Algorithm: $\mathcal{GPOPS}$, A MATLAB® Software for Solving Non-Sequential Multiple-Phase Optimal Control Problems Using the Gauss Pseudospectral Method

Anil V. Rao, Christopher L. Darby, Michael Patterson, Camila Francolin
University of Florida, Gainesville, FL 32611

David A. Benson
The Charles Stark Draper Laboratory, Inc., Cambridge, MA 02139

Geoffrey T. Huntington
Blue Origin, LLC, Kent, WA 98032

An algorithm is described to solve non-sequential multiple-phase optimal control problems using a recently developed numerical method called the *Gauss pseudospectral method*. The algorithm is well-suited for use in modern vectorized programming languages such as FORTRAN 95 and MATLAB®. The algorithm is designed in a modular way where the cost functional, differential-algebraic equations, and boundary (event) conditions are discretized in each phase of the optimal control problem. The phases are then connected using linkage conditions on the state and time. A large-scale nonlinear programming problem (NLP) arises from the discretization. The significant features of the NLP are then described in detail. A particular reusable MATLAB®, implementation of the algorithm, called $\mathcal{GPOPS}$, is applied to two classical optimal control problems to demonstrate its utility. The algorithm described in this paper will have lasting value as a tool for solving complex optimal control problems and will provide researchers and engineers with a useful reference when it is desired to implement the algorithm in other programming languages.

Categories and Subject Descriptors: G.1.4 [**Numerical Methods**]: Optimal Control, Nonlinear Optmization, Nonlinear Programming—*computational methods*

General Terms: Optimal Control, Optimization, Algorithms

Additional Key Words and Phrases: dynamic optimization, optimal control, nonlinear programming, phases

Name: Anil V. Rao: anilvrao@ufl.edu, Christopher L. Darby: cdarby@ufl.edu, Michael A. Patterson: mpatterson@ufl.edu, Camila Francolin: francoli@ufl.edu, David A. Benson: dbenson@draper.com, Geoffrey T. Huntington: geoff.huntington@gmail.com

1. INTRODUCTION

Due to the increasing complexity of engineering applications, over the past two decades the subject of optimal control has transitioned from theory to computation. In particular, computational optimal control has become a science in and of itself, resulting in a variety of numerical methods and corresponding software implementations of these methods. The vast majority of software implementations of optimal control today are those that involve the direct transcription of a continuous-time optimal control problem to a nonlinear program (NLP). The NLP is then solved using one of a variety of well-known software packages [Gill et al. 2002; Byrd et al. 2006; Betts and Frank 1994].

Over the last decade, a particular class of direct collocation methods, called *pseudospectral methods*, have risen to prominence in the numerical solution of optimal control problems [Elnagar et al. 1995; Elnagar and Kazemi 1998; Fahroo and Ross 2000; Fahroo and Ross 2001; Ross and Fahroo 2003; Rao 2003; Williams 2004b; Williams 2004a; Williams 2005; Ross and Fahroo 2004a; Ross and Fahroo 2004b; Ross and Fahroo 2008a; Ross and Fahroo 2008b; Benson 2004; Benson et al. 2006; Huntington 2007; Kameswaran and Biegler 2006]. In a pseudospectral method, the state and control are approximated using global polynomials and collocation of the differential-algebraic equations is performed at nodes typically obtained via some form of Gaussian quadrature. The three most commonly used set of collocation points are *Legendre-Gauss* (LG), *Legendre-Gauss-Radau* (LGR), and *Legendre-Gauss-Lobatto* (LGL) points. These three sets of points are obtained from the roots of a Legendre polynomial and/or linear combinations of a Legendre polynomial and its derivatives. All three sets of points are defined on the domain $[-1, 1]$, but differ significantly in that the LG points include *neither* of the endpoints, the LGR points include *one* of the endpoints, and the LGL points include *both* of the endpoints. In addition, the LGR points are asymmetric relative to the origin and are not unique in that they can be defined using either the initial point or the terminal point. As a result of these three collocation points, the following three mathematical methods have been developed: the *Legendre Pseudospectral method* (LPM) [Elnagar et al. 1995; Fahroo and Ross 2001], the *Radau pseudospectral method* (RPM) [Kameswaran and Biegler 2006], and the *Gauss pseudospectral method* (GPM) [Benson 2004; Benson et al. 2006; Huntington 2007]. In addition, methods have been developed to handle interior point constraints [Fahroo and Ross 2000; Ross and Fahroo 2004a], and non-sequential phases [Rao 2003].

While many pseudospectral methods and subsequent extensions have been described mathematically, these methods have to date not been described in the open literature in the form of *algorithms* that enable implementation in mathematical software. As a result, it is difficult for a researcher or an engineer to study these mathematical methods and arrive at a tractable software implementation after any reasonable investment of time. Instead, most users typically resort to commercial off-the-shelf (COTS) software programs to solve optimal control problems. Examples of COTS software include *SOCS* [Betts and Huffman 1997], *DIRCOL* [von Stryk 2000], *GESOP* [Jansch et al. 1994], *OTIS* [Vlases et al. 1990], *DIDO* [Ross and Fahroo 2001], *DIRECT* [Williams 2008], and *MISER* [Goh and Teo 1988]. The typical alternative to COTS software is "home-grown" software.

While users benefit greatly from COTS software, such programs require a great investment of time to learn. Moreover, even after overcoming the steep learning curve associated with *using* these programs, a user has gained little insight into the underlying algorithm used in the software. Thus, the underlying algorithm remains a "black box" and the user would still find it difficult to implement such an algorithm independently. Even worse, because using a "black-box" provides little insight about the underlying methodology, COTS programs are limited in their educational value.

The purpose of this paper is to enhance the understanding of the implementation of pseudospectral methods in mathematical software for solving optimal control problems. To achieve this goal, this paper provides a detailed description of an algorithm and a supporting software implementation of the algorithm. In particular, the algorithm developed in this paper utilizes the aforementioned *Gauss pseudospectral method* (GPM) [Benson 2004; Benson et al. 2006; Huntington 2007] for solving multiple-phase optimal control problems. The GPM is chosen as the basis for the algorithm of this paper because it has many useful numerical properties including the ability to generate highly accurate costates (adjoints) [Benson 2004; Benson et al. 2006; Huntington 2007] of the continuous-time optimal control problem.[1] It is noted that the authors of this paper have already developed customized (i.e., non-reusable) implementations of the algorithm described in this paper for applications in flight dynamics [Huntington 2007; Huntington et al. 2007; Huntington and Rao 2008b; Huntington and Rao 2007], but until now no detailed information has been provided to the research community on how to implement the GPM in a general-purpose software program. As a result, this paper is an attempt to fill the gap between the theory and the implementation of pseudospectral methods for optimal control by providing a general-purpose vectorized implementation of the Gauss pseudospectral method [Benson 2004; Benson et al. 2006; Huntington 2007; Huntington et al. 2007; Huntington and Rao 2008b; Huntington and Rao 2007]. The algorithm described in this article can be used in modern vectorized programming languages such as FORTRAN 95/98/2000 or MATLAB®. A particular MATLAB® implementation, called $\mathcal{GPOPS}$, is provided and is found to work well on a variety of complex multiple-phase continuous-time optimal control problems. After describing the algorithm, two examples are provided to demonstrate the flexibility and utility of the algorithm. The first example is a classical maximum-range optimal control problem [Bryson and Ho 1975] while the second example is a fairly complex multiple-phase launch vehicle ascent problem [Benson 2004; Huntington 2007; Huntington and Rao 2007]. In order to see the power of the algorithm, the second example is compared against the well known and commercially available program *Sparse Optimal Control Software* $\mathbb{SOCS}$ [Betts and Huffman 1997] where it is found that the performance of $\mathcal{GPOPS}$ using SNOPT [Gill et al. 2006] compares well with that of $\mathbb{SOCS}$.

[1] It is noted that, with some modifications, the algorithm described in this paper can be adapted to other pseudospectral methods such as the aforementioned RPM [Kameswaran and Biegler 2006] or the LPM [Elnagar et al. 1995; Fahroo and Ross 2001]

2. GENERAL MULTIPLE-PHASE OPTIMAL CONTROL PROBLEMS

The problems that $\mathcal{GPOPS}$ is capable of solving fall into the following general category. Given a set of P phases (where $p \in [1, \ldots, P]$), minimize the cost functional

$$J = \sum_{p=1}^{P} \left[\Phi^{(p)}(\mathbf{x}^{(p)}(t_0), t_0^{(p)}, \mathbf{x}^{(p)}(t_f), t_f^{(p)}; \mathbf{q}^{(p)}) + \int_{t_0^{(p)}}^{t_f^{(p)}} \mathcal{L}^{(p)}(\mathbf{x}^{(p)}(t), \mathbf{u}^{(p)}(t), t; \mathbf{q}^{(p)}) dt \right] \tag{1}$$

subject to the dynamic constraints

$$\dot{\mathbf{x}}^{(p)} = \mathbf{f}^{(p)}(\mathbf{x}^{(p)}, \mathbf{u}^{(p)}, t; \mathbf{q}^{(p)}), \qquad (p = 1, \ldots, P), \tag{2}$$

the inequality path constraints

$$\mathbf{C}_{\min}^{(p)} \leq \mathbf{C}^{(p)}(\mathbf{x}^{(p)}(t), \mathbf{u}^{(p)}(t), t; \mathbf{q}^{(p)}) \leq \mathbf{C}_{\max}^{(p)}, \qquad (p = 1, \ldots, P), \tag{3}$$

the boundary conditions

$$\boldsymbol{\phi}_{\min}^{(p)} \leq \boldsymbol{\phi}^{(p)}(\mathbf{x}^{(p)}(t_0), t_0^{(p)}, \mathbf{x}^{(p)}(t_f), t_f^{(p)}; \mathbf{q}^{(p)}) \leq \boldsymbol{\phi}_{\max}^{(p)}, \qquad (p = 1, \ldots, P), \tag{4}$$

and the linkage constraints [Betts 2001]

$$\mathbf{L}_{\min}^{(s)} \leq \mathbf{L}^{(s)}(\mathbf{x}^{(p_l^s)}(t_f), t_f^{(p_l^s)}; \mathbf{q}^{(p_l^s)}, \mathbf{x}^{(p_r^s)}(t_0), t_0^{(p_r^s)}; \mathbf{q}^{(p_r^s)}) \leq \mathbf{L}_{\max}^{(s)}, \left\{ \begin{array}{lcl} p_l, p_r & \in & [1, \ldots, P] \\ s & = & 1, \ldots, L \end{array} \right. \tag{5}$$

where $\mathbf{x}^{(p)}(t) \in \mathbb{R}^{n_x^{(p)}}$, $\mathbf{u}^{(p)}(t) \in \mathbb{R}^{n_u^{(p)}}$, $\mathbf{q}^{(p)} \in \mathbb{R}^{n_q^{(p)}}$, and $t \in \mathbb{R}$ are, respectively, the state, control, static parameters, and time in phase $p = [1, \ldots, P]$, L is the number of pairs of phases to be linked and $(p_l^{(s)}, p_r^{(s)}) \in [1, \ldots, P]$, $(s = 1, \ldots, L)$ are the "left" and "right" phase numbers, respectively. We note that the functions $\Phi^{(p)}$, $\mathcal{L}^{(p)}$, $\mathbf{f}^{(p)}$, $\mathbf{C}^{(p)}$, $\boldsymbol{\phi}^{(p)}$, and $\mathbf{L}^{(s)}$ are defined by the following mappings:

$$\begin{array}{lll} \Phi^{(p)} & : \mathbb{R}^{n_x^{(p)}} \times \mathbb{R} \times \mathbb{R}^{n_x^{(p)}} \times \mathbb{R} \times \mathbb{R}^{n_q^{(p)}} & \longrightarrow \mathbb{R} \\ \mathcal{L}^{(p)} & : \mathbb{R}^{n_x^{(p)}} \times \mathbb{R}^{n_u^{(p)}} \times \mathbb{R} \times \mathbb{R}^{n_q^{(p)}} & \longrightarrow \mathbb{R} \\ \mathbf{f}^{(p)} & : \mathbb{R}^{n_x^{(p)}} \times \mathbb{R}^{n_u^{(p)}} \times \mathbb{R} \times \mathbb{R}^{n_q^{(p)}} & \longrightarrow \mathbb{R}^{n_x^{(p)}} \\ \mathbf{C}^{(p)} & : \mathbb{R}^{n_x^{(p)}} \times \mathbb{R}^{n_u^{(p)}} \times \mathbb{R} \times \mathbb{R}^{n_q^{(p)}} & \longrightarrow \mathbb{R}^{n_c^{(p)}} \\ \boldsymbol{\phi}^{(p)} & : \mathbb{R}^{n_x^{(p)}} \times \mathbb{R} \times \mathbb{R}^{n_x^{(p)}} \times \mathbb{R} \times \mathbb{R}^{n_q^{(p)}} & \longrightarrow \mathbb{R}^{n_\phi^{(p)}} \\ \mathbf{L}^{(s)} & : \mathbb{R}^{n_x^{(p_l^s)}} \times \mathbb{R} \times \mathbb{R}^{n_q^{(p_l^s)}} \times \mathbb{R}^{n_x^{(p_r^s)}} \times \mathbb{R} \times \mathbb{R}^{n_q^{(p_r^s)}} & \longrightarrow \mathbb{R}^{n_L^{(s)}} \end{array} \tag{6}$$

where $n_x^{(p)}$, $n_u^{(p)}$, $n_q^{(p)}$, $n_c^{(p)}$, and $n_\phi^{(p)}$ are the dimensions of the state, control, static parameter vector, path constraint vector, and boundary condition vector in phase $p = [1, \ldots, P]$, and $n_L^{(s)}$ is the dimension of the vector formed by the s^{th} set of linkage constraints. While much of the time a user may want to solve a problem consisting of multiple phases, it is important to note that the phases *need not be sequential*. To the contrary, any two phases may be linked provided that the independent variable does not change direction (i.e., the independent variable moves in the same direction during each phase that is linked). It is noted that the approach to linking phases used in $\mathcal{GPOPS}$ is based on well-known formulations in the literature such as those given in Ref. Betts [2001] and Betts [1998]. A schematic of how phases can potentially be linked is given in Fig. 1.

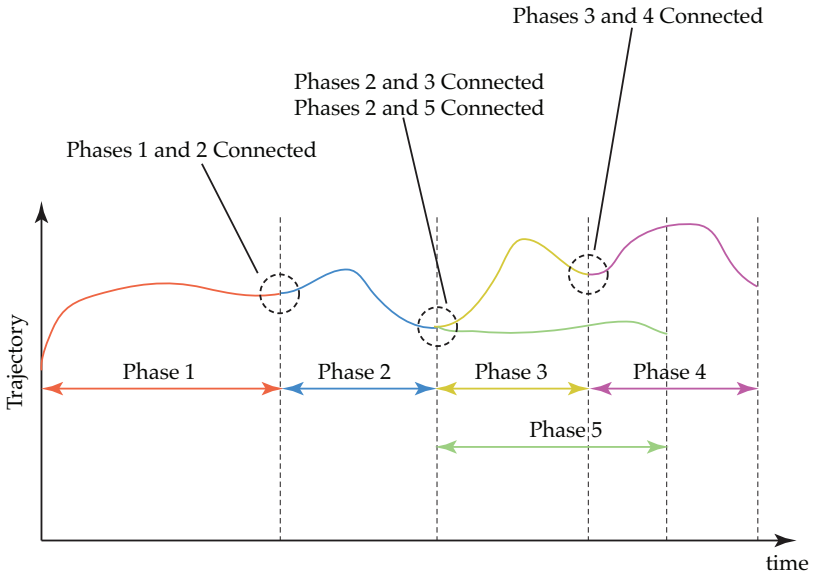

Fig. 1. Schematic of linkages for multiple-phase optimal control problem. The example shown in the picture consists of five phases where the ends of phases 1, 2, and 3 are linked to the starts of phases 2, 3, and 4, respectively, while the end of phase 2 is linked to the start of phase 5.

3. GAUSS PSEUDOSPECTRAL DISCRETIZATION

Before proceeding to a description of the $\mathcal{GPOPS}$ algorithm, in this section we provide a detailed description of the discretization of a multiple-phase optimal control problem via the GPM. For completeness, we restate some of the basic equations that have been previously developed for the GPM [Benson 2004; Benson et al. 2006; Huntington 2007].

Let $p \in [1, \ldots, P]$ be a particular phase of an optimal control problem and let $(\cdot)^{(p)}$ denote information in the p^{th} phase. Consider now the following transformation of the independent variable, $t^{(p)} \in [t_0^{(p)}, t_f^{(p)}]$, to a new variable $\tau^{(p)} \in [-1, 1]$ as follows:

$$t^{(p)} = \frac{t_f^{(p)} - t_0^{(p)}}{2}\tau^{(p)} + \frac{t_f^{(p)} + t_0^{(p)}}{2} \tag{7}$$

Furthermore, suppose we choose the collocation points in each phase to be the set of *Legendre-Gauss* (LG) points, $(\tau_1, \ldots, \tau_{N^{(p)}})$, which are the roots of the $[N^{(p)}]^{th}$ Legendre polynomial, $P_{N^{(p)}}(\tau)$, given as[2]

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

[2]It is noted that the Legendre-Gauss (LG) points are conveniently obtained by computing the eigenvalues of the $N^{(p)} \times N^{(p)}$ *Jacobi matrix* as is done in the pseudospectral differentiation matrix suite [Weideman and Reddy 2000].

Corresponding to the LG points are the LG *weights* which are computed as

$$w_i^{(p)} = \frac{2}{\left(1-\tau_i^{(p)}\right)\left[P'_{N^{(p)}}\right]^2}, \qquad (i=1,\ldots,N^{(p)}) \tag{9}$$

Finally, the *discretization points* used in the GPM are the LG points plus the points $\tau_0 = -1$ and $\tau_{N^{(p)}+1} = 1$ (thus resulting in a complete set of points $(\tau_0,\ldots,\tau_{N^{(p)}+1})$.

The discretization in each phase $p = [1,\ldots,P]$ of the problem can then be stated in terms of the independent variable $\tau^{(p)}$ as follows. First, the state is approximated using a basis of $N^{(p)}+1$ Lagrange interpolating polynomials, $L_i^{(p)}(\tau^{(p)})$, $(i = 0,\ldots,N^{(p)})$,

$$\mathbf{x}^{(p)}(\tau^{(p)}) \approx \mathbf{X}(\tau) = \sum_{i=0}^{N^{(p)}} \mathbf{X}^{(p)}(\tau_i^{(p)}) L_i(\tau^{(p)}) \tag{10}$$

where $L_i^{(p)}(\tau^{(p)})$ $(i=0,\ldots,N^{(p)})$ are defined as

$$L_i^{(p)}(\tau^{(p)}) = \prod_{j=0, j\neq i}^{N^{(p)}} \frac{\tau^{(p)} - \tau_j^{(p)}}{\tau_i^{(p)} - \tau_j^{(p)}} \tag{11}$$

It is known that the Lagrange polynomials $L_i^{(p)}(\tau^{(p)})$ $(i=0,\ldots,N^{(p)})$ satisfy the so called *isolation property*

$$L_i^{(p)}(\tau_j^{(p)}) = \begin{cases} 1 \ , & i = j \\ 0 \ , & i \neq j \end{cases} \tag{12}$$

The continuous cost functional of Eq. (1) is then approximated using the values of the state, control, and time at the LG points via a Gauss quadrature [Davis and Rabinowitz 1984] as

$$\begin{aligned} J = &\sum_{p=1}^{P} \Phi^{(p)}(\mathbf{X}_0^{(p)}, t_0^{(p)}, \mathbf{X}_f^{(p)}, t_f^{(p)}) \\ &+ \sum_{p=1}^{P} \frac{t_f^{(p)} - t_0^{(p)}}{2} \sum_{k=1}^{N^{(p)}} w_k^{(p)} \mathcal{L}^{(p)}(\mathbf{X}_k^{(p)}, \mathbf{U}_k^{(p)}, \tau_k^{(p)}; \mathbf{q}^{(p)}, t_0^{(p)}, t_f^{(p)}) \end{aligned} \tag{13}$$

Next, differentiating the expression in Eq. (10) with respect to $\tau^{(p)}$ gives

$$\left[\frac{d\mathbf{X}}{d\tau}\right]^{(p)} \approx \sum_{i=0}^{N^{(p)}} \mathbf{X}(\tau_i^{(p)}) \frac{dL_i^{(p)}(\tau^{(p)})}{d\tau^{(p)}} \tag{14}$$

The derivative of each Lagrange polynomial at the LG points can be represented compactly in the form of a *differentiation matrix*, $\mathbf{D}^{(p)} \in \mathbb{R}^{N^{(p)} \times N^{(p)}+1}$ as

$$\mathbf{D}_{ki}^{(p)} = \dot{L}_i^{(p)}(\tau_k^{(p)}) = \sum_{l=0}^{N^{(p)}} \frac{\prod_{j=0, j\neq i,l}^{N^{(p)}} \left(\tau_k^{(p)} - \tau_j^{(p)}\right)}{\prod_{j=0, j\neq i}^{N^{(p)}} \left(\tau_i^{(p)} - \tau_j^{(p)}\right)} \tag{15}$$

where $k = 1, \ldots, N^{(p)}$ and $i = 0, \ldots, N^{(p)}$. The dynamic constraint of Eq. (2) is then transcribed into algebraic constraints in terms of $\mathbf{D}^{(p)}$ as

$$\sum_{i=0}^{N} \mathbf{D}_{ki}^{(p)} \mathbf{X}_i^{(p)} - \frac{t_f^{(p)} - t_0^{(p)}}{2} \mathbf{f}^{(p)}(\mathbf{X}_k^{(p)}, \mathbf{U}_k^{(p)}, \tau_k^{(p)}; \mathbf{q}^{(p)}, t_0^{(p)}, t_f^{(p)}) = \mathbf{0} \quad (k = 1, \ldots, N^{(p)}) \tag{16}$$

where $\mathbf{X}_k^{(p)} \equiv \mathbf{X}^{(p)}(\tau_k^{(p)}) \in \mathbb{R}^{n^{(p)}}$ and $\mathbf{U}_k^{(p)} \equiv \mathbf{U}^{(p)}(\tau_k^{(p)}) \in \mathbb{R}^{m^{(p)}}$ $(k = 1, \ldots, N)$. Next, define an additional variable $\mathbf{X}_f^{(p)} \equiv \mathbf{X}_{N+1}^{(p)} \equiv \mathbf{X}^{(p)}(\tau_f^{(p)})$ as

$$\mathbf{X}_0^{(p)} = \mathbf{X}^{(p)}(\tau_0^{(p)}) \tag{17}$$

$$\mathbf{X}_f^{(p)} = \mathbf{X}_0^{(p)} + \frac{t_f^{(p)} - t_0^{(p)}}{2} \sum_{k=1}^{N} w_k^{(p)} \mathbf{f}^{(p)}(\mathbf{X}_k^{(p)}, \mathbf{U}_k^{(p)}, \tau_k^{(p)}; \mathbf{q}^{(p)}, t_0^{(p)}, t_f^{(p)}) \tag{18}$$

It is noted that, because we have introduced an additional variable, Eq. (18) is an additional constraint in the discretization (in order to maintain the same number of degrees of freedom). Next, the path constraints of Eq. (3) are discretized at the LG points as

$$\mathbf{C}_{\min}^{(p)} \leq \mathbf{C}^{(p)}(\mathbf{X}_k^{(p)}, \mathbf{U}_k^{(p)}, \tau_k^{(p)}; t_0^{(p)}, \mathbf{q}^{(p)}, t_f^{(p)}) \leq \mathbf{C}_{\max}^{(p)} \quad (k = 1, \ldots, N^{(p)}) \tag{19}$$

Furthermore, the boundary conditions of Eq. (4) are expressed as

$$\boldsymbol{\phi}_{\min}^{(p)} \leq \boldsymbol{\phi}^{(p)}(\mathbf{X}_0^{(p)}, t_0^{(p)}, \mathbf{X}_f^{(p)}, t_f^{(p)}) \leq \boldsymbol{\phi}_{\max}^{(p)} \tag{20}$$

Finally, the linkage constraints are mapped using the values at the termini and start, respectively, of the phase pairs $(p_l, p_r) \in [1, \ldots, P]$ $(l, r = [1, \ldots, P])$ as

$$\mathbf{L}_{\min}^{(s)} \leq \mathbf{L}^{(s)}(\mathbf{X}_f^{(p_l^s)}, t_f^{(p_l^s)}; \mathbf{q}^{(p_l^s)}, \mathbf{X}_0^{(p_u^s)}, t_0^{(p_u^s)}; \mathbf{q}^{(p_u^s)}) \leq \mathbf{L}_{\max}^{(s)}, \quad \begin{cases} p_l, p_u \in [1, \ldots, P] \\ s = 1, \ldots, L \end{cases} \tag{21}$$

4. ALGORITHM FOR SOLVING MULTIPLE-PHASE OPTIMAL CONTROL PROBLEMS USING THE GAUSS PSEUDOSPECTRAL METHOD

In this section we provide the algorithm for mapping of the multiple-phase GPM to an NLP in standard form. As mentioned in the introduction, we note that the approach described in this section is a generalization of many custom (non-reusable) MATLAB® programs that have been developed by the authors or MATLAB® programs co-developed by the authors and their colleagues [Rao 2003; Benson 2004; Huntington 2007; Huntington et al. 2007; Huntington and Rao 2008b; Huntington and Rao 2007; Huntington and Rao 2008a]. We note that the algorithm described is a formalization of much of this extensive programming experience.

In general, an NLP has the following standard form. Minimize the cost function

$$f(\mathbf{Z}) \tag{22}$$

subject to the algebraic equality and inequality constraints

$$\begin{array}{rcrcl} \mathbf{Z}_{\min} & \leq & \mathbf{Z} & \leq & \mathbf{Z}_{\max} \\ & & \mathbf{c}_E(\mathbf{Z}) & = & \mathbf{0} \\ \mathbf{c}_{I,\min} & \leq & \mathbf{c}_I(\mathbf{Z}) & \leq & \mathbf{c}_{I,\max} \end{array} \tag{23}$$

It is important to note in Eqs. (22) and (23) that the column vector $\mathbf{z} \in \mathbb{R}^{n_z}$ contains the NLP decision variables for the *entire* problem. Similar, $\mathbf{c}(\mathbf{z})$ is a column vector of constraints for the *entire* problem. Finally the subscripts "E" and "I" correspond to *equality* and *inequality* constraints, respectively. Recalling that the entire problem consists of P phases and denoting the decision vector within a given phase $p \in [1, \ldots, P]$ by $\mathbf{z}$, the complete vector of decision variables is a concatenation of the decision variables in each phase of the problem, i.e.,

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}^{(1)} \\ \vdots \\ \mathbf{z}^{(P)} \end{bmatrix} \tag{24}$$

In order to make the algorithm somewhat more manageable, the constraints are rearranged such that the constraints in each phase are a concatenation of the equality constraints and inequality constraints in the phase, i.e.,

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_E \\ \mathbf{c}_I \end{bmatrix} \tag{25}$$

where $\mathbf{c}$ is the subvector of constraints in phase $p = [1, \ldots, P]$. Finally, appended to the phase constraints are the *linkage* constraints, denoted $\mathbf{c}_L$. The total constraint vector is then given as

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}^{(1)} \\ \vdots \\ \mathbf{c}^{(P)} \\ \mathbf{c}_L \end{bmatrix} \tag{26}$$

4.1 Mapping of Decision Variables in a Single Phase

We now proceed to describe the mapping of the optimization vector within a particular phase $p \in [1, \ldots, P]$ to the values of the state, control, time, and static parameters. In order to make it easier to follow the details, the superscript $(\cdot)^{(p)}$ (denoting the phase number) is for the most part suppressed.

The total vector of optimization variables $\mathbf{z} \in \mathbb{R}^{n_x(N+2)+n_uN+n_q+2}$ in a particular phase $p = [1, \ldots, P]$ is given as

$$\mathbf{z}^{(p)} \equiv \mathbf{z} = \begin{bmatrix} \mathbf{z}_x \\ \mathbf{z}_u \\ \mathbf{q} \\ t_0 \\ t_f \end{bmatrix}, \qquad (p = 1, \ldots, P) \tag{27}$$

where $\mathbf{z}_x$ is the vector of variables associated with the values of the state at the discretization points, $\mathbf{z}_u$ is the vector of variables associated with the values of the control at the LG points, $\mathbf{q}$ is the vector of static optimization parameters, t_0 is the initial time, and t_f is the terminal time. The vector $\mathbf{z}_x$ is given as

$$\mathbf{z}_x = \begin{bmatrix} \boldsymbol{\chi}_1 \\ \vdots \\ \boldsymbol{\chi}_{n_x} \end{bmatrix} \tag{28}$$

where the column vectors $\boldsymbol{\chi}_j,\ (j = 1, \ldots, n_x)$ are given as

$$\boldsymbol{\chi}_j = \begin{bmatrix} x_{j1} \\ \vdots \\ x_{j,(N+2)} \end{bmatrix}, \qquad (j = 1, \ldots, n_x) \tag{29}$$

and $x_{jk},\ (j = 1, \ldots, n_x; k = 1, \ldots, N+2)$ is the value of the j^{th}component of the state at the k^{th} discretization point. Next, the vector $\mathbf{z}_u$ is given as

$$\mathbf{z}_u = \begin{bmatrix} \boldsymbol{\sigma}_1 \\ \vdots \\ \boldsymbol{\sigma}_{n_u} \end{bmatrix} \tag{30}$$

where the column vectors $\boldsymbol{\sigma}_j,\ (j = 1, \ldots, n_u)$ are given as

$$\boldsymbol{\sigma}_j = \begin{bmatrix} u_{j1} \\ \vdots \\ u_{j,N} \end{bmatrix}, \qquad (j = 1, \ldots, n_u) \tag{31}$$

and $u_{jk},\ (j = 1, \ldots, n_u; k = 1, \ldots, N)$ is the value of the j^{th} component of the control at the k^{th} collocation (LG) point. For use in vectorized operations in MATLAB®[The Mathworks, Inc. 2008], it is convenient to reshape the vectors $\mathbf{z}_x$ and $\mathbf{z}_u$ into matrices $\mathbf{Y}_x \in \mathbb{R}^{(N+2)\times n_x}$ and $\mathbf{Y}_x \in \mathbb{R}^{N\times n_u}$ whose rows contain a single vector value of the state and control, respectively. First, $\mathbf{Y}_x$ is obtained from $\mathbf{z}_x$ as

$$\mathbf{Y}_x = \mathbf{reshape}(\mathbf{z}_x, N+2, n_x) \tag{32}$$

where **reshape** is a pseudo-coded version of the MATLAB® "reshape" command. The vector $\mathbf{z}_x$ can be re-acquired from $\mathbf{Y}_x$ as

$$\mathbf{z}_x = \mathbf{Y}_x(:) \tag{33}$$

where ":" is a pseudo-coded version of the MATLAB®[The Mathworks, Inc. 2008] column-stacking operator. Next, $\mathbf{Y}_u$ is obtained from $\mathbf{z}_u$ as

$$\mathbf{Y}_u = \mathbf{reshape}(\mathbf{z}_u, N, n_u) \tag{34}$$

The vector $\mathbf{z}_u$ can be re-acquired from $\mathbf{Y}_u$ as

$$\mathbf{z}_u = \mathbf{Y}_u(:) \tag{35}$$

4.2 Construction of Total Vector of Decision Variables

To construct the total vector of NLP decision variables, the process described in the previous subsection is repeated for all phases $p = [1, \ldots, P]$. In other words, we construct the vector $\mathbf{z}$ by stacking the variables $\mathbf{z}^{(p)}$ using Eq. (24).

4.3 Construction of Constraint Vector within a Single Phase

Next, the algorithm for constructing the column vector of all NLP constraints within a given phase is described. The constraints within a single phase are obtained by stacking the collocated dynamic constraints, the collocated path constraints, and

the boundary conditions.[3] The vector of constraints within a single phase can then be written as

$$\mathbf{c}(\mathbf{z}) = \begin{bmatrix} \mathbf{c}_E(\mathbf{z}) \\ \mathbf{c}_I(\mathbf{z}) \end{bmatrix} \tag{36}$$

where, as before, the subscripts "E" and "I" correspond to *equality* and *inequality* constraints, respectively. A relatively straightforward way to map the equality constraints to a single column vector is as follows. First, suppose we define the *defect constraints*, $\boldsymbol{\delta}_k,\ k = 1, \ldots, N$, as

$$\boldsymbol{\delta}_k \ = \ \sum_{i=0}^{N} \mathbf{D}_{ki}\mathbf{X}_i - \frac{t_f - t_0}{2}\mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, t_k; \mathbf{q}), \qquad (k = 1, \ldots, N) \tag{37}$$

$$\boldsymbol{\delta}_{N+1} \ = \ \mathbf{X}_{N+1} - \mathbf{X}_0 - \frac{t_f - t_0}{2}\sum_{i=1}^{N} w_i \mathbf{f}(\mathbf{X}_i, \mathbf{U}_i, t_i; \mathbf{q}) \tag{38}$$

where we have assumed that $\mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, t_k; \mathbf{q})$ is a *row* vector of length n. Then we can define the matrix $\boldsymbol{\Delta}$ as

$$\boldsymbol{\Delta} = \begin{bmatrix} \boldsymbol{\delta}_1 \\ \vdots \\ \boldsymbol{\delta}_{N+1} \end{bmatrix} \tag{39}$$

It turns out that $\boldsymbol{\Delta}$ can be computed as

$$\boldsymbol{\Delta} = \begin{bmatrix} \mathbf{D}\mathbf{Y}_x(1:N+1,:) - \frac{t_f - t_0}{2}\mathbf{F} \\ \mathbf{Y}_x(\text{end},:) - \mathbf{Y}_x(1,:) - \frac{t_f - t_0}{2}\mathbf{w}^T\mathbf{F} \end{bmatrix} \tag{40}$$

where $\mathbf{w}$ is a column vector of LG weights, the notation "$1 : N+1$" is a pseudocode for the first $N+1$ rows of the matrix $\mathbf{Y}_x$, and $\mathbf{F}$ is a vectorized evaluation of the right-hand sides of the differential equations at the LG (collocation) points, i.e.,

$$\mathbf{F} = \begin{bmatrix} \mathbf{f}(\mathbf{X}_1, \mathbf{U}_1, t_1; \mathbf{q}) \\ \vdots \\ \mathbf{f}(\mathbf{X}_N, \mathbf{U}_N, t_N; \mathbf{q}) \end{bmatrix} \tag{41}$$

$$\equiv \mathbf{F}(\mathbf{Y}_x(2:N+1,:), \mathbf{Y}_u, \mathbf{t}(2:N+1); \mathbf{q})$$

where the column vector $\mathbf{t} \in \mathbb{R}^{N+2}$ is given as

$$\mathbf{t} = \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{N+1} \end{bmatrix} \tag{42}$$

The matrix $\boldsymbol{\Delta}$ can then be isomorphically mapped to the column vector of equality constraints $\mathbf{c}_E$ using a pseudo-coded version of the MATLAB® [The Mathworks,

[3]It is noted that the collocated dynamic constraints are *equality* constraints while the path constraints and boundary conditions are *inequality* constraints.

Inc. 2008] column-stacking operator ":" as

$$\mathbf{c}_E = \mathbf{\Delta}(:) \tag{43}$$

Next, let $\mathbf{H}_C$ be the matrix that arises from the evaluation of the path constraints at all of the LG points. Then $\mathbf{H}$ is given as

$$\mathbf{H}_C = \left[\begin{array}{c} \mathbf{C}(\mathbf{X}_1, \mathbf{U}_1, t_1; \mathbf{q}) \\ \vdots \\ \mathbf{C}(\mathbf{X}_N, \mathbf{U}_N, t_N; \mathbf{q}) \end{array} \right] \equiv \mathbf{C}(\mathbf{Y}_x(2:N+1,:), \mathbf{Y}_u, \mathbf{t}; \mathbf{q}) \tag{44}$$

Also, let $\mathbf{H}_\phi$ be the vector that arises from the evaluation of the boundary conditions. Noting that the boundary conditions are functions of only the state and time at the endpoints of the phase, we have

$$\mathbf{H}_\phi = \boldsymbol{\phi}(\mathbf{Y}_x(1,:), t_0, \mathbf{Y}_x(\text{end},:), t_f) \tag{45}$$

where it is noted that $\mathbf{Y}_x(1,:) = \mathbf{X}_0$ and $\mathbf{Y}_x(\text{end},:) = \mathbf{X}_f$, respectively. The inequality constraint vector is then given as

$$\mathbf{c}_I = \left[\begin{array}{c} \mathbf{H}_C(:) \\ \mathbf{H}_\phi \end{array} \right] \tag{46}$$

where we once again note the use of the MATLAB® [The Mathworks, Inc. 2008] column-stacking operator ":" to isomorphically map the matrix of path constraints to a column vector of length $n_c N$.

4.4 Construction of Linkage Constraint Vector

The vector of linkage constraints is obtained by using the values of the state and time at the terminus and start of each pair that is to be linked. We can write the evaluation of the s^{th} set of linkage constraints as follows:

$$\begin{array}{c} t_f^{(p_l^{(s)})} - t_0^{(p_r^{(s)})} \\ \mathbf{S}^{(s)}(\mathbf{Y}_x^{(p_l^{(s)})}(\text{end},:), \mathbf{q}^{(p_l^{(s)})}, \mathbf{Y}_x^{(p_r^{(s)})}(1,:), \mathbf{q}^{(p_r^{(s)})}) \end{array}, \qquad \left\{ \begin{array}{lcl} p_l, p_r & \in & [1, \ldots, P] \\ s & = & 1, \ldots, L \end{array} \right. \tag{47}$$

We note in Eq. (47) that the linkage conditions have been separated into those that link the independent variable (i.e., time) between phases and those that link the state and parameters between phases. The reason for this separation is that the linkage of the independent variable is a *linear* constraint and thus has a structure that can be taken advantage of when implementing the NLP in software.

5. CONSTRUCTION OF TOTAL VECTOR OF CONSTRAINTS

The total vector of constraints for all phases and the linkage constraints is then given as

$$\mathbf{c}(\mathbf{Z}) = \left\{ \begin{array}{c} \left[\begin{array}{c} \boldsymbol{\Delta}^{(1)}(:) \\ \mathbf{H}_C^{(1)}(:) \\ \mathbf{H}_\phi^{(1)} \end{array} \right] \\ \vdots \\ \left[\begin{array}{c} \boldsymbol{\Delta}^{(P)}(:) \\ \mathbf{H}_C^{(P)}(:) \\ \mathbf{H}_\phi^{(P)} \end{array} \right] \\ \left[\begin{array}{c} t_f^{(p_l^{(1)})} - t_0^{(p_r^{(1)})} \\ \mathbf{S}^{(1)}(\mathbf{Y}_x^{(p_l^{(1)})}(\text{end},:), \mathbf{q}^{(p_l^{(1)})}, \mathbf{Y}_x^{(p_r^{(1)})}(1,:), \mathbf{q}^{(p_r^{(1)})}) \end{array} \right] \\ \vdots \\ \left[\begin{array}{c} t_f^{(p_l^{(L)})} - t_0^{(p_r^{(L)})} \\ \mathbf{S}^{(L)}(\mathbf{Y}_x^{(p_l^{(L)})}(\text{end},:), \mathbf{q}^{(p_l^{(L)})}, \mathbf{Y}_x^{(p_r^{(L)})}(1,:), \mathbf{q}^{(p_r^{(L)})}) \end{array} \right] \end{array} \right\} \tag{48}$$

6. STRUCTURE OF GAUSS PSEUDOSPECTRAL NLP SPARSITY PATTERN

As it turns out, the NLP described in the previous section has an extremely well-defined structure that can be taken advantage of in a software implementation. First, the constraint Jacobian, $\partial \mathbf{c}/\partial \mathbf{z}$, is *sparse*. In order to see the structure of the constraint Jacobian more clearly, it is useful to examine the derivatives of the defect constraints, discretized path constraints, boundary conditions, and linkage conditions. First, suppose we let $\boldsymbol{\zeta}_i$, $(i = 1, \ldots, n)$ be the i^{th} column of the matrix $\boldsymbol{\Delta}$. Then $\boldsymbol{\Delta}$ can be written as

$$\boldsymbol{\Delta} = \left[\begin{array}{cccc} \boldsymbol{\zeta}_1 & \boldsymbol{\zeta}_2 & \cdots & \boldsymbol{\zeta}_n \end{array} \right], \qquad p = [1, \ldots, P] \tag{49}$$

Suppose further that we let $\boldsymbol{\eta}_i$ $(i = 1, \ldots, n)$ be i^{th} column of the matrix $\mathbf{F}$ [see Eq. (41)]. Then $\mathbf{F}$ can be written as

$$\mathbf{F} = \left[\begin{array}{cccc} \boldsymbol{\eta}_1 & \boldsymbol{\eta}_2 & \cdots & \boldsymbol{\eta}_{n_x} \end{array} \right], \qquad p = [1, \ldots, P] \tag{50}$$

Examining $\boldsymbol{\eta}_i$ $(i = 1, \ldots, n)$ further, it is seen that $\partial \boldsymbol{\eta}_i / \partial \boldsymbol{\chi}_j$, $(i, j = 1, \ldots, n_x)$ will be a structured matrix of size $(N+1) \times (N+2)$. The structure of $\partial \boldsymbol{\eta}_i / \partial \boldsymbol{\chi}_j$ will depend upon whether or not $\boldsymbol{\eta}_i$ is a function of $\boldsymbol{\chi}_j$. In the case where $\boldsymbol{\eta}_j$ *is* a function of $\boldsymbol{\chi}_j$, we have

$$\frac{\partial \boldsymbol{\eta}_i}{\partial \boldsymbol{\chi}_j} = \left[\begin{array}{ccc} \mathbf{0}_N & \mathbf{diag}\ (\mathbf{d}_{ij}) & \mathbf{0}_N \\ 0 & [\mathbf{w}.\mathbf{d}_{ij}]^T & 0 \end{array} \right] \tag{51}$$

where $\mathbf{0}_N$ is a column vector of zeros of length N, $\mathbf{w}.\mathbf{d}_i$ is a column vector obtained via element-by-element multiplication (i.e., the "." operator), and $\mathbf{d}_{ij}$ is a column vector defined as

$$\mathbf{d}_{ij} = \begin{bmatrix} \partial\eta_{i1}/\partial x_{j1} \\ \partial\eta_{i2}/\partial x_{j2} \\ \vdots \\ \partial\eta_{in_x}/\partial x_{jn_x} \end{bmatrix}, \qquad (i,j=1,\ldots,n) \tag{52}$$

Next, suppose we let

$$\begin{array}{rcl} \mathbf{D}_1 & = & \begin{bmatrix} \mathbf{D} & \mathbf{0}_N \end{bmatrix} \\ \mathbf{D}_2 & = & \begin{bmatrix} -1 & \mathbf{0}_N^T & 1 \end{bmatrix} \end{array} \tag{53}$$

In terms of $\mathbf{D}_1$ and $\mathbf{D}_2$ we define a new matrix $\tilde{\mathbf{D}}$ as

$$\tilde{\mathbf{D}} = \begin{bmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \end{bmatrix} \tag{54}$$

Then the terms in the defect constraints that do not involve $\mathbf{F}$ can be written as

$$\boldsymbol{\beta}_i = \tilde{\mathbf{D}}\boldsymbol{\chi}_i, \qquad (i=1,\ldots,n_x) \tag{55}$$

The matrix $\partial\boldsymbol{\beta}_i/\partial\boldsymbol{\chi}_j$ is then equal to either $\tilde{\mathbf{D}}$ or $\mathbf{0}_{(N+1)\times(N+2)}$ depending upon whether or not we are dealing with a diagonal block or an off-diagonal block, i.e.,

$$\frac{\partial\boldsymbol{\beta}_i}{\partial\boldsymbol{\chi}_j} = \left\{ \begin{array}{ll} \tilde{\mathbf{D}} & , \ i=j \\ \mathbf{0}_{(N+1)\times(N+2)} & , \ i\neq j \end{array} \right] \tag{56}$$

Combining all parts of the defect equations, the derivatives of the defect constraints with respect to each value of the discretized state are given as

$$\frac{\partial\boldsymbol{\zeta}_i}{\partial\boldsymbol{\chi}_j} = \tilde{\mathbf{D}} - \frac{t_f-t_0}{2}\frac{\partial\boldsymbol{\eta}_i}{\partial\boldsymbol{\chi}_j}, \qquad (i,j=1,\ldots,n_x) \tag{57}$$

Next, let

$$\mathbf{e}_{ij} = \begin{bmatrix} \partial\eta_{i1}/\partial u_{j1} \\ \partial\eta_{i2}/\partial u_{j2} \\ \vdots \\ \partial\eta_{in}/\partial u_{jn} \end{bmatrix}, \qquad (i=1,\ldots,n_x; j=1,\ldots,n_u) \tag{58}$$

Then the derivatives of the defect constraints with respect to the values of the control at the LG points are given as

$$\frac{\partial\boldsymbol{\zeta}_i}{\partial\boldsymbol{\sigma}_j} = -\frac{t_f-t_0}{2}\mathbf{diag}(\mathbf{e}_{ij}), \qquad (i=1,\ldots,n_x; j=1,\ldots,n_u) \tag{59}$$

Furthermore, the derivatives of the defect constraints with respect to the initial time, t_0, and terminal time, t_f, are given as

$$\begin{aligned}\frac{\partial \boldsymbol{\zeta}_i}{\partial t_0} &= \frac{1}{2}\boldsymbol{\eta}_i + \frac{t_f - t_0}{2}\frac{\partial \boldsymbol{\eta}_i}{\partial t_0} \\ \frac{\partial \boldsymbol{\zeta}_i}{\partial t_f} &= -\frac{1}{2}\boldsymbol{\eta}_i - \frac{t_f - t_0}{2}\frac{\partial \boldsymbol{\eta}_i}{\partial t_f}\end{aligned}, \qquad (i = 1, \ldots, n) \tag{60}$$

where it is noted that each quantity $\partial\boldsymbol{\zeta}/\partial t_0$ and $\partial\boldsymbol{\zeta}/\partial t_f$ is a *column vector* for a given value of p. Finally, the derivatives of the defect constraints with respect to the static parameters, $\mathbf{q}$, are given as

$$\frac{\partial \boldsymbol{\zeta}_i}{\partial \mathbf{q}} = -\frac{t_f - t_0}{2}\frac{\partial \boldsymbol{\eta}_i}{\partial \mathbf{q}} = -\frac{t_f - t_0}{2}\left[\begin{array}{cccc}\frac{\partial \boldsymbol{\eta}_i}{\partial q_1} & \frac{\partial \boldsymbol{\eta}_i}{\partial q_2} & \cdots & \frac{\partial \boldsymbol{\eta}_i}{\partial q_{n_q}}\end{array}\right] \tag{61}$$

Next, using an approach similar to that for the defect constraints, let the path constraint matrix defined in Eq. (44) be given column-wise as

$$\mathbf{H}_C = \left[\begin{array}{cccc}\boldsymbol{\xi}_1 & \boldsymbol{\xi}_2 & \cdots & \boldsymbol{\xi}_{n_c}\end{array}\right] \tag{62}$$

Furthermore, let

$$\mathbf{h}_{ij} = \left[\begin{array}{c}\partial\xi_{i1}/\partial x_{j1} \\ \partial\xi_{i2}/\partial x_{j2} \\ \vdots \\ \partial\xi_{iN}/\partial x_{jN}\end{array}\right], \qquad (i = 1, \ldots, n_c; j = 1, \ldots, n_x) \tag{63}$$

Then the derivatives of the path constraints with respect to the values of the state at the collocation points are given as

$$\frac{\partial \boldsymbol{\xi}_i}{\partial \boldsymbol{\chi}_j} = \mathbf{diag}(\mathbf{h}_{ij}), \qquad (i = 1, \ldots, n_c, j = 1, \ldots, n_x) \tag{64}$$

Furthermore, let

$$\mathbf{g}_{ij} = \left[\begin{array}{c}\partial\xi_{i1}/\partial u_{j1} \\ \partial\xi_{i2}/\partial u_{j2} \\ \vdots \\ \partial\xi_{in}/\partial u_{jn}\end{array}\right], \qquad (i = 1, \ldots, n_c, j = 1, \ldots, n_x) \tag{65}$$

Then the derivatives of the path constraints with respect to the values of the control at the collocation points are given as

$$\frac{\partial \boldsymbol{\xi}_i}{\partial \boldsymbol{\sigma}_j} = -\mathbf{diag}(\mathbf{g}_{ij}), \qquad (i = 1, \ldots, n_c, j = 1, \ldots, n_x \tag{66}$$

A graphical representation of the structure of a single phase constraint Jacobian as was described in this section can be seen in Fig. 2. Fig. 3 shows this structure over the entire problem, including all phases.

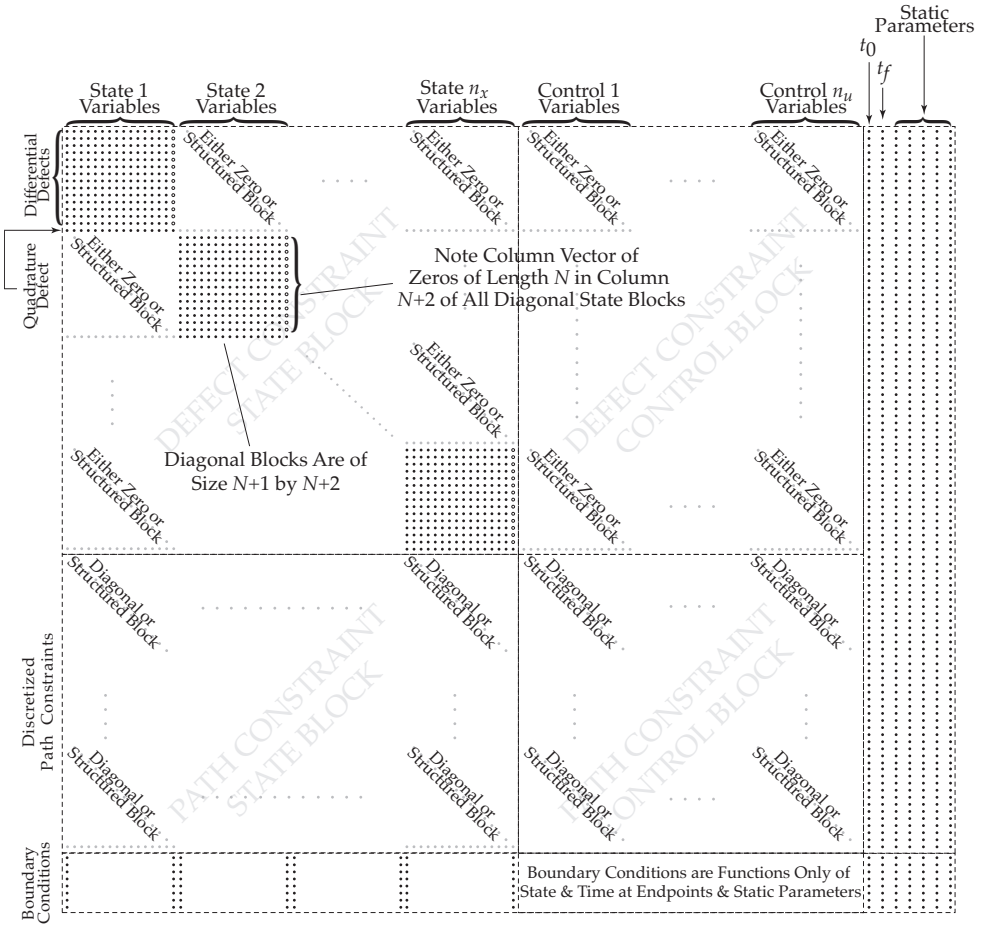

Fig. 2. Structure of Single Phase of Constraint Jacobian of discretization of Multiple-Phase Gauss Pseudospectral Method.

7. LAGRANGE MULTIPLIERS OF NLP AND COSTATE MAPPING

Each constraint in the NLP has a corresponding Lagrange multiplier. Suppose we let $\tilde{\mathbf{\Lambda}}_k^{(p)}$, $(k = 1, \ldots, N)$ and $\tilde{\mathbf{\Lambda}}_f^{(p)}$ be the Lagrange multipliers associated with the differential dynamic constraints of Eq. (16) and the quadrature constraint of Eq. (18), respectively, in each phase $p = [1, \ldots, P]$ of the optimal control problem. Furthermore, let $\tilde{\boldsymbol{\mu}}_k$, $(k = 1, \ldots, N)$ be the Lagrange multipliers associated with the discretized path constraints of Eq. (19). Finally, let $\tilde{\boldsymbol{\nu}}$ be the Lagrange multiplier associated with the discretized boundary conditions of Eq. (20). Then it has been shown [Benson 2004; Benson et al. 2006; Huntington 2007] that the costate (i.e., the adjoint) of the continuous-time optimal control problem is related to the Lagrange

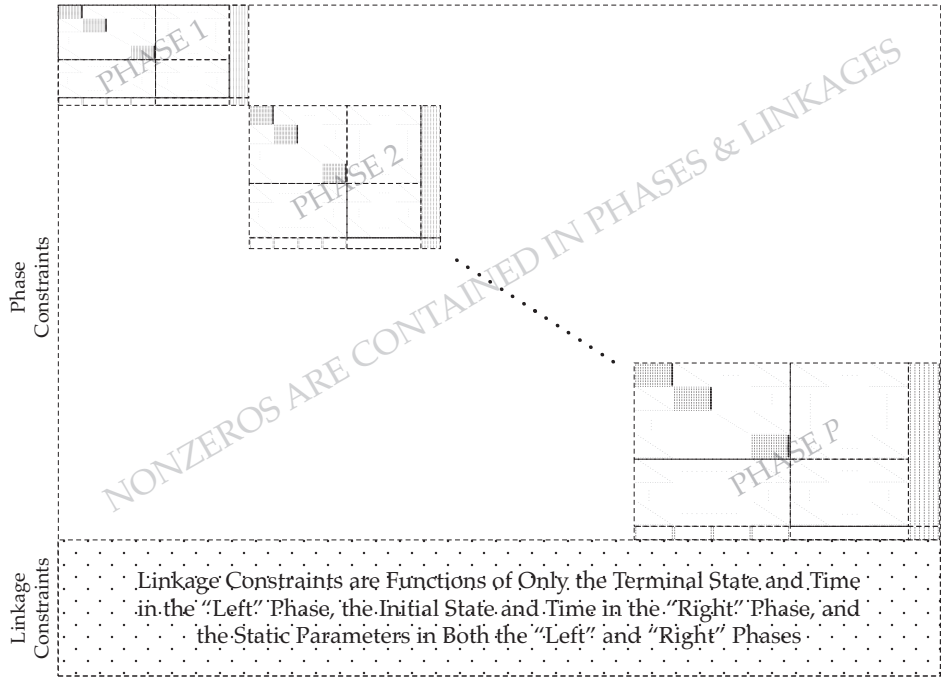

Fig. 3. Structure of Entire Sparsity Pattern for discretization of the Multiple-Phase Gauss Pseudospectral Method. Each Phase Has Nonzeros as Given in Fig. 2.

multipliers $\tilde{\mathbf{\Lambda}}_k$, $(k = 1, \ldots, N)$ and $\tilde{\mathbf{\Lambda}}_f$ as

$$\begin{aligned}
\mathbf{\Lambda}(t_f) &= \tilde{\mathbf{\Lambda}}_f \\
\mathbf{\Lambda}_k &= \mathbf{\Lambda}(t_f) + \tilde{\mathbf{\Lambda}}_k / w_k \\
\mathbf{\Lambda}(t_0) &= \mathbf{\Lambda}(t_f) - \sum_{i=1}^{N} D_{i0} \tilde{\mathbf{\Lambda}}_i \\
\boldsymbol{\mu}_k &= \frac{2}{t_f - t_0} \frac{\tilde{\boldsymbol{\mu}}_k}{w_k} \\
\boldsymbol{\nu} &= \tilde{\boldsymbol{\nu}}
\end{aligned} \tag{67}$$

The NLP described in Section 4 has a dual solution that includes the Lagrange multipliers for each category of discretized constraints. In particular, suppose we let $\mathbf{\Gamma}$ be the vector of Lagrange multipliers associated with the differential dynamic constraints and the quadrature constraints. Then, because the dynamics are discretized at N Legendre-Gauss points, we know that $\mathbf{\Gamma} \in \mathbb{R}^{n_x(N+1)}$ (i.e., we have $n_x N$ multipliers for the discretized dynamic constraints and another N multipliers for the quadrature constraints. The column vector $\mathbf{\Gamma}$ can be isomorphically mapped to a matrix of size $(N+1) \times n_x$ via a pseudo-coded version of the MATLAB® [The

Mathworks, Inc. 2008] command **reshape** [The Mathworks, Inc. 2008] as

$$\tilde{\mathbf{M}} = \mathbf{reshape}(\mathbf{\Gamma}, N+1, n_x) = \begin{bmatrix} \tilde{\mathbf{\Lambda}}_1 \\ \tilde{\mathbf{\Lambda}}_2 \\ \vdots \\ \tilde{\mathbf{\Lambda}}_{N+1} \end{bmatrix} \tag{68}$$

where $\tilde{\mathbf{\Lambda}}_i,\ (i = 1, \ldots, N+1)$ are row vectors. Next, we can construct a diagonal matrix $\mathbf{W}$ that contains the reciprocals of the Gauss weights as

$$\mathbf{W} = \mathbf{diag}(1/w_1, \ldots, 1/w_N) \tag{69}$$

The costate estimate given in Eq. (67) can then be obtained at the Legendre-Gauss points as

$$\mathbf{M} = \mathbf{W}\tilde{\mathbf{M}}(1 : \text{end} - 1, :) + \mathbf{repmat}(\tilde{\mathbf{\Lambda}}_f, N, 1) \tag{70}$$

In addition, the costate estimate at the *initial time* in phase $p = [1, \ldots, P]$ is computed as

$$\mathbf{\Lambda}(t_0) = \mathbf{repmat}(\tilde{\mathbf{\Lambda}}_f, N, 1) - \mathbf{sum}(\mathbf{repmat}(\mathbf{D}(:, 1), 1, n_x)\mathbf{M}(1 : \text{end} - 1, :), 1) \tag{71}$$

where we have used pseudo-coded versions of the MATLAB® commands **repmat** and **sum** (and, by virtue of the last argument being unity, the sum is taken along the columns). We can then augment the initial costate to the matrix $\mathbf{M}$ to obtain the complete matrix of costates as

$$\mathbf{M}_a = \begin{bmatrix} \mathbf{\Lambda}_0 \\ \mathbf{\Lambda}_1 \\ \mathbf{\Lambda}_2 \\ \vdots \\ \mathbf{\Lambda}_{N+1} \end{bmatrix} \tag{72}$$

8. SEPARATION OF CONSTANT AND NON-CONSTANT DERIVATIVES IN CONSTRAINT JACOBIAN

SNOPT [Gill et al. 2002] version 7 allows the user to choose the *SNOPTA* interface. One of the somewhat unique features of the SNOPTA interface is that the constant derivatives can be separated from the non-constant derivatives. This feature is described in the SNOPT 7 manual [Gill et al. 2006] but is repeated here in the context of the Gauss pseudospectral method. First, consider a problem such that each function (constraint or objective) can be written in the form

$$F_i(\mathbf{x}) = f_i(\mathbf{x}) + \sum_{j=1}^{n} A_{ij}\mathbf{x}_j \tag{73}$$

$f_i(\mathbf{x})$ is a nonlinear function, n is the total number of optimization (decision) variables, $\mathbf{x} \in \mathbb{R}^n$, and the elements A_{ij} are *constant*. Clearly the Jacobian of $\mathbf{F}$ is given as

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} + \mathbf{A} \tag{74}$$

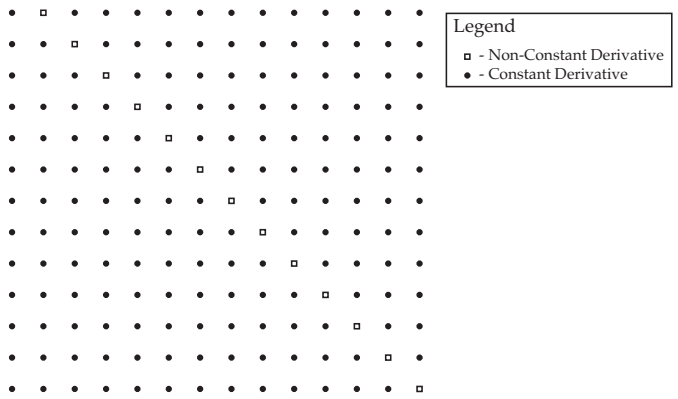

Fig. 4. Schematic Showing Constant and Non-Constant Derivatives Elements of $\partial\boldsymbol{\zeta}_i/\boldsymbol{\chi}_i$ of The Differential Equation Defect Constraints.

where

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} \tag{75}$$

Suppose now that there exist elements in the matrix $\mathbf{A}$ such that the corresponding elements in the matrix $\partial\mathbf{f}/\partial\mathbf{x}$ is *zero*. In other words, suppose that there are derivatives of $\mathbf{F}$ that are *constant*. Then these constant derivatives can be *stored* (i.e., they never need to be computed, but can be retrieved when necessary).

The Gauss pseudospectral method is an example of a situation where the exploitation of separability can be of great benefit. Specifically, it is known that the majority of the derivatives in the differential defect constraints of Eq. (37) are constant. In particular, examining Eq. (57), it is seen that the first N rows of the matrix $\partial\boldsymbol{\eta}_i/\boldsymbol{\chi}_i$[4] contain only elements along a diagonal that begins in the second column of the block and terminates at column N of the block. Consequently, elements that do not appear on this diagonal of $\partial\boldsymbol{\zeta}_i/\boldsymbol{\chi}_i$ are constant and are elements of the matrix $\mathbf{D}_1$. A schematic of the structure of the constant and non-constant elements of $\partial\boldsymbol{\zeta}_i/\boldsymbol{\chi}_i$ is seen in Fig. 4 while the sparsity pattern for the non-constant elements in a phase is given in Fig. 5. It is noted that the "off-diagonal" (in quotes because $\mathbf{D}_1$ is not square) elements can be off-loaded into the same matrix that stores the coefficients of the *linear* constraints. The SNOPTA interface then requires the row and column indices of the matrix coefficients to be stored as one-dimensional (column) vectors.

9. LINEAR CONSTRAINTS

As mentioned earlier, the constraints on the independent variable are *linear*. These linear constraints fall into two categories: monotonicity constraints and linkage

[4]Note the use of the index "i" in both $\boldsymbol{\zeta}$ and $\boldsymbol{\chi}$ because we are considering only the *diagonal* blocks of $\partial\boldsymbol{\zeta}_i/\boldsymbol{\chi}_j$ since these are the only blocks with constant derivatives

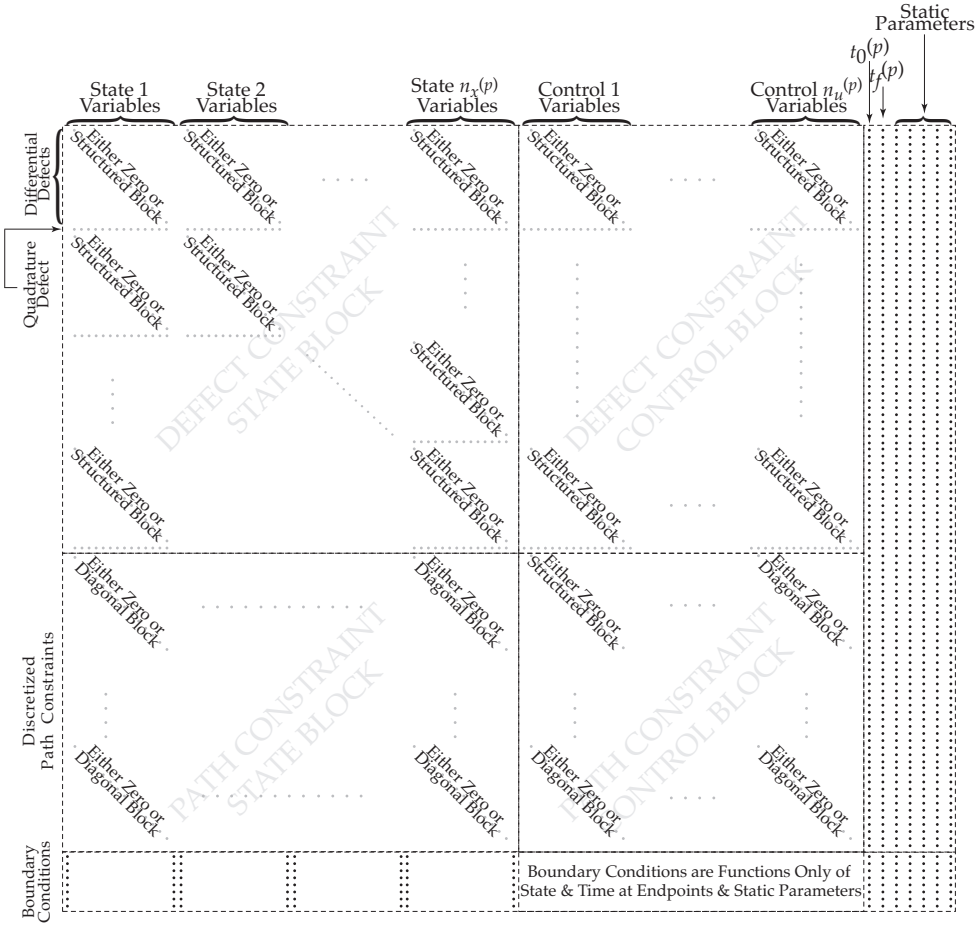

Fig. 5. Schematic Showing The Non-Constant Derivatives in a Single Phase of an Optimal Control Problem Discretized via the Gauss Pseudospectral Method..

constraints. First, we know that the independent variable must be *monotonic* (i.e., either increasing or decreasing throughout the problem). In order to ensure monotonicity, the following constraints must be placed on the independent variable in each phase $p \in [1, \ldots, P]$ of the problem:

$$\begin{array}{l} t_f - t_0 \geq 0 \text{ for an increasing independent variable} \\ t_f - t_0 \leq 0 \text{ for a decreasing independent variable} \end{array}, \qquad (p \in [1, \ldots, P]) \tag{76}$$

Next, the independent variable must be *continuous* at a phase interface. The continuity conditions on the independent variable are then restated from

$$t_f^{(p_l^{(s)})} - t_0^{(p_r^{(s)})}, \qquad (p_l, p_r \in [1, \ldots, L]) \tag{77}$$

where we recall that L is the number of linkage pairs. The entire set of linear constraints can then be written in the general linear constraint form

$$\mathbf{L}_{\min} \leq \mathbf{B}\mathbf{z} \leq \mathbf{L}_{\max} \tag{78}$$

where $\mathbf{B} \in \mathbb{R}^{n_l \times n_z}$ is matrix of coefficients for the linear constraints (in this case a matrix containing only zero, -1, and 1), $\mathbf{L}_{\min} \in \mathbb{R}^{n_l}$ are the lower bounds on the linear constraints, and $\mathbf{L}_{\max} \in \mathbb{R}^{n_l}$ are the upper bounds on the linear constraints. It is noted that the lower and upper bounds on the linear constraints are obtained directly from Eqs. (76) and (77), respectively. Finally, when treating the constraints on the independent variable as linear, these equations are removed from the general constraint vector given in Eq. (48).

10. COMPUTATION OF LEGENDRE-GAUSS POINTS, WEIGHTS, AND DIFFERENTIATION MATRIX

A key part of the algorithm described in this paper is the ability to compute the Legendre-Gauss points, weights, and differentiation matrix. It is noted that the Legendre-Gauss points are the roots of the N^{th} degree Legendre polynomial while the Legendre-Gauss weights are computed using Eq. (9). A convenient way to compute the Legendre-Gauss points is via the eigenvalues of the tri-diagonal *Jacobi* matrix. The Legendre-Gauss weights are then computed using Eq. (9) where the derivative of the N^{th}-degree Legendre polynomial, $\dot{P}_N$, is computed as

$$\dot{P}_N = \frac{-(n+1)P_N(\tau)}{1-\tau^2} \tag{79}$$

Finally, the Gauss pseudospectral differentiation matrix (which is a non-square matrix $\mathbf{D} \in \mathbb{R}^{N \times (N+1)}$) is given as

$$\mathbf{D}_{ki} = \begin{cases} \dfrac{(1+\tau_k)\dot{P}_N(\tau_k) + P_N(\tau_k)}{(\tau_k - \tau_i)\left[(1+\tau_i)\dot{P}_N(\tau_i) + P_N(\tau_i)\right]} & , \; i \neq k \\ \\ \dfrac{(1+\tau_i)\ddot{P}_N(\tau_i) + 2P_N(\tau_i)}{2\left[(1+\tau_i)\dot{P}_N(\tau_i) + P_N(\tau_i)\right]} & , \; i = k \end{cases} \tag{80}$$

11. COMPUTATION OF ENDPOINT CONTROLS

One of the attributes of the Gauss pseudospectral method is that the control is discretized at only the interior points (i.e., the Legendre-Gauss points). As a result, in order to obtain the controls at the endpoints of each phase, it is necessary to employ the Pontryagin minimum principle [Kirk 2004] using the endpoint values of the state and costate. The problem that needs to be solved is as follows. Determine either $\mathbf{U}(t_0)$ or $\mathbf{U}(t_f)$ (depending upon whether the initial or terminal control is desired) that minimizes the Hamiltonian

$$H = \mathcal{L} + \boldsymbol{\lambda}^T \mathbf{f} \tag{81}$$

subject to constraint

$$\mathbf{C}_{\min} \leq \mathbf{C} \leq \mathbf{C}_{\max} \tag{82}$$

The optimization problem given in Eqs. (81) and (82) is implemented by using the values of the state and costate at the endpoints of each phase obtained by solving the GPM NLP and minimizing over the allowable set of controls. The endpoint control optimization problem is small (having only a number of variables equal to the number of controls in each phase) and, thus, can be solved by a dense optimizer such as NPSOL.

12. AUTOMATIC SCALING OF NLP AND DETERMINATION OF DEPENDENCIES

An extremely important issue that arises in the solution of any optimization problem is *scaling*. A poorly scaled problem can lead to either extremely slow convergence or divergence [Betts 2001]. While not a great deal of information exists in the literature on scaling (primarily because scaling is more of an art than a science), it is a practical matter that needs to be dealt with in any implementation. In this section we describe a scaling procedure [Betts 2001]. It has been found that this scaling procedure works extremely well on a wide variety of problems and can be used as a substitute for the tedious work of manual scaling. Finally, it is noted that the procedure described here scales the functions of the *optimal control problem* as opposed to scaling the functions of the NLP.

The scaling procedure used in the current algorithm is divided into two parts. The first part of the procedure automatically scales the variables based on the bounds specified by the user. Given the user-specified bounds on the time, state, control, and parameters in each phase of the problem, the unscaled NLP variables are given as box-bound constraints [see Eq. (23)] as

$$\mathbf{z}_{\min} \leq \mathbf{z} \leq \mathbf{z}_{\max} \tag{83}$$

Now let $\mathbf{S}_{\mathbf{z}}$ be a diagonal matrix whose diagonal elements contain the scale factors for the variables. Then the scaled value of $\mathbf{z}$, denoted $\tilde{\mathbf{z}}$, is given as

$$\tilde{\mathbf{z}} = \mathbf{S}_{\mathbf{z}}\mathbf{z} \tag{84}$$

Because the user may set some of the lower and upper limits to $\pm\infty$, the first step is to find the infinite limits and set the diagonal elements in $\mathbf{S}_{\mathbf{z}}$ to unity. Next, for the lower and upper limits that remain, the *larger* of $|\mathbf{z}_{\min}|$ and $|\mathbf{z}_{\max}|$ is determined. The corresponding diagonal elements of $\mathbf{S}_{\mathbf{z}}$ are then set to the reciprocal of the larger values. In this manner, the variables are scaled such that their scaled limits

either lie between -1 and 1, -1 and ∞, $-\infty$ and 1, or $-\infty$ and ∞. It is noted, however, that if sensible non-infinite bounds are chosen, all scaled limits will lie between -1 and 1.

The second step in the automatic scaling procedure is to scale the functions of the optimal control problem. First, as has been suggested in the literature [Betts 2001], the differential equation constraints (i.e., the defect constraints) are scaled using the same scale factors as used to scale the states. Next, suppose we let $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{q}$ be the state, control, and static parameters in a given phase $p \in [1, \ldots, P]$ of the optimal control problem with corresponding scaled values $\tilde{\mathbf{x}}$, $\tilde{\mathbf{u}}$, and $\tilde{\mathbf{q}}$. Furthermore, let $\mathbf{S_x}$, $\mathbf{S_u}$ and $\mathbf{S_q}$ be diagonal matrices whose diagonal elements contain the corresponding scale factors. We then have

$$\begin{aligned} \tilde{\mathbf{x}} &= \mathbf{S_x}\mathbf{x} \\ \tilde{\mathbf{u}} &= \mathbf{S_u}\mathbf{u} \\ \tilde{\mathbf{q}} &= \mathbf{S_q}\mathbf{q} \end{aligned} \tag{85}$$

Finally, let $\tilde{\mathbf{C}}$ be the scaled value of the path constraints with a corresponding diagonal scaling matrix $\mathbf{S_C}$. Then, in a manner similar to the variables, we have

$$\tilde{\mathbf{C}} = \mathbf{S_C}\mathbf{C} \tag{86}$$

Suppose now that the Jacobian of $\mathbf{C}$ is computed as

$$\left[\begin{array}{ccc} \dfrac{\partial \mathbf{C}}{\partial \mathbf{x}} & \dfrac{\partial \mathbf{C}}{\partial \mathbf{u}} & \dfrac{\partial \mathbf{C}}{\partial \mathbf{q}} \end{array} \right] \tag{87}$$

Then the scale factors for the path constraints are obtained by (1) evaluating the Jacobian given in Eq. (87) at a specified number of trial points $(\mathbf{x}_i, \mathbf{u}_i, \mathbf{q}_i)$, $(i = 1, \ldots, n_T)$ where n_T is the number of trials in the phase; (2) computing the row norm of the path constraint Jacobian at the trial points; and (3) taking the average of the row norms. The scale factor for each scalar path constraint is then applied at each collocation point. The boundary conditions and the linkage constraints are scaled in a similar manner.

Simultaneous with the determination of scale factors for the NLP, a procedure has been implemented to determine the dependencies of the differential equations and path constraints on the state and control. In particular, recall in Section 6 that the off-diagonal blocks in each phase are either zero or structured depending upon whether a particular differential equation or path constraint is a function of a particular variable. In the algorithm described here, the Jacobian *pattern* (i.e., a matrix of ones and zeros) is determined at the n_T trial points. If at any of the trial points a particular element in the Jacobian is nonzero, then this dependence is included in the NLP sparsity pattern and the particular off-diagonal block in Fig. 5 is nonzero.

13. NUMERICAL METHODS FOR COMPUTATION OF OBJECTIVE FUNCTION GRADIENT AND CONSTRAINT JACOBIAN

A key issue that arises in the solution of any NLP is the computation of the derivatives to obtain the objective function gradient and constraint Jacobian. $\mathcal{GPOPS}$ has three options for computation of these quantities. The first option is the built-in finite difference method in SNOPT [Gill et al. 2006]. While finite differencing will

work on some problems, it can be problematic due to the fact that derivative approximations obtained using finite differencing can be inaccurate, thereby leading to a failure of the NLP solver to converge to an optimal solution. As a result, the following derivative methods can also be used in $\mathcal{GPOPS}$: (1) automatic differentiation via the program *Matlab automatic differentiation* [Forth 2006; Forth and Edvall 2007] and complex-step differentiation [Martins et al. 2003]. In particular, a customized complex-step differentiation method has been implemented in $\mathcal{GPOPS}$. As has been discussed in Ref. [Martins et al. 2003], complex-step differentiation provides extremely high-accuracy derivatives and is insensitive to the choice of the step size (as compared with finite difference approximations which create roundoff error for small values of the step). It is noted, however, that the following functions in MATLAB® create issues with complex-step differentiation: "abs", "min", "max", and "dot.". In addition, the user needs to code the problem carefully and not use the standard transpose operation in matlab, but use the "dot-transpose" to ensure that a *real transpose* is taken (i.e., not a complex-conjugate transpose). It is noted that the complex-step method has been tested on all of the problems included with the $\mathcal{GPOPS}$ software and has been found to work extremely well in practice.

14. USER INTERFACE TO $\mathcal{GPOPS}$

$\mathcal{GPOPS}$ has an original user interface where the optimal control problem can be input in an intuitive yet compact manner. The key MATLAB® programming element that makes this user interface possible is the cell array. In particular, $\mathcal{GPOPS}$ utilizes multi-level cell arrays that enables compact specification of the lower and upper bounds on all variables and constraints in each phase of the problem and provides equal flexibility when specifying how the phases are to be linked. Finally, all of the functions in the optimal control problem (i.e., cost function, differential-algebraic equations, boundary conditions, and linkage conditions) use cell array inputs, thus being consistent with other inputs in the software. To the best knowledge of the authors, no user interface similar to the one in $\mathcal{GPOPS}$ has ever been previously developed or used. All of the details regarding the $\mathcal{GPOPS}$ interface are found in the $\mathcal{GPOPS}$ user's manual that accompanies this article.

15. APPLICATIONS OF $\mathcal{GPOPS}$

In this section we consider two examples of the use of a MATLAB® implementation of the previously described $\mathcal{GPOPS}$ algorithm. The first example is the classical maximum-range problem [Bryson and Ho 1975] while the second example is a multiple-stage (i.e., multiple-phase) launch vehicle ascent problem [Benson 2004]. As mentioned earlier, the MATLAB® version of the sparse NLP solver SNOPT [Gill et al. 2006] developed by Gill [Gill et al. 2006] was used. It is noted that for both examples that default optimization and feasibility tolerances were used in SNOPT along with a limited memory Hessian and an LU factorization with complete pivoting (see the SNOPT manual [Gill et al. 2006] for more details). Finally, both examples were solved with different forms of differentiation for the objective function gradient and constraint Jacobian (i.e., , finite differences, complex-step, and automatic differentiation). It is noted that automatic differentiation was achieved using the program *MATLAB Automatic Differentiation* [Forth

2006; Forth and Edvall 2007]. Both examples given below are in the software distribution of $\mathcal{GPOPS}$.

15.1 Example 1: Bryson Maximum-Range Problem

Consider the following well-known optimal control problem [Bryson and Ho 1975] classically known as the *maximum-range problem*. Maximize the cost functional

$$J = x(t_f) \tag{88}$$

subject to the dynamic constraints

$$\begin{array}{rcl} \dot{x} & = & vu_1 \\ \dot{y} & = & vu_2 \\ \dot{v} & = & a - gu_2 \end{array} \tag{89}$$

the path constraint

$$u_1^2 + u_2^2 = 1 \tag{90}$$

and the boundary conditions

$$\begin{array}{rcl} x(0) & = & 0 \\ y(0) & = & 0 \\ v(0) & = & 0 \\ y(t_f) & = & 0.1 \end{array} \tag{91}$$

with $g = 1$, $a = g/2$, and $t_f = 2$. The $\mathcal{GPOPS}$ solution of the Bryson maximum-range problem is summarized in the following three plots that contain the state ($x(t)$), costate ($\lambda(t)$), and the Hamiltonian (H), respectively, for the problem(where $H = L + \lambda f$). It is noted that this example was solved three times, once using each method for approximating the objective function gradient and constraint Jacobian. Table 1 summarizes the computation time taken by each method where the computations were performed on an Intel Core Duo 2.5 Ghz MacBook Pro running MATLAB R2007a. It is seen that the execution time using finite-differencing is significantly larger than complex-step differentiation, while the execution time for complex step differentiation is larger than automatic differentiation. On the other hand, the number of iterations taken by complex-step and automatic differentiation are the same, indicating that the significant difference between complex-step and automatic differentiation lies in the speed required to compute the derivative (as opposed to the accuracy of the derivative itself).

Table 1. Comparison of the performance of the $\mathcal{GPOPS}$ Algorithm Using Various Differentiation Methods for Computation of the Objective Function Gradient and Constraint Jacobian for the Bryson Maximum-Range Problem.

Differentiation Method	Major Iterations	Execution Time (s)
Finite-Difference	80	42.87
Complex-Step	32	11.73
Automatic	32	3.93

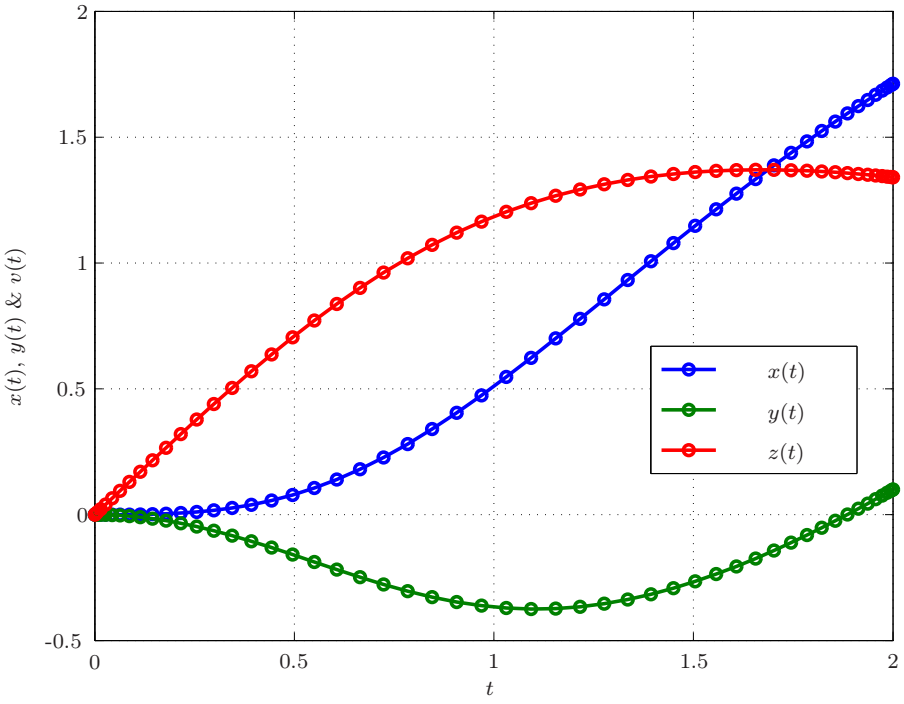

Fig. 6. Components of the state, $x(t)$, $y(t)$, and $v(t)$, vs. t for Bryson Maximum-Range Problem .

15.2 Example 2: Multiple-Stage Launch Vehicle Ascent Problem

The problem considered in this section is the ascent of a multiple-stage launch vehicle. The objective is to maneuver the launch vehicle from the ground to the target orbit while maximizing the remaining fuel in the upper stage. It is noted that this example is found in several places in the open literature [Benson 2004; Huntington 2007].

15.2.1 *Vehicle Properties.* The launch vehicle considered in this example has two main stages along with nine strap-on solid rocket boosters. The flight of the vehicle can be divided into *four* distinct phases. The first phase begins with the rocket at rest on the ground and at time t_0, the main engine and six of the nine solid boosters ignite. When the boosters are depleted at time t_1, their remaining dry mass is jettisoned. The final three boosters are then ignited, and along with the main engine, represent the thrust for the second phase of flight. These three remaining boosters are jettisoned when their fuel is exhausted at time t_2, and the main engine alone creates the thrust for the third phase. The fourth phase begins when the main engine fuel has been exhausted (MECO) and the dry mass associated with the main engine is ejected at time t_3. The thrust during phase four is from a second stage, which burns until the target orbit has been reached (SECO) at time t_4, thus completing the trajectory. The specific characteristics of these rocket motors can

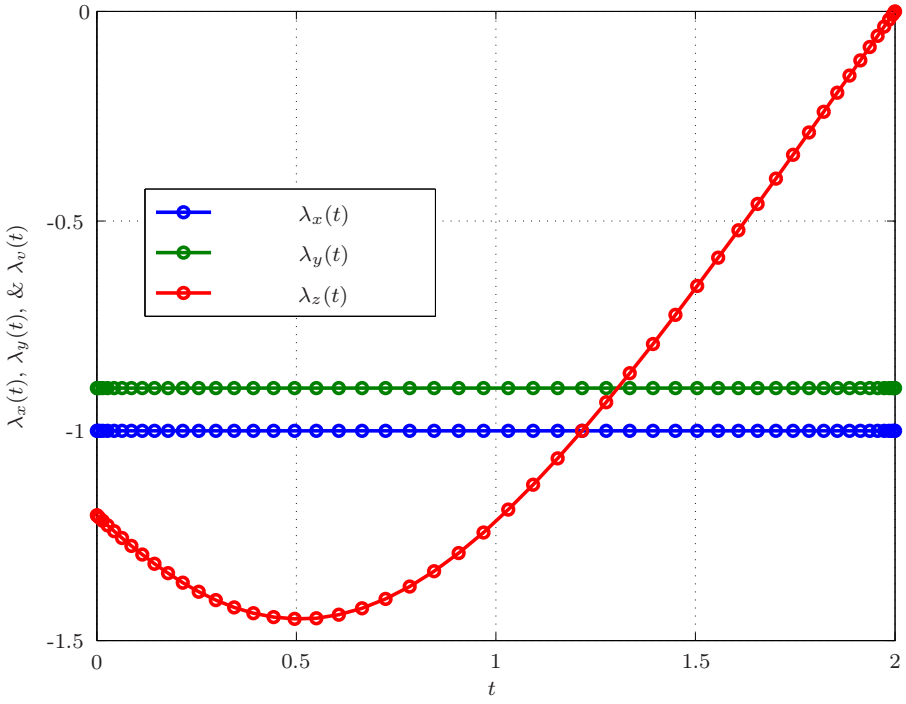

Fig. 7. Components of the costate, $\lambda_x(t)$, $\lambda_y(t)$, and $\lambda_v(t)$, vs. t for Bryson Maximum-Range Problem .

be seen in Table 2.[5] Note that the solid boosters and main engine burn for their entire duration (meaning t_1, t_2, and t_3 are fixed), while the second stage engine is shut off when the target orbit is achieved (t_4 is free).

Table 2. Mass and propulsion properties of the launch vehicle ascent problem.

	Solid Rocket Boosters	Stage 1	Stage 2
Total Mass (kg)	19290	104380	19300
Propellant Mass (kg)	17010	95550	16820
Engine Thrust (N)	628500	1083100	110094
Isp (s)	283.3334	301.6878	467.2131
Number of Engines	9	1	1
Burn Time (s)	75.2	261	700

15.2.2 *Dynamic Model.* The equations of motion for a non-lifting point mass in flight over a spherical rotating planet are expressed in Cartesian Earth centered

[5]It is noted that the values of specific impulse shown in Table 2 were obtained by computing $Tt_b/(g_0 m_{\text{prop}})$ where T is the thrust, t_b is the burn time, $g_0 = 9.80665$, and m_{prop} is the propellant mass

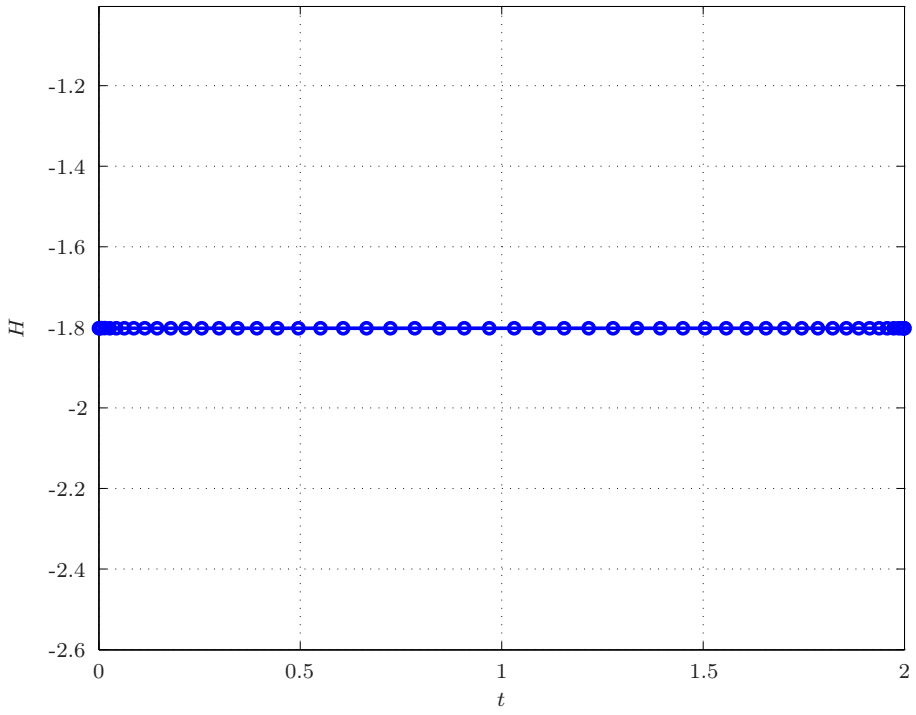

Fig. 8. H vs. t for Bryson Maximum-Range Problem. .

inertial (ECI) coordinates as

$$\begin{aligned}\dot{\mathbf{r}} &= \mathbf{v}\\ \dot{\mathbf{v}} &= -\frac{\mu}{\|\mathbf{r}\|^3}\mathbf{r} + \frac{T}{m}\mathbf{u} + \frac{\mathbf{D}}{m}\\ \dot{m} &= -\frac{T}{g_0 I_{sp}}\end{aligned} \tag{92}$$

where $\mathbf{r} = \left[\begin{array}{ccc} x & y & z \end{array}\right]^T$ is the Cartesian ECI position, $\mathbf{v} = \left[\begin{array}{ccc} v_x & v_y & v_z \end{array}\right]^T$ is the Cartesian ECI velocity, μ is the gravitational parameter, T is the vacuum thrust, m is the mass, g_0 is the acceleration due to gravity at sea level, I_{sp} is the specific impulse of the engine, $\mathbf{u} = \left[\begin{array}{ccc} u_x & u_y & u_z \end{array}\right]^T$ is the thrust direction, and $\mathbf{D} = \left[\begin{array}{ccc} D_x & D_y & D_z \end{array}\right]^T$ is the drag force. The drag force is defined in vector form as

$$\mathbf{D} = -\frac{1}{2}\rho S C_D \|\mathbf{v}_{rel}\|\mathbf{v}_{rel} \tag{93}$$

where C_D is the drag coefficient, S is the reference area, ρ is the atmospheric density, and $\mathbf{v}_{rel}$ is the Earth relative velocity, where $\mathbf{v}_{rel}$ is given as

$$\mathbf{v}_{rel} = \mathbf{v} - \mathbf{\Omega} \times \mathbf{r} \tag{94}$$

where $\mathbf{\Omega}$ is the angular velocity of the Earth relative to the inertial reference frame. The atmospheric density is modeled as the exponential function

$$\rho = \rho_0 \exp[-h/H] \tag{95}$$

where ρ_0 is the atmospheric density at sea level, $h = \|\mathbf{r}\| - R_e$ is the altitude, R_e is the equatorial radius of the Earth, and H is the density scale height. The numerical values for these constants can be found in Table 3.

Table 3. Constants used in the launch vehicle example.

Constant	Value
Payload Mass (kg)	4164
S (m^2)	4π
C_d	0.5
ρ_0 (kg/m^3)	1.225
H (m)	7200
t_1 (s)	75.2
t_2 (s)	150.4
t_3 (s)	261
R_e (m)	6378145 m
Ω (rad/s)	$7.29211585 \times 10^{-5}$

15.2.3 *Constraints.* The launch vehicle starts on the ground at rest (relative to the Earth) at time t_0, so that the ECI initial conditions are

$$\begin{aligned} \mathbf{r}(t_0) &= \mathbf{r}_0 = \left[\begin{array}{ccc} R_e \cos\phi_0 & 0 & R_e \sin\phi_0 \end{array} \right]^T \\ \mathbf{v}(t_0) &= \mathbf{v}_0 = \boldsymbol{\omega} \times \mathbf{r}_0 \\ m(t_0) &= m_0 = 301454 \quad \text{kg} \end{aligned} \tag{96}$$

where $\phi_0 = 28.5$ deg and corresponds to the geocentric latitude of Cape Canaveral, Florida and it is arbitrarily assumed that the inertially fixed axes are such that the initial inertial longitude is zero. The terminal constraints define the target geosynchronous transfer orbit (GTO), which is defined in terms of orbital elements as

$$\begin{aligned} a_f &= 24361.14\,\text{km}, \\ e_f &= 0.7308, \\ i_f &= 28.5\,\text{deg}, \\ \Omega_f &= 269.8\,\text{deg}, \\ \omega_f &= 130.5\,\text{deg} \end{aligned} \tag{97}$$

where a is the semi-major axis, e is the eccentricity, i is the inclination, Ω is the inertial longitude of the ascending node, and ω is the argument of perigee. It is noted that, because we are not specifying the location in terminal orbit that must be attained by the vehicle, the true anomaly, ν, is free. These orbital elements can be transformed into ECI coordinates via the transformation, T_{o2c} as given in [Bate et al. 1971].

In addition to the boundary constraints, there exists both a state path constraint and a control path constraint in this problem. A state path constraint is imposed

to keep the vehicle's altitude above the surface of the Earth, so that

$$|\mathbf{r}| \geq R_e \tag{98}$$

where R_e is the equatorial radius of the Earth. Next, the thrust direction is constrained to be of unit length via the equality path constraint

$$|\mathbf{u}| = 1 \tag{99}$$

Lastly, the phases are connected via the following set of linkage constraints at the terminus of phases 1, 2, and 3 and the start of phases 2, 3 and 4, respectively as

$$\begin{aligned} \mathbf{r}(t_f) - \mathbf{r}(t_0^{(p+1)}) &= \mathbf{0}, \\ \mathbf{v}(t_f^{(p)}) - \mathbf{v}(t_0^{(p)}) &= \mathbf{0}, \qquad (p = 1, \ldots, 3) \\ m(t_f^{(p)}) - m_{\text{dry}}^{(p)} - m(t_0^{(p+1)}) &= 0 \end{aligned} \tag{100}$$

It is noted that the linkage constraint on the mass at each of the phase interfaces includes an instantaneous drop of the dry mass of the particular stage. In this case, mass drops at the ends of phases 1, 2, and 3 are given, respectively, as is $6m_{\text{dry}}^{\text{srb}}$, $3m_{\text{dry}}^{\text{srb}}$, and $m_{\text{dry}}^{\text{first}}$, where the subscript "dry" denotes the dry mass (i.e., mass excluding fuel) and the superscripts "srb" and "first" denote a single solid rocket booster and the first stage, respectively. The objective of the problem is to determine the control (and corresponding trajectory) that minimizes the cost functional

$$J = -m(t_f^{(4)}) \tag{101}$$

subject to the conditions of Eqs. (92), (96), (97), (98), and (99). The problem was posed in SI units using the aforementioned auto-scaling procedure and 15 Legendre-Gauss points (i.e., nodes) in each phase. Finally, a constant initial guess was used for the position and velocity in each phase while a linear initial guess was used for the mass.

The $\mathcal{GPOPS}$ solution obtained for the launch vehicle ascent problem is summarized in the following Figs. 9–12 that contain the altitude, speed, components of control, and Hamiltonian, respectively, alongside the solution obtained using the commercial software program *Sparse Optimal Control Software* ($\mathbb{SOCS}$) [Betts and Huffman 1997]. It is seen that the $\mathcal{GPOPS}$ solution and the $\mathbb{SOCS}$ solution are in excellent agreement. With regard to accuracy, the optimal objective functions obtained using $\mathcal{GPOPS}$ and $\mathbb{SOCS}$ are 7529.7123 kg and 7529.7125 kg, respectively, (i.e., resulting in a difference of approximately 2.1×10^{-4} kg). In addition to the excellent agreement of between $\mathcal{GPOPS}$ and $\mathbb{SOCS}$, it is seen that the Hamiltonian obtained from $\mathcal{GPOPS}$ is piecewise constant. In this case it is known that the durations of the first three phases are *fixed* while the duration of the fourth phase is *free*. Because the Hamiltonian is not an explicit function of time for this problem, we know that its value in the first three phases should be constant (but not necessarily zero) while its value in the fourth phase should be zero (which, indeed, is the case). The additional result showing that the Hamiltonian is constant in each phase (and zero in the fourth phase) provides further validation of the accuracy of the $\mathcal{GPOPS}$ solution.

With regard to computational performance, Table 4 shows a high level set of statistics between the $\mathcal{GPOPS}$ solution using SNOPT and $\mathbb{SOCS}$ with SPRNLP.

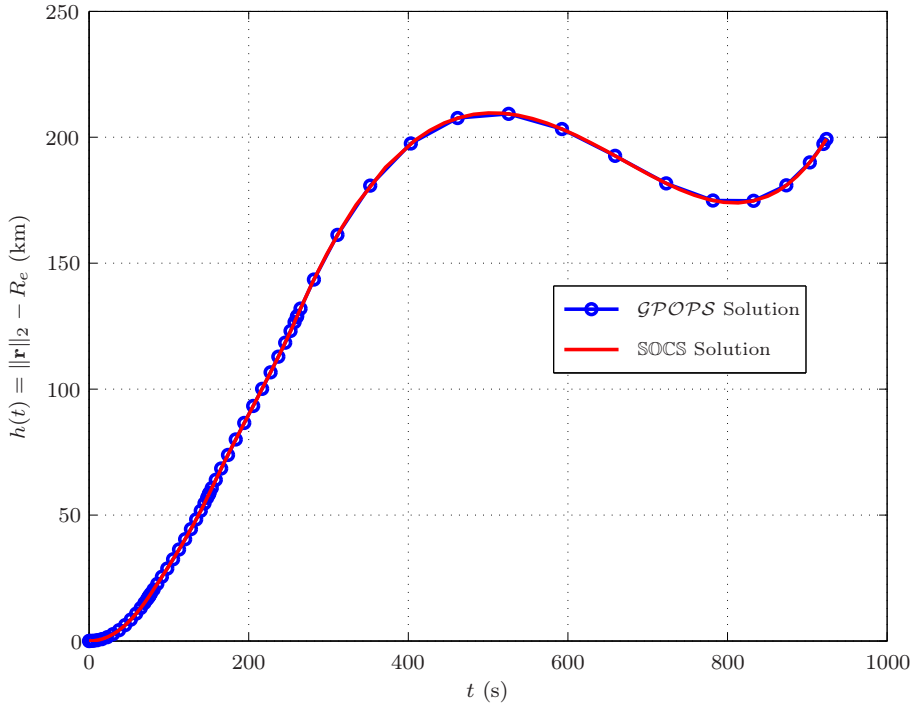

Fig. 9. Altitude vs. time for the launch vehicle ascent problem.

It is seen that a comparable number of major iterations were required by each software. In addition, it is noted that, while the execution time taken by $\mathbb{SOCS}$ is much less than that taken by $\mathcal{GPOPS}$, the $\mathbb{SOCS}$ code was run with a binary file (i.e., an executable file) compiled from FORTRAN source code using the Linux version of the Intel® FORTRAN 95 compiler whereas $\mathcal{GPOPS}$ was run using the SNOPT mex file in MATLAB® R2007a on an Intel Core Duo 2.5 Ghz MacBook Pro running MATLAB R2007a. While $\mathcal{GPOPS}$ is less computationally efficient than $\mathbb{SOCS}$, such a difference in performance is expected due to the difference in the underlying software (MATLAB mex file vs. FORTRAN binary) used in execution of each program.[6] On the other hand, $\mathbb{SOCS}$ needs to go through an elaborate grid refinement procedure (in this case utilizing four refinements of the time grid) and ultimately solves an NLP that is much larger (1822 variables and 1541 constraints) whereas the GPM solves the problem on a much coarser grid (664 variables and 534 constraints) and produces a solution of comparable quality to that of $\mathbb{SOCS}$. Furthermore, despite the complexity of the problem and the extremely poor initial guess (the guess is essentially a constant in each phase of the problem), it was possible to solve this problem in only 56 major iterations using automatic

[6]As a rule of thumb, efficient MATLAB® code will generally run anywhere from five to ten times slower than compiled FORTRAN code. Taking such a factor into account, it is expected for this example that a GPM code written in FORTRAN using automatic differentiation would execute in approximately 3 seconds.

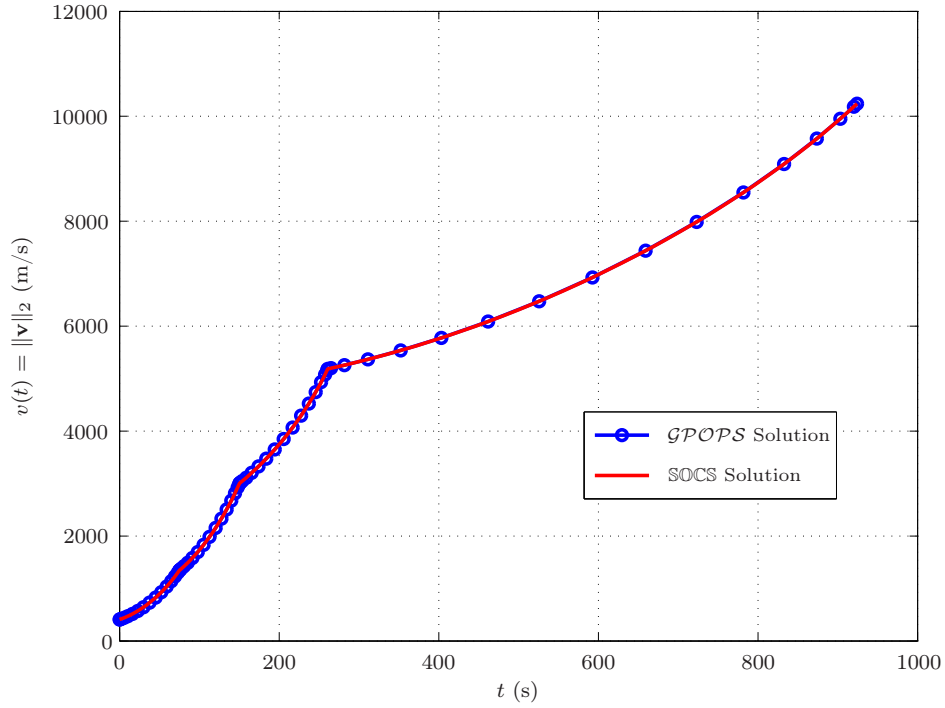

Fig. 10. Inertial speed vs. time for the launch vehicle ascent problem.

differentiation and 89 iterations using complex-step differentiation. Finally, this example shows the utility of the automatic scaling procedure because the problem is posed in SI units but is solved internally using scale factors computed as described in section 12.

Table 4. Brief table comparing the performance of the $\mathcal{GPOPS}$ Algorithm with SNOPT and SOCS using SPRNLP.

Optimal Control Solver	Major Iterations	Execution Time (s)	NLP Variables	NLP Constraints
$\mathcal{GPOPS}$ (Automatic)	37	33.0 (average)	664	534
$\mathcal{GPOPS}$ (Complex-Step)	39	200.0 (average)	664	534
SOCS	71	3.1	1822	1541

16. CONCLUSIONS

An algorithm is described for solving non-sequential multiple-phase optimal control problems using the Gauss pseudospectral method (GPM). In particular, a detailed explanation has been given of the mathematical structure of the nonlinear programming problem (NLP) that arises from the multiple-phase GPM. Many of the

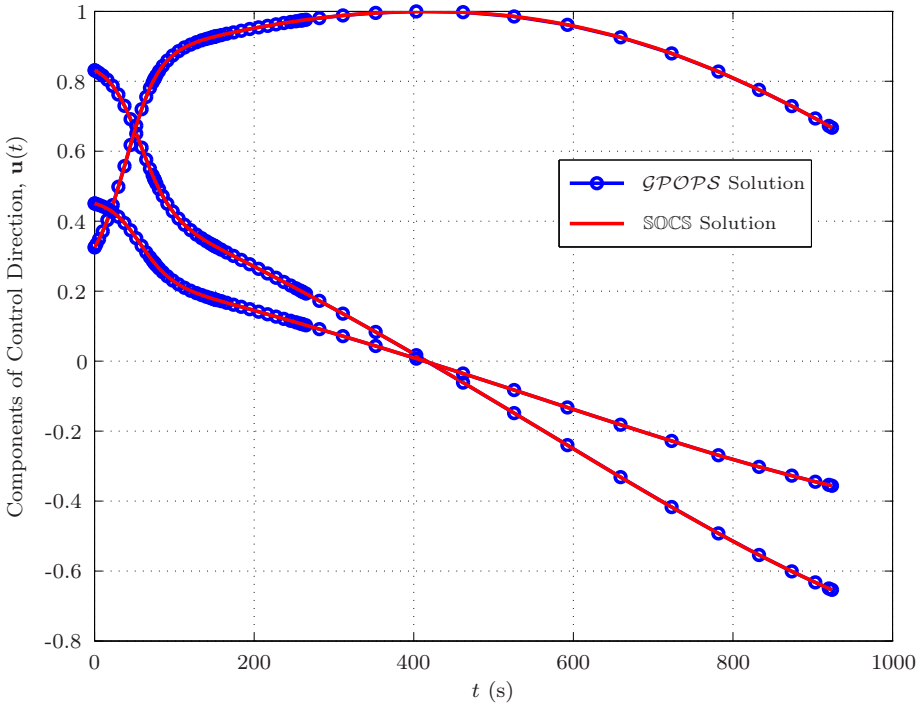

Fig. 11. Components of control vs. time for the launch vehicle ascent problem.

implementation details are subsequently discussed. The approach allows the user to design each phase of the optimal control problem independently and then connect any two phases via general linkage conditions on the state and time. In order to alleviate the burden of manually scaling the optimal control problem, an automatic scaling routine has been developed as part of the algorithm. Significant features of the algorithm include the separation of the constant derivatives of the constraint Jacobian from the non-constant derivatives, highly accurate computation of the costates (adjoints), and the use of automatic differentiation. A reusable MATLAB® software implementation, called $\mathcal{GPOPS}$, has been included with the algorithm developed and has been demonstrated successfully on two examples. It is found that the algorithm is highly capable of solving a wide range of complex optimal control problems and will be of long-term value to the optimal control community.

17. AVAILABILITY OF SOFTWARE

It is noted that the $\mathcal{GPOPS}$ implementation described in this article is available in the form of MATLAB® source code at no charge via download from either http://www.sourceforge.net/ or http://code.google.com/ by typing the project name $\mathcal{GPOPS}$ at the search prompt.

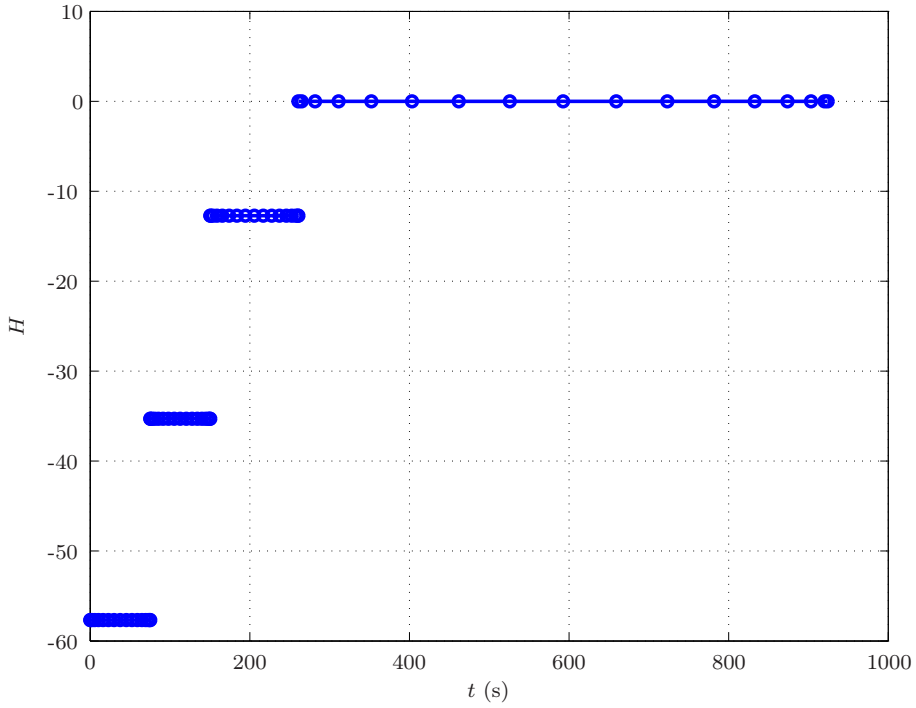

Fig. 12. Hamiltonian vs. time for the launch vehicle ascent problem ($\mathcal{GPOPS}$ solution only).

18. DISCLAIMER

The $\mathcal{GPOPS}$ implementation described in this paper is entirely original and has been developed utilizing only code that has been written by the authors. Furthermore, as stated in the article, the user interface to $\mathcal{GPOPS}$ is also entirely original, drawing either from earlier software implementations created by the authors or using MATLAB® language constructs that are not known to the authors to have ever been used before in the manner used in $\mathcal{GPOPS}$.

REFERENCES

Bate, R. R., Mueller, D. D., and White, J. E. 1971. *Fundamentals of Astrodynamics.* Dover Publications, New York.

Benson, D. A. 2004. A gauss pseudospectral transcription for optimal control. Ph.D. thesis, MIT.

Benson, D. A., Huntington, G. T., Thorvaldsen, T. P., and Rao, A. V. 2006. Direct trajectory optimization and costate estimation via an orthogonal collocation method. *Journal of Guidance, Control, and Dynamics 29,* 6 (November-December), 1435–1440.

Betts, J. T. 1998. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics 21,* 3 (September), 519–541.

Betts, J. T. 2001. *Practical Methods for Optimal Control Using Nonlinear Programming.* SIAM, Philadelphia, PA.

Betts, J. T. and Frank, P. D. 1994. A sparse nonlinear optimization algorithm. *Journal of Optimization Theory and Applications 82,* 2 (March-April), 193–207.

Betts, J. T. and Huffman, W. P. 1997. *Sparse Optimal Control Software - SOCS*, MEA-LR-085 ed. Boeing Information and Support Services, Seattle, Washington.

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

Byrd, R. H., Nocedal, J., and Waltz, R. 2006. *KNITRO: An Integrated Package for Nonlinear Optimization*. University of Colorado and Northwestern University, Boulder, Colorado and Evanston, Illinois.

Davis, P. and Rabinowitz, P. 1984. *Methods of Numerical Integration*. Academic Press, New York.

Elnagar, G. and Kazemi, M. 1998. Pseudospectral chebyshev optimal control of constrained nonlinear dynamical systems. *Computational Optimization and Applications 11,* 2, 195–217.

Elnagar, G., Kazemi, M., and Razzaghi, M. 1995. The pseudospectral legendre method for discretizing optimal control problems. *IEEE Transactions on Automatic Control 40,* 10, 1793–1796.

Fahroo, F. and Ross, I. M. 2000. A spectral patching method for direct trajectory optimization. *Journal of Astronautical Sciences 48,* 2-3 (Apr.-Sept.), 269–286.

Fahroo, F. and Ross, I. M. 2001. Costate estimation by a legendre pseudospectral method. *Journal of Guidance, Control, and Dynamics 24,* 2, 270–277.

Forth, S. A. 2006. An efficient overloaded implementation of forward mode automatic differentiation in matlab. *ACM Transactions on Mathematical Software 32,* 2, 195–222.

Forth, S. A. and Edvall, M. M. 2007. *User's Guide for MAD - a MATLAB Automatic Differentation Toolbox (TOMLAB/MAD) Version 1.4*. TOMLAB Optimization.

Gill, P. E., Murray, W., and Saunders, M. A. 2002. SNOPT: An sqp algorithm for large-scale constrained optimization. *SIAM Journal on Optimization 12,* 4, 979–1006.

Gill, P. E., Murray, W., and Saunders, M. A. 2006. *User's Guide for SNOPT Version 7: Software for Large Scale Nonlinear Programming*. Stanford University and University of California, San Diego.

Goh, C. J. and Teo, K. L. 1988. Miser: a fortran program for solving optimal control problems. *Advances in Engineering Software 10,* 2, 90–99.

Huntington, G. and Rao, A. V. 2007. A comparison of accuracy and computational efficiency of three pseudospectral methods. In *Proceedings of Guidance, Navigation and Control Conference*. American Institute of Aeronautics and Astrodynamics, Washington, DC.

Huntington, G. T. 2007. Advancement and analysis of a gauss pseudospectral transcription for optimal control. Ph.D. thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology.

Huntington, G. T., Benson, D. A., and Rao, A. V. 2007. Design of optimal tetrahedral spacecraft formations. *The Journal of the Astronautical Sciences 55,* 2 (April-June), 141–169.

Huntington, G. T. and Rao, A. V. 2008a. A comparison of global and local collocation methods for optimal control. *Journal of Guidance, Control, and Dynamics 31,* 2, 432–436.

Huntington, G. T. and Rao, A. V. 2008b. Optimal reconfiguration of spacecraft formations using the gauss pseudospecral method. *Journal of Guidance, Control, and Dynamics 31,* 3 (May-June), 689–698.

Jansch, C., Well, K. H., and Schnepper, K. 1994. *GESOP - Eine Software Umgebung Zur Simulation Und Optimierung*. Proceedings des SFB.

Kameswaran, S. and Biegler, L. T. 2006. Convergence rates for direct transcription of optimal control problems using collocation at radau points. *Computation Optimization and Applications Forthcoming*, TBD.

Kirk, D. E. 2004. *Optimal Control Theory: An Introduction*. Dover Publications, New York.

Martins, J. R. R., Sturdza, P., and Alonso, J. J. 2003. The complex-step derivative approximation. *ACM Transactions on Mathematical Software 29,* 3 (September), 245–262.

Rao, A. V. 2003. Extension of a pseudospectral legendre method for solving non-sequential multiple-phase optimal control problems. In AIAA Guidance, Navigation, and Control Con-

ference, *AIAA Paper 2003-5634*. Austin, Texas.

Ross, I. M. and Fahroo, F. 2001. *User's Manual for DIDO 2001 α: A MATLAB Application for Solving Optimal Control Problems*. Department of Aeronautics and Astronautics, Naval Postgraduate School, Technical Report AAS-01-03, Monterey, California.

Ross, I. M. and Fahroo, F. 2003. *Lecture Notes in Control and Information Sciences*. Springer-Verlag, Chapter Legendre Pseudospectral Approximations of Optimal Control Problems.

Ross, I. M. and Fahroo, F. 2004a. *Lecture Notes in Control and Information Sciences*. Vol. 295. Springer-Verlag, Heidelberg, Germany, Chapter Legendre Pseudospectral Approximations of Optimal Control Problems, 327–342.

Ross, I. M. and Fahroo, F. 2004b. Pseudospectral knotting methods for solving optimal control problems. *Journal of Guidance, Control, and Dynamics 27,* 3 (May-June), 397–405.

Ross, I. M. and Fahroo, F. 2008a. Advances in pseusospectral methods. *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2008-7309, Honolulu, Hawaii.

Ross, I. M. and Fahroo, F. 2008b. Convergence of the costates do not imply convergence of the controls. *Journal of Guidance, Control, and Dynamics 31,* 4, 1492–1497.

The Mathworks, Inc. 2008. *MATLAB: A Language for Technical Computing*. The Mathworks, Inc., Natick, MA.

Vlases, W. G., Paris, S. W., Martens, M. J., and Hargraves, C. R. 1990. Optimal trajectories by implicit simulation. Tech. rep., Boeing Aerospace and Electronics, WRDC-TR-90-3056.

von Stryk, O. 2000. *User's Guide for DIRCOL (Version 2.1): A Direct Collocation Method for the Numerical Solution of Optimal Control Problems*. Technical University, Munich, Germany.

Weideman, J. A. C. and Reddy, S. C. 2000. A matlab differentiation matrix suite. *ACM Transactions on Mathematical Software 26,* 4 (December), 465–519.

Williams, P. 2004a. Application of pseudospectral methods for receding horizon control. *Journal of Guidance, Control, and Dynamics 27,* 2, 310–314.

Williams, P. 2004b. Jacobi pseudospectral method for solving optimal control problems. *Journal of Guidance, Control, and Dynamics 27,* 2, 293–297.

Williams, P. 2005. Hermite-legendre-gauss-lobatto direct transcription methods in trajectory optimization. *Americal Astronautical Society*, Spaceflight Mechanics Meeting.

Williams, P. 2008. *User's Guide for DIRECT 2.0*. Royal Melbourne Institute of Technology, Melbourne, Australia.