Utilizing the Algorithmic Differentiation Package *ADiGator* for Solving Optimal Control Problems Using Direct Collocation

Matthew J. Weinstein[*]
Michael A. Patterson[†]
Anil V. Rao[‡]

University of Florida
Gainesville, FL 32611

Abstract

ADiGator is a newly developed free MATLAB algorithmic differentiation package. In this paper, we study the use of the *ADiGator* algorithmic differentiation tool in order to supply the first and second derivatives of the NLP arising from direct collocation of optimal control problems. While the methods of this paper may be applied to multiple direct collocation schemes, we focus on an *hp-adaptive Legendre-Gauss-Radau* scheme which has been coded in the MATLAB optimal control software *GPOPS-II*. The methods required to indirectly compute the first and second derivatives of the NLP using *ADiGator* are presented and three test cases are given. In the test cases, the method of supplying derivatives via *ADiGator* is shown to be highly efficient when compared to the classical method of finite-differencing.

1 Introduction

Over the past two decades the *direct collocation* method of solving optimal control problems has gained a great deal of popularity. In a direct collocation approach, the state is approximated using a set of trial (basis) functions and the dynamics are collocated at a specified set of points in the time interval. The state and control are discretized and the differential-algebraic equations are enforced at a set of discrete points. The new discretized problem is then solved by means of either a first- or second-derivative NLP solver. In a first-derivative (quasi-Newton) NLP solver, the objective function gradient and constraint Jacobian are used together with a dense quasi-Newton approximation of the Lagrangian Hessian. In a second-derivative (Newton) NLP solver,

[*]Ph.D. Candidate, Department of Mechanical and Aerospace Engineering, E-mail: mweinstein@ufl.edu.
[†]Visiting Faculty, Department of Mechanical and Aerospace Engineering. E-mail: mpatterson@ufl.edu.
[‡]Assistant Professor, Department of Mechanical and Aerospace Engineering. E-mail: anilvrao@ufl.edu. Associate Fellow, AIAA. Corresponding Author.

the objective function gradient and constraint Jacobian are used together with the Hessian of the NLP Lagrangian. While it is known that providing the Lagrangian Hessian can significantly improve the computational performance of an NLP solver, quasi-Newton methods are more commonly used due to the difficulty presented in computing the Lagrangian Hessian. Multiple optimal control algorithms which use direct collocation methods have been developed, among them are the well known programs *SOCS* [1], *DIDO* [2], *DIRCOL* [3], *GPOPS* [4], and *GPOPS-II* [5].

The classical approach of obtaining numerical derivative approximations of the resulting NLP is that of finite-differencing. In a finite-differencing approach, the derivative is approximated by evaluating the function at two neighboring points and dividing the difference of the function by the difference of the points. The use of finite differences is appealing due to the fact that only evaluations of the user function are required, however, the accuracy of the derivative is completely dependent upon the chosen spacing of the neighboring points. If the spacing is too small, then cancellation error reduces the significance of the function evaluations. If the spacing is too large, then truncation errors become significant. Moreover, even if the spacing is chosen optimally, one can only expect the derivative to be accurate to roughly 1/2 to 2/3 the significant digits of the original function. This ratio then becomes even smaller for second and higher order derivatives.

The desire for a method which accurately and efficiently computes derivatives automatically has lead to the field of research known as *automatic differentiation* (AD) or as it has been more recently referred, *algorithmic differentiation*. AD is defined as the process of determining accurate derivatives of a function defined by computer programs using the rules of differential calculus [6]. Assuming a computer program is differentiable, AD exploits the fact that a user program may be broken into a sequence of elementary operations, where each elementary operation has a corresponding derivative rule. Thus, given the derivative rules of each elementary operation, a derivative of the program is obtained by a systematic application of the chain rule, where the resulting derivative is exact up to machine precision.

ADiGator is a newly developed free MATLAB algorithmic differentiation package [7]. The tool utilizes source transformation via operator overloading in order to compute the derivatives of functions defined by MATLAB programs. The result of the method is a MATLAB program, which, when evaluated numerically, computes the possible non-zero derivatives of the original function program. The tool is appealing for computing the derivatives of direct collocation optimal control problems for a number of reasons. Namely, due to the fact that the resulting derivative code relies solely upon the native MATLAB library, no time penalties associated with operator overloading are incurred at the time of derivative evaluation. Additionally, *ADiGator* may be used in its *vectorized* mode to efficiently compute vectorized derivatives of vectorized functions, such as those resulting from direct collocation schemes. Furthermore, the method may be recursively applied in order to generate second-order derivative code, however, Hessian symmetry may not be exploited.

In this paper we study the use of the *ADiGator* algorithmic differentiation package in order to compute the the first and second derivatives of the NLP arising from direct collocation methods. While the methods of this paper may be applied to multiple direct collocation schemes, we focus on an *hp-adaptive Legendre-Gauss-Radau* [8, 9, 10, 11, 12, 13] scheme which has been coded in the commercial optimal control software *GPOPS-II*. This provides us with a good test environment due to the manner in which *GPOPS-II* requires only the derivatives of the control problem [5], together with the fact that it is implemented in the same language as the *ADiGator* tool. Moreover, *GPOPS-II* utilizes the open-source second derivative NLP solver *IPOPT* in order to solve the NLP which results from collocation. This paper is organized as follows. In Section 2 we introduce a generic transformed Bolza optimal control problem. In Section 3 we formulate the NLP which results from the hp Legendre-Gauss-Radau collocation of the transformed Bolza problem. In Section 4 we introduce sparse derivative notations which allow for the concise representation of the required derivatives of the NLP. In Section 5 we provide the methods used in order to use the *ADiGator* tool to provide the optimal control software *GPOPS-II* with the required derivative information. In Section 6 the method is tested on three problems. Finally, in Section 7, conclusions are provided.

2 Transformed Bolza Optimal Control Problem

In this paper we consider the following single phase continuous time Bolza optimal control problem transformed from the domain $t \in [t_0,\ t_f]$ to the fixed domain $\tau \in [-1,\ 1]$. This transformation is achieved by the affine transformation

$$\tau = \frac{2}{t_f - t_0} t + \frac{t_f + t_0}{t_f - t_0}. \tag{1}$$

The continuous time Bolza problem is then defined on the interval $[-1,\ 1]$ as follows. Determine the state, $\mathbf{y}(\tau) \in \mathbb{R}^n$, control, $\mathbf{u}(\tau) \in \mathbb{R}^m$, initial time, t_0, and final time, t_f, that minimizes the cost functional

$$J = \Phi(\mathbf{y}(-1), t_0, \mathbf{y}(+1), t_f) + \frac{t_f - t_0}{2} \int_{-1}^{+1} g(\mathbf{y}(\tau), \mathbf{u}(\tau)) d\tau, \tag{2}$$

subject to the dynamic constraints

$$\frac{d\mathbf{y}}{d\tau} = \frac{t_f - t_0}{2} \mathbf{f}(\mathbf{y}(\tau), \mathbf{u}(\tau)) \in \mathbb{R}^n, \tag{3}$$

the path constraints

$$\mathbf{c}_{\min} \leq \mathbf{c}(\mathbf{y}(\tau), \mathbf{u}(\tau)) \leq \mathbf{c}_{\max} \in \mathbb{R}^p, \tag{4}$$

and boundary conditions

$$\boldsymbol{\phi}(\mathbf{y}(-1), t_0, \mathbf{y}(+1), t_f) = \mathbf{0} \in \mathbb{R}^{\mathbf{q}}. \tag{5}$$

The optimal control problem of Eqs. (2)–(5) will be referred to as the *transformed continuous Bolza problem*. Here it is noted that a solution to the transformed continuous Bolza problem can be transformed from the domain $\tau \in [-1,\ +1]$ to the domain $t \in [t_0,\ t_f]$ via the affine transformation

$$t = \frac{t_f - t_0}{2} \tau + \frac{t_f - t_0}{2}. \tag{6}$$

3 Formulation of NLP Resulting from Radau Collocation

In this section the NLP which arises from the multiple-interval Legendre-Gauss-Radau (LGR) collocation of the transformed Bolza problem of Section 2 is given. In the LGR collocation method, the interval $\tau \in [-1,\ 1]$ is divided into a *mesh* consisting of K *mesh intervals* defined by the mesh points $-1 = T_0 < T_1 < \cdots < T_K = +1$. In each mesh interval k, the state is approximated by a $(N_k+1)^{th}$ degree Lagrange polynomial such that

$$\mathbf{y}^{(k)}(\tau) \approx \mathbf{Y}^{(k)}(\tau) = \sum_{i=1}^{N_k+1} \mathbf{Y}_i^{(k)} l_i^{(k)}(\tau), \quad l_i^{(k)}(\tau) = \prod_{\substack{j=1 \\ j \neq i}}^{N_k} \frac{\tau - \tau_j^{(k)}}{\tau_i^{(k)} - \tau_j^{(k)}}, \tag{7}$$

where $\mathbf{Y}^{(k)}(\tau)$ is the approximation of the state $\mathbf{y}^{(k)}(\tau)$ in the k^{th} mesh interval, $\tau \in [-1,\ 1]$, $l_i^{(k)}(\tau)$, $(i = 1, \ldots, N_k+1)$ is a basis of Lagrange polynomials, $\tau_1^{(k)}, \ldots, \tau_{N_k}^{(k)}$ are the Legendre-Gauss-Radau collocation points defined on the subinterval $\tau^{(k)} \in [T_{k-1},\ T_k)$, and $T_{N_k+1}^{(k)} = T_K$ is a noncollocated point. The dynamic constraints of Eq. (3) are enforced by either differentiating or integrating Eq. (7) with respect to τ to yield the defect constraints. Moreover, the quadrature rule used to approximate the integral of Eq. (2) is obtained by integrating Eq. (7) with respect to τ.

Using the variables $\mathbf{Y}_i^{(k)} \in \mathbb{R}^n$ and $\mathbf{U}_i^{(k)} \in \mathbb{R}^m$ to represent the discretized state and control corresponding to the point $\tau_i^{(k)}$, we define the collection of discrete state and control variables across the k^{th} interval by

$$\mathbf{Y}^{(k)} = \left[\begin{array}{cccc} \mathbf{Y}_1^{(k)} & \mathbf{Y}_2^{(k)} & \cdots & \mathbf{Y}_{N_k}^{(k)} \end{array} \right] \in \mathbb{R}^{n \times N}, \tag{8}$$

and

$$\mathbf{U}^{(k)} = \left[\begin{array}{cccc} \mathbf{U}_1^{(k)} & \mathbf{U}_2^{(k)} & \cdots & \mathbf{U}_{N_k}^{(k)} \end{array} \right] \in \mathbb{R}^{m \times N}, \tag{9}$$

respectively. It is noted that continuity in the state at the interior mesh points $k \in [1, \ldots, K-1]$ is enforced via the condition $\mathbf{Y}_{N_k+1}^{(k)} = \mathbf{Y}_1^{(k+1)}, \quad (k = 1, \ldots, K-1)$, where the *same* variable is used for both $\mathbf{Y}_{N_k+1}^{(k)}$ and $\mathbf{Y}_1^{(k+1)}$. The values of the state and control corresponding to the $N = \sum_{k=1}^{K} N_k$ LGR points are then collected into the matrices

$$\mathbf{Y} = \left[\begin{array}{cccc} \mathbf{Y}^{(1)} & \mathbf{Y}^{(1)} & \cdots & \mathbf{Y}^{(K)} \end{array} \right] \in \mathbb{R}^{n \times N}, \tag{10}$$

and

$$\mathbf{U} = \left[\begin{array}{cccc} \mathbf{U}^{(1)} & \mathbf{U}^{(2)} & \cdots & \mathbf{U}^{(K)} \end{array} \right] \in \mathbb{R}^{n \times N}, \tag{11}$$

respectively. Allowing $\mathbf{Y}_i$ and $\mathbf{U}_i$ to represent the i^{th} columns of $\mathbf{Y}$ and $\mathbf{U}$, the continuous functions $g(\mathbf{y}(\tau), \mathbf{u}(\tau))$, $\mathbf{f}(\mathbf{y}(\tau), \mathbf{u}(\tau))$, and $\mathbf{c}(\mathbf{y}(\tau), \mathbf{u}(\tau))$ of Section 2 *evaluated* at the points $(\mathbf{Y}_i, \mathbf{U}_i)$, $(i = 1, \ldots, N)$ are defined as

$$\mathbf{G}(\mathbf{Y}, \mathbf{U}) = \left[\begin{array}{cccc} g(\mathbf{Y}_1, \mathbf{U}_1) & g(\mathbf{Y}_2, \mathbf{U}_2) & \cdots & g(\mathbf{Y}_N, \mathbf{U}_N) \end{array} \right] \in \mathbb{R}^{1 \times N}, \tag{12}$$

$$\mathbf{F}(\mathbf{Y}, \mathbf{U}) = \left[\begin{array}{cccc} \mathbf{f}(\mathbf{Y}_1, \mathbf{U}_1) & \mathbf{f}(\mathbf{Y}_2, \mathbf{U}_2) & \cdots & \mathbf{f}(\mathbf{Y}_N, \mathbf{U}_N) \end{array} \right] \in \mathbb{R}^{n \times N}, \tag{13}$$

and

$$\mathbf{C}(\mathbf{Y}, \mathbf{U}) = \left[\begin{array}{cccc} \mathbf{c}(\mathbf{Y}_1, \mathbf{U}_1) & \mathbf{c}(\mathbf{Y}_2, \mathbf{U}_2) & \cdots & \mathbf{c}(\mathbf{Y}_N, \mathbf{U}_N) \end{array} \right] \in \mathbb{R}^{p \times N}, \tag{14}$$

respectively.

Allowing $\mathbf{Y}_{N+1} \in \mathbb{R}^n$ to be the state discretized at the point $\tau = +1$, the NLP may now be formed as follows. Determine the discrete variables $\mathbf{Y} \in \mathbb{R}^{n\times N}$, $\mathbf{Y}_{N+1} \in \mathbb{R}^n$, $\mathbf{U} \in \mathbb{R}^{m\times N}$, $t_0 \in \mathbb{R}$, and $t_f \in \mathbb{R}$ which minimize the cost function

$$J \approx \Phi(\mathbf{Y}_1, t_0, \mathbf{Y}_{N+1}, t_f) + \frac{t_f - t_0}{2}\mathbf{G}(\mathbf{Y}, \mathbf{U})\mathbf{w}, \tag{15}$$

subject to the defect constraints

$$\left[\begin{array}{cc} \mathbf{Y} & \mathbf{Y}_{N+1} \end{array}\right] \mathbf{A}^{\mathsf{T}} - \frac{t_f - t_0}{2}\mathbf{F}(\mathbf{Y}, \mathbf{U})\mathbf{B}^{\mathsf{T}} = \mathbf{0} \in \mathbb{R}^{n\times N}, \tag{16}$$

the path constraints

$$\mathbf{C}_{\min} \leq \mathbf{C}(\mathbf{Y}, \mathbf{U}) \leq \mathbf{C}_{\max} \in \mathbb{R}^{p\times N}, \tag{17}$$

and boundary conditions

$$\boldsymbol{\phi}(\mathbf{Y}_1, t_0, \mathbf{Y}_{N+1}, t_f) = \mathbf{0} \in \mathbb{R}^q, \tag{18}$$

where $\mathbf{w} \in \mathbb{R}^N$ is a column vector of LGR quadrature weights, and the matrices $\mathbf{A} \in \mathbb{R}^{N\times(N+1)}$ and $\mathbf{B} \in \mathbb{R}^{N\times N}$ are constant matrices whose values are determined by the given mesh together with whether a differentiation or integration scheme is being used. If an LGR differentiation scheme is being used, then the matrix $\mathbf{A}$ is the LGR differentiation matrix and $\mathbf{B}$ is the identity matrix. If an LGR integration scheme is being used, then the matrix $\mathbf{A}$ consists of entries of 0, -1, and $+1$, and the matrix $\mathbf{B}$ is the LGR integration matrix.

3.1 Required NLP Derivatives

In order to solve the NLP of Section 3 with a second-derivative (Newton) NLP solver, it is required that the objective gradient, constraint Jacobian, and Lagrangian Hessian be supplied. While one could simply obtain these derivatives by differentiating the objective and constraints of Section 3, it is more efficient to instead differentiate the optimal control functions and to then build the NLP derivatives. That is, because only the endpoint functions $\mathbf{\Phi}(\cdot)$ and $\phi(\cdot)$ and the vectorized functions $\mathbf{G}(\cdot)$, $\mathbf{F}(\cdot)$, and $\mathbf{C}(\cdot)$ may change, only the first and second derivatives of the endpoint functions and vectorized functions with respect to their given arguments are required in order to build the first and second derivatives of the NLP. In order to concisely write the required optimal control derivatives, we first introduce the variables

$$\mathbf{v} = \begin{bmatrix} \mathbf{Y}_1 \\ \mathbf{Y}_{N+1} \\ t_0 \\ t_f \end{bmatrix} \in \mathbb{R}^{2n+2} \tag{19}$$

and

$$\mathbf{Z} = \left[\begin{array}{cc} \mathbf{Y} & \mathbf{U} \end{array}\right] \in \mathbb{R}^{(n+m)\times N}. \tag{20}$$

Then, given the first and second derivatives of the endpoint functions $\mathbf{\Phi}(\mathbf{v})$ and $\boldsymbol{\phi}(\mathbf{v})$ with respect to $\mathbf{v}$, together with the first and second derivatives of the vectorized functions $\mathbf{G}(\mathbf{Z})$, $\mathbf{F}(\mathbf{Z})$, and $\mathbf{C}(\mathbf{Z})$ with respect to $\mathbf{Z}$, the first and second derivatives of the NLP may be built.

4 Sparse Derivative Representations

From Section 3.1 it is seen that in order to supply the NLP solver with first and second derivatives, it is required to compute the first and second derivatives of both endpoint and vectorized functions. While the required multi-dimensional derivative objects can be quite large, they typically contain many zeros, particularly when dealing with the vectorized functions. In this section we provide sparse representations of both endpoint function and vectorized function derivatives.

4.1 Sparse Representation of Endpoint Function Derivatives

Without loss of generality, consider a vector function of a vector $\mathbf{f}(\mathbf{x})$ where $\mathbf{f} : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. The *Jacobian* of the vector function $\mathbf{f}(\mathbf{x})$, denoted $\bigtriangledown_{\mathbf{x}}\mathbf{f}(\mathbf{x})$, is then an $m \times n$ matrix

$$\bigtriangledown_{\mathbf{x}}\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_m} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m\times n}. \tag{21}$$

Assuming the first derivative matrix $\bigtriangledown_{\mathbf{x}}\mathbf{f}(\mathbf{x})$ contains $p \leq mn$ *possible* non-zero elements, we denote $\mathbf{i}^{\mathbf{f}}_{\mathbf{x}} \in \mathbb{Z}^p_+$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{x}} \in \mathbb{Z}^p_+$, to be the row and column locations of the possible non-zero elements of $\bigtriangledown_{\mathbf{x}}\mathbf{f}(\mathbf{x})$. Furthermore, we denote $\mathbf{d}^{\mathbf{f}}_{\mathbf{x}} \in \mathbb{R}^p$ to be the possible non-zero elements of $\bigtriangledown_{\mathbf{x}}\mathbf{f}(\mathbf{x})$ such that

$$d^{\mathbf{f}}_{\mathbf{x}}(k) = \frac{\partial f_{\left[i^{\mathbf{f}}_{\mathbf{x}}(k)\right]}}{\partial x_{\left[j^{\mathbf{f}}_{\mathbf{x}}(k)\right]}}, \quad k = 1, \ldots, p, \tag{22}$$

where $d^{\mathbf{f}}_{\mathbf{x}}(k)$, $i^{\mathbf{f}}_{\mathbf{x}}(k)$, and $j^{\mathbf{f}}_{\mathbf{x}}(k)$ refer to the k^{th} elements of the vectors $\mathbf{d}^{\mathbf{f}}_{\mathbf{x}}$, $\mathbf{i}^{\mathbf{f}}_{\mathbf{x}}$, and $\mathbf{j}^{\mathbf{f}}_{\mathbf{x}}$, respectively. Using this sparse notation, the Jacobian $\bigtriangledown_{\mathbf{x}}\mathbf{f}(\mathbf{x})$ may be fully defined given the vectors $\mathbf{d}^{\mathbf{f}}_{\mathbf{x}}$, $\mathbf{i}^{\mathbf{f}}_{\mathbf{x}}$, and $\mathbf{j}^{\mathbf{f}}_{\mathbf{x}}$ together with the dimensions m and n. The *second* derivative of the function $\mathbf{f}(\mathbf{x})$ is then the three dimensional matrix $\bigtriangledown^2_{\mathbf{xx}}\mathbf{f}(\mathbf{x}) \in \mathbb{R}^{m\times n\times n}$. Assuming now that $\bigtriangledown^2_{\mathbf{xx}}\mathbf{f}(\mathbf{x})$ contains $q \leq mnn$ possible non-zero elements, we denote $\mathbf{i}^{\mathbf{f}}_{\mathbf{xx}} \in \mathbb{Z}^q_+$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{xx}} \in \mathbb{Z}^q_+$, and $\mathbf{k}^{\mathbf{f}}_{\mathbf{xx}} \in \mathbb{Z}^q_+$ to be the row, column, and layer (third dimension) locations of the non-zero elements of $\bigtriangledown^2_{\mathbf{xx}}\mathbf{f}(\mathbf{x})$. Moreover, we denote $\mathbf{d}^{\mathbf{f}}_{\mathbf{xx}} \in \mathbb{R}^q$ to be the non-zero entries of $\bigtriangledown^2_{\mathbf{xx}}\mathbf{f}(\mathbf{x})$ such that

$$d^{\mathbf{f}}_{\mathbf{xx}}(l) = \frac{\partial^2 f_{\left[i^{\mathbf{f}}_{\mathbf{xx}}(l)\right]}}{\partial x_{\left[j^{\mathbf{f}}_{\mathbf{xx}}(l)\right]}\partial x_{\left[k^{\mathbf{f}}_{\mathbf{xx}}(l)\right]}}, \quad l = 1, \ldots, q, \tag{23}$$

where $d^{\mathbf{f}}_{\mathbf{xx}}(l)$, $i^{\mathbf{f}}_{\mathbf{xx}}(l)$, $j^{\mathbf{f}}_{\mathbf{xx}}(l)$ and $k^{\mathbf{f}}_{\mathbf{xx}}(l)$ refer to the l^{th} elements of the vectors $\mathbf{d}^{\mathbf{f}}_{\mathbf{xx}}$, $\mathbf{i}^{\mathbf{f}}_{\mathbf{xx}}$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{xx}}$, and $\mathbf{k}^{\mathbf{f}}_{\mathbf{xx}}$, respectively. Using this sparse notation, it is then also the case that the second derivative matrix $\bigtriangledown^2_{\mathbf{xx}}\mathbf{f}(\mathbf{x})$ may be fully defined given the vectors $\mathbf{d}^{\mathbf{f}}_{\mathbf{xx}}$, $\mathbf{i}^{\mathbf{f}}_{\mathbf{xx}}$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{xx}}$, and $\mathbf{k}^{\mathbf{f}}_{\mathbf{xx}}$, together with the dimensions m, n, and n.

4.2 Sparse Representation of Vectorized Function Derivatives

Without loss of generality, consider a function $\mathbf{f}(\mathbf{x}(\sigma)) : \mathbb{R}^n \to \mathbb{R}^m$. Assuming the first derivative matrix $\bigtriangledown_{\mathbf{x}(\sigma)}\mathbf{f}(\mathbf{x}(\sigma)) \in \mathbb{R}^{m\times n}$ contains $p \leq mn$ elements, we again denote $\mathbf{i}^{\mathbf{f}}_{\mathbf{x}} \in \mathbb{Z}^p_+$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{x}} \in \mathbb{Z}^p_+$, to be the row and column locations of the possible non-zero elements of $\triangledown_{\mathbf{x}(\sigma)}\mathbf{f}(\mathbf{x}(\sigma))$. Likewise, assuming the second derivative matrix $\triangledown^2_{\mathbf{x}(\sigma)\mathbf{x}(\sigma)}\mathbf{f}(\mathbf{x}(\sigma)) \in \mathbb{R}^{m\times n\times n}$ contains $q \leq mnn$ non-zero entries, we again denote $\mathbf{i}^{\mathbf{f}}_{\mathbf{xx}} \in \mathbb{Z}^q_+$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{xx}} \in \mathbb{Z}^q_+$, and $\mathbf{k}^{\mathbf{f}}_{\mathbf{xx}} \in \mathbb{Z}^q_+$ to be the row, column, and layer (third dimension) locations of the possible non-zero elements of $\triangledown^2_{\mathbf{x}(\sigma)\mathbf{x}(\sigma)}\mathbf{f}(\mathbf{x}(\sigma))$. We now denote the vector $\mathbf{x}(\sigma)$ discretized at N points by the matrix $\mathbf{X}$ such that

$$\mathbf{X} = \left[\begin{array}{cccc} \mathbf{X}_1 & \mathbf{X}_2 & \cdots & \mathbf{X}_N \end{array}\right] \in \mathbb{R}^{n\times N}, \tag{24}$$

where $\mathbf{X}_h \in \mathbb{R}^n$, $h = 1,\ldots,N$, denotes the vector $\mathbf{x}(\sigma)$ discretized at the h^{th} point. The values of the function $\mathbf{f}(\mathbf{x}(\sigma))$ *evaluated* at the points $\mathbf{X}$ are then denoted by

$$\mathbf{F}(\mathbf{X}) = \left[\begin{array}{cccc} \mathbf{f}(\mathbf{X}_1) & \mathbf{f}(\mathbf{X}_2) & \cdots & \mathbf{f}(\mathbf{X}_N) \end{array}\right] \in \mathbb{R}^{n\times N}. \tag{25}$$

The first derivative of $\mathbf{F}(\mathbf{X})$ with respect to $\mathbf{X}$ is then the four-dimensional array $\triangledown_{\mathbf{X}}\mathbf{F}(\mathbf{X}) \in \mathbb{R}^{m\times N\times n\times N}$, where, due to the vectorized nature of the function $\mathbf{F}$, it is the case that only the entries

$$\frac{\partial F_{h,i}}{\partial X_{h,j}}, \quad \left(h = 1,\ldots,N, \quad i \in \mathbf{i}^{\mathbf{f}}_{\mathbf{x}}, \quad j \in \mathbf{j}^{\mathbf{f}}_{\mathbf{x}}\right) \tag{26}$$

are possibly non-zero. We now denote $\mathbf{d}^{\mathbf{f}}_{\mathbf{X}_h} \in \mathbb{R}^p$ to be the possible non-zero entries of the derivative matrix $\triangledown_{\mathbf{X}_h}\mathbf{f}(\mathbf{X}_h)$, laying in the $\mathbf{i}^{\mathbf{f}}_{\mathbf{x}}$ and $\mathbf{j}^{\mathbf{f}}_{\mathbf{x}}$ row and column locations. Using this notation, the possible non-zero entries of $\triangledown_{\mathbf{X}}\mathbf{F}(\mathbf{X})$ may be represented by the matrix

$$\mathbf{D}^{\mathbf{F}}_{\mathbf{X}} = \left[\begin{array}{cccc} \mathbf{d}^{\mathbf{f}}_{\mathbf{X}_1} & \mathbf{d}^{\mathbf{f}}_{\mathbf{X}_2} & \cdots & \mathbf{d}^{\mathbf{f}}_{\mathbf{X}_N} \end{array}\right] \in \mathbb{R}^{p\times N} \tag{27}$$

which may be mapped into $\triangledown_{\mathbf{X}}\mathbf{F}(\mathbf{X})$ given the indices $\mathbf{i}^{\mathbf{f}}_{\mathbf{x}}$ and $\mathbf{j}^{\mathbf{f}}_{\mathbf{x}}$. Thus, the four-dimensional first derivative matrix $\triangledown_{\mathbf{X}}\mathbf{F}(\mathbf{X})$ may be fully defined given the possible non-zero elements $\mathbf{D}^{\mathbf{F}}_{\mathbf{X}}$, the row and column indices $\mathbf{i}^{\mathbf{f}}_{\mathbf{x}}$ and $\mathbf{j}^{\mathbf{f}}_{\mathbf{x}}$ and the dimensions m, and n. Similarly, the second derivative of $\mathbf{F}(\mathbf{X})$ with respect to $\mathbf{X}$ is given by the six-dimensional array $\triangledown^2_{\mathbf{XX}}\mathbf{F}(\mathbf{X}) \in \mathbb{R}^{m\times N\times n\times N\times n\times N}$, where, again due to the vectorized nature of the function $\mathbf{F}$, it is the case that only the entries

$$\frac{\partial^2 F_{h,i}}{\partial X_{h,j}\partial X_{h,k}}, \quad \left(h = 1,\ldots,N, \quad i \in \mathbf{i}^{\mathbf{f}}_{\mathbf{xx}}, \quad j \in \mathbf{j}^{\mathbf{f}}_{\mathbf{xx}}, \quad k \in \mathbf{k}^{\mathbf{f}}_{\mathbf{xx}}\right) \tag{28}$$

may be non-zero. Moreover, the possible non-zero entries of $\triangledown^2_{\mathbf{XX}}\mathbf{F}(\mathbf{X})$ may be represented by the matrix

$$\mathbf{D}^{\mathbf{F}}_{\mathbf{XX}} = \left[\begin{array}{cccc} \mathbf{d}^{\mathbf{f}}_{\mathbf{X}_1\mathbf{X}_1} & \mathbf{d}^{\mathbf{f}}_{\mathbf{X}_2\mathbf{X}_2} & \cdots & \mathbf{d}^{\mathbf{f}}_{\mathbf{X}_N\mathbf{X}_N} \end{array}\right] \in \mathbb{R}^{q\times N}, \tag{29}$$

where $\mathbf{d}^{\mathbf{f}}_{\mathbf{X}_h\mathbf{X}_h}$ denotes the possible non-zero entries of $\triangledown_{\mathbf{X}_h\mathbf{X}_h}\mathbf{f}(\mathbf{X}_h)$. Similar to the first derivative case, the possible non-zero entries $\mathbf{D}^{\mathbf{F}}_{\mathbf{XX}}$ may then be mapped into $\triangledown^2_{\mathbf{XX}}\mathbf{F}(\mathbf{X})$ given the indices $\mathbf{i}^{\mathbf{f}}_{\mathbf{xx}}$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{xx}}$, and $\mathbf{k}^{\mathbf{f}}_{\mathbf{xx}}$. Thus, the six-dimensional second derivative matrix $\triangledown^2_{\mathbf{XX}}\mathbf{F}(\mathbf{X})$ may be fully defined given the matrix $\mathbf{D}^{\mathbf{F}}_{\mathbf{XX}}$, the row, column, and layer indices $\mathbf{i}^{\mathbf{f}}_{\mathbf{xx}}$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{xx}}$, and $\mathbf{k}^{\mathbf{f}}_{\mathbf{xx}}$, and the dimensions m and n.

5 Supplying Derivatives to GPOPS-II

Using the sparse derivative representations of Section 4 we may now concisely define the derivative information required to build the first and second derivatives of the NLP of Section 3. The required first derivative information is now given in Table 1 and the required second derivative information is given in Table 2. It is again stressed that the sparsity patterns of the first and second derivative matrices of the vectorized functions $\mathbf{G}(\mathbf{Z})$, $\mathbf{F}(\mathbf{Z})$, and $\mathbf{C}(\mathbf{Z})$ are fully defined by the sparsity patterns of the functions $\mathbf{g}(\mathbf{z}(\sigma))$, $\mathbf{f}(\mathbf{z}(\sigma)$, and $\mathbf{c}(\mathbf{z}(\sigma))$, respectively and this is conveyed in the referenced tables. It is also noted that the row locations of the scalar functions $\Phi(\mathbf{v})$ and $g(\mathbf{z}(\sigma))$ are omitted as they are simply a vector of ones.

Table 1: Required First Derivatives

Function	Row Locations	Column Locations	Possible Non-Zero Derivatives
$\Phi(\mathbf{v})$	-	$\mathbf{j}_{\mathbf{v}}^{\Phi}$	$\mathbf{d}_{\mathbf{v}}^{\Phi}$
$\boldsymbol{\phi}(\mathbf{v})$	$\mathbf{i}_{\mathbf{v}}^{\boldsymbol{\phi}}$	$\mathbf{j}_{\mathbf{v}}^{\boldsymbol{\phi}}$	$\mathbf{d}_{\mathbf{v}}^{\boldsymbol{\phi}}$
$\mathbf{G}(\mathbf{Z})$	-	$\mathbf{j}_{\mathbf{z}}^{g}$	$\mathbf{D}_{\mathbf{Z}}^{\mathbf{G}}$
$\mathbf{F}(\mathbf{Z})$	$\mathbf{i}_{\mathbf{z}}^{\mathbf{f}}$	$\mathbf{j}_{\mathbf{z}}^{\mathbf{f}}$	$\mathbf{D}_{\mathbf{Z}}^{\mathbf{F}}$
$\mathbf{C}(\mathbf{Z})$	$\mathbf{i}_{\mathbf{z}}^{\mathbf{c}}$	$\mathbf{j}_{\mathbf{z}}^{\mathbf{c}}$	$\mathbf{D}_{\mathbf{Z}}^{\mathbf{C}}$

Table 2: Required Second Derivatives

Function	Row Locations	Column Locations	Layer Locations	Possible Non-Zero Derivatives
$\Phi(\mathbf{v})$	-	$\mathbf{j}_{\mathbf{vv}}^{\Phi}$	$\mathbf{k}_{\mathbf{vv}}^{\Phi}$	$\mathbf{d}_{\mathbf{vv}}^{\Phi}$
$\boldsymbol{\phi}(\mathbf{v})$	$\mathbf{i}_{\mathbf{vv}}^{\boldsymbol{\phi}}$	$\mathbf{j}_{\mathbf{vv}}^{\boldsymbol{\phi}}$	$\mathbf{k}_{\mathbf{vv}}^{\boldsymbol{\phi}}$	$\mathbf{d}_{\mathbf{vv}}^{\boldsymbol{\phi}}$
$\mathbf{G}(\mathbf{Z})$	-	$\mathbf{j}_{\mathbf{zz}}^{g}$	$\mathbf{k}_{\mathbf{zz}}^{g}$	$\mathbf{D}_{\mathbf{ZZ}}^{\mathbf{G}}$
$\mathbf{F}(\mathbf{Z})$	$\mathbf{i}_{\mathbf{zz}}^{\mathbf{f}}$	$\mathbf{j}_{\mathbf{zz}}^{\mathbf{f}}$	$\mathbf{k}_{\mathbf{zz}}^{\mathbf{f}}$	$\mathbf{D}_{\mathbf{ZZ}}^{\mathbf{F}}$
$\mathbf{C}(\mathbf{Z})$	$\mathbf{i}_{\mathbf{zz}}^{\mathbf{c}}$	$\mathbf{j}_{\mathbf{zz}}^{\mathbf{c}}$	$\mathbf{k}_{\mathbf{zz}}^{\mathbf{c}}$	$\mathbf{D}_{\mathbf{ZZ}}^{\mathbf{C}}$

In Section 4 we referred to the index vectors (e.g. $\mathbf{i}_{\mathbf{z}}^{\mathbf{f}}$ and $\mathbf{j}_{\mathbf{z}}^{\mathbf{f}}$) as possible non-zero derivative locations, these may also be thought of as defining the function dependencies of a function with respect to its inputs, or the sparsity pattern of the derivative. In the default mode of *GPOPS-II*, the first derivative function dependencies of the optimal control problem are first found by evaluating the functions on MATLAB values of `NaN`, and determining which outputs are computed to likewise be `NaN`. The second derivative function dependencies are then estimated in the following manner. Without loss of generality consider a scalar function $g(\mathbf{x})$. Assuming the scalar function $g(\mathbf{x})$ is dependent upon both x_i and x_j, then it is assumed that the second derivative $\frac{\partial^2 g}{x_i x_j}$ is possibly non-zero. While this provides an over-estimate of the second-derivative sparsity patterns, it ensures that all possible non-zero derivatives will be calculated. Prior to solving the NLP, the first and second derivative sparsity patterns of the NLP are built and given to *IPOPT* using the sparsity patterns of the optimal control functions. Moreover, during the execution of the NLP the row and column locations of the first derivatives and the row, column and layer locations of the second derivatives are used together with a sparse finite-differencing algorithm in order to compute the possible non-zero derivatives of Tables 1 and 2. These possible non-zero derivatives are then used to build the required derivatives of the NLP.

5.1 Supplying Derivatives Using ADiGator

In this section we now give an introduction on the methods used by *ADiGator* to compute the required derivative information of Tables 1 and 2. The *ADiGator* software uses a source transformation via operator overloading approach of algorithmic differentiation [7]. The inputs to the software are a function to be differentiated together with information on the sizes of the inputs, and the derivative information of the inputs. The algorithm first transforms the given user source code into an intermediate source code, where the statements of the original code are augmented by calls to *ADiGator* specific transformation routines. The intermediate source is then *evaluated* multiple times on overloaded *CADA* objects [14]. Unlike conventional operator overloaded objects used in AD, the overloaded *CADA* objects are made to contain only information on the size of the objects, symbolic identifiers, and possible derivative non-zero locations. These overloaded evaluations result in the appropriate function and possible non-zero derivative calculations being printed to a derivative file, while simultaneously determining the associated sparsity pattern of the dependent output variables with respect to the independent input variables. The result of the entire method is then that the original user function is transformed into an executable MATLAB derivative function, which, when evaluated numerically, computes the non-zero derivatives of the original function. The generated derivative functions rely solely upon the native MATLAB library and may then be evaluated repeatedly at different numeric input values. Moreover, the process may be recursively executed in order to generate second-order derivative code and compute second-order derivative sparsity patterns. The process of computing the derivatives of Tables 1 and 2 using the *ADiGator* software is now explained.

Without loss of generality, it is first assumed that there exists a MATLAB function `EndpFun` which takes $\mathbf{v} \in \mathbb{R}^{2n+2}$ as an input and computes $\Phi(\mathbf{v}) \in \mathbb{R}$ and $\boldsymbol{\phi}(\mathbf{v}) \in \mathbb{R}^q$. The *ADiGator* tool is then given the function `EndpFun` together with the dimension of $\mathbf{v}$ and told to compute the derivatives of the function `EndpFun` with respect to $\mathbf{v}$. The result of the process is then the computation of the possible non-zero derivative locations, $\mathbf{j}_{\mathbf{v}}^{\Phi}$, $\mathbf{i}_{\mathbf{v}}^{\phi}$, and $\mathbf{j}_{\mathbf{v}}^{\phi}$ together with the generation of a new executable file `EndpGrd`. The function `EndpGrd` then takes as an input the vector $\mathbf{v} \in \mathbb{R}^{2n+2}$ and computes the possible non-zero derivatives $\mathbf{d}_{\mathbf{v}}^{\Phi}$ and $\mathbf{d}_{\mathbf{v}}^{\phi}$ (as well as the values of the original outputs). In order to generate a second derivative file, the *ADiGator* tool is given the function `EndpGrd` together with the dimension of $\mathbf{v}$ and told to compute the derivatives of the function `EndpGrd` with respect the input $\mathbf{v}$. The result of the process is then the computation of the possible non-zero derivative locations, $\mathbf{j}_{\mathbf{vv}}^{\Phi}$, $\mathbf{k}_{\mathbf{vv}}^{\Phi}$, $\mathbf{i}_{\mathbf{vv}}^{\phi}$, $\mathbf{j}_{\mathbf{vv}}^{\phi}$, and $\mathbf{k}_{\mathbf{vv}}^{\phi}$, together with the creation of a new executable file `EndpHes`. The function `EndpHes` then takes as an input the vector $\mathbf{v}$ and computes the possible non-zero second derivatives $\mathbf{d}_{\mathbf{vv}}^{\Phi}$ and $\mathbf{d}_{\mathbf{vv}}^{\phi}$ (together with the values of the original outputs).

Without loss of generality it is then assumed that there exists a MATLAB function `VectFun` which takes $\mathbf{Z} \in \mathbb{R}^{(n+m)\times N}$ $(N \geq 1)$ as an input and computes $\mathbf{G}(\mathbf{Z}) \in \mathbb{R}^{1\times N}$, $\mathbf{F}(\mathbf{Z}) \in \mathbb{R}^{n\times N}$, and $\mathbf{C}(\mathbf{Z}) \in \mathbb{R}^{p\times N}$. In order to efficiently generate associated derivative files, *ADiGator* is used in the *vectorized* mode. When used in the vectorized mode, the algorithm takes advantage of the sparse nature of vectorized function derivatives by storing information in a manner similar to those described in Section 4.2. As such, the *ADiGator* tool is given the function `VectFun`, and the fixed first dimension of $\mathbf{Z}$ and told to differentiate the function `VectFun` with respect to the input $\mathbf{Z}$, where the second dimension of $\mathbf{Z}$ is noted to be vectorized. The result of the transformation is then the computation of the possible non-zero derivative locations $\mathbf{j}^{g}_{\mathbf{z}}$, $\mathbf{i}^{\mathbf{f}}_{\mathbf{z}}$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{z}}$, $\mathbf{i}^{\mathbf{c}}_{\mathbf{z}}$, and $\mathbf{j}^{\mathbf{c}}_{\mathbf{z}}$ together with the creation of a new executable file `VectGrd`. The function `VectGrd` then takes as an input the matrix $\mathbf{Z} \in \mathbb{R}^{(n+m)\times N}$ ($N \geq 1$, $n+m$ fixed) and computes the possible non-zero derivatives $\mathbf{D}^{\mathbf{G}}_{\mathbf{Z}}$, $\mathbf{D}^{\mathbf{F}}_{\mathbf{Z}}$, and $\mathbf{D}^{\mathbf{C}}_{\mathbf{Z}}$. As with the endpoint functions, the process is then repeated in order to generate a second-derivative file. That is, the *ADiGator* tool is then given the function `VectGrd`, and the fixed first dimension of $\mathbf{Z}$ and told to differentiate the function `VectGrd` with respect to the input $\mathbf{Z}$, where the second dimension of $\mathbf{Z}$ is noted to be vectorized. The result of this second transformation is then the computation of the possible non-zero second derivative locations $\mathbf{j}^{g}_{\mathbf{zz}}$, $\mathbf{k}^{g}_{\mathbf{zz}}$, $\mathbf{i}^{\mathbf{f}}_{\mathbf{zz}}$, $\mathbf{j}^{\mathbf{f}}_{\mathbf{zz}}$, $\mathbf{k}^{\mathbf{f}}_{\mathbf{zz}}$, $\mathbf{i}^{\mathbf{c}}_{\mathbf{zz}}$, $\mathbf{j}^{\mathbf{c}}_{\mathbf{zz}}$, and $\mathbf{k}^{\mathbf{c}}_{\mathbf{zz}}$ together with the creation of a new executable file `VectHes`. The function `VectHes` then takes as an input the matrix $\mathbf{Z} \in \mathbb{R}^{(n+m)\times N}$ and computes the possible non-zero second derivatives $\mathbf{D}^{\mathbf{G}}_{\mathbf{ZZ}}$, $\mathbf{D}^{\mathbf{F}}_{\mathbf{ZZ}}$, and $\mathbf{D}^{\mathbf{C}}_{\mathbf{ZZ}}$.

The generation of the first derivative functions `EndpGrd` and `VectGrd`, and the second derivative functions `EndpHes` and `VectHes`, together with the computation of the possible non-zero derivative locations of Tables 1 and 2 is done entirely prior to the call to the *GPOPS-II* software. Once the *ADiGator* software has generated the derivative files, the software is then no longer needed as the derivative files may be evaluated using only the native MATLAB library. Thus, the *GPOPS-II* algorithm is given the possible non-zero derivative locations of Tables 1 and 2 (as computed by *ADiGator*) together with the names of the derivative functions, `EndpGrd`, `EndpHes`, `VectGrd`, and `VectHes`. *GPOPS-II* then uses the optimal control problem sparsity patterns in order to supply *IPOPT* with the first and second derivative sparsity patterns of the NLP. Moreover, at each iteration of the NLP when it is required to compute a first and/or second derivative, rather than using a finite-difference, *GPOPS-II* is able to call the given *ADiGator* generated derivative files and then use the outputs in order to build the proper NLP derivatives.

6 Examples

In this section we apply the method of Section 5 to three test cases. In all three examples we compare the efficiencies of using *ADiGator* versus the sparse central differencing algorithm of *GPOPS-II* in order to compute the required optimal control derivatives. The first test problem is the classical minimum-time-to-climb of a supersonic aircraft. The second test problem is that of a low-thrust orbital transfer, and the third test problem is a space station attitude control problem. All problems were solved using *GPOPS-II* in the second derivative mode, with *IPOPT* as the NLP solver and *MA57* as the linear solver. For the sake of conciseness, no MATLAB code or *GPOPS-II* solutions are provided, but rather only the efficiencies of the methods are explored. Table 3 shows the total time taken by the *ADiGator* algorithm in order to generate the required derivative files, where the total time is computed as the time taken to generate the first and second derivative files associated with the endpoint functions together with the first and second derivative files associated with the vectorized functions. All computations were performed on an Apple Mac Pro with Mac OS-X 10.9.2 (Mavericks) and a 2 × 2.4 GHz Quad-Core Intel Xeon processor with 24 GB 1066 MHz DDR3 RAM using MATLAB version R2014a.

Table 3: *ADiGator* Derivative File Generation Times

Example	1	2	3
Time (s)	3.281	6.140	4.492

Example 1: Minimum Time to Climb of Supersonic Aircraft

The problem considered in this section is the classical minimum time-to-climb of a supersonic aircraft. The objective is to determine the minimum-time trajectory and control from take-off to a specified altitude and speed. This problem was originally stated in the open literature in the work of Ref. [15], but the model used in this study was taken from Ref. [16] with the exception that a linear extrapolation of the thrust data as found in Ref. [16] was performed in order to fill in the "missing" data points.

The minimum time-to-climb problem for a supersonic aircraft is posed as follows. Minimize the cost functional

$$J = t_f \tag{30}$$

subject to the dynamic constraints

$$\begin{aligned}
\dot{h} &= v \sin\alpha && (31)\\
\dot{v} &= \frac{T\cos\alpha - D}{m} && (32)\\
\dot{\gamma} &= \frac{T\sin\alpha + L}{mv} + \left(\frac{v}{r} - \frac{\mu}{vr^2}\right)\cos\gamma && (33)\\
\dot{m} &= -\frac{T}{g_0 I_{sp}} && (34)
\end{aligned}$$

and the boundary conditions

$$\begin{aligned}
h(0) &= 0 \text{ ft} && (35)\\
v(0) &= 129.3144 \text{ m/s} && (36)\\
\gamma(0) &= 0 \text{ rad} && (37)\\
h(t_f) &= 19994.88 \text{ m} && (38)\\
v(t_f) &= 295.092 \text{ ft/s} && (39)\\
\gamma(t_f) &= 0 \text{ rad} && (40)
\end{aligned}$$

where h is the altitude, v is the speed, γ is the flight path angle, m is the vehicle mass, T is the magnitude of the thrust force, and D is the magnitude of the drag force. It is noted that this example uses table data obtained from Ref. [15].

This example was used to test the results of solving the problem using *GPOPS-II* by supplying derivatives with *ADiGator* versus the using the *GPOPS-II*'s sparse central differencing algorithm. The MATLAB code and solution produced by *GPOPS-II* using the sparse central differencing for this example may be seen in [17]. The solution produced by supplying derivatives via *ADiGator* is then within the NLP tolerences of that produced by using the finite differencing algorithm. The number of required *IPOPT* iterations and the *IPOPT* run times, however, are quite different. Table 4 shows the number of *IPOPT* iterations and *IPOPT* run times for each mesh refinement iteration using both AD and finite-differencing. From this table it is seen that by providing a more accurate derivative, the number of required *IPOPT* iterations is almost halved. Moreover, the total *IPOPT* run time using AD is less than one tenth of the run time using finite differences. This efficiency gain is due partly to the fewer number of required iterations, but also due to the way in which *ADiGator* is able to optimize the derivative computations by moving as much information as possible to the file generation phase.

Table 4: IPOPT Iteration and Run Time Results for Example 1 Using Algorithmic Differentiation and Sparse Central Finite Differencing.

(a) Algorithmic Differentiation

Mesh Number	IPOPT Iterations	IPOPT Run Time (s)
1	47	1.602
2	9	0.297
3	11	0.383
4	12	0.417
5	14	0.483
6	13	0.487
7	13	0.484
8	14	0.531
9	13	0.488
Totals:	146	5.173

(b) Finite Differencing

Mesh Number	IPOPT Iterations	IPOPT Run Time (s)
1	58	11.954
2	20	4.599
3	22	5.057
4	37	8.404
5	28	6.301
6	23	5.059
7	18	4.021
8	25	6.100
9	53	12.052
Totals:	284	63.548

Example 2: Low Thrust Orbit Transfer

In this example we consider the following low-thrust orbital transfer optimal control problem taken from Ref. [16]. The state of the system is given in modified equinoctial elements while the control is given in radial-transverse-normal coordinates. The goal is to determine the state

$$\mathbf{x} = (p, f, g, h, k, L, w), \tag{41}$$

the control

$$\mathbf{u} = (u_r, u_\theta, u_h), \tag{42}$$

and the throttle parameter, τ, that transfer the spacecraft from an initial orbit to a final orbit while maximizing the final weight of the spacecraft. The spacecraft starts in a circular low-Earth orbit with inclination $i(t_0) = 28.5\,\text{deg}$ and terminates in a highly elliptic low periapsis orbit with inclination $i(t_f) = 63.4\,\text{deg}$. The continuous-time optimal control problem corresponding to this orbital transfer problem can be stated in Mayer form as follows. Minimize the cost functional

$$J = -w(t_f) \tag{43}$$

subject to the dynamic constraints

$$\dot{\mathbf{x}} = \mathbf{A}(\mathbf{x})\boldsymbol{\Delta} + \mathbf{b}, \tag{44}$$

$$\dot{w} = -\frac{T(1+0.01\tau)}{I_{sp}}, \tag{45}$$

the path constraint

$$||\mathbf{u}|| = 1, \tag{46}$$

the parameter constraint

$$-50 \leq \tau \leq 0, \tag{47}$$

and the boundary conditions

$$\begin{array}{ll} p(t_0) = 21837080.052835 \text{ ft}, & p(t_f) = 40007346.015232 \text{ ft}, \\ f(t_0) = 0, & \sqrt{f^2(t_f) + g^2(t_f)} = 0.73550320568829, \\ g(t_0) = 0, & \sqrt{h^2(t_f) + k^2(t_f)} = 0.61761258786099, \\ h(t_0) = -0.25396764647494, & f(t_f)h(t_f) + g(t_f)k(t_f) = 0, \\ k(t_0) = 0, & g(t_f)h(t_f) - k(t_f)f(t_f) \leq 0, \\ L(t_0) = \pi \text{ rad}, & w(t_0) = 1 \text{ lbm}, \\ i(t_0) = 28.5 \text{ deg}, & i(t_f) = 63.4 \text{ deg}. \end{array} \tag{48}$$

The matrix $\mathbf{A}(\mathbf{x})$ in Eq. (44) is given as

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{2p}{q}\sqrt{\frac{p}{\mu}} & 0 \\ \sqrt{\frac{p}{\mu}}\sin(L) & \sqrt{\frac{p}{\mu}}\frac{1}{q}\big((q+1)\cos(L) + f\big) & -\sqrt{\frac{p}{\mu}}\frac{g}{q}\big(h\sin(L) - k\cos(L)\big) \\ -\sqrt{\frac{p}{\mu}}\cos(L) & \sqrt{\frac{p}{\mu}}\frac{1}{q}\big((q+1)\sin(L) + g\big) & \sqrt{\frac{p}{\mu}}\frac{f}{q}\big(h\sin(L) - k\cos(L)\big) \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{s^2\cos(L)}{2q} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{s^2\sin(L)}{2q} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\big(h\sin(L) - k\cos(L)\big) \end{bmatrix} \tag{49}$$

while the vector is

$$\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \sqrt{\mu p}(\frac{q}{p})^2 \end{bmatrix}, \tag{50}$$

where

$$\begin{array}{ll} q = 1 + f\cos(L) + g\sin(L), & r = p/q, \\ \alpha^2 = h^2 - k^2, & \chi = \sqrt{h^2 + k^2}, \\ s^2 = 1 + \chi^2. & \end{array} \tag{51}$$

The spacecraft acceleration is modeled as

$$\mathbf{\Delta} = \mathbf{\Delta}_g + \mathbf{\Delta}_T, \tag{52}$$

where $\mathbf{\Delta}_g$ is the acceleration due to the oblateness of the Earth while $\mathbf{\Delta}_T$ is the thrust specific force. The acceleration due to Earth oblateness is expressed in rotating radial coordinates as

$$\mathbf{\Delta}_g = \mathbf{Q}_r^{\mathsf{T}} \delta\mathbf{g}, \tag{53}$$

where $\mathbf{Q}_r$ is the transformation from rotating radial coordinates to Earth centered inertial coordinates. The matrix $\mathbf{Q}_r$ is given column-wise as

$$\mathbf{Q}_r = \begin{bmatrix}\mathbf{i}_r & \mathbf{i}_\theta & \mathbf{i}_h\end{bmatrix}, \tag{54}$$

where the basis vectors $\mathbf{i}_r$, $\mathbf{i}_\theta$, and $\mathbf{i}_h$ are given as

$$\mathbf{i}_r = \frac{\mathbf{r}}{||\mathbf{r}||} \quad , \quad \mathbf{i}_h = \frac{\mathbf{r}\times\mathbf{v}}{||\mathbf{r}\times\mathbf{v}||} \quad , \quad \mathbf{i}_\theta = \mathbf{i}_h \times \mathbf{i}_r. \tag{55}$$

Furthermore, the vector $\delta\mathbf{g}$ is defined as

$$\delta\mathbf{g} = \delta g_n \mathbf{i}_n - \delta g_r \mathbf{i}_r, \tag{56}$$

where $\mathbf{i}_n$ is the local North direction and is defined as

$$\mathbf{i}_n = \frac{\mathbf{e}_n - (\mathbf{e}_n^{\mathsf{T}}\mathbf{i}_r)\mathbf{i}_r}{||\mathbf{e}_n - (\mathbf{e}_n^{\mathsf{T}}\mathbf{i}_r)\mathbf{i}_r||} \tag{57}$$

and $\mathbf{e}_n = (0,0,1)$. The oblate earth perturbations are then expressed as

$$\delta g_r = -\frac{\mu}{r^2}\sum_{k=2}^{4}(k+1)\left(\frac{R_e}{r}\right)^k P_k(s)J_k, \tag{58}$$

$$\delta g_n = -\frac{\mu\cos(\phi)}{r^2}\sum_{k=2}^{4}\left(\frac{R_e}{r}\right)^k P_k^{'}(s)J_k, \tag{59}$$

where R_e is the equatorial radius of the earth, $P_k(s)$ $(s \in [-1,+1])$ is the k^{th}-degree Legendre polynomial, $P_k^{'}$ is the derivative of P_k with respect to s, $s = \sin(\phi)$, and J_k represents the zonal harmonic coefficients for $k = (2,3,4)$. Next, the acceleration due to thrust is given as

$$\mathbf{\Delta}_T = \frac{g_0 T(1+0.01\tau)}{w}\mathbf{u}. \tag{60}$$

Finally, the physical constants used in the problem are given as

$$\begin{array}{ll} I_{sp} = 450 \text{ s}, & T = 4.446618\times10^{-3} \text{ lbf}, \\ g_0 = 32.174 \text{ ft/s}^2, & \mu = 1.407645794\times10^{16} \text{ ft}^3/\text{s}^2, \\ R_e = 20925662.73 \text{ ft}, & J_2 = 1082.639\times10^{-6}, \\ J_3 = -2.565\times10^{-6}, & J_4 = -1.608\times10^{-6}. \end{array} \tag{61}$$

The manner in which the initial guess for this example is generated, together with the MATLAB code and solution produced by *GPOPS-II* using the sparse central differencing may be seen in [17]. The solution produced by supplying derivatives via *ADiGator* is within NLP tolerance of that produced via the sparse central differencing. The NLP iteration counts and NLP solve times using both AD and finite-differences may be seen in Table 5. In this table it is seen that the number of required NLP iterations are quite close, however the total NLP solve time using AD is roughly one third of the solve time using finite-differencing.

Table 5: IPOPT Iteration and Run Time Results for Example 2 Using Algorithmic Differentiation and Sparse Central Finite Differencing.

(a) Algorithmic Differentiation

Mesh Number	IPOPT Iterations	IPOPT Run Time (s)
1	27	1.769
2	52	3.898
3	50	6.950
4	28	5.209
5	11	2.429
6	10	2.432
7	11	2.651
8	8	2.068
9	8	2.077
Totals:	205	29.482

(b) Finite Differencing

Mesh Number	IPOPT Iterations	IPOPT Run Time (s)
1	28	6.229
2	59	16.352
3	52	21.460
4	28	14.745
5	11	6.394
6	10	6.124
7	11	6.827
8	8	5.070
9	8	5.118
Totals:	215	88.320

Example 3: Space Station Attitude Control

Consider the following space station attitude control optimal control problem taken from [18] and [16]. Minimize the cost functional

$$J = \tfrac{1}{2}\int_{t_0}^{t_f} \mathbf{u}^{\mathsf{T}}\mathbf{u}\,dt \tag{62}$$

subject to the dynamic constraints

$$\begin{aligned} \dot{\boldsymbol{\omega}} &= \mathbf{J}^{-1}\left\{\boldsymbol{\tau}_{gg}(\mathbf{r}) - \boldsymbol{\omega}^{\otimes}\left[\mathbf{J}\boldsymbol{\omega}+\mathbf{h}\right] - \mathbf{u}\right\}, \\ \dot{\mathbf{r}} &= \tfrac{1}{2}\left[\mathbf{r}\mathbf{r}^{\mathsf{T}} + \mathbf{I} + \mathbf{r}\right]\left[\boldsymbol{\omega} - \boldsymbol{\omega}(\mathbf{r})\right], \\ \dot{\mathbf{h}} &= \mathbf{u}, \end{aligned} \tag{63}$$

the inequality path constraint

$$\|\mathbf{h}\| \leq h_{\max}, \tag{64}$$

and the boundary conditions

$$\begin{aligned} t_0 &= 0, \\ t_f &= 1800, \\ \boldsymbol{\omega}(0) &= \bar{\boldsymbol{\omega}}_0, \\ \mathbf{r}(0) &= \bar{\mathbf{r}}_0, \\ \mathbf{h}(0) &= \bar{\mathbf{h}}_0, \\ \mathbf{0} &= \mathbf{J}^{-1}\left\{\boldsymbol{\tau}_{gg}(\mathbf{r}(t_f)) - \boldsymbol{\omega}^{\otimes}(t_f)\left[\mathbf{J}\boldsymbol{\omega}(t_f)+\mathbf{h}(t_f)\right]\right\}, \\ \mathbf{0} &= \tfrac{1}{2}\left[\mathbf{r}(t_f)\mathbf{r}^{\mathsf{T}}(t_f) + \mathbf{I} + \mathbf{r}(t_f)\right]\left[\boldsymbol{\omega}(t_f) - \boldsymbol{\omega}_0(\mathbf{r}(t_f))\right], \end{aligned} \tag{65}$$

where $(\boldsymbol{\omega}, \mathbf{r}, \mathbf{h})$ is the state and $\mathbf{u}$ is the control. In this formulation $\boldsymbol{\omega}$ is the angular velocity, $\mathbf{r}$ is the Euler-Rodrigues parameter vector, $\mathbf{h}$ is the angular momentum, and $\mathbf{u}$ is the input moment (and is the control).

$$
\begin{aligned}
\boldsymbol{\omega}_0(\mathbf{r}) &= -\omega_{\text{orb}}\mathbf{C}_2, \\
\boldsymbol{\tau}_{gg} &= 3\omega_{\text{orb}}^2\mathbf{C}_3^{\otimes}\mathbf{J}\mathbf{C}_3,
\end{aligned}
\tag{66}
$$

and $\mathbf{C}_2$ and $\mathbf{C}_3$ are the second and third column, respectively, of the matrix

$$
\mathbf{C} = \mathbf{I} + \frac{2}{1+\mathbf{r}^{\mathsf{T}}\mathbf{r}}\left(\mathbf{r}^{\otimes}\mathbf{r}^{\otimes} - \mathbf{r}^{\otimes}\right). \tag{67}
$$

In this example the matrix $\mathbf{J}$ is given as

$$
\mathbf{J} = \begin{bmatrix} 2.80701911616\times 10^{7} & 4.822509936\times 10^{5} & -1.71675094448\times 10^{7} \\ 4.822509936\times 10^{5} & 9.5144639344\times 10^{7} & 6.02604448\times 10^{4} \\ -1.71675094448\times 10^{7} & 6.02604448\times 10^{4} & 7.6594401336\times 10^{7} \end{bmatrix}, \tag{68}
$$

while the initial conditions $\bar{\boldsymbol{\omega}}_0$, $\bar{\mathbf{r}}_0$, and $\bar{\mathbf{h}}_0$ are

$$
\begin{aligned}
\bar{\boldsymbol{\omega}}_0 &= \begin{bmatrix} -9.5380685844896\times 10^{-6} \\ -1.1363312657036\times 10^{-3} \\ +5.3472801108427\times 10^{-6} \end{bmatrix}, \\
\bar{\mathbf{r}}_0 &= \begin{bmatrix} 2.9963689649816\times 10^{-3} \\ 1.5334477761054\times 10^{-1} \\ 3.8359805613992\times 10^{-3} \end{bmatrix}, \\
\bar{\mathbf{h}}_0 &= \begin{bmatrix} 5000 \\ 5000 \\ 5000 \end{bmatrix}.
\end{aligned}
\tag{69}
$$

A more detailed description of this problem, including all of the constants $\mathbf{J}$, $\bar{\boldsymbol{\omega}}_0$, $\bar{\mathbf{r}}_0$, and $\bar{\mathbf{h}}_0$, can be found in [18] or [16].

The manner in which this problem is coded in MATLAB together with the solution produced by *GPOPS-II* using the sparse central differencing may be seen in [5]. Again, the solution achieved by supplying *GPOPS-II* with *ADiGator* derivatives is within NLP tolerances of the solution produced by using the sparse central finite differencing scheme. In Table 6, the number of required NLP iterations together with the NLP solve times are given for both the AD and finite-differencing cases. In this table it is seen that, for this example, the NLP requires slightly more iterations to solve using the more accurate derivatives, however AD proves to be almost an order of magnitude more efficient than finite-differencing.

7 Conclusions

In this paper we have discussed the methods required in order to use the *ADiGator* algorithmic differentiation package to compute the first and second derivatives of the NLP which results from the hp-adaptive LGR collocation of optimal control problems. In Section 6, the methods were tested on three examples using the *GPOPS-II* optimal control software and compared against *GPOPS-II*'s sparse central differencing algorithm. In the first example we see that the NLP solver requires a significantly smaller amount of iterations in order to solve when using AD over finite-differencing.

Table 6: IPOPT Iteration and Run Time Results for Example 3 Using Algorithmic Differentiation and Sparse Central Finite Differencing.

(a) Algorithmic Differentiation

Mesh Number	IPOPT Iterations	IPOPT Run Time (s)
1	32	1.277
2	16	0.581
3	19	0.744
4	19	0.647
5	18	0.672
6	13	0.539
7	21	0.868
8	10	0.444
Totals:	148	5.772

(b) Finite Differencing

Mesh Number	IPOPT Iterations	IPOPT Run Time (s)
1	28	10.326
2	18	6.859
3	18	6.867
4	15	5.470
5	18	6.998
6	13	5.035
7	21	8.264
8	9	3.545
Totals:	140	53.365

This improvement is likely due to the fact that *ADiGator* is able to produce an exact second derivative sparsity pattern, together with the fact that derivatives supplied by AD are exact up to machine precision. This NLP behavior, however, was not witnessed in the second and third test cases. What is constant across all three test cases, however, is that by using *ADiGator*, the solutions were able to be obtained in a much smaller amount of time than when using the finite-differencing scheme. Even when factoring in the required derivative file generation time, we witness that the solutions of the three examples may be obtained in 13.3%, 40.3%, and 19%, respectively, of the time required when using the sparse central finite differencing scheme. We conclude that the method has been proven to be extremely efficient when compared to classical finite-differencing methods.

References

[1] Betts, J. T. and Huffman, W. P., “Sparse optimal control software SOCS,” *Mathematics and Engineering Analysis Technical Document MEA-LR-085, Boeing Information and Support Services, The Boeing Company, PO Box*, Vol. 3707, 1997, pp. 98124–2207.

[2] Ross, I. and Fahroo, F., “UserâĂŹs Manual for DIDO 2001 α: A MATLAB Application for Solving Optimal Control Problems,” Tech. rep., Tech. rep. AAS-01-03. Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA, 2001.

[3] von Stryk, O., “User’s Guide for DIRCOL, a Direct Collocation Method for the Numerical Solution of Optimal Control Problems. Version 2.1,” *Simulation, Systems Optimization and Robotics Group, Technical University of Darmstadt. http://www. sim. informatik.//tu-darmstadt. de/index/leftnav. html. en*, 1999.

[4] Rao, A. V., Benson, D. A., Darby, C., Patterson, M. A., Francolin, C., Sanders, I., and Huntington, G. T., “Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method,” *ACM Transactions on Mathematical Software (TOMS)*, Vol. 37, No. 2, 2010, pp. 22.

[5] Patterson, M. A. and Rao, A. V., “GPOPS- II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp–Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming,” *ACM Transactions on Mathematical Software*, Vol. 39, No. 3, 2013.

[6] Griewank, A., *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Frontiers in Appl. Mathematics*, SIAM Press, Philadelphia, Pennsylvania, 2008.

[7] Weinstein, M. J. and Rao, A. V., “A Source Transformation via Operator Overloading Method for Automatic Differentiation in MATLAB,” *ACM Transactions on Mathematical Software*, In Revision 2014.

[8] Garg, D., Patterson, M. A., Darby, C. L., Françolin, C., Huntington, G. T., Hager, W. W., and Rao, A. V., “Direct Trajectory Optimization and Costate Estimation of Finite-Horizon and Infinite-Horizon Optimal Control Problems via a Radau Pseudospectral Method,” *Computational Optimization and Applications*, Vol. 49, No. 2, June 2011, pp. 335–358.

[9] Garg, D., Patterson, M. A., Hager, W. W., Rao, A. V., Benson, D. A., and Huntington, G. T., “A Unified Framework for the Numerical Solution of Optimal Control Problems Using Pseudospectral Methods,” *Automatica*, Vol. 46, No. 11, November 2010, pp. 1843–1851.

[10] Garg, D., Hager, W. W., and Rao, A. V., “Pseudospectral Methods for Solving Infinite-Horizon Optimal Control Problems,” *Automatica*, Vol. 47, No. 4, April 2011, pp. 829–837.

[11] Darby, C. L., Hager, W. W., and Rao, A. V., “An *hp*–Adaptive Pseudospectral Method for Solving Optimal Control Problems,” *Optimal Control Applications and Methods*, Vol. 32, No. 4, July–August 2011, pp. 476–502.

[12] Darby, C. L., Hager, W. W., and Rao, A. V., "Direct Trajectory Optimization Using a Variable Low-Order Adaptive Pseudospectral Method," *Journal of Spacecraft and Rockets*, Vol. 48, No. 3, May–June 2011, pp. 433–445.

[13] Patterson, M. A. and Rao, A. V., "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Continuous-Time Optimal Control Problems," *Journal of Spacecraft and Rockets,*, Vol. 49, No. 2, March–April 2012, pp. 364–377.

[14] Patterson, M. A., Weinstein, M. J., and Rao, A. V., "An Efficient Overloaded Method for Computing Derivatives of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software,*, Vol. 39, No. 3, July 2013, pp. 17:1–17:36.

[15] Bryson, A. E. and Ho, Y.-C., *Applied Optimal Control*, Hemisphere Publishing, New York, 1975.

[16] Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, SIAM Press, Philadelphia, 2nd ed., 2009.

[17] Patterson, M. A. and Rao, A. V., "User's Guide for GPOPS-II, A General-Purpose MATLAB Software for Solving Multiple-Phase Optimal Control Problems. Version 2.0," 2014.

[18] Pietz, J. A., *Pseudospectral Collocation Methods for the Direct Transcription of Optimal Control Problems*, Master's thesis, Rice University, Houston, Texas, April 2003.