of the test function $F(\tau) = 2\tau + 2 - \tau^2$ has been computed. Results are then compared with the analytical integral, and with the trapezoidal rule (Fig. 1(d)). Numerically, we get exactly the analytical result, that is 3.3333, while the application of the trapezoidal rule by using the same number of nodes gives 3.3298. Once that a good approximation of the differential and integral operators have been described, we are ready to formulate the NLP problem which approximates the original OCP as follows:

Minimize (maximize) the cost function J, for n nodes, $i = 1,\dots,n$:

$$J = \Phi\left[t_f,\mathbf{X}_f,\mathbf{U}_f\right] + \frac{t_f-t_0}{2}\sum_{i=1}^{n}w_i\Psi[\mathbf{X}_i,\mathbf{U}_i] \tag{15}$$

subject to the nonlinear algebraic constraints

$$\mathbf{F} = \mathbf{D} \cdot \mathbf{X} - \frac{t_f - t_0}{2} \mathbf{f}(t, \mathbf{X}, \mathbf{U}) = \mathbf{0} \tag{16}$$

and to the path constraints

$$\mathbf{g}_L \leq \mathbf{G}\left(\mathbf{X}_i, \mathbf{U}_i\right) \leq \mathbf{g}_U \tag{17}$$

The discrete states and the controls are bounded, as in the continuous formulation.

$$\mathbf{x}_L \leq \mathbf{X}_i \leq \mathbf{x}_U \tag{18}$$

$$\mathbf{u}_L \leq \mathbf{U}_i \leq \mathbf{u}_U \tag{19}$$

This is the formal definition of the Nonlinear Programming Problem to solve. However, even if theoretically the problem could be solved, in practice further issues arise. In particular, the numerical conditioning of the problem, the exploitation of the Jacobian Matrix, and its computation play a major role in the quality of the results and the speed of the computation.

4 Hybridization of Jacobian matrix

Let us now consider the general structure of the Jacobian matrix associated with the NLP problem deriving from the application of fRPM, defined as follows.

$$\mathbf{Jac} = \left\{ \begin{array}{c} \nabla J \\ \nabla \mathbf{F} \\ \nabla \mathbf{G} \end{array} \right\} \tag{20}$$

The operator ∇ represents the vector of derivative w.r.t. the discrete state $\mathbf{X}_{NLP}$, that needs to be defined. An inspection of the NLP problem represented by Eqs. (15)-(19) suggests that this Jacobian matrix has a structure that can be exploited by looking at its parts. This is the subject of this section. In the most general case, considering n_s states, n_c controls, n_g constraints, n collocation points and unknown final time t_f, the Jacobian **Jac** associated with the transcription of an autonomous system of equations will be expressed as a matrix having the following dimensions

$$\dim(\mathbf{Jac}) = [n \cdot (n_s + n_g) + 1] \times [(n+1) \cdot n_s + n \cdot n_c + 1]\,. \tag{21}$$

To maintain a consistency between the states and the controls associated with each node, the following discrete state vector $\mathbf{X}_{NLP}$ is proposed[3].

$$\mathbf{X}_{NLP} = \left\{ \mathbf{X}_0 \middle| \mathbf{X}_1 \; \mathbf{U}_1 \middle| \mathbf{X}_2 \; \mathbf{U}_2 \middle| .. \; .. \middle| \mathbf{X}_n \; \mathbf{U}_n \middle| t_f \middle| \right\}^T \tag{22}$$

[3] Remark 3: Note that the final element is represented by t_f, in case the problem has finite open time. If not this variable is removed from the vector $\mathbf{X}_{NLP}$.

We can observe how the initial control $\mathbf{U_0}$ does not appear in Eq. (22). This is due to the choice of the fRPM as transcription method instead of the traditional RPM. The initial control indeed can be extrapolated once the NLP is solved. The Jacobian is by definition the matrix representing the partial derivatives of a given set of functions $\mathbf{C}(\mathbf{X}_{NLP})$ (i.e., our NLP constraints), which include the cost function J, the dynamics $\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_n\}$, and, when defined, the constraints $\mathbf{G} = \{\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_n\}$, so we can write

$$\mathbf{C}(\mathbf{X}_{NLP}) = \left\{ J \middle| \mathbf{f}_1 \ \mathbf{f}_2 \ ... \ \mathbf{f}_n \middle| \mathbf{g}_1 \ \mathbf{g}_2 \ ... \ \mathbf{g}_n \middle| \right\}^T \tag{23}$$

and the corresponding Jacobian matrix is

$$\mathbf{J} = \left[\frac{\partial \mathbf{C}}{\partial \mathbf{X}_{NLP}}\right] = \left[\begin{array}{c|cc|cc|c|cc|c}
\frac{\partial J}{\partial \mathbf{X}_0} & \frac{\partial J}{\partial \mathbf{X}_1} & \frac{\partial J}{\partial \mathbf{U}_1} & \frac{\partial J}{\partial \mathbf{X}_2} & \frac{\partial J}{\partial \mathbf{U}_2} & \cdot\cdot\ \cdot\cdot & \frac{\partial J}{\partial \mathbf{X}_n} & \frac{\partial J}{\partial \mathbf{U}_n} & \frac{\partial J}{\partial t_f} \\
\frac{\partial \mathbf{f}_1}{\partial \mathbf{X}_0} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{U}_2} & \cdot\cdot\ \cdot\cdot & \frac{\partial \mathbf{f}_1}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{U}_n} & \frac{\partial \mathbf{f}_1}{\partial t_f} \\
\frac{\partial \mathbf{f}_2}{\partial \mathbf{X}_0} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{U}_2} & \cdot\cdot\ \cdot\cdot & \frac{\partial \mathbf{f}_2}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{U}_n} & \frac{\partial \mathbf{f}_2}{\partial t_f} \\
\cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\
\frac{\partial \mathbf{f}_n}{\partial \mathbf{X}_0} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{U}_2} & \cdot\cdot\ \cdot\cdot & \frac{\partial \mathbf{f}_n}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{U}_n} & \frac{\partial \mathbf{f}_n}{\partial t_f} \\
\frac{\partial \mathbf{g}_1}{\partial \mathbf{X}_0} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{U}_2} & \cdot\cdot\ \cdot\cdot & \frac{\partial \mathbf{g}_1}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{U}_n} & \frac{\partial \mathbf{g}_1}{\partial t_f} \\
\cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\
\frac{\partial \mathbf{g}_n}{\partial \mathbf{X}_0} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{U}_2} & \cdot\cdot\ \cdot\cdot & \frac{\partial \mathbf{g}_n}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{U}_n} & \frac{\partial \mathbf{g}_n}{\partial t_f}
\end{array}\right] \tag{24}$$

This matrix can be computed numerically in different ways (e.g., analytically or with the classical finite-differences schemes). However, these are not the best approaches since they do not consider the theoretical knowledge contained in the definition of the discrete operator $\mathbf{D}$, nor do they take full advantage of the intrinsic sparsity associated with the use of PSMs. Instead, we propose to express the Jacobian matrix as sum of three different contributions.

$$\mathbf{Jac} = \mathbf{Jac}_{Ps} + \mathbf{Jac}_{Du} + \mathbf{Jac}_{Th} \tag{25}$$

We can now analyze each of these terms and describe how to compute them.

4.1 Pseudospectral Jacobian

This part of the Jacobian matrix is intrinsically related to the use of the fRPM. More specifically, it can be seen as the contribution to the Jacobian and to the constraints represented in Eq. (24) given by the use of the discrete differential matrix $\mathbf{D}$. In the frame of the discretization of the dynamics, it represents the term

$$\mathbf{D} \cdot \mathbf{X} \tag{26}$$

From a pure algebraic point of view, the differential operator can be seen as a set of linear combinations of the nodal values of each of the states. The Pseudospectral Jacobian is entirely defined once the matrix **D** is computed and expanded. More explicitly, it can be defined as follows

$$\mathbf{Jac}_{Ps} = \left[\begin{array}{ccccc} & \multicolumn{3}{c}{\mathbf{O}_{1\times[(n+1)\cdot n_s + n\cdot n_c + 1]}} & \\ \mathbf{D}_{1,0} & .. & .. & \mathbf{D}_{1,n} & \\ .. & .. & .. & .. & \mathbf{O}_{[n\cdot(n_s+n_g)+1\times 1]} \\ \mathbf{D}_{n,0} & .. & .. & \mathbf{D}_{n,n} & \\ & \multicolumn{3}{c}{\mathbf{O}_{n_g\times[(n+1)\cdot n_s + n\cdot n_c + 1]}} & \end{array} \right] \tag{27}$$

where

$$\mathbf{D}_{i,j} = D_{i,j} \cdot \mathbf{I}_{n_s}, \; i \in [1,n], \; j \in [0,n] \tag{28}$$

and $\mathbf{I}_{n_s}$ is the identity matrix of dimension n_s. The elements $D_{i,j}$ are the time derivative of the polynomials defined in Eq. (7), evaluated in the collocation nodes. The Pseudospectral Jacobian can then be entirely computed just once, before the beginning of the real optimization process. Moreover, the accuracy of its computation is a consequence of how good the estimate of the roots of the Legendre-Radau Polynomials is, and not of the errors given by the approximation due to the use of numerical differentiation techniques.

4.2 Dual Jacobian

The Dual Jacobian refers to the cost function of Eq. (15), the right-hand side of the differential equations of Eq. (16), and the path constraints of Eq. (17). This contribution is computed by using the dual number theory, which will be briefly described in the next section.

4.2.1 Dual Numbers

In linear algebra, the dual numbers extend the real numbers by adjoining one new element ε with the property $\varepsilon^2 = 0$ (ε is nilpotent). The collection of dual numbers forms a particular two-dimensional commutative associative algebra over the real numbers [5] . Every dual number has the form

$$z = a + b\varepsilon \tag{29}$$

with a and b uniquely determined real numbers and, in particular,

$$\begin{array}{ll} a = real(z), & \text{Real Part} \\ b = dual(z), & \text{Dual Part} \end{array}$$

Dual numbers extend the real numbers in a similar way to the complex numbers. Indeed, as the dual numbers, the complex numbers adjoin a new element i, for which $i^2 = -1$, and every complex number has the form $z = a + bi$ where a and b are real numbers. The definition given in Eq. (29) relies on the idea that $\varepsilon^2 = 0$ with $\varepsilon \neq 0$. To implement the dual numbers, algebraic operations on these numbers should be properly defined. It is important to underline that the dual number algebra is a non-division algebra; given two dual numbers, division is possible only if the real part of the divisor is different from zero. The dual numbers have been implemented in MATLAB as a new class of numbers [3], using operator overloading. The class includes definitions for standard algebraic operations, logical comparison operations, and other more general functions such as the exponential or the trigonometric functions. This class definition file allows a real-valued analysis code to be easily converted to operate on dual numbers by just changing the variable type declarations, while the structure of the code remains unchanged. The use of the dual numbers allows us to compute exact first derivatives, as it will be explained in the next section.

4.2.2 Dual-Step Differentiation Method

The dual-step differentiation method uses the dual numbers to provide exact first order derivatives. Consider the Taylor series of a function $f(x)$ for $x \in \mathbb{R}$ for a given perturbation value a.

$$f(x+a) = f(x) + af'(x) + \frac{1}{2!}a^2 f''(x) + \frac{a^3 f'''(x)}{3!} + \dots. \tag{30}$$

If we assume that the perturbation a is the dual number

$$a = a_1 \varepsilon \qquad \text{with} \qquad \varepsilon^2 = 0 \qquad \text{and} \qquad \varepsilon \neq 0 \tag{31}$$

we can expand in Taylor series around the center x the function $f(x)$ by using a dual step, so that $a^2 = 0$, $a^3 = 0$, $\dots$, and the Taylor series in Eq. (30) truncates exactly at the first-derivative term, yielding the properties of the approximation that we are seeking:

$$f(x+a) = f(x) + a_1 f'(x)\varepsilon. \tag{32}$$

So, to get $f'(x)$ it is necessary to simply read off the ε component and divide by a_1, yielding the dual-step first derivative formula[4].

$$f'(x) = \frac{Dual[f(x+a)]}{a_1}. \tag{33}$$

[4] From the inspection of Eq. (32) it is possible to observe that each function extended in the dual plane *hides* its derivative in its dual part. Indeed, the dual-number algebra is such that, when operations are carried out on the real part of the number, derivative information for those operations is formed and stored in the non-real part of the number. The disadvantage is a larger computational cost and, in addition, the need of working with analytical functions.

This formula clearly shows the advantages of the use of the dual-step differentiation method over the central difference and the complex-step approximations. Indeed, since the dual-step derivative approximation does not involve a difference operation and no terms of the Taylor series are ignored, this formula is subject neither to truncation error, nor to round-off error. There is no need to make the step size small and the simplest choice is $a_1 = 1$, which eliminates the need to divide by the step size. Therefore, using the dual-step method, the error between numerical and analytical derivative ($\eta = |f' - f'_{ref}|/|f'_{ref}|$) is machine zero regardless of the selected step size, as illustrated in Fig. 2, where the derivative of the function $2e^{-t^4}\sin(t)$ is computed by using central difference schemes with 3, 5 and 7 points (CD3, CD5, CD7), complex-step (CS), and dual-step (DS). It is clear that the dual-step approach provides exact results, even in presence of highly nonlinear functions. Indeed, con-

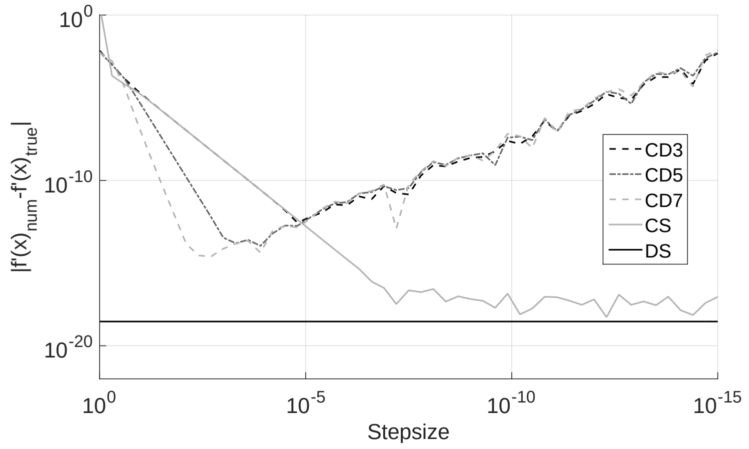


Fig. 2 Comparison of numerical methods for first-derivative computation of the function $2e^{-t^4}\sin(t)$.

sidering the central difference (CD) and the complex-step approximations, instead, Fig. 2 shows that, as the stepsize decreases, the error decreases according to the order of the truncation error of the method. However, after a certain value the error for the central difference approximations tends to grow, while the error for the complex-step approximation continuously decreases. This shows the effect of the round-off error, which affects the central differences but not the first derivative complex-step approximation. While central differences and complex-step provide approximated derivatives for these terms, the use of Dual Number Theory permits permits the computation of zero-epsilon derivatives. The only limit for the use of this technique is the same associated with the use of the complex-step, that is, the need to have analytical functions, i.e., no look-up tables are allowed.

4.2.3 Dual-number based Jacobian

In case we are dealing with analytical functions, it is possible to compute their contribution to the Jacobian matrix (i.e., considering the matrix **D** equal to **0**), excluding the last column,

$$\mathbf{Jac}_{Du} = \left[\frac{\partial \mathbf{C}}{\partial \mathbf{X}_{NLP}}\right]_{\mathbf{D}=\mathbf{0}} = -k_t \left[\begin{array}{ccccccccc}
 & \frac{\partial J}{\partial \mathbf{X}_1} & \frac{\partial J}{\partial \mathbf{U}_1} & \frac{\partial J}{\partial \mathbf{X}_2} & \frac{\partial J}{\partial \mathbf{U}_2} & \cdot\cdot\;\cdot\cdot & \frac{\partial J}{\partial \mathbf{X}_n} & \frac{\partial J}{\partial \mathbf{U}_n} & \\
 & \frac{\partial \mathbf{f}_1}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{U}_2} & \cdot\cdot\;\cdot\cdot & \frac{\partial \mathbf{f}_1}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{U}_n} & \\
 & \frac{\partial \mathbf{f}_2}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{U}_2} & \cdot\cdot\;\cdot\cdot & \frac{\partial \mathbf{f}_2}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{U}_n} & \\
\mathbf{O}_{[n\cdot(n_s+n_g)+1\times n_s]} & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot\;\cdot\cdot & \cdot\cdot & \cdot\cdot & \mathbf{O}_{[n\cdot(n_s+n_g)+1\times 1]} \\
 & \frac{\partial \mathbf{f}_n}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{U}_2} & \cdot\cdot\;\cdot\cdot & \frac{\partial \mathbf{f}_n}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{U}_n} & \\
 & \frac{\partial \mathbf{g}_1}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{U}_2} & \cdot\cdot\;\cdot\cdot & \frac{\partial \mathbf{g}_1}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{g}_1}{\partial \mathbf{U}_n} & \\
 & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot\;\cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\
 & \frac{\partial \mathbf{g}_n}{\partial \mathbf{X}_1} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{U}_1} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{X}_2} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{U}_2} & \cdot\cdot\;\cdot\cdot & \frac{\partial \mathbf{g}_n}{\partial \mathbf{X}_n} & \frac{\partial \mathbf{g}_n}{\partial \mathbf{U}_n} &
\end{array}\right] \tag{34}$$

where k_t is equal to the time scale factor defined as $\frac{t_f - t_0}{2}$ for the elements related to the functions $\mathbf{f}$, and equal to 1 for all the other terms of $\mathbf{Jac}_{Du}$. Each of the elements of $\mathbf{Jac}_{Du}$ can be rewritten in dual form. We can therefore write

$$\mathbf{Jac}_{Du} = -k_t Dual\left[\mathbf{C}(\mathbf{X}_{NLP} + \varepsilon)\right]_{\mathbf{D}=\mathbf{0}} \tag{35}$$

The differentiation operation becomes then an evaluation of the single elements of $\mathbf{C}(\mathbf{X}_{NLP})$ in dual sense, and the extraction of the dual part.

4.3 Theoretical Jacobian

Finally, a third contribution, the Theoretical Jacobian, arises in case we deal with problems having an open final time. In this case the NLP state vector $\mathbf{X}_{NLP}$ will have a further variable, that is t_f. The Jacobian associated with this term is simply proportional to the output of the continuous functions by a factor $\frac{t_f - t_0}{2}$ introduced in virtue of the mapping between physical and pseudospectral time of Eq. (10), the Jacobian associated with this term is proportional to the output of the continuous functions in virtue of the time factor $\frac{t_f - t_0}{2}$.

$$\mathbf{Jac}_{Th} = -\frac{1}{2}\left[\begin{array}{cc}
 & 0 \\
 & \mathbf{f}_1 \\
 & \mathbf{f}_2 \\
\mathbf{O}_{[n\cdot(n_s+n_g)+1]\times[(n+1)\cdot n_s + n\cdot n_c]} & \cdot\cdot \\
 & \mathbf{f}_n \\
 & \mathbf{O}_{n\cdot n_g\times 1}
\end{array}\right] \tag{36}$$

The hybridization of the Jacobian matrix makes the computation of the NLP problems solution more accurate, as no approximations are taken, except those associated with the transcription process. Hence, significant CPU time is saved when solving the NLP problem, as we will see.

5 Automatic scaling: Projected Jacobian Rows Normalization

Let us reformulate the NLP of Eqs. (15)-(19). If we group the differential constraints $\mathbf{f}_i$, and the algebraic constraints $\mathbf{g}_i$, $i = 1,\dots,n$ as

$$\mathbf{F} = \left\{ \mathbf{f}_1 \ \mathbf{f}_2 \ \dots \ \mathbf{f}_n \right\}^T, \ \mathbf{G} = \left\{ \mathbf{g}_1 \ \mathbf{g}_2 \ \dots \ \mathbf{g}_n \right\}^T \tag{37}$$

the core of the NLP can be rewritten in the following compact form as function of the vector $\mathbf{X}$[5].

$$\begin{gathered} \min \ J(\mathbf{X}), \\ \text{s.t. } \mathbf{F}(\mathbf{X}) = 0 \\ \mathbf{g}_L \leq \mathbf{G}(\mathbf{X}) \leq \mathbf{g}_U \\ \mathbf{X}_L \leq \mathbf{X} \leq \mathbf{X}_U \end{gathered} \tag{38}$$

A measure of the quality of a scaling method is the condition number (C.N.) of the Jacobian of the NLP (38), which in the general case is a rectangular matrix given by Eq.(20). Since the Jacobian matrix is involved in the KKT conditions required to solve the NLP, a well-conditioned Jacobian is essential for solving the problem defined in Eq. (38) without excessive rounding errors. This implies that the scaling is not a secondary aspect in the transcription of the optimal control problems. Note that the effective scaling involves two steps: the scaling of the states $\mathbf{X}$, which will be transformed into scaled states $\tilde{\mathbf{X}}$, and the scaling of the constraints $\mathbf{F}$, transformed into the corresponding $\tilde{\mathbf{F}}$. Their combination will result in the scaling of the Jacobian matrix.

5.1 Scaling of NLP States

The states $\mathbf{X}$ of the NLP problem are scaled using the standard linear transformation given in [2], regardless of the NLP scaling method that we use. Specifically, the scaled state $\tilde{\mathbf{X}}$ is given by

$$\tilde{\mathbf{X}} = \mathbf{K}_\mathbf{x} \cdot \mathbf{X} + \mathbf{b}_\mathbf{x} \tag{39}$$

where $\mathbf{K}_\mathbf{x}$ is a diagonal matrix, and $\mathbf{b}_\mathbf{x}$ is a vector having the same dimensions as $\mathbf{X}$. Since we always deal with bounded states and control, the diagonal elements of the matrices $\mathbf{K}_\mathbf{x}$ and $\mathbf{b}_\mathbf{x}$ are defined as:

[5] here $\mathbf{X}$ is meant to be the one defined in Eq. (22), with the subscript dropped to avoid heavy notation.

$$\mathbf{K_x}_{i,i} = \frac{1}{\mathbf{X_U}_i - \mathbf{X_L}_i},\ \mathbf{b_x}_i = -\frac{\mathbf{X_L}_i}{\mathbf{X_U}_i - \mathbf{X_L}_i} \tag{40}$$

Note that the transformation (40) yields scaled states $\tilde{\mathbf{X}}$ which always lie in the interval $[0,1]$. In case of unbounded states, artificial upper and lower boundaries are usually introduced [2].

5.2 Constraints scaling - state of the art

Linear scaling techniques use a scaling of the form (41).

$$\tilde{\mathbf{F}} = \mathbf{K_f} \cdot \mathbf{F},\ \tilde{\mathbf{G}} = \mathbf{K_g} \cdot \mathbf{G} \tag{41}$$

$\mathbf{K_f}$ and $\mathbf{K_g}$ are diagonal matrices. The isoscaling (IS) method is one such technique whereby the constraints $\mathbf{F}$ are scaled exactly like the states, that is,

$$\mathbf{K_f} = \mathbf{K_x},$$

where $\mathbf{K_x}$ is given by Eq. (40), see [2]-[11]. Note that isoscaling does not help in scaling the constraints $\mathbf{G}$. A possible refinement of this approach has been suggested by Rao [12], who uses randomly sampled points around the vector $\mathbf{X}$, and computes the mean of the norms of the Jacobian rows of $\mathbf{F}$ and $\mathbf{G}$ instead of the norm of the Jacobian rows. Unfortunately, this technique significantly increases the CPU time needed to compute the scaling coefficients, since the Jacobian matrix must be evaluated many more times. Next, we introduce a simple linear scaling technique which does not require additional Jacobian matrix evaluations, and hence is less computationally expensive.

5.3 Projected Jacobian Rows Normalization

Isoscaling bases the scaling of the constraints solely on the scaling of the states. In other words, it does not take into account the relationship between the states and the constraints, which is represented in linearized form by the Jacobian matrix. Conversely, Jacobian rows normalization (JRN) only considers this relationship, without involving the states' normalization in the process. Specifically, in the JRN technique, the diagonal elements of $\mathbf{K_f}$ and $\mathbf{K_g}$ are given by

$$\mathbf{K_f}_{i,i} = \underset{k}{\text{mean}} \frac{1}{|\nabla \mathbf{F}|_i},\ \mathbf{K_g}_{i,i} = \underset{k}{\text{mean}} \frac{1}{|\nabla \mathbf{G}|_i} \tag{42}$$

where k represents the number of random samples generated to compute the scaling factors. The projected Jacobian rows normalization (PJRN) technique which we propose considers both the states and the constraints' magnitude. Specifically, in the PJRN, the diagonal elements of $\mathbf{K_f}$ and $\mathbf{K_g}$ are given by

$$\mathbf{K_f}_{i,i} = \frac{1}{\left|\nabla \mathbf{F} \cdot \mathbf{K_x^{-1}}\right|_i}, \ \mathbf{K_g}_{i,i} = \frac{1}{\left|\nabla \mathbf{G} \cdot \mathbf{K_x^{-1}}\right|_i} \tag{43}$$

and this scaling generally leads to a better-conditioned Jacobian matrix, and to more uniformly distributed singular values. The Jacobian of the PJRN-scaled NLP can be therefore computed as

$$\tilde{\mathbf{Jac}} = \begin{Bmatrix} \tilde{\nabla}\tilde{J} \\ \tilde{\nabla}\tilde{\mathbf{F}} \\ \tilde{\nabla}\tilde{\mathbf{G}} \end{Bmatrix} = \begin{Bmatrix} K_J \cdot \nabla J \cdot \mathbf{K_x^{-1}} \\ \mathbf{K_F} \cdot \nabla \mathbf{F} \cdot \mathbf{K_x^{-1}} \\ \mathbf{K_G} \cdot \nabla \mathbf{G} \cdot \mathbf{K_x^{-1}} \end{Bmatrix} \tag{44}$$

where K_J is a parameter which normalizes the cost function J. $\mathbf{K_x}$ is given by using Eq. (40), while $\mathbf{K_f}$ and $\mathbf{K_g}$ are computed by using Eq. (42). Note that K_J can be either manually selected, or automatically computed by means of the PJRN, which is the choice adopted in this work. This completes the self-scaling procedure. We can observe the effects of the hybridization of the Jacobian matrix, and the self-scaling procedure in two significant examples, illustrated in the next section.

6 Numerical Examples

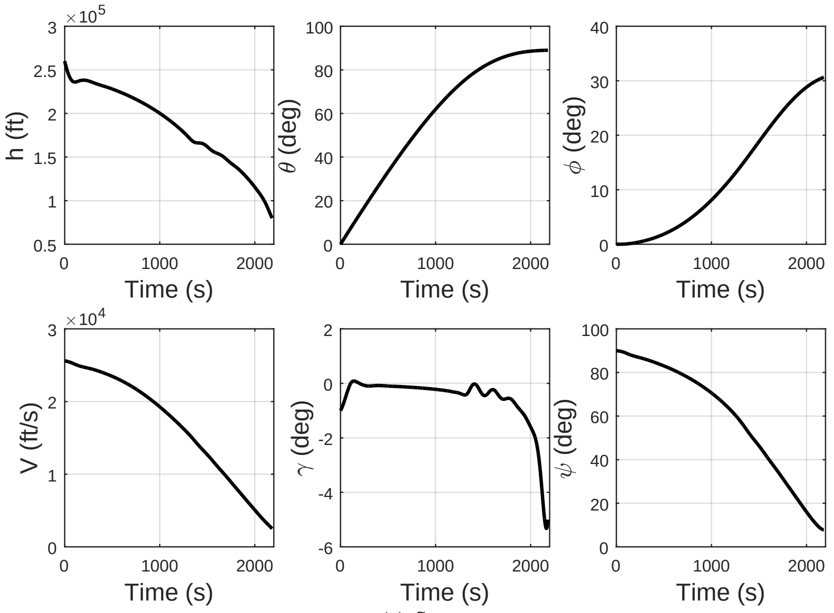
Two examples are proposed to show the application of the proposed improved pseudospectral method for EDL applications, specifically. In the first example the optimal Space Shuttle entry guidance problem is solved. In the second example the asteroid descent and landing problem for a soft touchdown on an asteroid is shown[6].

6.1 Entry: Space Shuttle Guidance

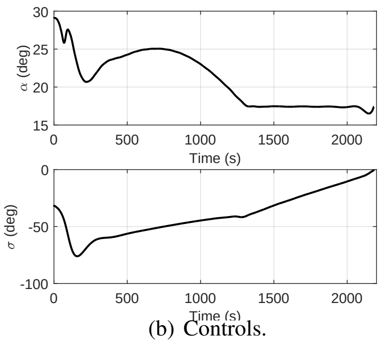
This problem deals with the maximization of the crossrange (corresponding in this case to the latitude) of the Space Shuttle during the atmospheric entry phase, while satisfying the maximum heat-rate limit, and final conditions on altitude, velocity and flight-path angle. A full description of the problem can be found in [2]. Results are depicted in Figs. 3(a)-3(c).

It can be observed that the results are fully consistent with the ones generated by Betts. The combined use of the hybrid jacobian and scaling techniques leads to accurate and faster results with respect to the standard methods. Indeed, the solution

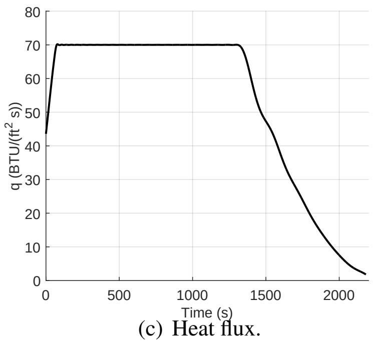
[6] The tests are repeated three times for a better characterization of the obtained CPU times.



(a) States.



(b) Controls.



(c) Heat flux.

Fig. 3 Space Shuttle entry guidance example.

is generated in 8.81 s and 9.72 s when JRN and PJRN techniques are used in comparison to 51 s and 54 s respectively, when the non-hybrid Jacobian is employed.

The differences becomes larger when IPOPT is used (73 s and 155 s when the JRN and PJRN with the hybrid jacobian are used, respectively), versus 817 s and 570 s when no knowledge of the Jacobian matrix is exploited. The use of scaling techniques improves the initial conditioning of the problem by several orders of magnitude (the C.N. goes from $7.468 \cdot 10^9$ to 418 for the PJRN and 24813 when the JRN is adopted).

6.2 Descent and Landing: JAXA-DLR Trojan Mission

This problem deals with the descent and landing of a small lander, in the frame of a JAXA-DLR joint-study for the design of a mission targeting Jupiter's Trojan asteroids. The objective is the maximization of the final mass, while having a synchronized, soft touchdown of the lander on the asteroid surface's nadir point at the beginning of the descent phase. To guarantee the synchronization, together with radial position r, radial and tangential components of velocity V_r and V_t, the motion is described by using the relative angle θ_{ast} w.r.t. the landing point, fixed on the surface of the asteroid. The in-plane dynamics is therefore described as

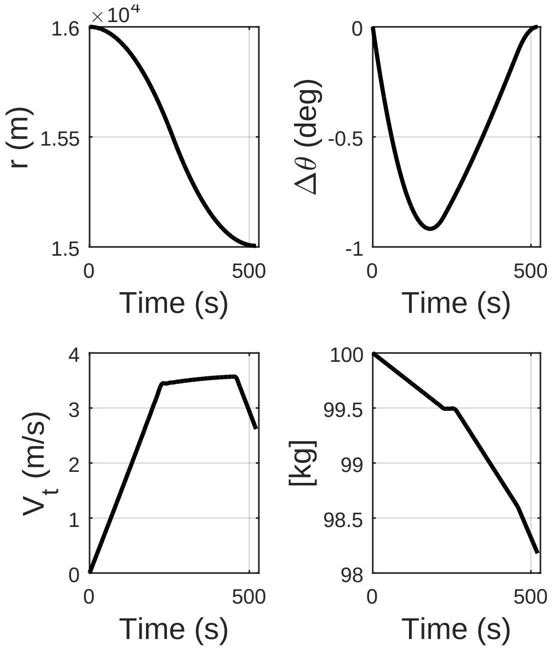
$$\begin{aligned} \dot{r} &= V_r \\ \dot{\theta}_{ast} &= \frac{V_t}{r} - \omega_{ast} \\ \dot{V}_r &= \frac{V_t^2}{r} - \frac{\mu_{ast}}{r^2} + \frac{T_r}{m} \\ \dot{V}_t &= -\frac{V_r V_t}{r} + \frac{T_t}{m} \\ \dot{m} &= \frac{\|T_r\| + \|T_t\|}{I_{sp} g_0} \end{aligned} \tag{45}$$

with the asteroid having a gravitational parameter μ_{ast} equal to 3.774 10^6 m^3/^{2}, and a radius of 15 km. The initial altitude w.r.t. the surface is 1 km. The final desired altitude is 5 m, without side-velocity w.r.t. the landing point. The lander initial mass is 100 kg with a specific impulse of 68 s. Limits equal to 3 N for the radial component of the thrust and 1.5 for the tangential one are also taken into account.

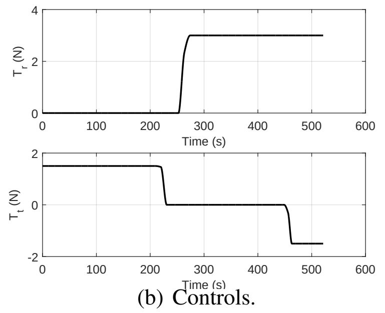
$$\begin{aligned} |T_r| &\leq 3 \quad \text{N} \\ |T_t| &\leq 1.5 \text{ N} \end{aligned} \tag{46}$$

Results are depicted in Figs. 4(a)-4(c), where states, controls, and the trajectory are shown.

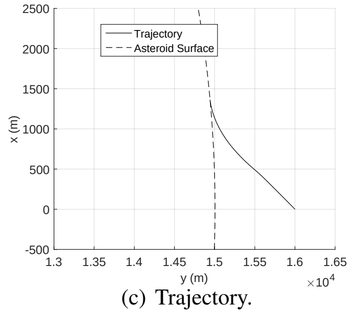
One can see that the solution follows a bang-bang structure, as expected. At the end of the mission the relative angle θ_{ast} is equal to 0, and the tangential velocity V_t becomes exactly equal to the one of the landing point on the asteroid surface. Radially, the gravity is initially exploited to accelerate the lander towards the asteroid. The thrusters are only used to decelerate the lander during the second-half of the descent, to guarantee a soft touchdown. This solution maximizes the remaining propellant at the cost of larger time, equal to 521 s. The trajectory confirms that the lander smoothly reaches the prescribed final point. In terms of numerical performance, if we look at the solution having 50 nodes the use of the PJRN permits to reduce the condition number to 170.5 (while the unscaled condition number is equal to 19400.1 and the JRN generates a condition number equal to 2564.1. Therefore, the condition number is improved by more than one order of magnitude with respect



(a) States.



(b) Controls.



(c) Trajectory.

Fig. 4 Trojan asteroid descent and landing example.

to standard literature methods. The CPU time required to compute a valid solution by using the hybrid Jacobian structure is equal to 43.3 s (PJRN) and 34.7 s (JRN). If we do not exploit the Jacobian structure the required time to compute an optimal solution is equal to 331.4 s (PJRN) and 179.1 s (JRN), respectively. In this case the PJRN does not make so much difference with respect to the JRN, while the hybrid Jacobian computation is highly effective in reducing the required CPU time. Note that when the Jacobian structure is exploited the time is dramatically reduced even if the dual-number class, involving more operations, is invoked.

7 Conclusions

In this paper we have given an overview on pseudospectral methods, together with some key-improvements with respect to the standard transcription described in literature, which include an exploitation of the Jacobian matrix, exactly computed by using the dual-number theory, and a self-scaling approach, which ensures a better numerical conditioning of the problem we want to solve. The proposed techniques have been implemented in SPARTAN, the first European tool implementing the flipped-Radau pseudospectral method. The method, and the tool can be used for preliminary analysis of complex, nonlinear problems involving entry, descent, and landing applications. As examples an entry mission, based on the Space Shuttle entry guidance, and a descent and landing mission for a soft-touchdown on a Trojan asteroid, have been implemented.

Results show the validity of SPARTAN as state-of-the-art tool for entry, descent, and landing guidance analysis, leading to reduced CPU time with respect to standard methods, and a significant improvement of the numerical conditioning of the problems, here measured by the condition number. Its generic structure encourages the use for further scenarios and problems involving complex dynamics and multiple constraints, where no analytical solutions are available, e.g., lunar landing missions or mars descent phases. Future work will include the extension of the method to deal with uncertainties to compute stochastic optimal trajectories, and the implementation of new cases (e.g., low-thrust interplanetary maneuvers) to explore the use of the tool in different scenarios.

References

[1] M. Abramovitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover Publications, 1695.

[2] J. T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, 2nd ed.* SIAM, Philadelphia, 2010.

[3] V. D'Onofrio. Implementation of advanced differentiation methods for optimal trajectory computation. Master's thesis, University of Naples Federico II, Naples, 2015.

[4] Elissar. Description of dido optimal control software, June 2015. URL http://www.elissarglobal.com/.

[5] J. Fike and J.Alonso. The development of hyper-dual numbes for exact second-derivative calculations. In *49th AIAA Aerospace Sciences meeting including the New Horizons Forum and Aerospace Exposition , Orlando, USA, 2011*, number AIAA paper 2011-886.

[6] D. Garg. *Advances in Global Pseudospectral Methods for Optimal Control*. PhD thesis, University of Florida, Gainesville, 2011.

[7] P. E. Gill, W. Murray, and M. A. Saunders. *User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming*. University of California, San Diego, USA, 2008.

[8] Q. Gong, I. M. Ross, W. Kang, and F. Fahroo. Connections between the covector mapping theorem and convergence of pseudospectral methods for optimal control. *Comput Optim Appl, 2008*, 2008. doi: 10.1007/s10589-007-9102-4.

[9] L. Huneker, M. Sagliano, and Y.E. Arslantas. Spartan: An improved global pseudospectral algorithm for high-fidelity entry-descent-landing guidance analysis. In *30th International Symposium on Space Technology and Science, Kobe, Japan, 2015*, 2015.

[10] J. R. R. Martins, P. Sturdza, and J. J. Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software, Vol. 29, No. 3, September 2003, Pages 245262*, 2003. doi: 10.1145/838250.838251.

[11] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[12] A. V. Rao. A survey of numerical methods for optimal control. In *AAS/AIAA Astrodynamics Specialist Conference, AAS Paper 09-334, Pittsburgh, PA, August 10 - 13*, 2009.

[13] I. M. Ross, P. Sekhavat, A. Fleming, and Q. Gong. Pseudospectral feedback control: Foundations, examples and experimental results. In *AIAA Guidance, Navigation, and Control Conference, Keystone, USA,*, 2006. doi: 10.2514/6.2006-6354.

[14] M. Sagliano. Performance analysis of linear and nonlinear techniques for automatic scaling of discretized control problems. *Operations Research Letters, Vol.42 Issue 3, May 2014, pp. 213-216*, 2014. doi: 10.1016/j.orl.2014.03.003.

[15] M. Sagliano. *Development of a Novel Algorithm for High Performance Reentry Guidance*. phdthesis, 2016. URL http://elib.suub.uni-bremen.de/edocs/00105082-1.pdf.

[16] M. Sagliano and S. Theil. Hybrid jacobian computation for fast optimal trajectories generation. In *AIAA Guidance, Navigation, and Control Conference, Boston, USA,*, 2013. doi: 10.2514/6.2013-4554.

[17] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter linesearch algorithm for large-scale nonlinear programming. *Math. Program. 106(1) , Springer-Verlag, New York, 2006*, 2006.