Hybrid Jacobian Computation for Fast Optimal Trajectories Generation

M. Sagliano[1],S. Theil[2]
Deutsches Zentrum für Luft und Raumfahrt (German Aerospace Center)

Robert Hooke Straße 7, Bremen, Germany, 28359

Nowadays the new, increased capabilities of CPUs have constantly encouraged researchers and engineers towards the investigation of numerical optimization as an analysis and synthesis tool in order to generate optimal trajectories and the controls to track them. In particular, one of the most promising techniques is represented by direct collocation methods. Among these, Pseudospectral Methods are gaining popularity for their straightforward implementation and some useful properties, like the possibility to remove the Runge phenomenon present in traditional interpolation techniques and the "spectral" convergence observable in the case of smooth problems. Experience shows that the quality of the results and the computation time are strongly affected by the jacobian matrix describing the transcription of the optimal control problem as an NLP. In this paper, the structure of the Jacobian matrix is analyzed, taking advantage of the sparse nature of such matrices. Additionally, its systematic "hybridization" will be discussed and implemented in order to speed up the simulations. Two different problems will be then described and solved with this approach and the results will be shown. Finally, a quantitative analysis of the performances deriving from the use of the hybrid jacobian compared to a traditional numerical technique will be shown as well.

Nomenclature

C_L	=	Lift Coefficient
C_D	=	Drag coefficient
L	=	Lift Force
D	=	Drag Force
NLP	=	NonLinear Programming Problem
Φ	=	Mayer term of cost function
Ψ	=	Lagrange term of cost function
OCP	=	Optimal Control Problem
CFD	=	Computational Fluid Dynamics
RPM	=	Radau PseudoSpectral Method
$FRPM$	=	Flipped Radau PseudoSpectral Method
Jac	=	Jacobian
I_{Ns}	=	Ns × Ns Identity Matrix
GPM	=	Gauss PseudoSpectral Method
LPM	=	Lobatto PseudoSpectral Method
LU	=	Unitary Length
$M_{cost,\Phi,t}$	=	Mayer Cost Function Dependency Matrix with respect to Time
$M_{cost,\Phi,x}$	=	Mayer Cost Function Dependency Matrix with respect to State
$M_{cost,\Phi,u}$	=	Mayer Cost Function Dependency Matrix with respect to Control
$M_{cost,\Psi,t}$	=	Lagrange Cost Function Dependency Matrix with respect to Time
$M_{cost,\Psi,x}$	=	Lagrange Cost Function Dependency Matrix with respect to State

[1] GNC Engineer, marco.sagliano@dlr.de
[2] Head of GNC Department, stephan.theil@dlr.de

$M_{cost,\Psi,u}$ = Lagrange Cost Function Dependency Matrix with respect to Control
$M_{dyn,t}$ = Dynamics Function Dependency Matrix with respect to Time
$M_{dyn,x}$ = Dynamics Function Dependency Matrix with respect to State
$M_{dyn,u}$ = Dynamics Function Dependency Matrix with respect to Control
$M_{cons,t}$ = Constraints Function Dependency Matrix with respect to Time
$M_{cons,x}$ = Constraints Function Dependency Matrix with respect to State
$M_{cons,u}$ = Constraints Function Dependency Matrix with respect to Control
M_{GD} = Global Dependency Matrix
OR = Logical Operator
PS = PseudoSpectral
Re = Real part
Im = Imaginary part
t = time
τ = normalized time
t_0 = initial time
t_F = final time
g = gravity acceleration
m = mass
h = altitude
ϕ = longitude
θ = latitude
V = speed
γ = flight-path angle
ψ = velocity azimuth angle
α = angle of attack
β = bank angle
ρ = air density
q = heat rate
μ = canonical gravitational parameter
r = radius
φ = azimuth angle
V_r = radial speed
V_t = tangential speed
δ = thrust angle
SNOPT = Sparse Nonlinear Optimizer
T = thrust magnitude
TU = Unitary Time
X_{NLP} = NLP state vector
X_i = state variable in the *i-th* collocation node
U_i = control variable in the *i-th* collocation node
w.r.t. = with respect to

I. Introduction

Nowadays, the new increased capabilities of CPUs have constantly encouraged researchers and engineers towards the investigation of numerical optimization as an analysis and synthesis tool in order to generate optimal trajectories and the controls to track them. In particular, one of the most promising techniques is represented by direct collocation methods. Among these, Pseudospectral Methods are gaining popularity for their straightforward implementation and some interesting properties which are associated to their use; such as the possibility to remove the Runge phenomenon associated to the traditional interpolation techniques and the "spectral" convergence observable in case of smooth problems. This class of methods has been significantly explored in the last years, especially the GPM, LPM and RPM versions [4,6,7,10]. Experience shows that the computation time and, in some cases, the quality of the results are strongly affected by the Jacobian computation method. This paper starts with a short review of PseudoSpectral Methods, with an emphasis on the Radau PseudoSpectral Method and its

flipped version FRPM. Then, a transcription process from a generic Optimal Control Problem to the Nonlinear Programming Problem using the FRPM is shown. Next, in a smiliar manner to what was done in [2], the Jacobian associated to the NLP is analysed in deeper way, in particular for the Flipped Radau Pseudospectral Method. In the first place, an analysis of the continuous functions representing the differential equations, the cost function and the constraints (if any) is performed in order to extract the dependency information needed for a smarter computation of the Jacobian. Additionally, it is possible to investigate the theoretical structure of the FRPM in order to exploit the "Pseudospectral" content of the Jacobian with no need of numerical techniques to compute it. Finally, in the case of unknown final time, the NLP will have an extra variable to take this information into account. This results in a third contribution, whose knowledge is entirely contained in the continuous functions representing the differential equations to be solved. All these segments together will be then used as a basis to perform the "hybridization" of the Jacobian matrix given by the NLP which represents the initial Optimal Control Problem. Section II of this paper describes the main features of the PS methods, and how they can be used to transcribe optimal control problems. Section III describes the procedure implemented for analysing the continuous functions describing the differential equations, the cost index and the constraints involved in the problems considered. Section IV focuses on the computation of the Jacobian while Section V shows the numerical results obtained. Finally, in Section VI the comparison in terms of cpu time between the hybrid and full numerical jacobian is reported for both the problems here analysed, using different number of nodes. In particular two well-known optimization problems, the Orbit Raising problem and the Space Shuttle Reentry Guidance, have been here considered.

II. Pseudospectral Methods

Pseudospectral Methods represent a particular area of interest in the frame of the wider class of direct methods, which is well known in the Computational Fluid Dynamics community [11]. The attention of the aerospace research community towards these techniques is becoming higher and higher during the last years [4,6,7,10]. The basic idea behind these methods is, as in the other direct methods, to *collocate* the differential equations, the cost function and the constraints (if any) in a finite number of points in order to treat them as a set of nonlinear algebraic constraints. In this way, the continuous Optimal Control Problem is reduced to a discrete NLP problem having finite dimensions (despite the continuous problem, which has "infinite" dimensions), which can then be efficiently solved with one of the well-known available software packages, e.g. SNOPT. The main difference with respect to many of the classical collocation methods shown in [1] resides in the choice of the discrete set of points (called *collocation points*) which are chosen as linear combinations of the roots of Legendre Polynomials or their derivatives. In particular, looking at the methods developed over the last years we can distinguish between two subcategories of Pseudospectral methods: the symmetrical methods, like the Gauss Pseudospectral Method (GPM) and the Lobatto PseudoSpectral Method (LPM) and the asymmetrical methods represented by the Radau Pseudospectral Method (RPM) in its direct and flipped form [4]. In this paper our attention is focused on the advantages deriving from the use of flipped version of the RPM: the Flipped Radau PseudoSpectral Method, which is based on the flipped distributions of points w.r.t. the classical RPM. In particular, it has been shown that for the FRPM, as well as for all the PS methods, the following properties are valid:

1) "Spectral" convergence in the case of a smooth problem
2) The Runge phenomenon is avoided
3) Straightforward implementation
4) Sparse structure of the associated NLP problem
5) 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 [6].

It can be then useful to have a look to the FRPM and how it can be conveniently employed to solve OCPs.

II.A Discretization using RPM and FRPM

As all the other PseudoSpectral Methods, the RPM and FRPM are based on the transformation of a physical domain of the independent variable (generally this is "time") into a normalized independent variable τ.

$$\tau \in [-1,1] \tag{1}$$

This means, given a physical domain t,

$$t \in [t_0, t_f] \tag{2}$$

the mapping between the two domains can be performed using the following linear transformations

$$\begin{aligned} t &= \frac{t_f - t_0}{2}\tau + \frac{t_f + t_0}{2} \\ \tau &= \frac{2}{t_f - t_0}t - \frac{t_f + t_0}{t_f - t_0} \end{aligned} \tag{3}$$

This change of domain is due to the fact that the Legendre Polynomials (a subset of more general Jacobi Polynomials) represent a set of orthogonal functions only in the domain [-1,1]. This property gives us the opportunity to approximate a generic continuous function F(τ) as follows:

$$F(\tau) = \sum_{i=0}^{N} F_i P_i(\tau) \tag{4}$$

where

$$P_i(\tau) = \prod_{\substack{j=0 \\ j \neq i}}^{N} \left(\frac{\tau - \tau_j}{\tau_i - \tau_j} \right) \tag{5}$$

and F_i are the values that the continuous function F assumes collocation points τ_i.

Specifically, in the FRPM the points

$$\tau_j, \quad j \in [1, N] \tag{6}$$

are the roots of the combination of two different Legendre Polynomials of order N and N-1:

$$P_N(\tau) = L_N(\tau) - L_{N-1}(\tau) \tag{7}$$

where, by definition,

$$L_N(\tau) = \frac{1}{2^N N!}\frac{d^N}{d\tau^N}\left[\left(\tau^2 - 1\right)^N \right] \tag{8}$$

In particular, the asymmetrical nature of the RPM and the FRPM allow us to impose only one of the extreme points on the interval (2). In the FRPM only the initial point is specified, while no conditions could be traditionally imposed on the final point. This is partially true, as this information can be included in the transcription process in other ways [10].

II.B Transcription of the Optimal Control Problem as NLP

Let us consider the structure of the classical Bolza Optimal Control Problem we want to solve. The purpose is to minimize a cost function

$$J=\Phi\left[x\left(t_0\right),\left(t_F\right)\right]+\int_{t_0}^{t_F}\Psi\left[x\left(\varsigma\right)\right]d\varsigma \tag{9}$$

subject to the dynamics

$$\dot{x}(t)=f_c(t,x,u) \tag{10}$$

with

$$\begin{aligned} x_L \le x(t) \le x_U \\ u_L \le u(t) \le u_U \end{aligned} \tag{11}$$

In some cases the solution must also satisfy some global constraints.

$$g_L \le g(x,t,u) \le g_U \tag{12}$$

Since our attention will be given to the FRPM only the initial state can be explicitly specified, while the final state will be part of the solution of the NLP problem. The continuous states and controls can be substituted with polynomials which interpolate the values in the nodes. This means that the entire information related to the states and the controls is enclosed in their nodal values. In other words,

$$\begin{aligned} x(t_i) \cong X_i, \quad i \in \left[0,N\right] \\ u(t_j) \cong U_j, \quad j \in \left[1,N\right] \end{aligned} \tag{13}$$

Of course, the boundaries valid for the continuous form will also be applied to the discrete representation of the functions.

$$\begin{aligned} x_L \le X_i \le x_U, \quad i \in \left[0,N\right] \\ u_L \le U_j \le u_U, \quad j \in \left[1,N\right] \end{aligned} \tag{14}$$

The difference in the indexing in (13) and (14) is due to the distinction between discretization and collocation. While the discretization includes (in the FRPM) the initial point, the collocation does not. Hence, the controls will be approximated with a polynomial having a lower order and the NLP problem will not provide the initial values for the controls. These can in some cases be part of the initial set of known inputs, otherwise they can be extrapolated from the generated polynomial interpolating the N values of controls in the collocation nodes.

Cost Function

The cost function can be rewritten as

$$J=\sum_{i=0}^{N} v_i\Phi_i+\frac{t_F-t_0}{2}\sum_{i=0}^{N} w_i\Psi_i \tag{15}$$

where

$$v=\left[v_1,v_2,...,v_N\right]^T \tag{16}$$

represents a vector having at least one unitary element (the first and/or the last) and all the other elements are zero, and

$$w = \left[w_1, w_2, ..., w_N \right]^T \tag{17}$$

are the Gaussian quadrature weigths associated to the Radau collocation nodes, which can be computed as

$$w = \ flip(\tilde{w}) \tag{18}$$

where

$$\begin{aligned} \tilde{w}_j &= \frac{1}{\left(1-\tau_j\right)\dot{P}_{N-1}^2}, \quad j \in \left[1...N\right] \\ \tilde{w}_0 &= \frac{2}{N} \end{aligned} \tag{19}$$

The operator *flip* simply multiplies the input by -1 and then sorts the results in increasing order. Φ and Ψ are the respective Mayer and Lagrange terms evaluated in the considered set of collocation points.

Dynamics

The dynamics of the system can be rewritten in terms of residuals, i.e. in terms of algebraic nonlinear constraints evaluated in the collocation points,

$$f_i = \sum_{j=0}^{N} D_{i,j} \cdot X_j - \tfrac{t_F - t_0}{2} f_C(\tau_i, X_i, U_i) = 0, \quad i \in \left[1, N\right], j \in \left[0, N\right] \tag{20}$$

where f and f_C represent respectively the residuals associated to the *collocated* dynamics and the continuous functions describing the differential equations of the considered system. D_{ij} is the (i-th,j-th) element of the Radau discrete matrix acting as differential operator, which can be computed as

$$D = reduce(\hat{D}) \ , \tag{21}$$

with

$$\hat{D}_{i,j} = \frac{\sum_{\substack{k=0 \\ k \neq j}}^{N} \left[\prod_{\substack{m=0 \\ m \neq j,k}}^{N} \left(\tau_i - \tau_m\right) \right]}{\prod_{\substack{k=0 \\ k \neq j}}^{N} \left(\tau_j - \tau_k\right)}, \quad i, j \in \left[0, N\right] \tag{22}$$

The operator *reduce* will delete the last or the first row respectively if the method selected is the RPM or the FRPM. The resulting matrix will be a full-rank N-1 × N matrix.

Constraints

The global constraints can be simply treated as further algebraic constraints to be imposed in the collocation points

$$g_L \leq g(\tau_i, X_i, U_i) \leq g_U, \quad i \in \left[1, ..., N\right] \tag{23}$$

We have then completely defined a NLP having the cost function (15) to be minimized while satisfying the constraints defined in (14), (20), (23).

III. Analysis of Continuous System

III.A Perturbation of Continuous Functions

Experience shows that, while for simple systems a more detailed analysis of Jacobian can be avoided, in complex problems like atmospheric reentry a solid knowledge of its structure is very helpful and significantly increases the speed of computation and in some cases the quality of the results. It is then convenient to look at the continuous functions representing the specific problem. The first step is to perform a systematic analysis of these functions in order to extract the information needed to compute the Jacobian in a more efficient way, as we will see in section IV. The idea is then to create a set of perturbation states and controls and evaluate the cost function, the dynamics of the system and the constraints. To do this, random values of the states and controls are generated using the initial guesses provided to the NLP solver. More specifically, assuming that no *a priori* knowledge of the solution is known, the initial guess along the trajectory can be computed as linear interpolation between the initial and final values ($\underline{x}_0$,$\underline{u}_0$) and ($\underline{x}_F$,$\underline{u}_F$). We can define their mean value as follows.

$$\begin{aligned} \underline{x}_m &= \frac{\underline{x}_0 + \underline{x}_F}{2} \\ \underline{u}_m &= \frac{\underline{u}_0 + \underline{u}_F}{2} \end{aligned} \tag{24}$$

Assuming that the states and the controls are bounded, the random values of the variables needed to evaluate the functions can simply be computed as normal distributions around these mean values.

$$\begin{aligned} \underline{x}_p &= N(\underline{x}_m, \underline{\sigma}_x^2) \\ \underline{u}_p &= N(\underline{u}_m, \underline{\sigma}_u^2) \end{aligned} \tag{25}$$

where $\underline{\sigma}_x$ and $\underline{\sigma}_u$ are the standard deviations assumed as proportional to the difference between the upper and lower boundaries of the variables. In this way a certain number of perturbation vectors can be generated. It is important to stress that here we are only interested in generating the output from the continuous functions, so it´s not important in this phase to have real states and controls data (i.e. data which satisfy our optimal control problem).

III.B Dependency Matrices

The functions are evaluated using the inputs expressed in (25). The objective is to generate the dependency matrices related to the cost function, the dynamics and the constraints. These matrices will have a column for each classical continuous variable, i.e. time, states and controls, and a row for each constraining function. The elements of the matrices are defined equal to 0 when no dependency is recognized and 1 when a dependency is recognized.

	$variable_1$	$variable_2$
$function_1$	0 (no dependency)	1 (dependency)

Table 1: Definition of Dependency Matrix

For example, assuming a system with two states and one control channel,

$$\begin{aligned} \dot{x}_1(t) &= 2x_1^3 + 3x_2 + u_1^2 \\ \dot{x}_2(t) &= 5\sin(x_1) - 3u_1 \end{aligned} \tag{26}$$

the function describing the dynamics will have the following associated 2x4 dependency matrix.

	t	x_1	x_2	u_1
$\dot{x}_1$	0	1	1	1
$\dot{x}_2$	0	1	0	1

Table 2: Dynamics Dependency Matrix for 2-states system

In case of Space Shuttle Reentry (as treated in section V) the dependency matrices are expressed in the tables (3-5) above. The Dependency Matrix for the cost function has dimensions 2x9. For clearity the Mayer and Lagrange contributions are treated in separate way.

	t	h	ϕ	θ	V	γ	ψ	α	β
J_{MAYER}	0	0	0	1	0	0	0	0	0
$J_{LAGRANGE}$	0	0	0	0	0	0	0	0	0

Table 3: Cost Function Dependency Matrix for Space Shuttle Reentry case

The Dependency Matrix for the Dynamics has dimensions 6x9.

	t	h	ϕ	θ	V	γ	ψ	α	β
$\dot{h}$	0	0	0	0	1	1	0	0	0
$\dot{\phi}$	0	1	0	1	1	1	1	0	0
$\dot{\theta}$	0	1	0	0	1	1	1	0	0
$\dot{V}$	0	1	0	0	1	1	0	1	0
$\dot{\gamma}$	0	1	0	0	1	1	0	1	1
$\dot{\psi}$	0	1	0	1	1	1	1	1	1

Table 4: Dynamics Dependency Matrix for Space Shuttle Reentry case

The only constraint is represented by the heat rate, which has a 1x9 dependency matrix.

	t	h	ϕ	θ	V	γ	ψ	α	β
q	0	1	0	0	1	0	0	1	0

Table 5: Constraint Dependency Matrix for Space Shuttle Reentry case

This information can be rearranged in the matrices defined in (27), which will be used to compute the Jacobian in efficient way, as it will be shown in the next section.

$$
\begin{aligned}
M_{cost_{\Phi,t}} &= 0 \\
M_{cost_{\Psi,t}} &= 0 \\
M_{cost_{\Phi,x}} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \\
M_{cost_{\Psi,x}} &= O_{1\times 6} \\
M_{cost_{\Phi,u}} &= O_{1x2} \\
M_{cost_{\Psi,u}} &= O_{1\times 2} \\
M_{dyn,t} &= O_{6x1} \\
M_{dyn,x} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \\
M_{dyn,u} &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \\
M_{cons,t} &= 0, \\
M_{cons,x} &= \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \\
M_{cons,u} &= \begin{bmatrix} 1 & 0 \end{bmatrix}
\end{aligned}
\tag{27}
$$

IV. Hybridization of Jacobian

Let us now consider the general structure of the Jacobian associated to the NLP problem deriving from the application of FRPM, and its specific application to the already cited case of Space Shuttle Guidance.

IV.A General structure of Jacobian associated to the FRPM

In the most general case, considering n_s states, n_c controls, n_g constraints, n collocation points and unknown final time, the Jacobian associated to the transcription of an autonomous system of equations (as in the examples here treated) will be expressed as a matrix having the following dimensions and structure.

$$\dim(Jac) = \left[n \cdot \left(n_s + n_g \right) + 1 \right] \times \left[\left(n+1 \right) \cdot n_s + n \cdot n_c + 1 \right] \tag{28}$$

While the dimensions of the Jacobian do not vary once the variables and the constraints are given, the kind of sparsity patterns which will appear in it depends on the ordering of the NLP state vector. In order to take advantage of the information provided by the dependency matrices (as we will see in section IV.C) and to maintain a consistence between the states and the controls associated to each node, the following order for the NLP variable is proposed:

$$\underline{X}_{NLP} = \left\{ \underline{x}_0 \;\middle|\; \underline{x}_1 \;\; \underline{u}_1 \;\middle|\; \underline{x}_2 \;\; \underline{u}_2 \;\middle|\; .. \;\; .. \;\middle|\; \underline{x}_N \;\; \underline{u}_N \;\middle|\; t_F \right\}^T \tag{29}$$

We can observe how the initial control $\underline{u}_0$ does not appear in (29). 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. Since the Jacobian is by definition the matrix representing the partial derivatives of a given set of functions (i.e. our NLP constraints) w.r.t. their variables, this set and its order must be defined. We will then consider all the constraints defined during the transcription of the problem. The cost function is treated as another constraint, generally the first [5]. This is mathematically included in the Jacobian in order to generate a NLP Problem which is compatible with NLP solver SNOPT interface.

$$\underline{C}(X_{NLP}) = \left[\;\middle|\; J \;\middle|\; \underline{f}_1 \;\; \underline{f}_2 \;\; .. \;\; \underline{f}_N \;\middle|\; \underline{g}_1 \;\; .. \;\; \underline{g}_N \;\middle|\; \right]^T \tag{30}$$

The Jacobian deriving from these definitions is the following:

$$Jac = \left[\frac{\partial \underline{C}}{\partial \underline{X}_{NLP}} \right] = \begin{bmatrix} \frac{\partial J}{\partial \underline{x}_0} & \frac{\partial J}{\partial \underline{x}_1} & \frac{\partial J}{\partial \underline{u}_1} & \frac{\partial J}{\partial \underline{x}_2} & \frac{\partial J}{\partial \underline{u}_2} & .. & .. & \frac{\partial J}{\partial \underline{x}_N} & \frac{\partial J}{\partial \underline{u}_N} & \frac{\partial J}{\partial t_F} \\ \frac{\partial \underline{f}_1}{\partial \underline{x}_0} & \frac{\partial \underline{f}_1}{\partial \underline{x}_1} & \frac{\partial \underline{f}_1}{\partial \underline{u}_1} & \frac{\partial \underline{f}_1}{\partial \underline{x}_2} & \frac{\partial \underline{f}_1}{\partial \underline{u}_2} & .. & .. & \frac{\partial \underline{f}_1}{\partial \underline{x}_N} & \frac{\partial \underline{f}_1}{\partial \underline{u}_N} & \frac{\partial \underline{f}_1}{\partial t_F} \\ \frac{\partial \underline{f}_2}{\partial \underline{x}_0} & \frac{\partial \underline{f}_2}{\partial \underline{x}_1} & \frac{\partial \underline{f}_2}{\partial \underline{u}_1} & \frac{\partial \underline{f}_2}{\partial \underline{x}_2} & \frac{\partial \underline{f}_2}{\partial \underline{u}_2} & .. & .. & \frac{\partial \underline{f}_2}{\partial \underline{x}_N} & \frac{\partial \underline{f}_2}{\partial \underline{u}_N} & \frac{\partial \underline{f}_2}{\partial t_F} \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ \frac{\partial \underline{f}_N}{\partial \underline{x}_0} & \frac{\partial \underline{f}_N}{\partial \underline{x}_1} & \frac{\partial \underline{f}_N}{\partial \underline{u}_1} & \frac{\partial \underline{f}_N}{\partial \underline{x}_2} & \frac{\partial \underline{f}_N}{\partial \underline{u}_2} & .. & .. & \frac{\partial \underline{f}_N}{\partial \underline{x}_N} & \frac{\partial \underline{f}_N}{\partial \underline{u}_N} & \frac{\partial \underline{f}_N}{\partial t_F} \\ \frac{\partial \underline{g}_1}{\partial \underline{x}_0} & \frac{\partial \underline{g}_1}{\partial \underline{x}_1} & \frac{\partial \underline{g}_1}{\partial \underline{u}_1} & \frac{\partial \underline{g}_1}{\partial \underline{x}_2} & \frac{\partial \underline{g}_1}{\partial \underline{u}_2} & .. & .. & \frac{\partial \underline{g}_1}{\partial \underline{x}_N} & \frac{\partial \underline{g}_1}{\partial \underline{u}_N} & \frac{\partial \underline{g}_1}{\partial t_F} \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ \frac{\partial \underline{g}_N}{\partial \underline{x}_0} & \frac{\partial \underline{g}_N}{\partial \underline{x}_1} & \frac{\partial \underline{g}_N}{\partial \underline{u}_1} & \frac{\partial \underline{g}_N}{\partial \underline{x}_2} & \frac{\partial \underline{g}_N}{\partial \underline{u}_2} & .. & .. & \frac{\partial \underline{g}_N}{\partial \underline{x}_N} & \frac{\partial \underline{g}_N}{\partial \underline{u}_N} & \frac{\partial \underline{g}_N}{\partial t_F} \end{bmatrix} \tag{31}$$

This Jacobian matrix can be computed numerically in different ways (ex. with the classical finite differences scheme or using the complex-step derivative technique [3,12]). This is not the best approach since it does not consider the theoretical knowledge contained in the definition of the discrete operator D, nor does it take full advantage from the intrinsic sparsity associated to the use of Pseudospectral methods. Instead, the approach followed in [2] brings better results, and will be here particularized for the FRPM. On this purpose, let us then express the Jacobian as sum of three different contributions.

$$Jac = Jac_{PseudoSpectral} + Jac_{Numerical} + Jac_{Theoretical} \tag{32}$$

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

IV.B 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 (20) given by the use of the discrete differential matrix D. In the frame of the discretization of the dynamics, it represents

$$\underline{\underline{D}} \cdot \underline{x} \tag{33}$$

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. More explicitly, the Pseudospectral Jacobian Matrix can be defined as follows

$$Jac_{PseudoSpectral} = \begin{bmatrix} & O_{1\times\left[(n+1)\cdot n_s + n\cdot n_c + 1\right]} & \\ \tilde{D}_{1,0} \;\; .. & .. \;\; \tilde{D}_{1,n} & \\ .. \;\; .. & .. \;\; .. & O_{\left[n\cdot(n_s+n_g)+1\right]\times 1} \\ \tilde{D}_{n,0} \;\; .. & .. \;\; \tilde{D}_{n,n} & \\ & O_{n_g\times\left[(n+1)\cdot n_s + n\cdot n_c + 1\right]} & \end{bmatrix} \tag{34}$$

where

$$\tilde{D}_{i,j} = D_{i,j} \cdot I_{n_S}, \quad i \in [1, N], j \in [0, N] \tag{35}$$

and I_{ns} is the identity matrix of dimension n_S. 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 numerical differentiation techniques.

IV.C Numerical Jacobian

The numerical Jacobian can be computed using the classical finite differences or the complex-step derivative methods [3,12]. In particular, the second technique is very interesting as it removes the cancellation errors which are a consequence of the use of the classical finite-differences scheme. This method is based on the hyphotesis of having continuous functions, and consequently the functions defining the OCP can be seen as real parts of complex functions. Under these premises, the Cauchy-Riemann conditions for the complex function $u + iv = f(x + iy)$ are valid.

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \tag{36}$$

Then, by definition, the derivative of the real part of the function f can be approximated as

$$\mathrm{Re}\left[\frac{df}{dx}\right] = \frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \cong \frac{\mathrm{Im}\left[f(x+ih)\right]}{h} \tag{37}$$

Where h is the numerical step used to perturbate the functions, generally assumed very small.

This approach will be applied to all the elements of the Jacobian recognized by the dependency matrices. The chosen ordering for the state vector makes this operation quite intuitive as the nonzero elements of the dependency matrices will represent the patterns of computation for the numerical derivatives. We can hence express this contribution as

$$Jac_{Numerical} = \left(Jac_{Continuous} \circ M_{GD} \right) \tag{38}$$

where $Jac_{Continuous}$ is the Jacobian considering only the continuous functions defining the problem (i.e. considering the matrix D equal to 0), excluding the last column,

$$Jac_{Continuous} = \left[\frac{\partial \underline{C}}{\partial \underline{X}_{NLP}} \right]_{D=0} = -\frac{t_F - t_0}{2} \begin{bmatrix} \frac{\partial J}{\partial \underline{x}_0} & \frac{\partial J}{\partial \underline{x}_1} & \frac{\partial J}{\partial \underline{u}_1} & \frac{\partial J}{\partial \underline{x}_2} & \frac{\partial J}{\partial \underline{u}_2} & .. & .. & \frac{\partial J}{\partial \underline{x}_N} & \frac{\partial J}{\partial \underline{u}_N} \\ \frac{\partial \underline{f}_{C1}}{\partial \underline{x}_0} & \frac{\partial \underline{f}_{C1}}{\partial \underline{x}_1} & \frac{\partial \underline{f}_{C1}}{\partial \underline{u}_1} & \frac{\partial \underline{f}_{C1}}{\partial \underline{x}_2} & \frac{\partial \underline{f}_{C1}}{\partial \underline{u}_2} & .. & .. & \frac{\partial \underline{f}_{C1}}{\partial \underline{x}_N} & \frac{\partial \underline{f}_{C1}}{\partial \underline{u}_N} \\ \frac{\partial \underline{f}_{C2}}{\partial \underline{x}_0} & \frac{\partial \underline{f}_{C2}}{\partial \underline{x}_1} & \frac{\partial \underline{f}_{C2}}{\partial \underline{u}_1} & \frac{\partial \underline{f}_{C2}}{\partial \underline{x}_2} & \frac{\partial \underline{f}_{C2}}{\partial \underline{u}_2} & .. & .. & \frac{\partial \underline{f}_{C2}}{\partial \underline{x}_N} & \frac{\partial \underline{f}_{C2}}{\partial \underline{u}_N} \\ .. & .. & .. & .. & .. & .. & .. & .. & .. \\ \frac{\partial \underline{f}_{CN}}{\partial \underline{x}_0} & \frac{\partial \underline{f}_{CN}}{\partial \underline{x}_1} & \frac{\partial \underline{f}_{CN}}{\partial \underline{u}_1} & \frac{\partial \underline{f}_{CN}}{\partial \underline{x}_2} & \frac{\partial \underline{f}_{CN}}{\partial \underline{u}_2} & .. & .. & \frac{\partial \underline{f}_{CN}}{\partial \underline{x}_N} & \frac{\partial \underline{f}_{CN}}{\partial \underline{u}_N} \\ \frac{\partial g_1}{\partial \underline{x}_0} & \frac{\partial g_1}{\partial \underline{x}_1} & \frac{\partial g_1}{\partial \underline{u}_1} & \frac{\partial g_1}{\partial \underline{x}_2} & \frac{\partial g_1}{\partial \underline{u}_2} & .. & .. & \frac{\partial g_1}{\partial \underline{x}_N} & \frac{\partial g_1}{\partial \underline{u}_N} \\ .. & .. & .. & .. & .. & .. & .. & .. & .. \\ \frac{\partial g_N}{\partial \underline{x}_0} & \frac{\partial g_N}{\partial \underline{x}_1} & \frac{\partial g_N}{\partial \underline{u}_1} & \frac{\partial g_N}{\partial \underline{x}_2} & \frac{\partial g_N}{\partial \underline{u}_2} & .. & .. & \frac{\partial g_N}{\partial \underline{x}_N} & \frac{\partial g_N}{\partial \underline{u}_N} \end{bmatrix} \quad O_{\left[n \cdot \left(n_s + n_g \right) + 1 \right] \times 1} \tag{39}$$

M_{GD} is a matrix having the same dimensions of the Jacobian, and defined as:

$$M_{GD} = \begin{bmatrix} OR\left(M_{cost_{\Phi,x}}, M_{cost_{\Psi,x}} \right) & M_{cost_{\Psi,x}} & M_{cost_{\Psi,u}} & ... & OR\left(M_{cost_{\Phi,x}}, M_{cost_{\Psi,x}} \right) & OR\left(M_{cost_{\Phi,u}}, M_{cost_{\Psi,u}} \right) \\ M_{dyn_x} & M_{dyn_x} & M_{dyn_u} & ... & M_{dyn_x} & M_{dyn_u} \\ ... & ... & ... & ... & ... & ... \\ M_{dyn_x} & M_{dyn_x} & M_{dyn_u} & ... & M_{dyn_x} & M_{dyn_u} \\ M_{cons_x} & M_{cons_x} & M_{cons_u} & ... & M_{cons_x} & M_{cons_u} \\ ... & ... & ... & ... & ... & ... \\ M_{cons_x} & M_{cons_x} & M_{cons_u} & ... & M_{cons_x} & M_{cons_u} \end{bmatrix} \quad O_{\left[n \cdot \left(n_s + n_g \right) + 1 \right] \times 1} \tag{40}$$

and the operator $\circ$ represents the Hadamard product.

It will then be possible to apply the knowledge acquired from the continuous functions to know which elements must be computed numerically (i.e. the nonzero elements of the matrix M_{GD}).

IV.D Theoretical Jacobian

Finally, a third contribution, the Theoretical Jacobian, arises in case we deal with problems having an unknown final time. The NLP state vector will then have a further variable. In this case, the Jacobian associated to this term is proportional to the output of the continuous functions f_C evaluated at all the collocation points, as it can be easily verified from (20).

$$Jac_{Theoretical} = -\frac{1}{2}\begin{bmatrix} O_{\left[n\cdot\left(n_s+n_g\right)+1\right]\times\left[\left(n+1\right)\cdot n_s+n\cdot n_c+1\right]} & \begin{matrix} 0 \\ \underline{f}_{C,1} \\ \underline{f}_{C,2} \\ .. \\ \underline{f}_{C,N} \end{matrix} \\ & O_{n\cdot n_g\times 1} \end{bmatrix} \tag{41}$$

The hybridization of the Jacobian matrix has as consequence that only a small part (between 4.68% and 8.84% in the cases analysed) of its elements need numerical computation techniques. Hence, significant CPU time is saved when solving the optimal control problem, or more specifically, the NLP problem representing it. In order to see which are the results, let´s see two significant examples.

V. Numerical Examples

V.A Orbit Raising Problem

This problem, as stated by Conway [8], has been reproposed more than once in literature [13], and deals with the maximization of the energy of an orbit in a given fixed time. It can be expressed as follows:

Maximize the specific energy

$$J = \frac{1}{r(t_F)} - \left\{\frac{1}{2}\left[V_r^2(t_F) + V_t^2(t_F)\right]\right\} \tag{42}$$

of an orbit subject to the following dynamics (expressed in canonical units)

$$\begin{aligned} \frac{dr}{dt} &= V_r \\ \frac{d\varphi}{dt} &= \frac{V_t}{r} \\ \frac{dV_r}{dt} &= \frac{V_t^2}{r} - \frac{\mu}{r^2} + T\sin\left(\delta\right) \\ \frac{dV_t}{dt} &= -\frac{V_r V_t}{r} + T\cos\left(\delta\right) \end{aligned} \tag{43}$$

where T is the specific force, assumed to be constant and equal to 0.01, μ is the normalized gravitational parameter and δ is the angle between the direction of the thrust and the tangential velocity. In this case the final time is equal to 50 TU. The solution is generated using between 25 and 100 nodes. Since the final time is known, the Jacobian here will only consist of the pseudosepctral (in red) and the numerical contributions (in blue for the differential equations, in black for the cost function). All the figures showing states, controls and constraints are associated to the solutions obtained using 100 nodes, while the jacobian structure derive from a NLP having 5 nodes (chosen for a better visualization of the patterns).

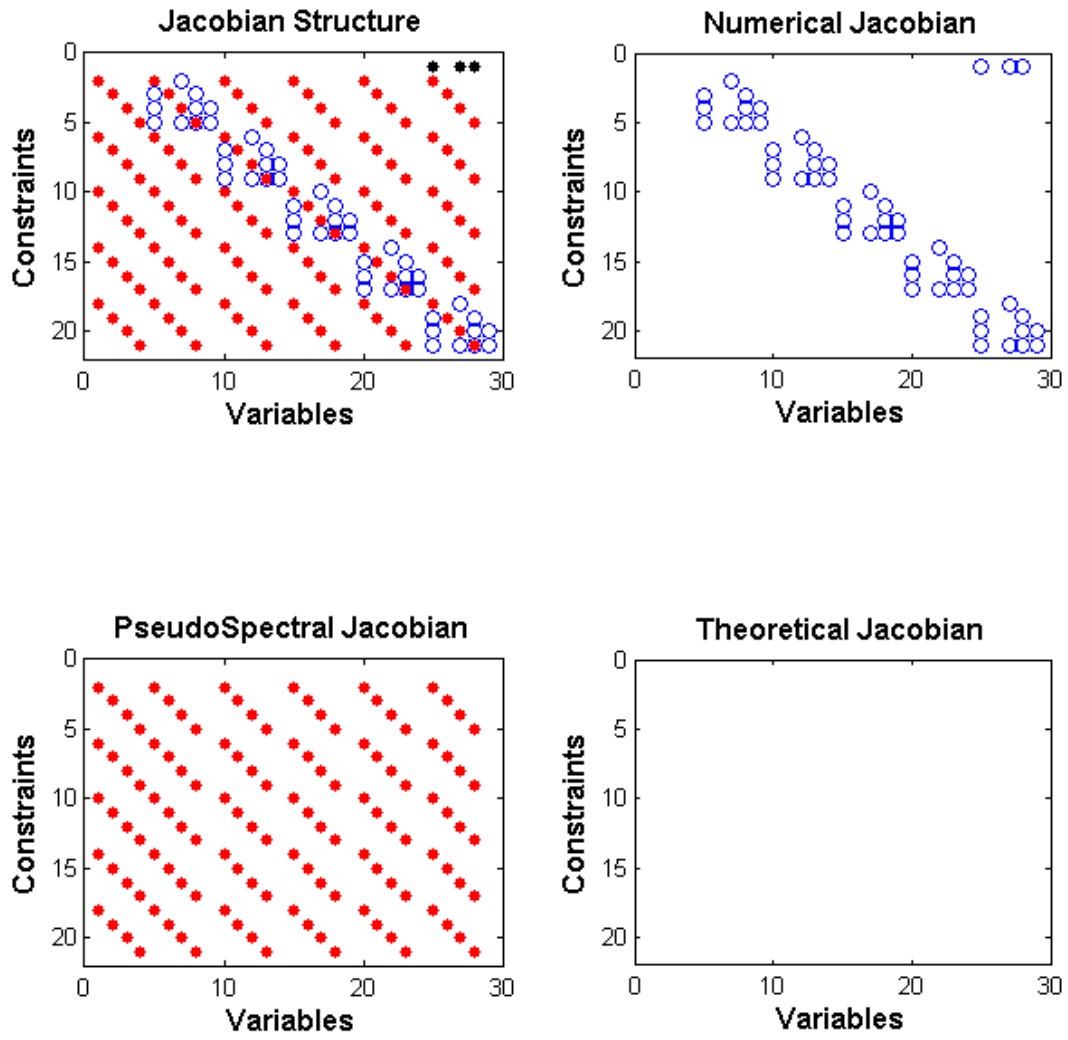

Figure 1. Jacobian Matrix Sparsity Patterns for Orbit Raising Problem.

The Pseudospectral Solution to this problem is represented in the following figures:

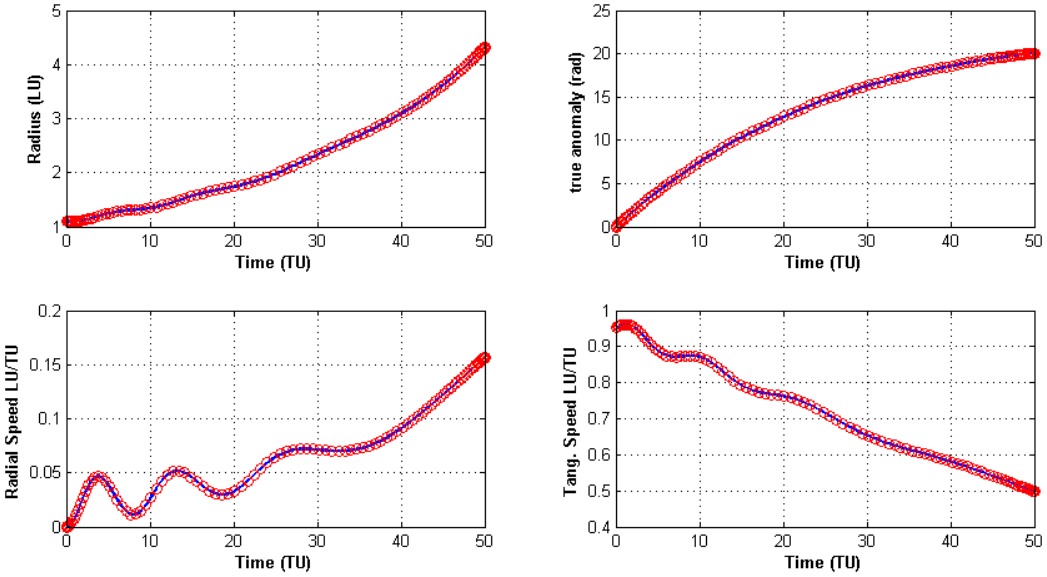

Figure 2. States Evolution for Orbit Raising Problem

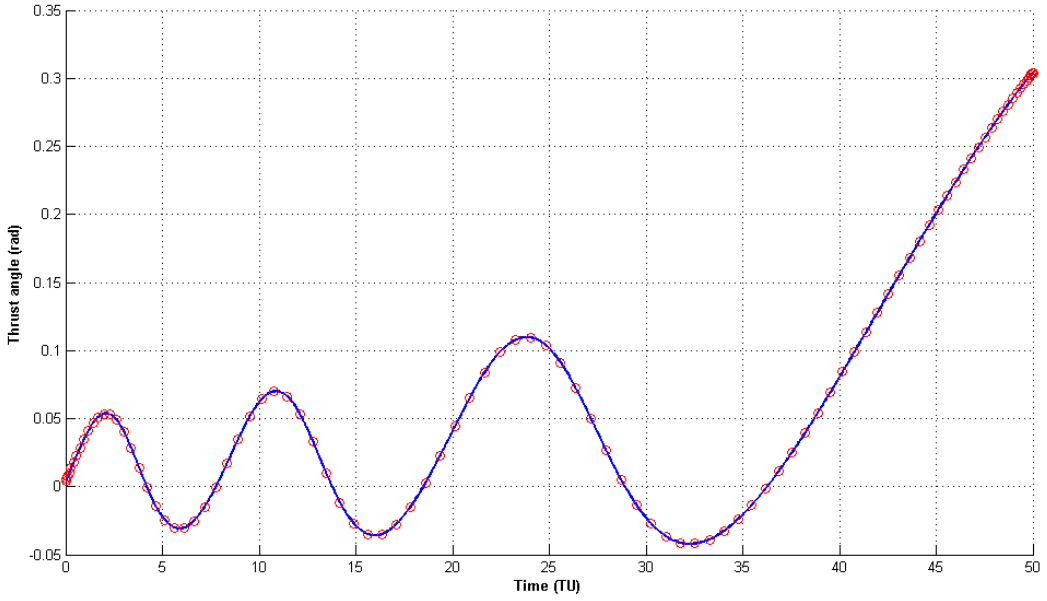

Figure 3. Control Evolution for Orbit Raising Problem

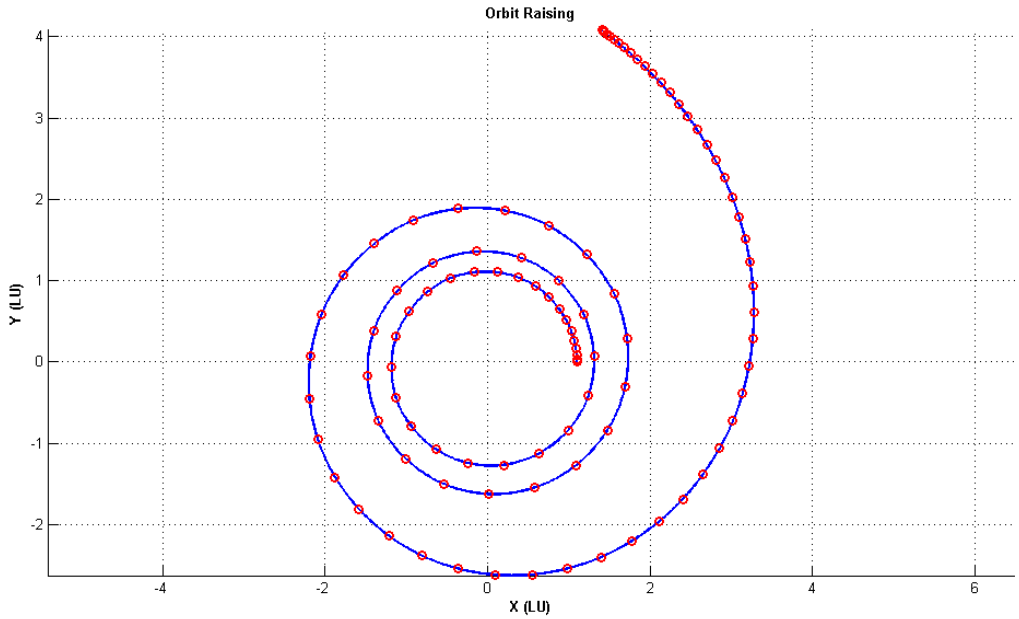

Figure 4. Orbit Evolution

V.B Space Shuttle Reentry

In this case the Shuttle Reentry Guidance [1] is implemented and solved. The problem is stated as follows:

Maximize the Cost Function

$$J=\theta(t_F) \tag{44}$$

subject to the dynamics

$$\begin{aligned}
\dot{h} &= V\sin(\gamma) \\
\dot{\phi} &= \tfrac{V}{r}\cos(\gamma)\sin(\psi)/\cos(\theta) \\
\dot{\theta} &= \tfrac{V}{r}\cos(\gamma)\cos(\psi) \\
\dot{V} &= -\tfrac{D}{m} - g\sin(\gamma) \\
\dot{\gamma} &= \tfrac{L}{mV}\cos(\beta) + \cos(\gamma)\left(\tfrac{V}{r} - \tfrac{g}{V}\right) \\
\dot{\psi} &= \tfrac{1}{mV\cos(\gamma)} L\sin(\beta) + \tfrac{V}{r\cos(\theta)}\cos(\gamma)\sin(\psi)\sin(\theta)
\end{aligned} \tag{45}$$

where

$$\begin{aligned}
L &= \tfrac{1}{2}\rho\cdot V^2\cdot S\cdot C_L \\
D &= \tfrac{1}{2}\rho\cdot V^2\cdot S\cdot C_D
\end{aligned} \tag{46}$$

C_L and C_D are respectively linear and quadratic functions of the Angle of Attack,

The initial conditions are

$$
\begin{aligned}
h &= 260000\ ft \\
\phi &= 0° \\
\theta &= 0° \\
V &= 25600\ ft/s \\
\gamma &= -1° \\
\psi &= 90°
\end{aligned}
\tag{47}
$$

and the final time is unknown. In this case the Jacobian has all the three components (red for PseudoSpectral, blue, light blue and black for numerical contributions respectively of dynamics, constraints and cost function, and green for the analytical contributions) presented in the section IV. The patterns representing the Jacobian are shown in the following figures.

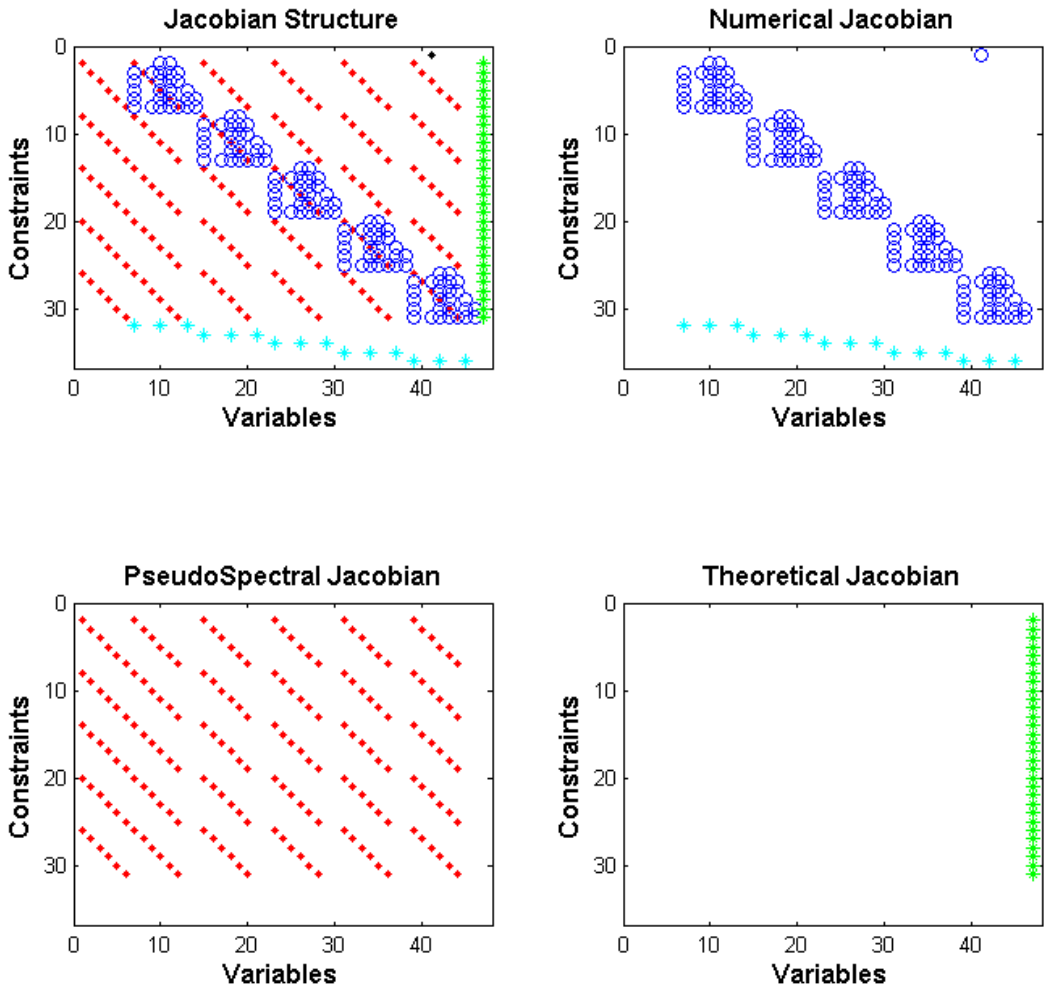

Figure 5. Jacobian Matrix Sparsity Patterns for Space Shuttle Reentry Problem.

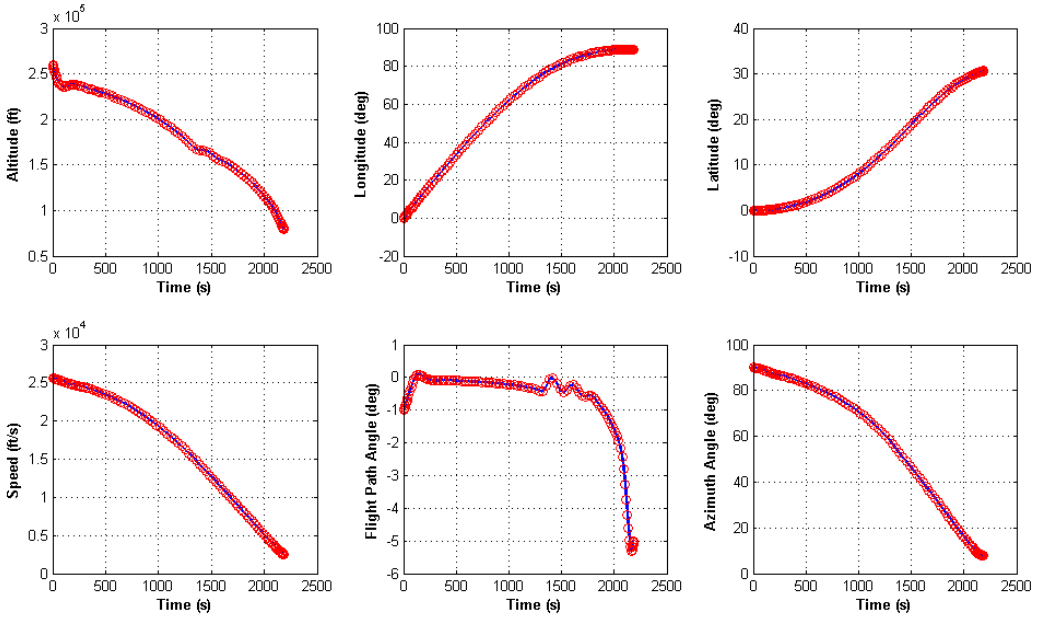

Figure 6. States Evolution for Space Shuttle Reentry Problem

.

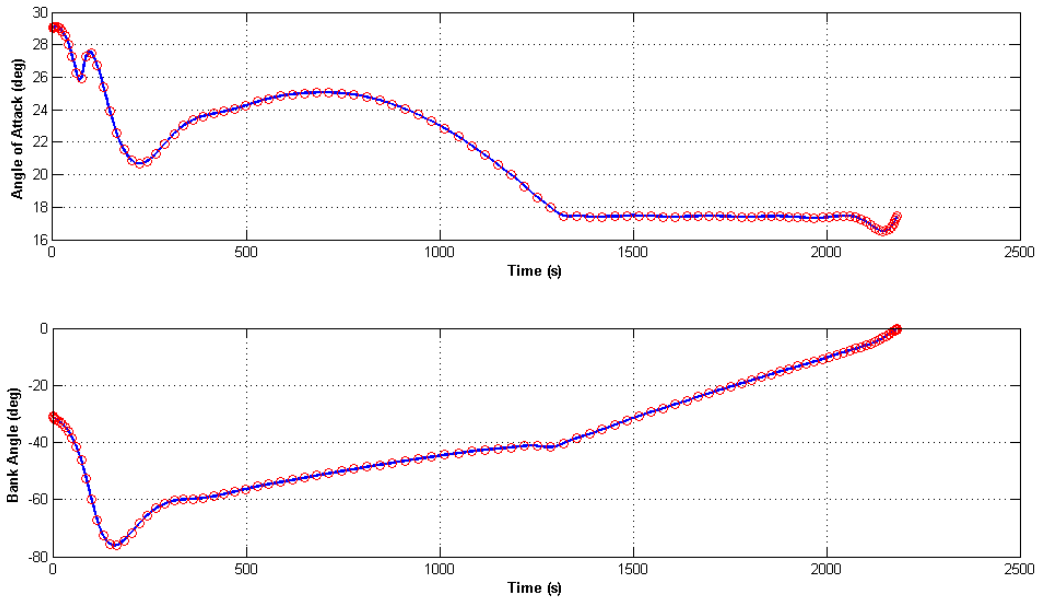

Figure 7. Controls Evolution for Space Shuttle Reentry Problem

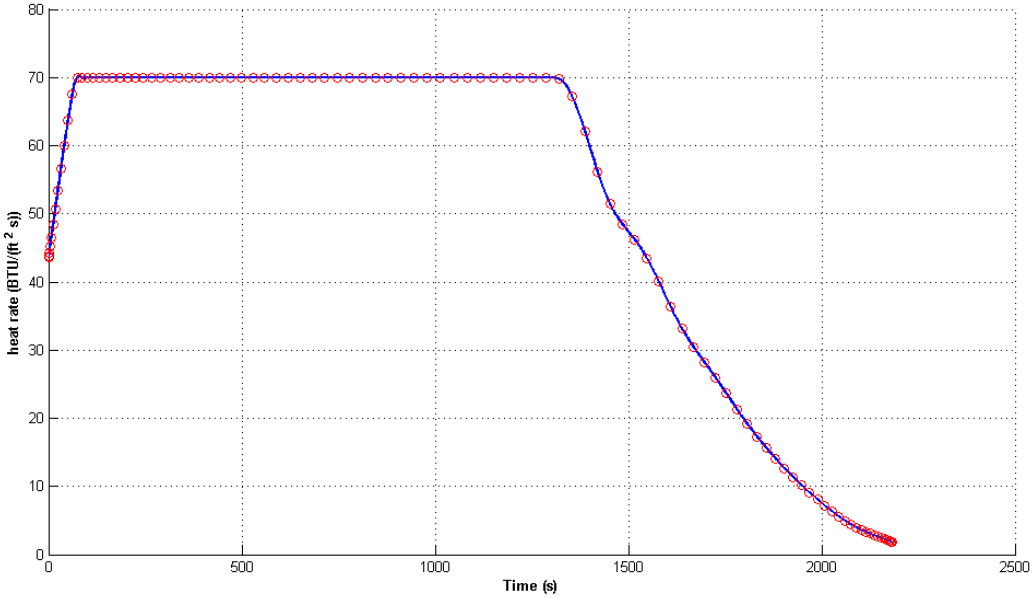

Figure 8. Heat Rate Evolution for Space Shuttle Reentry Problem

VI. Cpu Time Comparison

Here a comparison between the CPU time required to compute the solution of the full nonlinear Jacobian NLP problems and the hybrid Jacobian problems is performed. In particular, in the following table, four cases for each of the problems considered are compared. All the simulations have been performed with a laptop having a i7M640 CPU with a clock frequency of 2.80 GHz and 4 GB of RAM. All the examples considered do not have any meaningful initial guess from previous solutions. In other words all the problems were solved with cold start.

Nodes	CPU Time [s] (Full NLP)	CPU Time [s] (Hybrid NLP)	Offline Time (Common) [s]
25	11.1512	0.4517	0.3726
50	22.0853	1.5566	0.7162
75	71.4989	2.6371	1.7659
100	202.3225	12.2964	5.7351

Table 6 CPU Time Required for Orbit Raising Problem

Nodes	CPU Time [s] (Full NLP)	CPU Time [s] (Hybrid NLP)	Offline Time (Common) [s]
25	15.1166	0.8816	0.6147
50	42.8415	4.5443	1.4301
75	171.6521	18.2937	3.9240
100	594.1716	50.5697	10.8373

Table 7 CPU Time Required for Space Shuttle Reentry Problem

VII. Conclusions

In this paper a deeper analysis of the Jacobian structure associated to the use of Pseudospectral Methods has been performed, and in particular on the use of Flipped Radau PseudoSpectral Method. This analysis separated the Jacobian into three different terms which have been exploited and derived in the frame of the method implemented, bringing as result a faster computation of the NLP problems representing their respective Optimal Control Problems (OCPs) described here, and which can be efficiently computed using the aforementioned hybridization approach. The improvements in terms of cpu time have been compared to the use of the traditional techniques to compute the Jacobian. This significant save in CPU time brings new perspectives in the use of the optimal control problems as part of wider analysis tools like MonteCarlo or Worst-Case analyses, with no need to use more expensive solutions for what concerns the CPU capabilities required with such a complex problems. Further efforts can be performed in terms of optimization of matlab-based offline-code, and in terms of real-time implementation of the techniques here shown for real flying missions.

References

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

[2]Patterson, M. A., Rao, A. V., "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Optimal Control Problems", *Journal of Spacecraft and Rockets, Vol. 49 No.2 March-April 2012,*

[3]Rao, A. V., "A Survey of Numerical Methods for Optimal Control" *AAS/AIAA Astrodynamics Specialist Conference*, AAS Paper 09-334,

[4]Garg D., "Advances in Global PseudoSpectral Methods for Optimal Control," Ph.D. Dissertation, Aeronautics and Astronautics Dept., University of Florida., 2011.

[5]Gill, P., Murray,W., Saunders M.A., User's Guide for SNOPT Version 7:Software for Large-Scale Nonlinear Programming, University of California, 2007.

[6]Fahroo, F. Ross, M., "On Discrete-Time Optimality Conditions for Pseudospectral Methods," *Proc. AIAA/AAS Astrodynamics Specialist Conference,* Keystone, Colorado, August 2006. AIAA-2006-6304.

[7]Huntington G. D., "Advancement and Analysis of a Gauss Pseudospectral Transcription for Optimal Control Problems," Ph.D. Dissertation, Aeronautics and Astronautics Dept., Massachusetts Institute of Technology, 2007.

[8]Herman, A. L., Conway B., "Direct Optimization Using Collocation Based on High-Order Gauss-Lobatto Quadrature Rules" *Journal of Guidance, Control and Dynamics, Vol. 19 No.3 May-June 1996,*

[9]Sagliano, M., Theil S., "A Radau Pseudospectral Method-Based Guidance Reentry Algorithm" 63rd International Astronautical Congress, Naples, Italy, 2012.

[10]Garg D., Patterson M. A., Francolin C., Darby C.L., Huntington G.T., Hager W.V., Rao A.V., "Direct trajectory optimization and costate estimation of finite-horizon and infinite-horizon optimal control problems using a Radau pseudospectral method," *Computational Optimization and Applications, Volume 49 Issue 2, June 2011, Pages 335-358.*

[11]Trefethen, L.N., Spectral Methods in Matlab, Oxford University, 2000, SIAM, Philadelphia.

[12]Martins, J. R. R. A., Sturdza P., Alonso J. J., "The Complex-Step Derivative Approximation," *ACM Transaction on Mathematical Software, Vol. 29, No. 3, September 2003, Pages 245-262.*

[13]Fumenti, F. et al. "Collocation Points Distributions for optimal spacecraft trajectories," *Commun Nonlinear Sci Numer Simulat (2012), http://dx.doi.org/10.1016/j.cnsns.2012.07.023*