

User's Manual for *GPOPS* Version 1.0:
**A MATLAB[®] Package for Dynamic Optimization Using
the Gauss Pseudospectral Method**

Anil V. Rao
University of Florida
Gainesville, FL 32607

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

Geoffrey T. Huntington
Blue Origin, LLC
Seattle, WA

Camila Francolin
Christopher L. Darby
Michael Patterson
University of Florida
Gainesville, FL 32607

THIS PAGE IS INTENTIONALLY LEFT BLANK

Acknowledgments

The software *GPOPS* was developed in response to a demand from the research and academic community for a customizable implementation of the Gauss pseudospectral method for solving optimal control problems. Indeed, while the authors of *GPOPS* have published extensively in the open literature on the theory and application of the Gauss pseudospectral method, no effort has been made to date to provide source code of actual software that can be utilized and adapted by a broad community. The *GPOPS* software is an attempt to fill that void and enable researchers, educators, and others involved in solving complex optimal control problems to take advantage of a code that can be customized to one's particular needs. The authors of *GPOPS* hope sincerely that the code is useful.

Disclaimer

This software is provided “as is” and free-of-charge. Neither the authors nor their employers assume any responsibility for any harm resulting from the use of this software. The authors do, however, hope that users will find this software useful for research and other purposes.

Preface to The *GPOPS* Software

It is noted that *GPOPS* has been designed to work with the nonlinear programming solver SNOPT (Gill, et. al., 2007). In addition, it has the option of using automatic differentiation for computing the objective function gradient and the constraint Jacobian. In particular, *GPOPS* is designed to work with the program *MAD* (MATLAB Automatic Differentiation) developed at Cranfield Institute of Technology and available for sale through TOMLABTM Optimization (URL:tomopt.com). While automatic differentiation is not required, it is highly recommended. Finally, the software can be adapted to work with other NLP solvers, but these implementation have not as of yet been developed.

Licensing Agreement

The software *GPOPS* is distributed under the following license agreement. Regardless of how you obtain a copy of *GPOPS*, you must abide by the following terms:

```
%-----%
%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!%
% THIS NOTICE MAY NOT BE DELETED UNDER CIRCUMSTANCES DURING ANY          %
% EXECUTION OF THE GPOPS SOFTWARE OR TRANSMITTAL OF GPOPS OR ANY          %
% DOCUMENTATION ASSOCIATED WITH GPOPS                                     %
%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!%
%-----%
% GPOPS Copyright (c) Anil V. Rao, Geoffrey T. Huntington, David            %
% Benson, Michael Patterson, and Christopher Darby.                       %
%-----%
%
%-----%
% PLEASE READ THE FOLLOWING LICENSE AGREEMENT CAREFULLY.                  %
%-----%
%
% By downloading, using, modifying, or distributing GPOPS, you            %
% to the terms of this license agreement. Your agreement with              %
% this license gives you extremely GENEROUS RIGHTS, so if you do           %
% not agree to the terms of this agreement, you may not proceed            %
% further with using, using, modifying, or distributing GPOPS,            %
%
% License for GPOPS Software (Based on the Simple Public License)          %
%
% This is a license for the software Gauss Pseudospectral Optimal          %
% Control Software (GPOPS). The license for GPOPS is based on the          %
% Simple Public License. In the same spirit as the Simple Public            %
% License, the language for the GPOPS License is similar to that          %
% of GPL 2.0. The words are different, but the goal is the same:          %
% to guarantee for all users the freedom to share and change              %
% software. If anyone wonders about the meaning of the GPOPS             %
% License, they should interpret it as consistent with GPL 2.0.           %
%
% LICENSE FOR GPOPS
% The GPOPS License applies to the software's source and object
% code and comes with any rights that I have in it (other than
% trademarks). You agree to the GPOPS License simply by
% downloading, copying, distributing, or making a derivative work
% of the software.
%
% You get the royalty-free right to:
```

```

%
%      * Use the software for any purpose;
%      * Make derivative works of it (this is called a
%      "Derived Work");
%      * Copy and distribute it and any Derived Work.
%
% If you distribute the software or a Derived Work, you must give
% back to the community by:
%
%      * Prominently noting the date of any changes you make;
%      * Leaving other people's copyright notices, warranty
%      disclaimers, and license terms in place;
%      * Providing the source code, build scripts, installation
%      scripts, and interface definitions in a form that is easy
%      to get and best to modify;
%      * Licensing it to everyone under SimPL, or substantially
%      similar terms (such as GPL 2.0), without adding further
%      restrictions to the rights provided;
%      * Conspicuously announcing that it is available under
%      that license.
%
% Restrictions for Use and Distribution of GPOPS:
%      * Unless you are one of the copyright holders, you may NOT
%      sell either the complete GPOPS software or any individual
%      files within the software
%      * The only exception to the sales restriction above is that
%      you may include the GPOPS software as a whole as part of
%      free operating system distributions (e.g., Linux)
%
% There are some things that you must shoulder:
%
%      * You get NO WARRANTIES. None of any kind;
%      * If the software damages you in any way, you may only
%      recover direct damages up to the amount you paid for it
%      (that is zero if you did not pay anything). You may not
%      recover any other damages, including those called
%      "consequential damages." (The state or country where you
%      live may not allow you to limit your liability in this
%      way, so this may not apply to you);
%
% The GPOPS License continues perpetually, except that your
% license rights end automatically if:
%
%      * You do not abide by the "give back to the community" terms
%      (your licensees get to keep their rights if they abide);
%      * Anyone prevents you from distributing the software under
%      the terms of the SimPL.
%      * You sell the software in any manner with the one exception
%      listed above.
%
% THE GPOPS LICENSE IS NOT THE SIMPL, BUT IS BASED ON THE SIMPL
%
% In addition, to the license given above, the following is a
% condition of using or modifying the GPOPS software in any manner:

```

0

0

Contents

1	Introduction to Gauss Pseudospectral Optimization Software (<i>GPOPS</i>)	10
1.1	Gauss Pseudospectral Method Employed by <i>GPOPS</i>	10
1.2	Gauss Pseudospectral Discretization of Continuous Bolza Problem	12
1.3	KKT Conditions of the NLP	13
1.4	First-Order Optimality Conditions of Continuous Bolza Problem	14
1.5	Gauss Pseudospectral Discretized Necessary Conditions	15
1.6	Costate Estimate	16
1.7	Computation of Boundary Controls	17
1.8	Organization of <i>GPOPS</i>	19
1.9	Key MATLAB Data Structures Used by <i>GPOPS</i>	19
1.10	Notation Used Throughout Remainder of This Manual	19
2	Constructing an Optimal Control Problem in <i>GPOPS</i>	20
2.1	Preliminary Information	20
2.2	Initialization of <i>GPOPS</i>	20
2.3	Call to <i>GPOPS</i>	21
2.4	Syntax for Setup Structure	21
2.5	Specifying Function Names Used in Optimal Control Problem	22
2.6	Syntax for limits Cell Array	22
2.7	Syntax for connections Cell Array	27
2.8	Syntax of Each Function in Optimal Control Problem	27
2.9	Specifying an Initial Guess of The Solution	32
2.10	Scaling of Optimal Control Problem	33
2.11	Different Options for Specification of Derivatives	33
2.12	Output of Execution of <i>GPOPS</i>	34
2.13	Useful Information for Debugging a <i>GPOPS</i> Problem	34
3	Examples of Using <i>GPOPS</i>	36
3.1	Hyper-Sensitive Problem	36
3.2	Bryson-Denham Problem	39
3.3	Multiple-Stage Launch Vehicle Ascent Problem	45
3.4	Minimum Time-to-Climb of a Supersonic Aircraft	56
3.5	Some Concluding Remarks	62

Chapter 1

Introduction to Gauss Pseudospectral Optimization Software (*GPOPS*)

Gauss Pseudospectral Optimization Software (*GPOPS*) is a software program written in MATLAB¹® for solving multiple-phase optimal control problems of the following form. Given a set of P phases (where $p = 1, \dots, P$), minimize the cost functional

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

subject to the dynamic constraint

$$\dot{\mathbf{x}}^{(p)} = \mathbf{f}^{(p)}(\mathbf{x}^{(p)}, \mathbf{u}^{(p)}, t; \mathbf{q}^{(p)}), \quad (p = 1, \dots, P), \quad (1-2)$$

the boundary conditions

$$\phi_{\min} \leq \phi^{(p)}(\mathbf{x}^{(p)}(t_0), t_0^{(p)}, \mathbf{x}^{(p)}(t_f), t_f^{(p)}; \mathbf{q}^{(p)}) \leq \phi_{\max}, \quad (p = 1, \dots, P), \quad (1-3)$$

the inequality path constraints

$$\mathbf{C}^{(p)}(\mathbf{x}^{(p)}(t), \mathbf{u}^{(p)}(t), t; \mathbf{q}^{(p)}) \leq \mathbf{0}, \quad (p = 1, \dots, P), \quad (1-4)$$

and the phase continuity (linkage) constraints

$$\mathbf{P}^{(s)}(\mathbf{x}^{(p_l^s)}(t_f), t_f^{(p_l^s)}; \mathbf{q}^{(p_l^s)}, \mathbf{x}^{(p_u^s)}(t_0), t_0^{(p_u^s)}; \mathbf{q}^{(p_u^s)}) = \mathbf{0}, \quad (p_l, p_u \in [1, \dots, P], s = 1, \dots, L) \quad (1-5)$$

where $\mathbf{x}^{(p)}(t) \in \mathbb{R}^{n_p}$, $\mathbf{u}^{(p)}(t) \in \mathbb{R}^{m_p}$, $\mathbf{q}^{(p)} \in \mathbb{R}^{q_p}$, and $t \in \mathbb{R}$ are, respectively, the state, control, static parameters, and time in phase $p \in [1, \dots, P]$, L is the number of phases to be linked, $p_l^s \in [1, \dots, P]$, ($s = 1, \dots, L$) are the “left” phase numbers, and $p_u^s \in [1, \dots, P]$, ($s = 1, \dots, L$) are the “right” phase numbers.

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). A schematic of how phases can potentially be linked is given in Fig. 1.1.

1.1 Gauss Pseudospectral Method Employed by *GPOPS*

The method employed by *GPOPS* is the *Gauss Pseudospectral Method* (GPM). The GPM is an orthogonal collocation method where the collocation points are the *Legendre-Gauss* points. The theory of the GPM can

¹MATLAB is a registered trademark of The Mathworks, Inc., One Apple Hill, Natick, MA

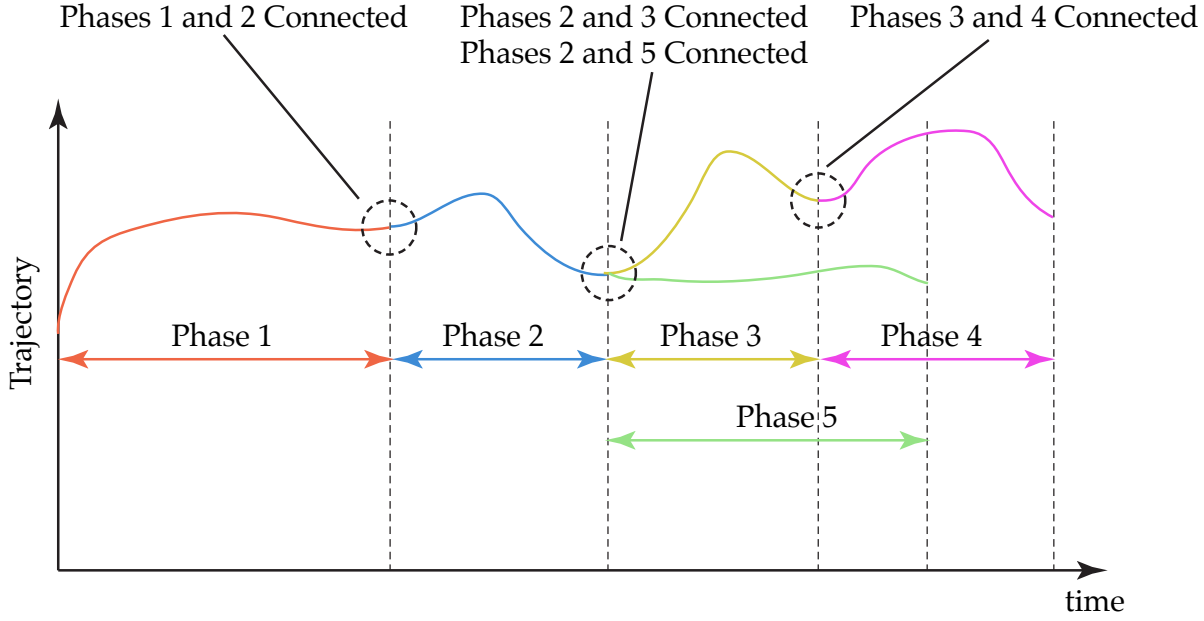


Figure 1.1 Schematic of linkages for multiple-phase optimal control problem. The example shown in the picture consists of five phases where the ends of phases 1, 2, and 3 are linked to the starts of phases 2, 3, and 4, respectively, while the end of phase 3 is linked to the start of phase 5.

be found in (Benson, 2004; Benson, et al., 2006) while applications of the GPM can be found in (Huntington, et al., 2005a; Huntington, et al., 2005b; Huntington, et al., 2005c; Huntington and Rao, 2007a). Strictly speaking, no knowledge of the GPM is required for using *GPOPS*. However, for completeness, in this section we describe the mathematics of the GPM. It is noted that this section is taken largely from Benson, et al. (2006).

1.1.1 Continuous Bolza Problem

Without loss of generality, consider the following optimal control problem. Determine the state, $\mathbf{x}(\tau) \in \mathbb{R}^n$, control, $\mathbf{u}(\tau) \in \mathbb{R}^m$, initial time, t_0 , and final time, t_f , that minimize the cost functional

$$J = \Phi(\mathbf{x}(-1), t_0, \mathbf{x}(1), t_f) + \frac{t_f - t_0}{2} \int_{-1}^1 g(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau; t_0, t_f) d\tau \quad (1-6)$$

subject to the constraints

$$\frac{d\mathbf{x}}{d\tau} = \frac{t_f - t_0}{2} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau; t_0, t_f) \quad (1-7)$$

$$\phi(\mathbf{x}(-1), t_0, \mathbf{x}(1), t_f) = \mathbf{0} \quad (1-8)$$

$$\mathbf{C}(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau; t_0, t_f) \leq \mathbf{0} \quad (1-9)$$

The optimal control problem of Eqs. (1-6)–(1-9) will be referred to as the *continuous Bolza problem*. It is noted that the optimal control problem of Eqs. (1-6)–(1-9) can be transformed from the time interval $\tau \in [-1, 1]$ to the time interval $t \in [t_0, t_f]$ via the affine transformation

$$t = \frac{t_f - t_0}{2} \tau + \frac{t_f + t_0}{2} \quad (1-10)$$

1.2 Gauss Pseudospectral Discretization of Continuous Bolza Problem

The direct approach to solving the continuous Bolza optimal control problem of Section 1.1.1 is to discretize and transcribe Eqs. (1–6)–(1–9) to a nonlinear programming problem (NLP). The Gauss pseudospectral method, like Legendre and Chebyshev methods, is based on approximating the state and control trajectories using interpolating polynomials. The state is approximated using a basis of $N + 1$ Lagrange interpolating polynomials, L ,

$$\mathbf{x}(\tau) \approx \mathbf{X}(\tau) = \sum_{i=0}^N \mathbf{X}(\tau_i) L_i(\tau) \quad (1-11)$$

where $L_i(\tau)$ ($i = 0, \dots, N$) are defined as

$$L_i(\tau) = \prod_{j=0, j \neq i}^N \frac{\tau - \tau_j}{\tau_i - \tau_j} \quad (1-12)$$

Additionally, the control is approximated using a basis of N Lagrange interpolating polynomials $L_i^*(\tau)$, ($i = 1, \dots, N$) as

$$\mathbf{u}(\tau) \approx \mathbf{U}(\tau) = \sum_{i=1}^N \mathbf{U}(\tau_i) L_i^*(\tau) \quad (1-13)$$

where

$$L_i^*(\tau) = \prod_{j=1, j \neq i}^N \frac{\tau - \tau_j}{\tau_i - \tau_j} \quad (1-14)$$

It can be seen from Eqs. (1–12) and (1–14) that $L_i(\tau)$ ($i = 0, \dots, N$) and $L_i^*(\tau)$ ($i = 1, \dots, N$) satisfy the properties

$$L_i(\tau_j) = \begin{cases} 1 & , \quad i = j \\ 0 & , \quad i \neq j \end{cases} \quad (1-15)$$

$$L_i^*(\tau_j) = \begin{cases} 1 & , \quad i = j \\ 0 & , \quad i \neq j \end{cases} \quad (1-16)$$

Differentiating the expression in Eq. (1–11), we obtain

$$\dot{\mathbf{x}}(\tau) \approx \dot{\mathbf{X}}(\tau) = \sum_{i=0}^N x(\tau_i) \dot{L}_i(\tau) \quad (1-17)$$

The derivative of each Lagrange polynomial at the LG points can be represented in a differential approximation matrix, $D \in \mathbb{R}^{N \times N+1}$. The elements of the differential approximation matrix are determined offline as follows:

$$D_{ki} = \dot{L}_i(\tau_k) = \sum_{l=0}^N \frac{\prod_{j=0, j \neq i, l}^N (\tau_k - \tau_j)}{\prod_{j=0, j \neq i}^N (\tau_i - \tau_j)} \quad (1-18)$$

where $k = 1, \dots, N$ and $i = 0, \dots, N$. The dynamic constraint is transcribed into algebraic constraints via the differential approximation matrix as follows:

$$\sum_{i=0}^N D_{ki} \mathbf{X}_i - \frac{t_f - t_0}{2} \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f) = \mathbf{0} \quad (k = 1, \dots, N) \quad (1-19)$$

where $\mathbf{X}_k \equiv \mathbf{X}(\tau_k) \in \mathbb{R}^n$ and $\mathbf{U}_k \equiv \mathbf{U}(\tau_k) \in \mathbb{R}^m$ ($k = 1, \dots, N$). Note that the dynamic constraint is collocated only at the LG points and *not* at the boundary points (this form of collocation differs from other well known pseudospectral methods such as those found in (Elnagar, et al., 1995) and (Elnagar and Kazemi, 1998)). Additional variables in the discretization are defined as follows: $\mathbf{X}_0 \equiv \mathbf{X}(-1)$, and \mathbf{X}_f , where \mathbf{X}_f is defined in terms of \mathbf{X}_k , ($k = 0, \dots, N$) and $\mathbf{U}(\tau_k)$ ($k = 1, \dots, N$) via the Gauss quadrature (Davis, 1975)

$$\mathbf{X}_f \equiv \mathbf{X}_0 + \frac{t_f - t_0}{2} \sum_{k=1}^N w_k \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f) \quad (1-20)$$

The continuous cost function of Eq. (1-6) is approximated using a Gauss quadrature (Davis, 1975) as

$$J = \Phi(\mathbf{X}_0, t_0, \mathbf{X}_f, t_f) + \frac{t_f - t_0}{2} \sum_{k=1}^N w_k g(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f) \quad (1-21)$$

where w_k are the Gauss weights. Next, the boundary constraint of Eq. (1-8) is expressed as

$$\phi(\mathbf{X}_0, t_0, \mathbf{X}_f, t_f) = \mathbf{0} \quad (1-22)$$

Furthermore, the path constraint of Eq. (1-9) is evaluated at the LG points as

$$\mathbf{C}(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f) \leq \mathbf{0} \quad (k = 1, \dots, N) \quad (1-23)$$

The cost function of Eq. (1-21) and the algebraic constraints of Eqs. (1-19), (1-20), (1-22), and (1-23) define an NLP whose solution is an approximate solution to the continuous Bolza problem. Finally, it is noted that the above discretization can be employed in multiple-phase problems by transcribing the problem in each phase using the above discretization and connecting the phases by *linkage* constraints (as described above).

1.3 KKT Conditions of the NLP

The first-order optimality conditions (i.e., the Karush-Kuhn-Tucker (KKT) conditions) of the NLP can be obtained using the augmented cost function or Lagrangian. The augmented cost function is formed using the Lagrange multipliers $\tilde{\mathbf{\Lambda}}_k \in \mathbb{R}^n$, $\tilde{\boldsymbol{\mu}}_k \in \mathbb{R}^c$, $k = 1, \dots, N$, $\tilde{\mathbf{\Lambda}}_F \in \mathbb{R}^n$, and $\tilde{\boldsymbol{\nu}} \in \mathbb{R}^q$ as

$$\begin{aligned} J_a = & \Phi(\mathbf{X}_0, t_0, \mathbf{X}_f, t_f) + \frac{t_f - t_0}{2} \sum_{k=1}^N w_k g(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f) - \tilde{\boldsymbol{\nu}}^T \phi(\mathbf{X}_0, t_0, \mathbf{X}_f, t_f) \\ & - \sum_{k=1}^N \tilde{\boldsymbol{\mu}}_k^T \mathbf{C}(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f) - \sum_{k=1}^N \tilde{\mathbf{\Lambda}}_k^T \left(\sum_{i=0}^N D_{ki} \mathbf{X}_i - \frac{t_f - t_0}{2} \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f) \right) \\ & - \tilde{\mathbf{\Lambda}}_F^T \left(\mathbf{X}_f - \mathbf{X}_0 - \frac{t_f - t_0}{2} \sum_{k=1}^N w_k \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f) \right) \end{aligned} \quad (1-24)$$

The KKT conditions are found by setting equal to zero the derivatives of the Lagrangian with respect to \mathbf{X}_0 , \mathbf{X}_k , \mathbf{X}_f , \mathbf{U}_k , $\tilde{\mathbf{\Lambda}}_k$, $\tilde{\boldsymbol{\mu}}_k$, $\tilde{\mathbf{\Lambda}}_F$, $\tilde{\boldsymbol{\nu}}$, t_0 , and t_f . The solution to the NLP of Section (1.2) must satisfy the following KKT conditions:

$$\sum_{i=0}^N \mathbf{X}_i D_{ki} = \frac{t_f - t_0}{2} \mathbf{f}_k \quad (1-25)$$

$$\begin{aligned} & \sum_{i=1}^N \left(\frac{\tilde{\mathbf{\Lambda}}_i^T}{w_i} + \tilde{\mathbf{\Lambda}}_F^T \right) D_{ki}^\dagger + \tilde{\mathbf{\Lambda}}_F^T D_{kN+1}^\dagger = \\ & \frac{t_f - t_0}{2} \left(-\frac{\partial g_k}{\partial \mathbf{X}_k} - \left(\frac{\tilde{\mathbf{\Lambda}}_k^T}{w_k} + \tilde{\mathbf{\Lambda}}_F^T \right) \frac{\partial \mathbf{f}_k}{\partial \mathbf{X}_k} + \frac{2}{t_f - t_0} \frac{\tilde{\boldsymbol{\mu}}_k^T}{w_k} \frac{\partial \mathbf{C}_k}{\partial \mathbf{X}_k} \right) \end{aligned} \quad (1-26)$$

$$\mathbf{0} = \frac{\partial g_k}{\partial \mathbf{U}_k} + \left(\frac{\tilde{\Lambda}_k^T}{w_k} + \tilde{\Lambda}_F^T \right) \frac{\partial \mathbf{f}_k}{\partial \mathbf{U}_k} - \frac{2}{t_f - t_0} \frac{\tilde{\mu}_k^T}{w_k} \frac{\partial \mathbf{C}_k}{\partial \mathbf{U}_k} \quad (1-27)$$

$$\phi(\mathbf{X}_0, t_0, \mathbf{X}_f, t_f) = \mathbf{0} \quad (1-28)$$

$$\tilde{\Lambda}_0^T = -\frac{\partial \Phi}{\partial \mathbf{X}_0} + \tilde{\nu}^T \frac{\partial \phi}{\partial \mathbf{X}_0} \quad (1-29)$$

$$\tilde{\Lambda}_F^T = \frac{\partial \Phi}{\partial \mathbf{X}_f} - \tilde{\nu}^T \frac{\partial \phi}{\partial \mathbf{X}_f} \quad (1-30)$$

$$-\frac{t_f - t_0}{2} \sum_{k=1}^N w_k \frac{\partial \tilde{\mathcal{H}}_k}{\partial t_0} + \frac{1}{2} \sum_{k=1}^N w_k \tilde{\mathcal{H}}_k = \frac{\partial \Phi}{\partial t_0} - \tilde{\nu}^T \frac{\partial \phi}{\partial t_0} \quad (1-31)$$

$$\frac{t_f - t_0}{2} \sum_{k=1}^N w_k \frac{\partial \tilde{\mathcal{H}}_k}{\partial t_f} + \frac{1}{2} \sum_{k=1}^N w_k \tilde{\mathcal{H}}_k = -\frac{\partial \Phi}{\partial t_f} + \tilde{\nu}^T \frac{\partial \phi}{\partial t_f} \quad (1-32)$$

$$\mathbf{C}_k \leq \mathbf{0} \quad (1-33)$$

$$\tilde{\mu}_{jk} = 0, \text{ when } C_{jk} < 0 \quad (1-34)$$

$$\tilde{\mu}_{jk} \leq 0, \text{ when } C_{jk} = 0 \quad (1-35)$$

$$\mathbf{X}_f = \mathbf{X}_0 + \frac{(t_f - t_0)}{2} \sum_{k=1}^N w_k \mathbf{f}_k \quad (1-36)$$

$$\tilde{\Lambda}_F = \tilde{\Lambda}_0 + \frac{t_f - t_0}{2} \sum_{k=1}^N w_k \left(-\frac{\partial g_k}{\partial \mathbf{X}_k} - \left(\frac{\tilde{\Lambda}_k^T}{w_k} + \tilde{\Lambda}_F^T \right) \frac{\partial \mathbf{f}_k}{\partial \mathbf{X}_k} + \frac{2}{t_f - t_0} \frac{\tilde{\mu}_k^T}{w_k} \frac{\partial \mathbf{C}_k}{\partial \mathbf{X}_k} \right) \quad (1-37)$$

where the shorthand notation $g_k \equiv g(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f)$, $\mathbf{f}_k \equiv \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f)$, $\mathcal{H}_k \equiv \mathcal{H}(\mathbf{X}_k, \mathbf{A}_k, \boldsymbol{\mu}_k, \mathbf{U}_k, \tau_k; t_0, t_f)$, and $C_{jk} \equiv C_j(\mathbf{X}_k, \mathbf{U}_k, \tau_k; t_0, t_f)$ is used. Note that the augmented Hamiltonian, $\tilde{\mathcal{H}}_k$, is defined as

$$\tilde{\mathcal{H}}_k \equiv g_k + \left(\frac{\tilde{\Lambda}_k^T}{w_k} + \tilde{\Lambda}_F^T \right) \mathbf{f}_k - \frac{2}{t_f - t_0} \frac{\tilde{\mu}_k^T}{w_k} \mathbf{C}_k \quad (1-38)$$

and $\tilde{\Lambda}_0$ is defined as

$$\tilde{\Lambda}_0^T \equiv -\frac{\partial \Phi}{\partial \mathbf{X}_0} + \tilde{\nu}^T \frac{\partial \phi}{\partial \mathbf{X}_0} \quad (1-39)$$

1.4 First-Order Optimality Conditions of Continuous Bolza Problem

The indirect approach to solving the continuous Bolza problem of Eqs. (1-6)–(1-9) in Section 1.1.1 is to apply the calculus of variations and Pontryagin's Maximum Principle (Pontryagin, et al., 1962) to obtain first-order necessary conditions for optimality (Kirk, 1970). These variational conditions are typically derived using the augmented Hamiltonian, \mathcal{H} , defined as

$$\mathcal{H}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{u}, \tau; t_0, t_f) = g(\mathbf{x}, \mathbf{u}, \tau; t_0, t_f) + \boldsymbol{\lambda}^T(\tau) \mathbf{f}(\mathbf{x}, \mathbf{u}, \tau; t_0, t_f) - \boldsymbol{\mu}^T(\tau) \mathbf{C}(\mathbf{x}, \mathbf{u}, \tau; t_0, t_f) \quad (1-40)$$

where $\lambda(\tau) \in \mathbb{R}^n$ is the costate and $\mu(\tau) \in \mathbb{R}^c$ is the Lagrange multiplier associated with the path constraint. The continuous-time first-order optimality conditions can be shown to be

$$\begin{aligned}
\frac{d\mathbf{x}}{d\tau} &= \frac{t_f - t_0}{2} \mathbf{f}(\mathbf{x}, \mathbf{u}, \tau; t_0, t_f) = \frac{t_f - t_0}{2} \frac{\partial \mathcal{H}}{\partial \lambda} \\
\frac{d\lambda}{d\tau} &= \frac{t_f - t_0}{2} \left(-\frac{\partial g}{\partial \mathbf{x}} - \lambda^T \frac{\partial \mathbf{f}}{\partial \mathbf{x}} + \mu^T \frac{\partial \mathbf{C}}{\partial \mathbf{x}} \right) = -\frac{t_f - t_0}{2} \frac{\partial \mathcal{H}}{\partial \lambda} \\
\mathbf{0} &= \frac{\partial g}{\partial \mathbf{u}} + \lambda^T \frac{\partial \mathbf{f}}{\partial \mathbf{u}} - \mu^T \frac{\partial \mathbf{C}}{\partial \mathbf{u}} = \frac{\partial \mathcal{H}}{\partial \mathbf{u}} \\
\phi(\mathbf{x}(\tau_0), t_0, \mathbf{x}(\tau_f), t_f) &= \mathbf{0} \\
\lambda(\tau_0) &= -\frac{\partial \Phi}{\partial \mathbf{x}(\tau_0)} + \nu^T \frac{\partial \phi}{\partial \mathbf{x}(\tau_0)}, \quad \lambda(\tau_f) = \frac{\partial \Phi}{\partial \mathbf{x}(\tau_f)} - \nu^T \frac{\partial \phi}{\partial \mathbf{x}(\tau_f)} \\
\mathcal{H}(t_0) &= \frac{\partial \Phi}{\partial t_0} - \nu^T \frac{\partial \phi}{\partial t_0}, \quad \mathcal{H}(t_f) = -\frac{\partial \Phi}{\partial t_f} + \nu^T \frac{\partial \phi}{\partial t_f} \\
\mu_j(\tau) &= 0, \text{ when } C_j(\mathbf{x}, \mathbf{u}, \tau; t_0, t_f) < 0, \quad j = 1, \dots, c \\
\mu_j(\tau) &\leq 0, \text{ when } C_j(\mathbf{x}, \mathbf{u}, \tau; t_0, t_f) = 0, \quad j = 1, \dots, c
\end{aligned} \tag{1-41}$$

where $\nu \in \mathbb{R}^q$ is Lagrange multiplier associated with the boundary condition ϕ . It can be shown that the augmented Hamiltonian at the initial and final times can be written, respectively, as

$$\mathcal{H}(t_0) = -\frac{t_f - t_0}{2} \int_{-1}^1 \frac{\partial \mathcal{H}}{\partial t_0} d\tau + \frac{1}{2} \int_{-1}^1 \mathcal{H} d\tau \tag{1-42}$$

$$\mathcal{H}(t_f) = \frac{t_f - t_0}{2} \int_{-1}^1 \frac{\partial \mathcal{H}}{\partial t_f} d\tau + \frac{1}{2} \int_{-1}^1 \mathcal{H} d\tau \tag{1-43}$$

1.5 Gauss Pseudospectral Discretized Necessary Conditions

In order to discretize the variational conditions of Section (1.4) using the Gauss pseudospectral discretization, it is necessary to form an appropriate approximation for the costate. In this method, the costate, $\lambda(\tau)$, is approximated as follows:

$$\lambda(\tau) \approx \Lambda(\tau) = \sum_{i=1}^{N+1} \lambda(\tau_i) L_i^\dagger(\tau) \tag{1-44}$$

where $L_i^\dagger(\tau)$ ($i = 1, \dots, N + 1$) are defined as

$$L_i^\dagger(\tau) = \prod_{j=1, j \neq i}^{N+1} \frac{\tau - \tau_j}{\tau_i - \tau_j} \tag{1-45}$$

It is emphasized that the costate approximation is *different* from the state approximation. In particular, the basis of $N + 1$ Lagrange interpolating polynomials $L_i^\dagger(\tau)$ ($i = 1, \dots, N + 1$) includes the costate at the *final* time (as opposed to the initial time which is used in the state approximation). This (non-intuitive) costate approximation is necessary in order to provide a complete mapping between the KKT conditions and the variational conditions.

Using the costate approximation of Eq. (1-44), The first-order necessary conditions of the continuous Bolza problem in Eq. (1-41) are discretized as follows. First, the state and control are approximated using Eqs. (1-11) and (1-13), respectively. Next, the costate is approximated using the basis of $N + 1$ Lagrange interpolating polynomials as defined in Eq. (1-44). The continuous-time first-order optimality conditions of Eq. (1-41) are discretized using the variables $\mathbf{X}_0 \equiv \mathbf{X}(-1)$, $\mathbf{X}_k \equiv \mathbf{X}(\tau_k) \in \mathbb{R}^n$, and $\mathbf{X}_f \equiv \mathbf{X}(1)$ for the state, $\mathbf{U}_k \equiv \mathbf{U}(\tau_k) \in \mathbb{R}^m$ for the control, $\Lambda_0 \equiv \Lambda(-1)$, $\Lambda_k \equiv \Lambda(\tau_k) \in \mathbb{R}^n$, and $\Lambda_f \equiv \Lambda(1)$ for the costate, and $\mu_k \equiv \mu(\tau_k) \in \mathbb{R}^c$, for the Lagrange multiplier associated with the path constraints at the LG points

$k = 1, \dots, N$. The other unknown variables in the problem are the initial and final times, $t_0 \in \mathbb{R}$, $t_f \in \mathbb{R}$, and the Lagrange multiplier, $\boldsymbol{\nu} \in \mathbb{R}^q$. The total number of variables is then given as $(2n + m + c)N + 4n + q + 2$. These variables are used to discretize the continuous necessary conditions of Eq. (1–41) via the Gauss pseudospectral discretization. Note that the derivative of the state is approximated using Lagrange polynomials based on $N+1$ points consisting of the N LG points and the initial time, τ_0 , while the derivative of the costate is approximated using Lagrange polynomials based on $N+1$ points consisting of the N LG points and the final time, τ_f . The resulting algebraic equations that approximate the continuous necessary conditions at the LG points are given as

$$\sum_{i=0}^N \mathbf{X}_i D_{ki} = \frac{t_f - t_0}{2} \mathbf{f}_k \quad (1-46)$$

$$\sum_{i=1}^N \Lambda_i D_{ki}^\dagger + \Lambda_f D_{kN+1}^\dagger = \frac{t_f - t_0}{2} \left(-\frac{\partial g_k}{\partial \mathbf{X}_k} - \Lambda_k^T \frac{\partial \mathbf{f}_k}{\partial \mathbf{X}_k} + \boldsymbol{\mu}_k^T \frac{\partial \mathbf{C}_k}{\partial \mathbf{X}_k} \right) \quad (1-47)$$

$$\mathbf{0} = \frac{\partial g_k}{\partial \mathbf{U}_k} + \Lambda_k^T \frac{\partial \mathbf{f}_k}{\partial \mathbf{U}_k} - \boldsymbol{\mu}_k^T \frac{\partial \mathbf{C}_k}{\partial \mathbf{U}_k} \quad (1-48)$$

$$\phi(\mathbf{X}_0, t_0, \mathbf{X}_f, t_f) = \mathbf{0} \quad (1-49)$$

$$\Lambda_0 = -\frac{\partial \Phi}{\partial \mathbf{X}_0} + \boldsymbol{\nu}^T \frac{\partial \phi}{\partial \mathbf{X}_0} \quad (1-50)$$

$$\Lambda_f = \frac{\partial \Phi}{\partial \mathbf{X}_f} - \boldsymbol{\nu}^T \frac{\partial \phi}{\partial \mathbf{X}_f} \quad (1-51)$$

$$-\frac{t_f - t_0}{2} \sum_{k=1}^N w_k \frac{\partial \mathcal{H}_k}{\partial t_0} + \frac{1}{2} \sum_{k=1}^N w_k \mathcal{H}_k = \frac{\partial \Phi}{\partial t_0} - \boldsymbol{\nu}^T \frac{\partial \phi}{\partial t_0} \quad (1-52)$$

$$\frac{t_f - t_0}{2} \sum_{k=1}^N w_k \frac{\partial \mathcal{H}_k}{\partial t_f} + \frac{1}{2} \sum_{k=1}^N w_k \mathcal{H}_k = -\frac{\partial \Phi}{\partial t_f} + \boldsymbol{\nu}^T \frac{\partial \phi}{\partial t_f} \quad (1-53)$$

$$\mu_{jk} = 0, \text{ when } C_{jk} < 0 \quad (1-54)$$

$$\mu_{jk} \leq 0, \text{ when } C_{jk} = 0 \quad (1-55)$$

for $k = 1, \dots, N$ and $j = 1, \dots, c$. The final two equations that are required (in order to link the initial and final state and costate, respectively) are

$$\mathbf{X}_f = \mathbf{X}_0 + \frac{t_f - t_0}{2} \sum_{k=1}^N w_k \mathbf{f}_k \quad (1-56)$$

$$\Lambda_f = \Lambda_0 + \frac{t_f - t_0}{2} \sum_{k=1}^N w_k \left(-\frac{\partial g_k}{\partial \mathbf{X}_k} - \Lambda_k^T \frac{\partial \mathbf{f}_k}{\partial \mathbf{X}_k} + \boldsymbol{\mu}_k^T \frac{\partial \mathbf{C}_k}{\partial \mathbf{X}_k} \right) \quad (1-57)$$

The total number of equations in set of discrete necessary conditions of Eqs. (1–46)–(1–57) is $(2n + m + c)N + 4n + q + 2$ (the same number of unknown variables). Solving these nonlinear algebraic equations would be an indirect solution to the optimal control problem.

1.6 Costate Estimate

One of the key features of the Gauss pseudospectral method is the ability to map the KKT multipliers of the NLP to the costates of the continuous-time optimal control problem. In particular, using the results of Sections 1.3 and 1.5, a costate estimate for the continuous Bolza problem can be obtained at the Legendre-Gauss points and the boundary points. This costate estimate is taken from (Benson, 2004) and is summarized below via the *Gauss Pseudospectral Costate Mapping Theorem*:

Theorem 1 (Gauss Pseudospectral Costate Mapping Theorem) *The Karush-Kuhn-Tucker (KKT) conditions of the NLP are exactly equivalent to the discretized form of the continuous first-order necessary conditions of the continuous Bolza problem when using the Gauss pseudospectral discretization. Furthermore, a costate estimate at the initial time, final time, and the Legendre-Gauss points can be found from the KKT multipliers, $\tilde{\Lambda}_k$, $\tilde{\mu}_k$, $\tilde{\Lambda}_F$, and $\tilde{\nu}$,*

$$\Lambda_k = \frac{\tilde{\Lambda}_k}{w_k} + \tilde{\Lambda}_F, \quad \mu_k = \frac{2}{t_f - t_0} \frac{\tilde{\mu}_k}{w_k}, \quad \nu = \tilde{\nu} \quad (1-58)$$

$$\Lambda(t_0) = \tilde{\Lambda}_0, \quad \Lambda(t_f) = \tilde{\Lambda}_F$$

Proof of Theorem 1 Using the substitution of Eq. (1-58), it is seen that Eqs. (1-25)–(1-37) are exactly the same as Eqs. (1-46)–(1-57).

Theorem 1 indicates that solving the NLP derived from the Gauss pseudospectral transcription of the optimal control problem is equivalent to applying the Gauss pseudospectral discretization to the continuous-time variational conditions. Fig. 1.2 shows the solution path for both the direct and indirect methods.

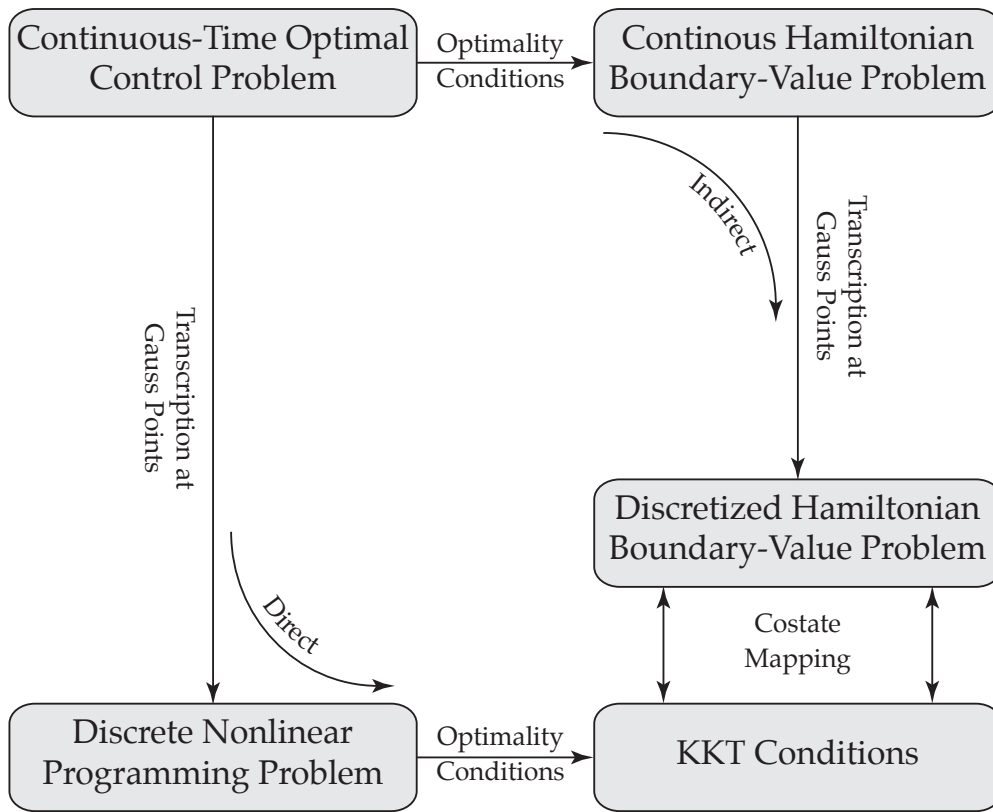


Figure 1.2 Equivalence of indirect and direct forms using the Gauss pseudospectral discretization.

1.7 Computation of Boundary Controls

It is seen in the GPM that the control is discretized only at the LG points and is *not* discretized at either the initial or the terminal point. Consequently, the solution of the NLP defined by Eqs. (1-19), (1-20), (1-21), (1-22), and (1-23) does not produce values of the controls at the boundaries. The ability to obtain accurate initial and terminal controls can be important in many applications, particularly in guidance where real-time computation of the initial control is of vital interest.

At first glance, it may seem that the lack of control information at the boundaries can be overcome simply via extrapolation of the control at the LG points. However, multiple reasons exist as to why this is not the best approach. First, no particular functional form for the control is assumed in the GPM discretization. As a result, the best function to use for extrapolation is ambiguous. Second, any reasonable extrapolation of the control (e.g., linear, quadratic, cubic, or spline) may violate a path constraint which, in general, will render the extrapolated control infeasible. Third, even if the extrapolated control is feasible, it will not satisfy the required optimality conditions at the boundaries (i.e., the control will be suboptimal with respect to the NLP). Consequently, it is both practical and most rigorous to develop a systematic procedure to compute the boundary controls. We now show how to compute the boundary controls from the primal and dual solutions of the NLP arising from the Gauss pseudospectral method. It is noted that the algorithm described in this section is taken from Huntington, et. al. (2007b)

Because the approach for computing the initial and terminal control is identical, we focus on the computation of the initial control. First, recalling the augmented Hamiltonian, $\tilde{\mathcal{H}}_a$, for the continuous-time optimal control problem, we have

$$\tilde{\mathcal{H}}_a(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \equiv g + \boldsymbol{\lambda}^T \mathbf{f} - \boldsymbol{\mu}^T \mathbf{C} \quad (1-59)$$

where shorthand notation is used. Now recall that, from the minimum principle of Pontryagin, at every instant of time the optimal control is the control $\mathbf{u}^*(\tau) \in \mathcal{U}$ that satisfies the condition

$$\tilde{\mathcal{H}}_a(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \leq \tilde{\mathcal{H}}_a(\mathbf{x}^*, \mathbf{u}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \quad (1-60)$$

where \mathcal{U} is the feasible control set. Consequently, for a given instant of time τ where $\mathbf{x}^*(\tau)$, $\boldsymbol{\lambda}^*(\tau)$, and $\boldsymbol{\mu}^*(\tau)$ are known, Eq. (1-60) is a constrained optimization problem in the $\mathbf{u}(\tau) \in \mathbb{R}^m$. In order to solve this constrained optimization problem at the *initial* time, it is necessary to know $\mathbf{x}^*(\tau_0)$, $\boldsymbol{\lambda}^*(\tau_0)$, and $\boldsymbol{\mu}^*(\tau_0)$.

Consider now the information that can be obtained by solving the NLP associated with the GPM. In particular, the primal solution to the NLP produces $\mathbf{X}(\tau_0)$ while the dual solution to the NLP can be manipulated algebraically to obtain the initial costate, $\boldsymbol{\Lambda}^*(\tau_0)$. However, because the NLP does not evaluate that path constraint at the boundaries, there is no associated Lagrange multiplier $\tilde{\boldsymbol{\mu}}(\tau_0)$. This apparent impediment can be overcome by applying the minimum principle in a manner somewhat different from that given in Eq. (1-60). In particular, suppose we let \mathcal{H} be the *Hamiltonian* (not the augmented Hamiltonian), where \mathcal{H} is defined as

$$\mathcal{H}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) \equiv g + \boldsymbol{\lambda}^T \mathbf{f} \quad (1-61)$$

It is seen in Eq. (1-61) that the term involving the path constraint is not included. The path constraint is instead incorporated into the feasible control set. In particular, suppose we let \mathcal{V}_0

$$\mathcal{V}_0 = \mathcal{U} \cap \mathcal{C}_0 \quad (1-62)$$

where \mathcal{V}_0 is the intersection of the original set of feasible controls at time τ_0 , denoted \mathcal{U} , with the set of all controls at time τ_0 that satisfy the inequality constraint of Eq. (1-23), denoted \mathcal{C}_0 . Then, using the values of $\mathbf{X}(\tau_0)$ and $\boldsymbol{\Lambda}(\tau_0)$, the following modified optimization problem in m variables $\mathbf{U}(\tau_0) \in \mathbb{R}^m$ can be solved to obtain the initial control, $\mathbf{U}(\tau_0)$:

$$\begin{aligned} & \text{minimize} && \mathcal{H}(\mathbf{X}(\tau_0), \mathbf{U}(\tau_0), \boldsymbol{\Lambda}(\tau_0), \tau_0; t_0, t_f) \\ & \mathbf{U}(\tau_0) \in \mathcal{V}_0 \end{aligned} \quad (1-63)$$

It is noted that, because \mathcal{V}_0 is restricted by the inequality path constraint at τ_0 , the solution of $\mathbf{U}(\tau_0)$ is equivalent to the solution of the following problem:

$$\begin{aligned} & \text{minimize} && \mathcal{H}(\mathbf{X}(\tau_0), \mathbf{U}(\tau_0), \boldsymbol{\Lambda}(\tau_0), \tau_0; t_0, t_f) \\ & \mathbf{U}(\tau_0) \in \mathcal{U} \\ & \text{subject to} && \mathbf{C}(\mathbf{X}(\tau_0), \mathbf{U}(\tau_0), \tau_0; t_0, t_f) \leq \mathbf{0} \end{aligned} \quad (1-64)$$

Interestingly, if the constraint is *active*, then the initial path constraint multiplier, $\tilde{\mu}(\tau_0)$, will also be determined by the minimization problem of Eq. (1–64). Finally, as alluded to above, the control at the terminal time, $\mathbf{U}(\tau_f)$, can be obtained by solving the minimization problem of Eq. (1–64) at $\tau = \tau_f$, i.e.,

$$\begin{aligned} & \text{minimize} && \mathcal{H}(\mathbf{X}(\tau_f), \mathbf{U}(\tau_f), \mathbf{\Lambda}(\tau_f), \tau_f; t_0, t_f) \\ & \mathbf{U}(t_f) \in \mathcal{U} \\ & \text{subject to} && \mathbf{C}(\mathbf{X}(\tau_f), \mathbf{U}(\tau_f), \tau_f; t_0, t_f) \leq \mathbf{0} \end{aligned} \tag{1-65}$$

1.8 Organization of *GPOPS*

GPOPS is organized as follows. In order to specify the optimal control problem that is to be solved, the user must write MATLAB functions that define the following functions in each phase of the problem:

- (1) the cost functional
- (2) the right-hand side of the differential equations and the path constraints(i.e., the differential-algebraic equations)
- (3) the boundary conditions (i.e., event conditions)
- (4) the path constraints

In addition, the user must write a MATLAB function that describes how the phases are connected (i.e., a linkage function). The user must also specify the lower and upper limits on every component of the following quantities:

- (1) the state at the following points in time:
 - at the beginning of the phase
 - during the phase
 - at the end of the phase
- (2) the control
- (3) the initial and terminal boundary conditions
- (4) the path constraints

It is noted that each of the functions must be defined for each phase of the problem. Finally, the user must also specify the the upper and lower bounds on the phase-connect (i.e., linkage) conditions. The remainder of this document is devoted to describing in detail the MATLAB[®] syntax for describing the optimal control problem and each of the constituent functions.

1.9 Key MATLAB Data Structures Used by *GPOPS*

Because *GPOPS* is a MATLAB software, it takes advantage of many of the key data structures that are built in to the MATLAB software. In particular, a key data structure used throughout the *GPOPS* software is that of a cell array. A detailed discussion of cell arrays is beyond the scope of this manual. The user is urged to read the relevant documentation on cell arrays posted on the The Mathworks, Inc., website at the following URL: www.mathworks.com.

1.10 Notation Used Throughout Remainder of This Manual

The following notation is adopted for use throughout the remainder of this manual. First, all user-specified names will be denoted by *slanted* characters (not *italic*, but *slanted*). Second, any item denoted by **boldface characters** are pre-defined and cannot be changed by the user. Finally, users with color capability will see the slanted characters in *red* and will see the boldface characters in *blue*.

Chapter 2

Constructing an Optimal Control Problem in *GPOPS*

We now proceed to describe the constructs required to specify an optimal control problem in *GPOPS*. We note once again that the key MATLAB data structure used in constructing an optimal control problem is the cell array.

2.1 Preliminary Information

Before proceeding to the details of setting up a problem in *GPOPS*, the following few preliminary details are useful. First, it is important to understand that the *GPOPS* interface is laid out in *phases*. Using a phase-based approach, it is possible to describe each segment of the problem independently of the other segments. The segments are then *linked* together using linkage conditions (or phase-connect conditions). Second, it is important to note that *GPOPS* uses the vectorization capabilities of MATLAB. In this vein all matrices and vectors in *GPOPS* are oriented *column-wise* for maximum efficiency. As you read through this chapter, please keep in mind the column-wise orientation of all matrices used in *GPOPS*.

2.2 Initialization of *GPOPS*

Before beginning to set up any problem in *GPOPS*, a call to the function `gpopsInitialize` will make subsequent coding easier. Simply put the line `gpopsInitialize` at the top of your program (after, of course, any instantiations of “clear” or “clear all”). Calling `gpopsInitialize` will put into MATLAB memory the variables shown in Table 2.1

Table 2.1 Table of integers placed in MATLAB workspace from call to function `gpopsInitialize`.

Index	Value	Meaning of Cell Array Index
<code>imin</code>	1	Lower Bound
<code>imax</code>	2	Upper Bound
<code>itime</code>	1	Time
<code>istate</code>	2	State
<code>icontrol</code>	3	Control
<code>iparameter</code>	4	Static Parameters
<code>ipath</code>	5	Path Constraint
<code>ievent</code>	6	Event Constraint
<code>iduration</code>	7	Phase Duration
<code>icostate</code>	5	Costate
<code>ihamilt</code>	7	Hamiltonian
<code>imayer</code>	8	Mayer Cost
<code>ilagrange</code>	9	Lagrange Cost
<code>iconnect</code>	3	Phase-Connect (Linkage) Constraint
<code>fcost</code>	1	Cost Functional
<code>fdae</code>	2	Differential-Algebraic Equation Function
<code>fevent</code>	3	Event Constraint Function
<code>fconnect</code>	4	Phase-Connect (Linkage) Function

The user need not be concerned about the actual *values* of the various integers above, only that the integers correspond to different elements in the various cell arrays that contain the lower and upper bounds on the variables and constraints and the different parts of the solution. As will be seen, the integers allocated in the MATLAB workspace through the function `gpopsInitialize` make subsequent coding easier and more intuitive. Thus, rather than referring to cell array elements by number, these elements will be referred to using the integer variables given above. It will be seen in reading the remainder of the manual how the function `gpopsInitialize` aids the user in setting up the problem.

2.3 Call to *GPOPS*

The call to *GPOPS* is deceptively simple and is given as follows:

output=gpops(setup)

The input *setup* is a user-defined structure that contains all of the information about the optimal control problem to be solved¹. Finally, the variable *output* is a structure that contains all of the information from the original problem *plus* the information from the run itself (i.e., the solution)². The remainder of this chapter is devoted to describing the fields in the structure *setup*.

2.4 Syntax for Setup Structure

The user-defined structure *setup* contains required fields and optional fields. The required fields in the structure *problem* are as follows:

- **name**: a string containing the name of the problem.
- **funcs**: a cell array whose elements contain the names of the user-defined function in the problem.
- **direction**: a string that indicates the direction of the independent variable. The two possible values for this string are “increasing” and “decreasing”.

¹see the detailed description of *problem* in Section 2.4

²See the detailed description of the output in Section 2.12.

- **nodes**: a row vector of length P (where P is the number of phases) containing integers that specify the number of nodes (i.e., the number of Legendre-Gauss points) in each phase of the problem.
- **limits**: a cell farray that contains the information about the lower and upper limits on the variables and constraints in each phase of the problem (see Section 2.6 below).
- **solinit**: a double-indexed cell array that contains a guess of the solution in each phase of the problem (see Section 2.9 below).

The optional fields (and their default values) are as follows:

- **autoscale**: a string that indicates whether or not the user would like the optimal control problem to be scaled automatically before it is solved. (default="off").
- **derivatives**: a string indicating if numerical or automatic differentiation should be used. Possible values for this string are "numerical", "complex", "automatic" (default="numerical").

2.5 Specifying Function Names Used in Optimal Control Problem

The syntax for specifying the names of the MATLAB functions is done by setting the fields in the structure **FUNCS** and is given as follows:

```

setup.funcs{1} = 'costfun.m'
setup.funcs{2} = 'dae fun.m'
setup.funcs{3} = 'eventfun.m'
setup.funcs{4} = 'connectfun.m'

```

It is emphasized that the *values* of the elements of the cell array **funcs** can be any names that the user desires. Also, it is extremely important to note that the functions must be ordered properly, i.e.,

funcs{1} \Leftrightarrow Cost Functional

Example of Specifying Function Names for Use in *GPOPS*

Suppose we have a problem whose cost functional, differential-algebraic equations, path constraints, event constraints, and linkage constraints are defined, respectively, via the *user-defined* functions *mycostfun.m*, *mydae fun.m*, *mypathfun.m*, *myeventfun.m*, and *mylinkfun.m*. Then the syntax for specifying these functions for use in *GPOPS* is given as follows:

```

problem.funcs{1} = 'gpopsCost';
problem.funcs{2} = 'gpopsDae';
problem.funcs{3} = 'gpopsEvent';
problem.funcs{4} = 'gpopsConnect';

```

2.6 Syntax for **limits** Cell Array

Once the user-defined structure **setup** has been defined, the next step in setting up a problem for use with *GPOPS* is to create a cell array called **limits**, where **limits** is a field of the structure **setup**. The cell array **limits** is double-dimensional as follows. First, **limits{1}** and **limits{2}** contain the lower and upper limits, respectively, on the variables and constraints in all of the phases in the problem.³ Next, both **limits{1}** and **limits{2}** are themselves cell arrays of size $P \times 7$, where P is the number of phases in the problem. Each row of **limits{1}** and **limits{2}** contain the lower and upper limits on the variables and constraints in the p^{th} phase of the problem. In particular, each row of **limits{1}** and **limits{2}** contain the following information:

³It is noted that the index "1" (i.e., **limits{1}**) corresponds to information about the lower limits while the index "2" (i.e., **limits{2}**) corresponds to information about the upper limits

- **nodes**: a positive integer that specifies the number of nodes (Gauss points) in phase $p \in [1, \dots, P]$
- **limits{1}{p,1}** and **limits{2}{p,1}**: row vectors, each of length two, that contain the information about the lower and upper limits, respectively, on the initial and terminal time in phase $p \in [1, \dots, P]$. The row vectors **limits{1}{p,1}** and **limits{2}{p,1}** have the following form:

$$\begin{aligned} \mathbf{limits}\{1\}\{p,1\} &= \begin{bmatrix} t_0^{\min} & t_f^{\min} \end{bmatrix} \\ \mathbf{limits}\{2\}\{p,1\} &= \begin{bmatrix} t_0^{\max} & t_f^{\max} \end{bmatrix} \end{aligned}$$

- **limits{1}{p,2}** and **limits{2}{p,2}**: matrices, each of size $n_p \times 3$, that contain the lower and upper limits, respectively, on the state in phase $p \in [1, \dots, P]$. Each of the columns of the matrices **limits{1}{p,2}** and **limits{2}{p,2}** are given as follows:
 - **limits{1}{p,2}{:;1}**: a column vector containing the lower (upper) limits on the state at the *start* of phase $p \in [1, \dots, P]$.
 - **limits{1}{p,2}{:;2}**: a column vector containing the lower (upper) limits on the state at the *during* phase $p \in [1, \dots, P]$.
 - **limits{1}{p,2}{:;3}**: a column vector containing the lower (upper) limits on the state at the *terminus* of phase $p \in [1, \dots, P]$.

The matrices **limits{1}{p,2}** and **limits{2}{p,2}** then have the following form:

$$\begin{aligned} \mathbf{limits}\{1\}\{p,2\} &= \begin{bmatrix} x_{10}^{\min} & x_1^{\min} & x_{1f}^{\min} \\ \vdots & \vdots & \vdots \\ x_{n0}^{\min} & x_n^{\min} & x_{nf}^{\min} \end{bmatrix} \\ \mathbf{limits}\{2\}\{p,2\} &= \begin{bmatrix} x_{10}^{\max} & x_1^{\max} & x_{1f}^{\max} \\ \vdots & \vdots & \vdots \\ x_{n0}^{\max} & x_n^{\max} & x_{nf}^{\max} \end{bmatrix} \end{aligned}$$

- **limits{1}{p,3}** and **limits{2}{p,3}**: column vectors, each of length m_p , that contain the lower and upper limits, respectively, on the controls in phase $p \in [1, \dots, P]$. The column vectors **limits{1}{p,3}** and **limits{2}{p,3}** have the following form:

$$\begin{aligned} \mathbf{limits}\{1\}\{p,3\} &= \begin{bmatrix} u_1^{\min} \\ \vdots \\ u_m^{\min} \end{bmatrix} \\ \mathbf{limits}\{2\}\{p,3\} &= \begin{bmatrix} u_1^{\max} \\ \vdots \\ u_m^{\max} \end{bmatrix} \end{aligned}$$

- **limits{1}{p,4}** and **limits{2}{p,4}**: column vectors, each of length q_p , that contain the lower and upper limits, respectively, on the static parameters in phase $p \in [1, \dots, P]$. The column vectors **limits{1}{p,4}** and **limits{2}{p,4}** have the following form:

$$\begin{aligned} \mathbf{limits}\{1\}\{p,4\} &= \begin{bmatrix} q_1^{\min} \\ \vdots \\ q_{q_p}^{\min} \end{bmatrix} \\ \mathbf{limits}\{2\}\{p,4\} &= \begin{bmatrix} q_1^{\max} \\ \vdots \\ q_{q_p}^{\max} \end{bmatrix} \end{aligned}$$

- **limits{1}{p,5}** and **limits{2}{p,5}**: column vectors, each of length r_p , that contain the lower and upper limits, respectively, on the path constraints in phase $p \in [1, \dots, P]$. The column vectors **limits{1}{p,5}** and **limits{2}{p,5}** have the following form:

$$\begin{aligned} \mathbf{limits}\{1\}\{p, 5\} &= \begin{bmatrix} c_1^{\min} \\ \vdots \\ c_{r_p}^{\min} \end{bmatrix} \\ \mathbf{limits}\{2\}\{p, 5\} &= \begin{bmatrix} c_1^{\max} \\ \vdots \\ c_{r_p}^{\max} \end{bmatrix} \end{aligned}$$

- **limits{1}{p,6}** and **limits{2}{p,6}**: column vectors, each of length e_p , that contain the lower and upper limits on the event constraints in phase $p \in [1, \dots, P]$. The column vectors **limits{1}{p,6}** and **limits{2}{p,6}** have the following form:

$$\begin{aligned} \mathbf{limits}\{1\}\{p, 6\} &= \begin{bmatrix} \phi_1^{\min} \\ \vdots \\ \phi_{e_p}^{\min} \end{bmatrix} \\ \mathbf{limits}\{2\}\{p, 6\} &= \begin{bmatrix} \phi_1^{\max} \\ \vdots \\ \phi_{e_p}^{\max} \end{bmatrix} \end{aligned}$$

- **limits{1}{p,7}** and **limits{2}{p,7}**: scalars that contain the lower and upper limits on the duration of phase $p \in [1, \dots, P]$. The scalars **limits{1}{p,7}** and **limits{2}{p,7}** have the following form:

$$\begin{aligned} \mathbf{limits}\{1\}\{p, 7\} &= T^{\min} \\ \mathbf{limits}\{2\}\{p, 7\} &= T^{\max} \end{aligned}$$

Example of Setting Up a Limits Cell Array

As an example of setting up a limits cell array in *GPOPS*, consider the following two-phase optimal control problem. In particular, suppose that *phase 1* of the problem has 3 states, 2 controls, 2 path constraints, and 5 event constraints. Suppose further that the lower and upper limits on the initial and terminal time in the first phase are given as

$$\begin{aligned} 0 &\leq t_0^{(1)} \leq 0 \\ 50 &\leq t_f^{(1)} \leq 100 \end{aligned}$$

Next, suppose that the lower and upper limits on the states at the *start* of the first phase are given, respectively, as

$$\begin{aligned} 1 &\leq x_1(t_0^{(1)}) \leq 1 \\ -3 &\leq x_2(t_0^{(1)}) \leq 0 \\ 0 &\leq x_3(t_0^{(1)}) \leq 5 \end{aligned}$$

Similarly, suppose that the lower and upper limits on the states *during* the first phase are given, respectively, as

$$\begin{aligned} 1 &\leq x_1(t^{(1)}) \leq 10 \\ -50 &\leq x_2(t^{(1)}) \leq 50 \\ -20 &\leq x_3(t^{(1)}) \leq 20 \end{aligned}$$

Finally, suppose that the lower and upper limits on the states at the *terminus* of the first phase are given, respectively, as

$$\begin{aligned} 5 &\leq x_1(t_f^{(1)}) \leq 7 \\ 2 &\leq x_2(t_f^{(1)}) \leq 2.5 \\ -\pi &\leq x_2(t_f^{(1)}) \leq \pi \end{aligned}$$

Next, suppose that the lower and upper limits on the controls *during* the first phase are given, respectively, as

$$\begin{aligned} -50 &\leq u_1(t^{(1)}) \leq 50 \\ -100 &\leq u_2(t^{(1)}) \leq 100 \end{aligned}$$

Next, suppose that the lower and upper limits on the path constraints *during* the first phase are given, respectively, as

$$\begin{aligned} -10 &\leq p_1(t^{(1)}) \leq 10 \\ 1 &\leq p_2(t^{(1)}) \leq 1 \end{aligned}$$

Next, suppose that the lower and upper limits on the event constraints of the first phase are given, respectively, as

$$\begin{aligned} 0 &\leq \phi_1^{(1)} \leq 1 \\ -2 &\leq \phi_2^{(1)} \leq 4 \\ 8 &\leq \phi_3^{(1)} \leq 20 \\ 3 &\leq \phi_4^{(1)} \leq 3 \\ 10 &\leq \phi_5^{(1)} \leq 10 \end{aligned}$$

In a similar manner, suppose that *phase 2* of the problem contains the following information: 4 states, 3 controls, 1 path constraint, and 4 event constraints. Also, suppose now that the lower and upper limits on the initial and terminal time in the first phase are given, respectively, as

$$\begin{aligned} 50 &\leq t_0^{(2)} \leq 100 \\ 100 &\leq t_f^{(2)} \leq 200 \end{aligned}$$

Next, suppose that the lower and upper limits on the states at the *start* of the second phase are given, respectively, as

$$\begin{aligned} 3 &\leq x_1(t_0^{(2)}) \leq 3 \\ -10 &\leq x_2(t_0^{(2)}) \leq 4 \\ 7 &\leq x_3(t_0^{(2)}) \leq 18 \\ 25 &\leq x_4(t_0^{(2)}) \leq 75 \end{aligned}$$

Similarly, suppose that the lower and upper limits on the states *during* the second phase are given, respectively, as

$$\begin{aligned} -200 &\leq x_1(t^{(2)}) \leq 200 \\ -50 &\leq x_2(t^{(2)}) \leq 50 \\ -20 &\leq x_3(t^{(2)}) \leq 20 \\ -80 &\leq x_4(t^{(2)}) \leq 80 \end{aligned}$$

Finally, suppose that the lower and upper limits on the states at the *terminus* of the second phase are given, respectively, as

$$\begin{aligned} 12 &\leq x_1(t_f^{(2)}) \leq 12 \\ -60 &\leq x_2(t_f^{(2)}) \leq 30 \\ -90 &\leq x_3(t_f^{(2)}) \leq 10 \\ 100 &\leq x_4(t_f^{(2)}) \leq 500 \end{aligned}$$

Next, suppose that the lower and upper limits on the controls *during* the second phase are given, respectively, as

$$\begin{aligned} -90 &\leq u_1(t^{(2)}) \leq 90 \\ -120 &\leq u_2(t^{(2)}) \leq 120 \end{aligned}$$

Next, suppose that the lower and upper limits on the path constraints *during* the second phase are given, respectively, as

$$\begin{aligned} -10 &\leq p_1(t^{(2)}) \leq 10 \\ 1 &\leq p_2(t^{(2)}) \leq 1 \end{aligned}$$

Finally, suppose that the lower and upper limits on the events constraints of the second phase are given, respectively, as

$$\begin{aligned} 0 &\leq \phi_1^{(2)} \leq 1 \\ -2 &\leq \phi_2^{(2)} \leq 4 \\ 8 &\leq \phi_3^{(2)} \leq 20 \\ 3 &\leq \phi_4^{(2)} \leq 3 \end{aligned}$$

Then a MATLAB code that would generate the above specification is given as follows:

```
% Use auxiliary variables to make the code easier to read and reduce
% the possibility of a coding error.
imin = 1; imax = 2;
itime = 1; istate = 2; icontrol = 3; iparameter = 4; ipath = 5; ievent = 6;

iphase = 1; % Set the phase number to 1
limits{imin}{iphase,itime} = [0 50];
limits{imax}{iphase,itime} = [0 100];
limits{imin}{iphase,istate} = [1 1 5; -3 -50 2; 0 -20 -pi];
limits{imax}{iphase,istate} = [1 10 7; 0 50 2.5; 5 20 pi];
limits{imin}{iphase,icontrol} = [-50; -100];
limits{imax}{iphase,icontrol} = [ 50; 100];
limits{imin}{iphase,iparameter} = [];
limits{imax}{iphase,iparameter} = [];
limits{imin}{iphase,ipath} = [-10; 1];
limits{imax}{iphase,ipath} = [10; 1];
limits{imin}{iphase,ievent} = [0; -2; 8; 3; 10];
limits{imax}{iphase,ievent} = [1; 4; 20; 3; 10];

iphase = 2; % Set the phase number to 2
limits{imin}{iphase,itime} = [50 100];
limits{imax}{iphase,itime} = [100 200];
limits{imin}{iphase,istate} = [3 -200 12; -10 -50 -60; 7 -20 -90; 25 -80 100];
limits{imax}{iphase,istate} = [3 200 12; 4 50 30; 18 20 10; 75 80 500];
limits{imin}{iphase,icontrol} = [-90; -120];
limits{imax}{iphase,icontrol} = [ 90; 120];
limits{imin}{iphase,iparameter} = [];
limits{imax}{iphase,iparameter} = [];
limits{imin}{iphase,ipath} = [-10; 10];
limits{imax}{iphase,ipath} = [1; 1];
limits{imin}{iphase,ievent} = [0; -2; 8; 3];
limits{imax}{iphase,ievent} = [1; 4; 20; 3];

setup.limits = limits;
```

Note: in order to make it easier to follow the MATLAB code, we have introduced auxiliary integer variables such as **imin**, **imax**, etc., so that the user can keep track of the quantities whose bounds are being specified.

The use of such auxiliary variables is not necessary, but it makes the code easier to read. Also, it allows for a substantial portion of the code to be cut and paste from phase to phase, thus reducing the chance of a coding error.

2.7 Syntax for **connections** Cell Array

Another required field in the structure **setup** is a cell array called **connections** that defines the way that the phases are to be connected. If there is only one phase in the problem, then **setup.connections** is set to "[]". If the problem contains more than a single phase, then **connections** is a cell array of size $L \times 3$ (where L is the number of pairs of phases to be connected). The elements of the cell array **connections** are as follows:

- **connections**{*s*,1}: a column vector of length l_s containing the lower limits on the s^{th} pair of phase-connect
- **connections**{*s*,2}: a column vector of length l_s containing the upper limits on the s^{th} pair of phase-connect
- **connections**{*s*,1}: a row vector of length two containing the s^{th} pair of phases to be connected

2.8 Syntax of Each Function in Optimal Control Problem

Now that we know *which* functions *GPOPS* will use, the next step is to discuss the syntax of each of these functions. In general, the syntax for each function will differ because the quantities being evaluated are different in nature. In this section we will explain the syntax of each function.

2.8.1 Syntax of Function Used to Evaluate Cost

The syntax used to evaluate a user-defined cost functional is given as follows:

function [Mayer,Lagrange]=mycostfun(solcost,iphase);

where **mycostfun.m** is the name of the MATLAB function, **solcost** and **iphase** are inputs to the function, and **Mayer** and **Lagrange** are the outputs. The inputs **solcost** are a cell array and an integer, respectively, while **Mayer** and **Lagrange** are the endpoint cost and the integrand of the integrated cost, respectively. The input cell array **solcost** has the following elements (note that N =number of LG points which are on the interior of the time interval):

- **solcost**{1,1}: the initial time in phase **iphase**
- **solcost**{1,2}: the initial state in phase **iphase**
- **solcost**{1,3}: the terminal time in phase **iphase**
- **solcost**{1,4}: the terminal state in phase **iphase**
- **solcost**{2,1}: a column vector of length N that contains the time (excluding the initial and terminal points) in phase **iphase**
- **solcost**{2,2}: a matrix of size $N \times n$ (where n is the number of states) that contains the values of the state (excluding the initial and terminal points) in phase **iphase**
- **solcost**{2,3}: a matrix of size $N \times m$ (where m is the number of controls) that contains the values of the control (excluding the initial and terminal points) in phase **iphase**
- **solcost**{2,4}: a column vector of length q that contains the values of the static parameters in the phase

Finally, the outputs of **mycostfun** are as follows:

- **Mayer**: a *scalar*, i.e., size 1×1
- **Lagrange**: a *column* vector of size $N \times 1$

2.8.2 Warning About Outputs to Cost Function

For many optimal control problems the output **Lagrange** in the user-defined cost function **mycostfun** is **zero**. As such, it is appealing to set **Lagrange** to zero by the MATLAB command

$$\text{Lagrange}=0; \quad (2-1)$$

However, *the integrand cannot be set to a scalar value!*. Instead, the integrand *must* be set to a **column vector of zeros!**. The way to set the integrand to zero and that *will work in all cases* (i.e., numerical or automatic differentiation) is as follows:

$$\text{Lagrange}=\text{zeros}(\text{size}(\text{solcost}\{2,1\})); \quad (2-2)$$

The user is urged to use the syntax of Eq. (2-2) whenever the integrand is identically zero.

Example of a Cost Functional

Suppose we have a two-phase optimal control problem that uses a cost functional named “mycostfun.m”. Suppose further that the dimension of the state in each phase is 2 while the dimension of the control in each phase is 2. Also, suppose that the endpoint and integrand cost in phase 1 are given, respectively, as

$$\begin{aligned} \Phi^{(1)}(\mathbf{x}^{(1)}(t_0), t_0, \mathbf{x}^{(1)}(t_f), t_f) &= \mathbf{x}^T(t_f) \mathbf{S} \mathbf{x}(t_f) \\ \mathcal{L}^{(1)}(\mathbf{x}^{(1)}(t), \mathbf{u}^{(1)}(t), t) &= \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} \end{aligned}$$

while the endpoint and integrand in phase 2 are given, respectively, as

$$\begin{aligned} \Phi^{(2)}(\mathbf{x}^{(2)}(t_0^{(1)}), t_0, \mathbf{x}^{(1)}(t_f^{(1)}), t_f^{(1)}) &= \mathbf{x}^T(t_f) \mathbf{x}(t_f) \\ \mathcal{L}^{(2)}(\mathbf{x}^{(2)}(t), \mathbf{u}^{(2)}(t), t) &= \mathbf{u}^T \mathbf{R} \mathbf{u} \end{aligned}$$

Then the syntax of the above cost functional is given as follows:

```
function [endpoint,integrand]=mycostfun(solcost,iphase);

Q = [5 0; 0 2];
R = [1 0; 0 3];
S = [1 5; 5 1];
t0 = solcost{1,1};
x0 = solcost{1,2};
tf = solcost{1,3};
xf = solcost{1,4};
t = solcost{2,1};
x = solcost{2,2};
u = solcost{2,3};
p = solcost{2,4};

if iphase==1,
    Mayer = dot(xf',S*xf');
    Lagrange = dot(x,x*Q',2)+dot(u,u*R',2); % Note transposes
elseif iphase==2,
    Mayer = dot(xf,xf);
    Lagrange = dot(u,u*R',2); % Note transposes
    Lagrange = Lagrange'; % Lagrange must be a column vector
end;
```

It is noted in the above function call that the third argument in the command **dot** takes the dot product across the *rows*, thereby producing a *column vector*.

2.8.3 Syntax of Function Used to Evaluate Differential-Algebraic Equations

The calling syntax used to evaluate the right-hand side of a user-defined vector of differential equations is given as follows:

```
function dae=mydaefun(soldae,iphase);
```

where *mydaefun.m* is the name of the MATLAB function, *soldae* and *iphase* are the inputs to the function, and *xdot* is the output (i.e., the right-hand side of the differential equations and the values of the path constraints). The input *soldae* is a cell array while the output *dae* is a matrix of size $N \times (n + c)$ where n is the number of differential equations, c is the number of path constraints, and N is the number of LG points. The input structure *soldae* has the following fields:

- *soldae{1}*: a column vector of length N that contains the time (excluding the initial and terminal points) in phase *iphase*
- *soldae{2}*: a matrix of size $N \times n$ (where n is the number of states) that contains the values of the state (excluding the initial and terminal points) in phase *iphase*
- *soldae{3}*: a matrix of size $N \times m$ (where m is the number of controls) that contains the values of the control (excluding the initial and terminal points) in phase *iphase*
- *soldae{4}*: a column vector of length q that contains the values of the static parameters in the phase

Finally, the output of *mydaefun* are as follows:

- *dae*: a matrix of size $N \times (n + c)$ containing the values of the right-hand side of the n differential equations and the c path constraints evaluated at the N LG points

Example of a Differential-Algebraic Equation

Suppose we have a two-phase optimal control problem that uses a differential equation function called “mydaefun.m”. Suppose further that the dimension of the state in each phase is 2, the dimension of the control in each phase is 2. Furthermore, suppose that there are no path constraints in phase 1 and one path constraint in phase 2. Next, suppose that the differential equations in phase 1 are given as

$$\begin{aligned}\dot{x}_1 &= -x_1^2 - x_2^2 + u_1 u_2 \\ \dot{x}_2 &= -x_1 x_2 + 2(u_1 + u_2)\end{aligned}$$

Also, suppose that the differential equations in phase 2 are given as

$$\begin{aligned}\dot{x}_1 &= \sin(x_1^2 + x_2^2) + u_1 u_2^2 \\ \dot{x}_2 &= -\sin x_1 \cos x_2 + 2u_1 u_2\end{aligned}$$

Finally, suppose that the path constraint in phase 2 is given as

$$u_1^2 + u_2^2 = 1$$

Then a MATLAB code that will evaluate the above system of differential-algebraic equations is given as follows:

```
function dae = mydaefun(soldae,iphase);

t = soldae{1};
x = soldae{2};
u = soldae{3};
p = soldae{4};
```

```

if iphase==1,
    x1dot = -x(:,1).^2-x(:,2).^2 + u(:,1).*u(:,2);
    x2dot = -x(:,1).*x(:,2) + 2*(u(:,1)+u(:,2));
    path = [];
elseif iphase==2,
    x1dot = -x(:,1).*x(:,2)+u(:,1).*u(:,2).^2;
    x2dot = -sin(x(:,1)).*cos(x(:,2))+2*u(:,1).*u(:,2);
    path = u(:,1).^2+u(:,2).^2;
end;
dae = [x1dot x2dot path];

```

2.8.4 Syntax of Function Used to Evaluate Event Constraints

The syntax used to evaluate a user-defined vector of event constraints is given as follows:

function events=myeventfun(solevents,iphase);

where **myeventfun.m** is the name of the MATLAB function, **solevents** and **iphase** are the inputs to the function, and **events** is the output (i.e., the value of the event constraints). The inputs **solevents** and **iphase** are a cell array and an integer, respectively, while the output **eventspath** is a *column vector* of length e where e is the number of event constraints. The input cell array **solevents** has the following elements:

- **solevents{1,1}**: the time at the start of the phase
- **solevents{1,2}**: the state at the start of the phase
- **solevents{1,3}**: the time at the terminus of the phase
- **solevents{1,4}**: the state at the terminus of the phase
- **solevents{1,5}**: the static parameters in the phase

Example of Event Constraints

Suppose we have a one-phase optimal control problem that has two initial event constraints and three terminal event constraints. Suppose further that the number of states in the phase is six and that the function that computes the values of these constraints is called “myeventfun.m”. Finally, let the two initial event constraints be given as

$$\begin{aligned}\phi_{01} &= x_1(t_0)^2 + x_2(t_0)^2 + x_3(t_0)^2 \\ \phi_{02} &= x_4(t_0)^2 + x_5(t_0)^2 + x_6(t_0)^2\end{aligned}$$

while the three terminal event constraints are given as

$$\begin{aligned}\phi_{f1} &= \sin(x_1(t_f)) \cos(x_2(t_f) + x_3(t_f)) \\ \phi_{f2} &= \tan(x_4^2(t_f) + x_5^2(t_f) + x_6^2(t_f)) \\ \phi_{f3} &= x_4(t_f) + x_5(t_f) + x_6(t_f)\end{aligned}$$

Then the syntax of the above event function is given as

```

function events = myeventfun(solevents,iphase);

t0 = solevents{1};
x0 = solevents{2};
tf = solevents{3};

```

```

xf = solevents{4};

ei1 = dot(x0(1:3),x0(1:3));
ei2 = dot(x0(4:6),x0(4:6));
ef1 = sin(xf(1))*cos(xf(2)+xf(3));
ef2 = tan(dot(xf(4:6),xf(4:6)));
ef3 = xf(4)+xf(5)+xf(6);

events = [ei1;ei2;ef1;ef2;ef3];

```

Finally, it is noted that each event constraint need not be a function of either the initial or the terminal state, but can also be functions that contain *both* the initial and terminal state and/or the initial and terminal time. As an example of an event constraint that contains both the initial and terminal state, consider the following example.

Example of Event Constraint Containing Both Initial and Terminal State

Suppose we have a one-phase optimal control problem that contains only a single state. Furthermore, suppose that the problem contains a single event constraint on the *difference* between the terminal value of the state and the initial value of the state. Finally, suppose that the function that computes the values of these constraints is called “myeventfun.m”. Then the event constraint is evaluated as

$$\phi = x(t_f) - x(t_0)$$

Then the syntax of the above event function is given as

```

function events = myeventfun(solevents);

t0 = solevents{1};
x0 = solevents{2};
tf = solevents{3};
xf = solevents{4};

events = xf-x0;

```

2.8.5 Syntax of Function Used to Evaluate Linkage Constraints

The syntax used to define the *user-defined* vector of linkage constraints between two phases is given as follows:

```
function links=mylinkfun(sollink,left_phase,right_phase);
```

where *mylinkfun.m* is the name of the MATLAB function, *sollink*, *left_phase*, *right_phase* are the inputs to the function, and *links* is the output (i.e., the value of the linkage constraints). The inputs *sollink*, *left_phase*, and *right_phase* are a cell array and two integers, respectively, while the output *links* is a *column vector* of length *l*, where *l* is the number of event constraints. The input cell array *sollink* has the following elements:

- *sollink*{1}: the state at the terminus of phase *left_phase*
- *sollink*{2}: the static parameters in phase *left_phase*
- *sollink*{3}: the state at the start of phase *right_phase*

- **sollink{4}**: the static parameters in phase *right_phase*

The terms *left* and *right* are conventions adopted to help the user orient the phases on a page from left to right.

2.9 Specifying an Initial Guess of The Solution

The field **solinit** of the user-defined structure **setup** contains the initial guess for the problem. The field **solinit** is a cell array of size $P \times 4$ (where P is the number of phases in the problem). The p^{th} row of the cell array **solinit** contains the initial guess of the problem in phase $p \in [1, \dots, P]$. The elements of the cell array **solinit** are given as follows:

- **solinit{p,1}**: a *column* vector of length s where s is the number of time points used in the guess
- **solinit{p,2}**: a matrix of size $s \times n$ where s is the number of time points and n is the number of states in the phase
- **solinit{p,3}**: a matrix of size $s \times m$ where s is the number of time points and m is the number of controls in the phase
- **solinit{p,4}**: a column vector of length q where q is the number of static parameters in the phase

It is noted that the element **solinit{p,1}** must be monotonic and in the same direction as that specified by the field **independent.variable** of the structure **setup**. Schematically, in each phase of the problem the guess for the time, states, controls, and parameters is structured as follows:

$$\begin{aligned}
 \mathbf{solinit}\{p, 1\} &= \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ \vdots \\ t_s \end{bmatrix} \\
 \mathbf{solinit}\{p, 2\} &= \begin{bmatrix} x_{10} & x_{20} & \cdots & x_{n0} \\ x_{11} & x_{21} & \cdots & x_{n1} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1s} & x_{2s} & \cdots & x_{ns} \end{bmatrix} \\
 \mathbf{solinit}\{p, 3\} &= \begin{bmatrix} u_{10} & u_{20} & \cdots & u_{m0} \\ u_{11} & u_{21} & \cdots & u_{m1} \\ \vdots & \vdots & \vdots & \vdots \\ u_{1s} & u_{2s} & \cdots & u_{ms} \end{bmatrix} \\
 \mathbf{solinit}\{p, 4\} &= \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_q \end{bmatrix}
 \end{aligned}$$

Example of Specifying an Initial Guess

Suppose we have a two-phase problem that has three states and two controls in phase 1 while it has two states and one control in phase 2. Furthermore, suppose that we choose five time points for the guess in phase 1 while we choose 3 time points for the guess in phase 2. A MATLAB code that would create such an initial guess is given below.

```

itime = 1; istate = 2; icontrol = 3; iparameter = 4;
iphase = 1;
solinit{iphase,itime} = [0; 1; 3; 5; 7];
solinit{iphase,istate}(:,1) = [1.27; 3.1; 5.8; 9.6; -13.7272];
solinit{iphase,istate}(:,2) = [-4.2; -9.6; 8.5; 25.73; 100.00];
solinit{iphase,istate}(:,3) = [18.727; 1.827; 25.272; -14.272; 26.84];
solinit{iphase,icontrol}(:,1) = [8.4; -13.7; -26.5; 19; 87];
solinit{iphase,icontrol}(:,2) = [-1.2; 5.8; -3.77; 14; 19.787];
solinit{iphase,iparameter} = [];

iphase = 2;
solinit{2,itime} = [7; 7.5; 8];
solinit{2,istate}(:,1) = [0.5; 1.5; 8];
solinit{2,istate}(:,2) = [-0.5; -2.5; 19];
solinit{2,istate}(:,3) = [8.4; -13.7; -26.5; 19; 87];
solinit{iphase,iparameter} = [];

setup.solinit = solinit;

```

It is noted again that, for the above example, auxiliary integer variables were used to minimize the cumbersomeness of coding and to minimize the chance of error.

2.10 Scaling of Optimal Control Problem

As with any numerical optimization procedure, the approach employed by *GPOPS* requires a well-scaled optimal control problem. In general, it is recommended that the user scale the problem in accordance with any known large discrepancies either in the sizes of various quantities (i.e., state, control) or the sizes of the derivatives of such quantities. While it is beyond the scope of this user's manual to provide a general procedure for scaling, in an attempt to reduce the burden on the user an automatic scaling procedure has been developed for use in *GPOPS*. This procedure is based on the scaling algorithm developed in (Betts, 2001). In order to invoke the automatic scaling routine, the user must set the field **autoscale** in the user-defined structure **setup** to the string "on".

The automatic scaling procedure operates as follows. The bounds on the variables are used to scale all components of the state, control, parameters, and time to lie between -1 and 1. As a result, it is essential that the user provide *sensible* bounds on all quantities (e.g., do not provide unreasonably large bounds as this will result in a poorly scaled problem). Next, the constraints are scaled to make the row norms of the Jacobians of the respective functions approximately unity. The automatic scaling procedure is by no means foolproof, but it has been found in practice to work well on many problems that otherwise would require scaling by hand. The advice given here is to try the automatic scaling procedure, but not to use it for too long if it is proving to be unsuccessful.

2.11 Different Options for Specification of Derivatives

The user has three choices for the computation of the derivatives of the objective function gradient and the constraint Jacobian for use within the NLP solver. As stated above, these choices are "numerical", "complex", and "automatic". By setting **setup.derivatives** to "numerical", the default finite-differencing algorithm within SNOPT is used. By setting **setup.derivatives** to "complex", a complex-step differentiation method is used. Finally, by setting **setup.derivatives** to "automatic", automatic differentiation is used (if the program MAD is installed on your computer).

Of these three methods, the complex-step differentiation method is most preferred because it is part of the *GPOPS* software (i.e., the user does not have to obtain any third-party software). Also, complex-step differentiation provides excellent accuracy and robustness for obtaining a solution. One drawback with complex-step differentiation, however, is that certain functions need to be handled with great care. In particular, the functions **min**, **max**, **abs**, and **dot** need to be redefined for use in complex-step differentiation (see Ref. Martins, et al. (2003) and the URL <http://mdolab.utias.utoronto.ca/resources/complex-step/complexify.f90> for details). Finally, the transpose operator must be replaced with a dot-transpose (i.e., a *real transpose*) because the standard transpose in MATLAB produces a complex conjugate transpose and it is necessary to maintain a real transpose when computing derivatives via complex-step differentiation.

2.12 Output of Execution of *GPOPS*

Upon execution of *GPOPS*, new fields are created in the output structure **output**. In particular, upon completion of the execution of *GPOPS*, the following new fields are created (in addition to the fields that were created prior to running *GPOPS* on the problem):

- **solution**: a cell array of size length $P \times 8$ (where P is the number of phases) containing the solution in each phase

The p^{th} row in the cell array **solution** contains the solution in phase $p \in [1, \dots, P]$. The elements of the cell array **solution** are as follows:

- **solution{p,1}**: a column vector of size $M \times 1$ containing the time at each point along the trajectory (where $M = N + 2$ is the number of time points and N is the number of LG points)
- **solution{p,2}**: a matrix of size $M \times n$ such that the rows contain the state at the time points along the trajectory
- **solution{p,3}**: a matrix of size $M \times m$ such that the rows contain the state at the time points along the trajectory
- **solution{p,4}**: a column vector of length q containing the static parameters
- **solution{p,5}**: a matrix of size $M \times n$ such that the rows contain the costate at each time point along the trajectory
- **solution{p,6}**: a structure containing the Lagrange multipliers of the path constraints
- **solution{p,7}**: a column vector of size $M \times 1$ that contains the Hamiltonian at each time point along the trajectory
- **solution{p,8}**: The Mayer part of the cost along the trajectory
- **solution{p,9}**: The Lagrange (integrated) cost along the trajectory

Analysis of the output of *GPOPS* is deferred until the next chapter where several specific examples are studied.

2.13 Useful Information for Debugging a *GPOPS* Problem

2.13.1 Debugging Code when Using Matlab Automatic Differentiation

As stated above, one of the options in *GPOPS* is to use MATLAB Automatic Differentiation (MAD). In particular, MAD is invoked when the user sets the field **setup.derivatives** to “automatic”. As is often the case, the user may want to break in the various functions to ensure proper coding of the functions. In the case where numerical derivatives are used, the variables can be printed in the MATLAB command window

by breaking in the function and typing the appropriate variable name. However, when MAD is being used, all variables are so-called *fmad* objects (see the TOMLAB/MAD documentation). An *fmad* object is not a real-valued variable, but contains information about both the *value* of the variables and the *derivatives* of the variables. If a user wants to obtain the value of a variable, the command is “`getvalue`”. For example, when using automatic differentiation, the value of a variable named “*y*” is obtained by typing the command “`getvalue(y)`” (and not simply by typing “*y*”). The user is urged to keep this in mind when using MAD to compute derivatives.

2.13.2 Dimensions of Arrays When Debugging *GPOPS* Code

One aspect of *GPOPS* that may appear confusing when debugging code pertains to the dimensions of the arrays and the corresponding time values. It is important to remember that *GPOPS* uses collocation at *Legendre-Gauss* points. Because the Legendre-Gauss points lie on the interior of the time interval of interest, the dynamics, path constraints, and integrand cost are computed only at the Legendre-Gauss points. While this may appear to be a bit strange, the fundamental point here is that Gaussian quadrature (which is used in *GPOPS*) only evaluates the functions at the Legendre-Gauss points. Do not try to “fool” *GPOPS* by adding the endpoints to the computation of the dynamics, path constraints, or integrand cost. If you do this, you will get an error because the dimensions are incorrect. For a more complete mathematical description of the collocation method used in *GPOP*, see either Chapter 1 of this manual or the references contained in the bibliography at the end of this manual.

Chapter 3

Examples of Using *GPOPS*

In this Chapter we provide three complete examples of using *GPOPS*. For each example the optimal control problem is first described quantitatively, then the *GPOPS* code is provided.

3.1 Hyper-Sensitive Problem

Consider the following optimal control problem. Minimize the cost functional

$$J = \frac{1}{2} \int_0^{t_f} [x^2 + u^2] dt \quad (3-1)$$

subject to the dynamic constraint

$$\dot{x} = -x^3 + u \quad (3-2)$$

and the boundary conditions

$$\begin{aligned} x(0) &= 1.5 \\ x(t_f) &= 1 \end{aligned} \quad (3-3)$$

with $t_f = 50$. It is noted that this problem is taken from Rao (2000). The *GPOPS* code that solves this problem is shown below. In particular, the following three MATLAB functions are defined:

- hyperSensitiveMain.m: MATLAB m-file (main driver) for problem
- hyperSensitiveCost.m: MATLAB function that evaluates the cost functional
- hyperSensitiveDae.m: MATLAB function that evaluates the differential-algebraic equations

The beginning and end of each function is labeled by a MATLAB comment.

```
%-----  
% BEGIN: script hyperSensitiveMain.m  
%-----  
% This m-file is the main file for the following optimal control  
% problem:  
% minimize  
%   J = 0.5*(x^2+u^2)  
% subject to  
%   dx/dt = -x^3 + u  
%   x(0) = 1.5  
%   x(tf) = 1  
%-----  
% This example is taken from the following reference:  
% Rao, A. V. and Mease, K. D., "Eigenvector Approximate Dichotomic
```

```

% Basis Method for Solving Hypersensitive Optimal Control
% Problems," Optimal Control Applications and Methods, Vol. 21,
% No. 1, 2000, pp. 1-19.
%-----
gpopsInitialize;

x0 = 1.5;
xf = 1;
xmin = -50;
xmax = 50;
umin = -50;
umax = 50;

iphase = 1;
limits{imin}{iphase,itime} = [0 50];
limits{imax}{iphase,itime} = [0 50];
limits{imin}{iphase,istate} = [x0 xmin xf];
limits{imax}{iphase,istate} = [x0 xmax xf];
limits{imin}{iphase,icontrol} = umin;
limits{imax}{iphase,icontrol} = umax;
limits{imin}{iphase,iparameter} = [];
limits{imax}{iphase,iparameter} = [];
limits{imin}{iphase,ipath} = [];
limits{imax}{iphase,ipath} = [];
limits{imin}{iphase,ievent} = [];
limits{imax}{iphase,ievent} = [];
solinit{iphase,1} = [0; 20];
solinit{iphase,2} = [x0; x0];
solinit{iphase,3} = [0; 0];

solinit{iphase,4} = []; % No parameters in Phase 1

clear x0 xf xmin xmax umin umax

setup.name = 'HyperSensitive-Problem';
setup.funcs = {'hyperSensitiveCost','hyperSensitiveDae'};
setup.nodes = 50;
setup.limits = limits;
setup.solinit = solinit;
setup.connections = [];
setup.derivatives = 'complex';
setup.direction = 'increasing';
setup.autoscale = 'off';

output = gpops(setup);
solution = output.solution;

function dae = hyperSensitiveDae(sol,iphase);

t = sol{1};
x = sol{2};
u = sol{3};
p = sol{4};

dae = -x.^3+u;

function [Mayer,Lagrange] = hyperSensitiveCost(sol,iphase);

t0 = sol{1,1};
x0 = sol{1,2};
tf = sol{1,3};
xf = sol{1,4};
t = sol{2,1};
x = sol{2,2};
u = sol{2,3};
p = sol{2,4};

```

```
Mayer = zeros(size(t0));
Lagrange = 0.5*(x.^2+u.^2);
```

The output of the above code from *GPOPS* is summarized in the following three plots that contain the state (x), costate (λ), and the Hamiltonian (H), respectively, for the problem (where $H = L + \lambda f$ where $L = 0.5(x^2 + u^2)$ and $f = -x^3 + u$).

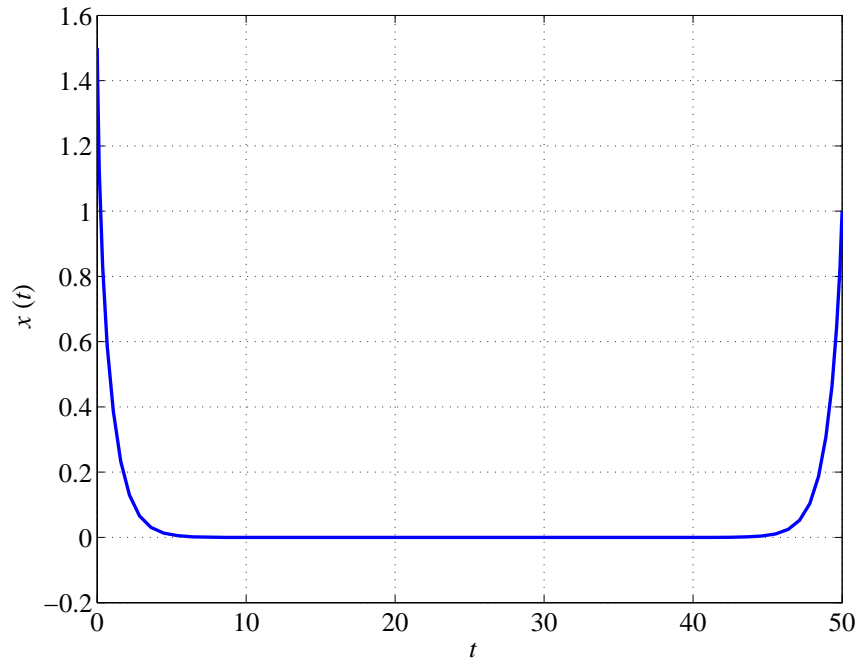


Figure 3.1 $x(t)$ vs. t for one-dimensional problem.

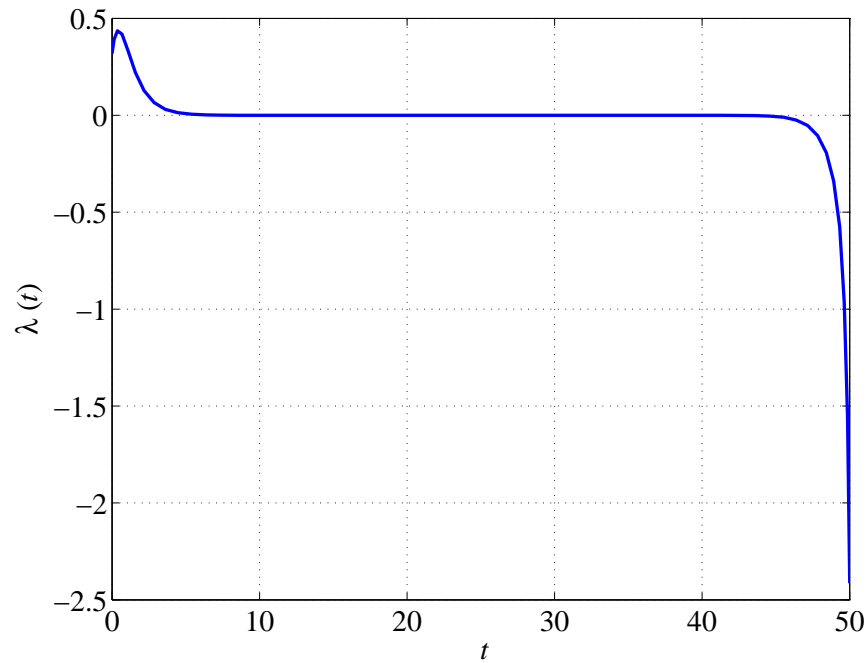


Figure 3.2 $\lambda(t)$ vs. t for one-dimensional problem.

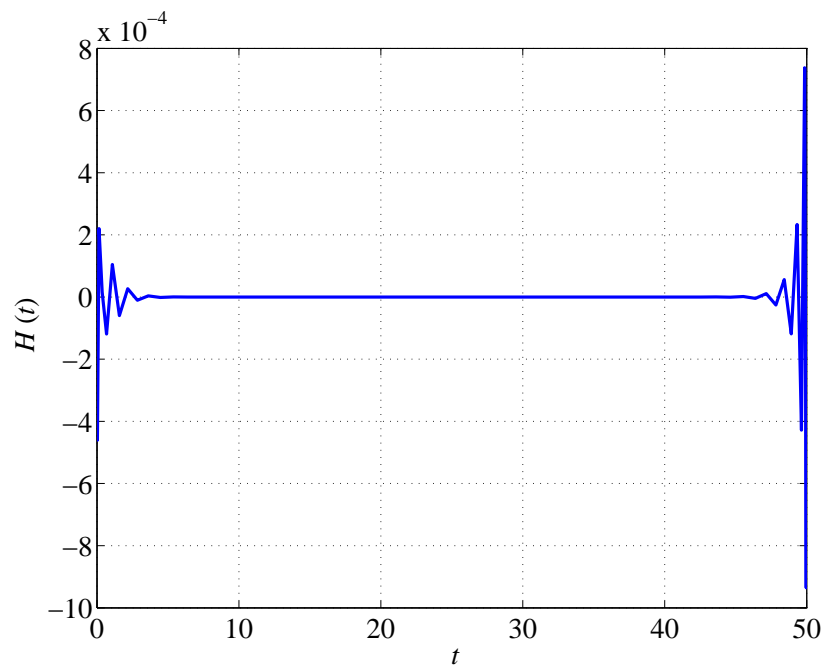


Figure 3.3 H vs. t for one-dimensional problem.

3.2 Bryson-Denham Problem

Consider the following optimal control problem. Minimize the cost functional

$$J = x_3(t_f) \quad (3-4)$$

subject to the dynamic constraints

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= u \\ \dot{x}_3 &= \frac{1}{2}u^2\end{aligned}\tag{3-5}$$

the path constraint

$$0 \leq x_1(t) \leq 1/9\tag{3-6}$$

and the boundary conditions

$$\begin{aligned}x_1(0) &= 0 \\ x_2(0) &= 1 \\ x_3(0) &= 0 \\ x_1(t_f) &= 0 \\ x_2(t_f) &= -1\end{aligned}\tag{3-7}$$

The above problem was originally formulated by Bryson and Denham (need reference) and is referred to as the *Bryson-Denham* problem. The *GPOP* code that solves the Bryson-Denham problem is shown below. In particular, the following three MATLAB functions are defined:

- `brysonDenhamMain.m`: MATLAB m-file (main driver) for problem
- `brysonDenhamCost.m`: MATLAB function that evaluates the cost functional
- `brysonDenhamDae.m`: MATLAB function that evaluates the differential-algebraic equation
- `brysonDenhamEvent.m`: MATLAB function that evaluates the event constraints

The beginning and end of each function is labeled by a MATLAB comment. It is noted that while all five boundary conditions are simple bounds (and are, thus, linear, they are treated as general event constraints in order to demonstrate the proper use of an event function.

```
% -----
% Bryson-Denham Example Problem
% -----
% -----
% This example is taken from the following reference:
% -----
% Bryson, A. E., Denham, W. F., and Dreyfus, S. E., "Optimal
% Programming Problems with Inequality Constraints. I: Necessary
% Conditions for Extremal Solutions, AIAA Journal, Vol. 1, No. 11,
% November, 1963, pp. 2544-2550.
gpopsInitialize;

x10 = 0;
x20 = 1;
x30 = 0;
x1f = 0;
x2f = -1;
x1min = -10;
x1max = 10;
x2min = x1min;
x2max = x1max;
x3min = x1min;
x3max = x1max;

param_min = [];
param_max = [];
path_min = 0;
path_max = 1/9;
event_min = [x10; x20; x30; x1f; x2f];
event_max = [x10; x20; x30; x1f; x2f];
duration_min = [];
duration_max = [];
```

```

iphase = 1;
limits{imin}{iphase,itime} = [0 0];
limits{imax}{iphase,itime} = [0 50];
limits{imin}{iphase,istate}(1,:) = [x1min x1min x1min];
limits{imax}{iphase,istate}(1,:) = [x1max x1max x1max];
limits{imin}{iphase,istate}(2,:) = [x2min x2min x2min];
limits{imax}{iphase,istate}(2,:) = [x2max x2max x2max];
limits{imin}{iphase,istate}(3,:) = [x3min x3min x3min];
limits{imax}{iphase,istate}(3,:) = [x3max x3max x3max];
limits{imin}{iphase,icontrol} = -5000;
limits{imax}{iphase,icontrol} = 5000;
limits{imin}{iphase,iparameter} = param_min;
limits{imax}{iphase,iparameter} = param_max;
limits{imin}{iphase,ipath} = path_min;
limits{imax}{iphase,ipath} = path_max;
limits{imin}{iphase,ievent} = event_min;
limits{imax}{iphase,ievent} = event_max;
limits{imin}{iphase,iduration} = [];
limits{imax}{iphase,iduration} = [];
solinit{iphase,itime} = [0; 0.5];
solinit{iphase,istate}(:,1) = [x10; x1f];
solinit{iphase,istate}(:,2) = [x20; x2f];
solinit{iphase,istate}(:,3) = [x30; x30];
solinit{iphase,icontrol} = [0; 0];
solinit{iphase,iparameter} = []; % No parameters in Phase 1

clear x10 x20 x30 x1f x2f x1min x1max x2min x2max x3min x3max param_min param_max path_min path_max event_min eve

setup.name = 'Bryson-Denham-Problem';
setup.funcs{1} = 'brysonDenhamCost';
setup.funcs{2} = 'brysonDenhamDae';
setup.funcs{3} = 'brysonDenhamEvent';
setup.funcs{4} = 'brysonDenhamConnect';
setup.nodes = 55;
setup.limits = limits;
setup.solinit = solinit;
setup.connections = [];
setup.derivatives = 'complex';
setup.direction = 'increasing';
setup.autoscale = 'off';

output = gpops(setup);
solution = output.solution;

function dae = brysonDenhamDae(sol,iphase);

t = sol{1};
x = sol{2};
u = sol{3};
x1dot = x(:,2);
x2dot = u;
x3dot = u.^2/2;
path = x(:,1);

dae = [x1dot x2dot x3dot path];

function [Mayer,Lagrange]=brysonDenhamCost(sol,iphase);

t0 = sol{1,1};
x0 = sol{1,2};
tf = sol{1,3};
xf = sol{1,4};
t = sol{2,1};
x = sol{2,2};
u = sol{2,3};
p = sol{2,4};

```

```

Mayer = xf(3);
Lagrange = zeros(size(t));

function events = brysonDenhamEvent(sol,iphase);

t0 = sol{1};
x0 = sol{2};
tf = sol{3};
xf = sol{4};

events(1:5,:) = [x0; xf(1:2)];

```

The output from *GPOPS* of the Bryson-Denham problem coded above is summarized in the following plots that contain the components of the state, the component of the costate, and the control, respectively.

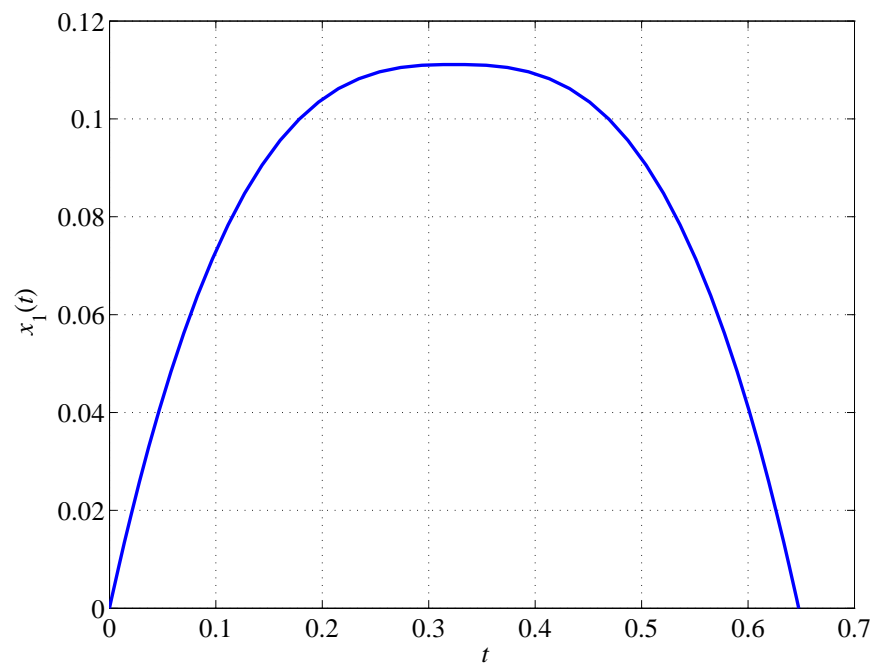


Figure 3.4 $x_1(t)$ vs. t for Bryson-Denham problem.

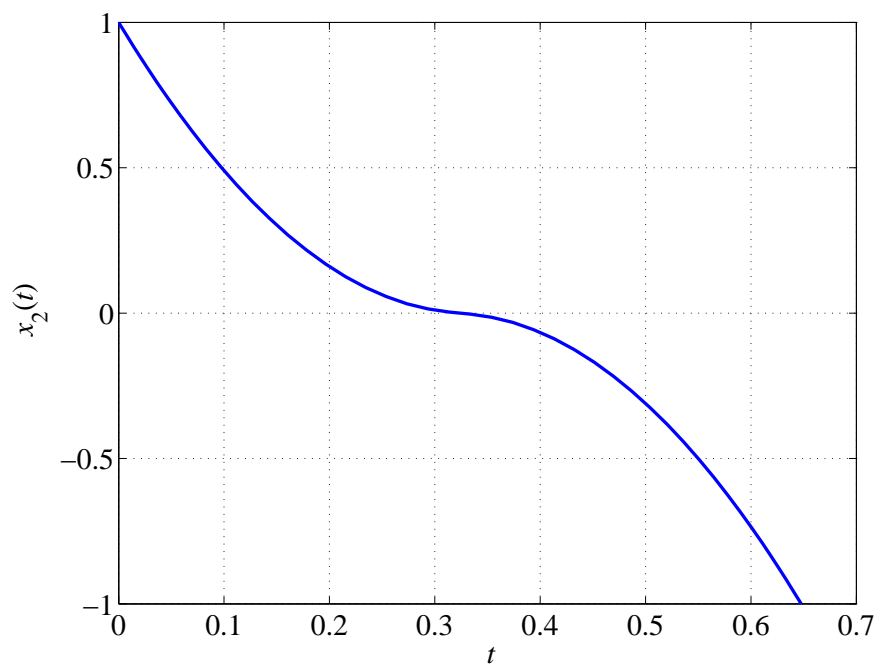


Figure 3.5 $x_2(t)$ vs. t for Bryson-Denham problem.

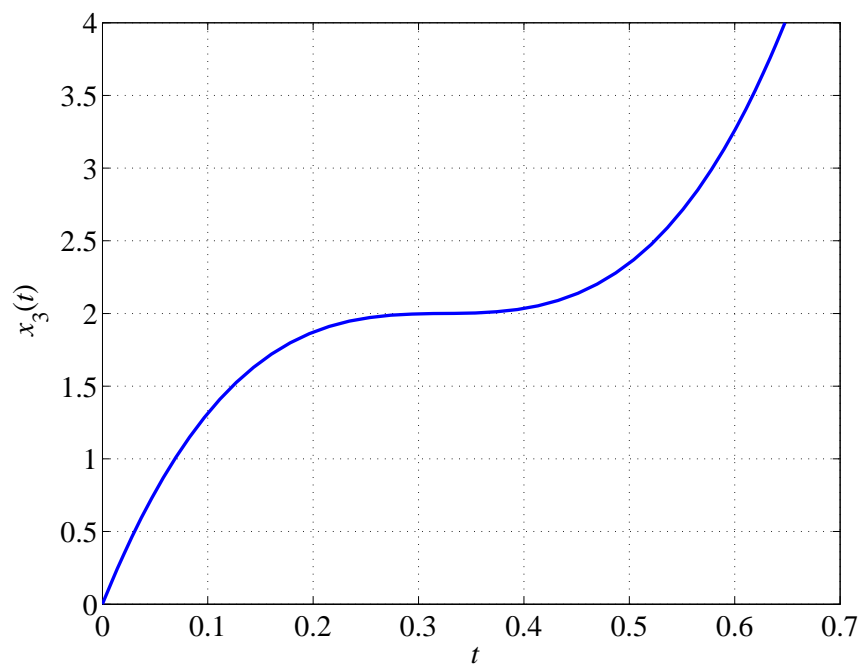


Figure 3.6 $x_3(t)$ vs. t for Bryson-Denham problem.

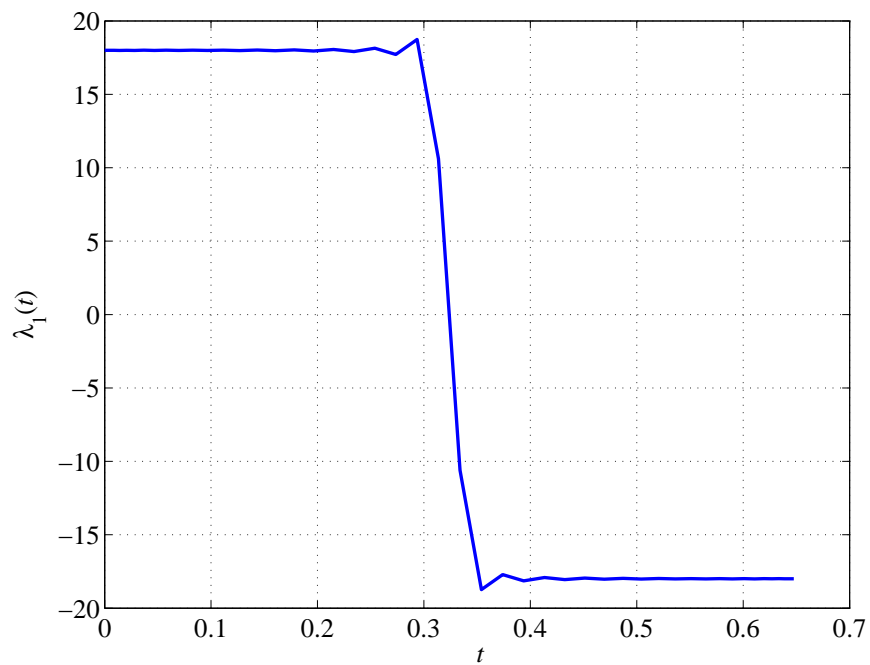


Figure 3.7 $\lambda_1(t)$ vs. t for Bryson-Denham problem.

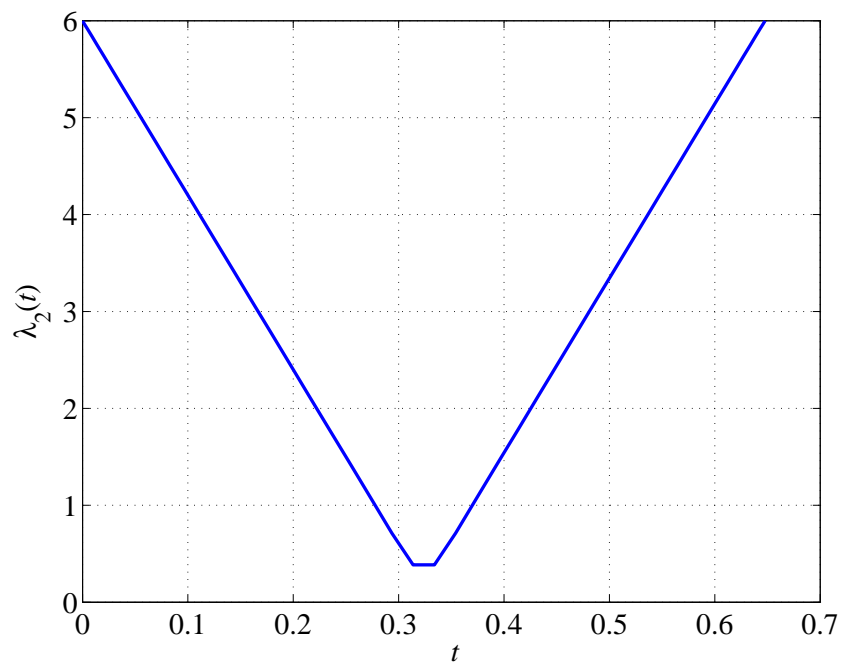


Figure 3.8 $\lambda_2(t)$ vs. t for Bryson-Denham problem.

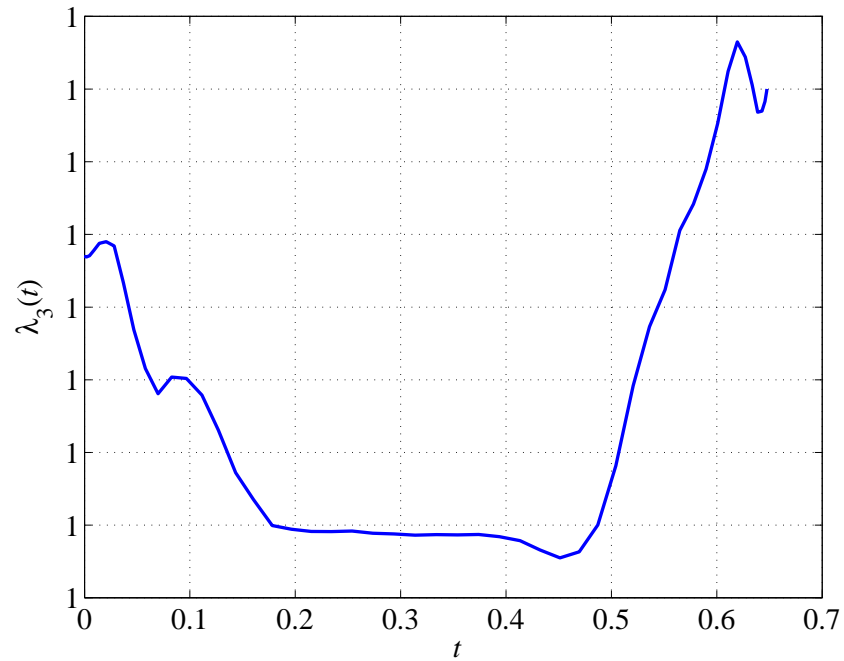


Figure 3.9 $\lambda_3(t)$ vs. t for Bryson-Denham problem.

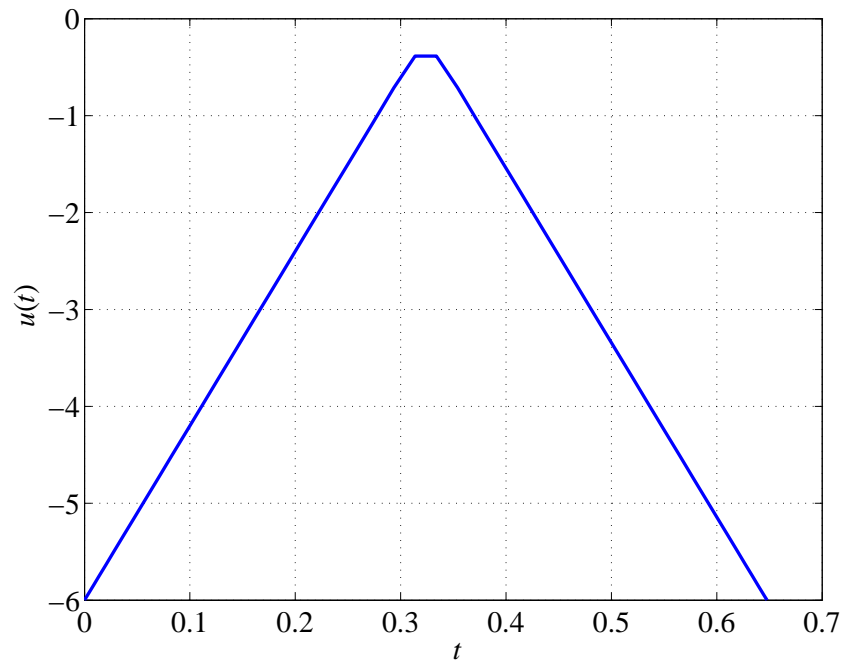


Figure 3.10 $u(t)$ vs. t for Bryson-Denham problem.

3.3 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 taken verbatim from Benson (2004).

3.3.1 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 be seen in Table 3.1. 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 3.1 Mass and propulsion properties of the launch vehicle ascent problem.

	Solid Boosters	Stage 1	Stage 2
Total Mass (kg)	19290	104380	19300
Propellant Mass (kg)	17010	95550	16820
Engine Thrust (N)	628500	1083100	110094
Isp (sec)	284	301.7	462.4
Number of Engines	9	1	1
Burn Time (sec)	75.2	261	700

3.3.2 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 inertial (ECI) coordinates as

$$\begin{aligned}
 \dot{\mathbf{r}} &= \mathbf{v} \\
 \dot{\mathbf{v}} &= -\frac{\mu}{\|\mathbf{r}\|^3}\mathbf{r} + \frac{T}{m}\mathbf{u} + \frac{\mathbf{D}}{m} \\
 \dot{m} &= -\frac{T}{g_0 I_{sp}}
 \end{aligned} \tag{3-8}$$

where $\mathbf{r}(t) = [x(t) \ y(t) \ z(t)]^T$ is the position, $\mathbf{v} = [v_x(t) \ v_y(t) \ v_z(t)]^T$ 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, $\mathbf{u} = [u_x \ u_y \ u_z]^T$ is the thrust direction, and $\mathbf{D} = [D_x \ D_y \ D_z]^T$ is the drag force. The drag force is defined as

$$\mathbf{D} = -\frac{1}{2}C_D A_{ref} \rho \|\mathbf{v}_{rel}\| \mathbf{v}_{rel} \tag{3-9}$$

where C_D is the drag coefficient, A_{ref} 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

$$\mathbf{v}_{rel} = \mathbf{v} - \boldsymbol{\omega} \times \mathbf{r} \tag{3-10}$$

where $\boldsymbol{\omega}$ is the angular velocity of the Earth relative to inertial space. The atmospheric density is modeled as the exponential function

$$\rho = \rho_0 \exp[-h/h_0] \tag{3-11}$$

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_0 is the density scale height. The numerical values for these constants can be found in Table 3.2.

Table 3.2 Constants used in the launch vehicle example.

Constant	Value
Payload Mass (kg)	4164
A_{ref} (m ²)	4π
C_d	0.5
ρ_0 (kg/m ³)	1.225
h_0 (km)	7.2
t_1 (s)	75.2
t_2 (s)	150.4
t_3 (s)	261
R_e (km)	6378.14
V_E (km/s)	7.905

3.3.3 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

$$\begin{aligned} \mathbf{r}(t_0) &= \mathbf{r}_0 = \begin{bmatrix} 5605.2 & 0 & 3043.4 \end{bmatrix}^T \text{ km} \\ \mathbf{v}(t_0) &= \mathbf{v}_0 = \begin{bmatrix} 0 & 0.4076 & 0 \end{bmatrix}^T \text{ km/s} \\ m(t_0) &= m_0 = 301454 \text{ kg} \end{aligned} \quad (3-12)$$

which corresponds to the Cape Canaveral launch site. The terminal constraints define the target geosynchronous transfer orbit (GTO), which is defined in orbital elements as

$$\begin{aligned} a_f &= 24361.14 \text{ km}, \\ e_f &= 0.7308, \\ i_f &= 28.5 \text{ deg}, \\ \Omega_f &= 269.8 \text{ deg}, \\ \omega_f &= 130.5 \text{ deg} \end{aligned} \quad (3-13)$$

The orbital elements, a , e , i , Ω , and ω represent the semi-major axis, eccentricity, inclination, right ascension of the ascending node (RAAN), and argument of perigee, respectively. Note that the true anomaly, ν , is left undefined since the exact location within the orbit is not constrained. These orbital elements can be transformed into ECI coordinates via the transformation, T_{o2c} , where T_{o2c} is given in (Bate, et al., 2001).

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

$$\|\mathbf{r}\| \geq R_e \quad (3-14)$$

where R_e is the radius of the Earth, as seen in Table 3.2. Next, a path constraint is imposed on the control to guarantee that the control vector is unit length, so that

$$\|\mathbf{u}\| = 1 \quad (3-15)$$

Lastly, each of the four phases in this trajectory is linked to the adjoining phases by a set of linkage conditions. These constraints force the position and velocity to be continuous and also account for the mass

ejections, as

$$\begin{aligned} \mathbf{r}^{(p)}(t_f) - \mathbf{r}^{(p+1)}(t_0) &= \mathbf{0}, \\ \mathbf{v}^{(p)}(t_f) - \mathbf{v}^{(p+1)}(t_0) &= \mathbf{0}, \\ m^{(p)}(t_f) - m_{dry}^{(p)} - m^{(p+1)}(t_0) &= 0 \end{aligned} \quad (p = 1, \dots, 3) \quad (3-16)$$

where the superscript (p) represents the phase number.

The optimal control problem is then to find the control, \mathbf{u} , that minimizes the cost function

$$J = -m^{(4)}(t_f) \quad (3-17)$$

subject to the conditions of Eqs. (3-8), (3-12), (3-13), (3-14), and (3-15).

The MATLAB code that solves the multiple-stage launch vehicle ascent problem using *GPOPS* is shown below. In particular, this problem requires the specification of a function that computes the cost functional, the differential-algebraic equations (which, it is noted, include both the differential equations *and* the path constraints), and the event constraints in each phase of the problem along with the phase-connect (i.e., link-age) constraints. The problem was posed in SI units and the built-in autoscaling procedure was used.

```
% -----
% Multiple-Stage Launch Vehicle Ascent Example
% -----
%
% This example can be found in one of the following three references:
% Benson, D. A., A Gauss Pseudospectral Transcription for Optimal
% Control, Ph.D. Thesis, Department of Aeronautics and
% Astronautics, Massachusetts Institute of Technology, November 2004.
%
% Huntington, G. T. Advancement and Analysis of a Gauss
% Pseudospectral Transcription for Optimal Control, Ph.D. Thesis,
% Department of Aeronautics and Astronautics, Massachusetts
% Institute of Technology, May 2007.
%
% Huntington, G. T., Benson, D. A., Kanizay, N., Darby, C. L.,
% How, J. P., and Rao, A. V., "Computation of Boundary Controls
% Using a Gauss Pseudospectral Method," 2007 Astrodynamics
% Specialist Conference, Mackinac Island, Michigan, August 19-23, 2007.
% -----
gpopsInitialize;

global CONSTANTS

omega = 7.29211585e-5; % Earth rotation rate (rad/s)
omega_matrix = [0 -omega 0; omega 0 0; 0 0 0];
CONSTANTS.omega_matrix = omega_matrix; % Rotation rate matrix (rad/s)
CONSTANTS.mu = 3.986012e14; % Gravitational parameter (m^3/s^2)
CONSTANTS.cd = 0.5; % Drag coefficient
CONSTANTS.sa = 4*pi; % Surface area (m^2)
CONSTANTS.rho0 = 1.225; % sea level gravity (kg/m^3)
CONSTANTS.H = 7200.0; % Density scale height (m)
CONSTANTS.Re = 6378145.0; % Radius of earth (m)
CONSTANTS.g0 = 9.80665; % sea level gravity (m/s^2)

lat0 = 28.5*pi/180; % Geocentric Latitude of Cape Canaveral
x0 = CONSTANTS.Re*cos(lat0); % x component of initial position
z0 = CONSTANTS.Re*sin(lat0); % z component of initial position
y0 = 0;
r0 = [x0; y0; z0];
v0 = CONSTANTS.omega_matrix*r0;

bt_srb = 75.2;
bt_first = 261;
bt_second = 700;

t0 = 0;
```

```

t1 = 75.2;
t2 = 150.4;
t3 = 261;
t4 = 961;

m_tot_srb      = 19290;
m_prop_srb     = 17010;
m_dry_srb      = m_tot_srb-m_prop_srb;
m_tot_first    = 104380;
m_prop_first   = 95550;
m_dry_first    = m_tot_first-m_prop_first;
m_tot_second   = 19300;
m_prop_second  = 16820;
m_dry_second   = m_tot_second-m_prop_second;
m_payload      = 4164;
thrust_srb     = 628500;
thrust_first   = 1083100;
thrust_second  = 110094;
mdot_srb       = m_prop_srb/bt_srb;
ISP_srb        = thrust_srb/(CONSTANTS.g0*mdot_srb);
mdot_first     = m_prop_first/bt_first;
ISP_first      = thrust_first/(CONSTANTS.g0*mdot_first);
mdot_second    = m_prop_second/bt_second;
ISP_second     = thrust_second/(CONSTANTS.g0*mdot_second);

af = 24361140;
ef = 0.7308;
incf = 28.5*pi/180;
Omf = 269.8*pi/180;
omf = 130.5*pi/180;
nuguess = 0;
cosincf = cos(incf);
cosOmf = cos(Omf);
cosomf = cos(omf);
oe = [af ef incf Omf omf nuguess];
[rout,vout] = launchoe2rv(oe,CONSTANTS.mu);
rout = rout';
vout = vout';

m10 = m_payload+m_tot_second+m_tot_first+9*m_tot_srb;
m1f = m10-(6*mdot_srb+mdot_first)*t1;
m20 = m1f-6*m_dry_srb;
m2f = m20-(3*mdot_srb+mdot_first)*(t2-t1);
m30 = m2f-3*m_dry_srb;
m3f = m30-mdot_first*(t3-t2);
m40 = m3f-m_dry_first;
m4f = m_payload;

r0 = r0;
v0 = v0;
t0 = t0;
t1 = t1;
t2 = t2;
t3 = t3;
t4 = t4;
dt = 10;
m_tot_srb      = m_tot_srb;
m_prop_srb     = m_prop_srb;
m_dry_srb      = m_dry_srb;
m_tot_first    = m_tot_first;
m_prop_first   = m_prop_first;
m_dry_first    = m_dry_first;
m_tot_second   = m_tot_second;
m_prop_second  = m_prop_second;
m_dry_second   = m_dry_second;
m_payload      = m_payload;
thrust_srb     = thrust_srb;

```

```

thrust_first = thrust_first;
thrust_second = thrust_second;
ISP_srb = ISP_srb;
ISP_first = ISP_first;
ISP_second = ISP_second;

CONSTANTS.thrust_srb = thrust_srb;
CONSTANTS.thrust_first = thrust_first;
CONSTANTS.thrust_second = thrust_second;
CONSTANTS.ISP_srb = ISP_srb;
CONSTANTS.ISP_first = ISP_first;
CONSTANTS.ISP_second = ISP_second;

m10 = m10;
m1f = m1f;
m20 = m20;
m2f = m2f;
m30 = m30;
m3f = m3f;
m40 = m40;
m4f = m4f;
af = af;

rmin = -2*CONSTANTS.Re;
rmax = -rmin;
vmin = -10000;
vmax = -vmin;

iphase = 1;
limits{imin}{iphase,itime} = [t0 t1];
limits{imax}{iphase,itime} = [t0 t1];
limits{imin}{iphase,istate}(1,:) = [r0(1) rmin rmin];
limits{imax}{iphase,istate}(1,:) = [r0(1) rmax rmax];
limits{imin}{iphase,istate}(2,:) = [r0(2) rmin rmin];
limits{imax}{iphase,istate}(2,:) = [r0(2) rmax rmax];
limits{imin}{iphase,istate}(3,:) = [r0(3) rmin rmin];
limits{imax}{iphase,istate}(3,:) = [r0(3) rmax rmax];
limits{imin}{iphase,istate}(4,:) = [v0(1) vmin vmin];
limits{imax}{iphase,istate}(4,:) = [v0(1) vmax vmax];
limits{imin}{iphase,istate}(5,:) = [v0(2) vmin vmin];
limits{imax}{iphase,istate}(5,:) = [v0(2) vmax vmax];
limits{imin}{iphase,istate}(6,:) = [v0(3) vmin vmin];
limits{imax}{iphase,istate}(6,:) = [v0(3) vmax vmax];
limits{imin}{iphase,istate}(7,:) = [m10 m1f m1f];
limits{imax}{iphase,istate}(7,:) = [m10 m10 m10];
limits{imin}{iphase,icontrol}(1,:) = -1;
limits{imax}{iphase,icontrol}(1,:) = 1;
limits{imin}{iphase,icontrol}(2,:) = -1;
limits{imax}{iphase,icontrol}(2,:) = 1;
limits{imin}{iphase,icontrol}(3,:) = -1;
limits{imax}{iphase,icontrol}(3,:) = 1;
limits{imin}{iphase,4} = [];
limits{imax}{iphase,4} = [];
limits{imin}{iphase,5} = 1;
limits{imax}{iphase,5} = 1;
solinit{iphase,1} = [t0; t1];
solinit{iphase,istate}(:,1) = [r0(1); r0(1)];
solinit{iphase,istate}(:,2) = [r0(2); r0(2)];
solinit{iphase,istate}(:,3) = [r0(3); r0(3)];
solinit{iphase,istate}(:,4) = [v0(1); v0(1)];
solinit{iphase,istate}(:,5) = [v0(2); v0(2)];
solinit{iphase,istate}(:,6) = [v0(3); v0(3)];
solinit{iphase,istate}(:,7) = [m10; m1f];
solinit{iphase,icontrol}(:,1) = [1; 1];
solinit{iphase,icontrol}(:,2) = [0; 0];
solinit{iphase,icontrol}(:,3) = [0; 0];
solinit{iphase,4} = [];

```

```

iphase = 2;
limits{imin}{iphase,itime} = [t1 t2];
limits{imax}{iphase,itime} = [t1 t2];
limits{imin}{iphase,istate}(1,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(1,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(2,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(2,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(3,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(3,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(4,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(4,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(5,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(5,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(6,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(6,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(7,:) = [m2f m2f m2f];
limits{imax}{iphase,istate}(7,:) = [m20 m20 m20];
limits{imin}{iphase,icontrol}(1,:) = -1;
limits{imax}{iphase,icontrol}(1,:) = 1;
limits{imin}{iphase,icontrol}(2,:) = -1;
limits{imax}{iphase,icontrol}(2,:) = 1;
limits{imin}{iphase,icontrol}(3,:) = -1;
limits{imax}{iphase,icontrol}(3,:) = 1;
limits{imin}{iphase,4} = [];
limits{imax}{iphase,4} = [];
limits{imin}{iphase,5} = 1;
limits{imax}{iphase,5} = 1;
solinit{iphase,1} = [t1; t2];
solinit{iphase,istate}(:,1) = [r0(1); r0(1)];
solinit{iphase,istate}(:,2) = [r0(2); r0(2)];
solinit{iphase,istate}(:,3) = [r0(3); r0(3)];
solinit{iphase,istate}(:,4) = [v0(1); v0(1)];
solinit{iphase,istate}(:,5) = [v0(2); v0(2)];
solinit{iphase,istate}(:,6) = [v0(3); v0(3)];
solinit{iphase,istate}(:,7) = [m20; m2f];
solinit{iphase,icontrol}(:,1) = [1; 1];
solinit{iphase,icontrol}(:,2) = [0; 0];
solinit{iphase,icontrol}(:,3) = [0; 0];
solinit{iphase,4} = [];

iphase = 3;
limits{imin}{iphase,1} = [t2 t3];
limits{imax}{iphase,1} = [t2 t3];
limits{imin}{iphase,istate}(1,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(1,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(2,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(2,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(3,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(3,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(4,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(4,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(5,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(5,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(6,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(6,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(7,:) = [m3f m3f m3f];
limits{imax}{iphase,istate}(7,:) = [m30 m30 m30];
limits{imin}{iphase,icontrol}(1,:) = -1;
limits{imax}{iphase,icontrol}(1,:) = 1;
limits{imin}{iphase,icontrol}(2,:) = -1;
limits{imax}{iphase,icontrol}(2,:) = 1;
limits{imin}{iphase,icontrol}(3,:) = -1;
limits{imax}{iphase,icontrol}(3,:) = 1;
limits{imin}{iphase,4} = []; % No parameters
limits{imax}{iphase,4} = []; % No parameters
limits{imin}{iphase,5} = 1; % One Path Constraint

```

```

limits{imax}{iphase,5} = 1; % One Path Constraint
solinit{iphase,1} = [t2; t3];
solinit{iphase,istate}(:,1) = [rout(1); rout(1)];
solinit{iphase,istate}(:,2) = [rout(2); rout(2)];
solinit{iphase,istate}(:,3) = [rout(3); rout(3)];
solinit{iphase,istate}(:,4) = [vout(1); vout(1)];
solinit{iphase,istate}(:,5) = [vout(2); vout(2)];
solinit{iphase,istate}(:,6) = [vout(3); vout(3)];
solinit{iphase,istate}(:,7) = [m30; m3f];
solinit{iphase,icontrol}(:,1) = [0; 0];
solinit{iphase,icontrol}(:,2) = [0; 0];
solinit{iphase,icontrol}(:,3) = [1; 1];
solinit{iphase,4} = [];

iphase = 4;
limits{imin}{iphase,1} = [t3 t3];
limits{imax}{iphase,1} = [t3 t4];
limits{imin}{iphase,istate}(1,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(1,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(2,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(2,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(3,:) = [rmin rmin rmin];
limits{imax}{iphase,istate}(3,:) = [rmax rmax rmax];
limits{imin}{iphase,istate}(4,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(4,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(5,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(5,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(6,:) = [vmin vmin vmin];
limits{imax}{iphase,istate}(6,:) = [vmax vmax vmax];
limits{imin}{iphase,istate}(7,:) = [m4f m4f m4f];
limits{imax}{iphase,istate}(7,:) = [m40 m40 m40];
limits{imin}{iphase,icontrol}(1,:) = -1;
limits{imax}{iphase,icontrol}(1,:) = 1;
limits{imin}{iphase,icontrol}(2,:) = -1;
limits{imax}{iphase,icontrol}(2,:) = 1;
limits{imin}{iphase,icontrol}(3,:) = -1;
limits{imax}{iphase,icontrol}(3,:) = 1;
limits{imin}{iphase,4} = [];
limits{imax}{iphase,4} = [];
limits{imin}{iphase,5} = 1;
limits{imax}{iphase,5} = 1;
limits{imin}{iphase,6} = [af; ef; incf; Omf; omf];
limits{imax}{iphase,6} = [af; ef; incf; Omf; omf];
solinit{iphase,1} = [t3; t4];
solinit{iphase,istate}(:,1) = [rout(1) rout(1)];
solinit{iphase,istate}(:,2) = [rout(2) rout(2)];
solinit{iphase,istate}(:,3) = [rout(3) rout(3)];
solinit{iphase,istate}(:,4) = [vout(1) vout(1)];
solinit{iphase,istate}(:,5) = [vout(2) vout(2)];
solinit{iphase,istate}(:,6) = [vout(3) vout(3)];
solinit{iphase,istate}(:,7) = [m40; m4f];
solinit{iphase,icontrol}(:,1) = [0; 0];
solinit{iphase,icontrol}(:,2) = [0; 0];
solinit{iphase,icontrol}(:,3) = [1; 1];
solinit{iphase,4} = [];

iconnmin = 1; % Lower bounds on connection conditions
iconnmax = 2; % Upper bounds on connection conditions
iconnect = 3; % Index correspond to phases that need to be connected.

ipair = 1; % First pair of phases to connect
connections{ipair,iconnect} = [1 2];
connections{ipair,iconnmin} = [0; 0; 0; 0; 0; 0; 0; -6*m_dry_srb];
connections{ipair,iconnmax} = [0; 0; 0; 0; 0; 0; 0; -6*m_dry_srb];

ipair = 2; % Second pair of phases to connect
connections{ipair,iconnect} = [2 3];

```

```

connections{ipair,iconnmin} = [0; 0; 0; 0; 0; 0; 0; -3*m_dry_srb];
connections{ipair,iconnmax} = [0; 0; 0; 0; 0; 0; 0; -3*m_dry_srb];

ipair = 3; % Third pair of phases to connect
connections{3,iconnect} = [3 4];
connections{3,iconnmin} = [0; 0; 0; 0; 0; 0; 0; -m_dry_first];
connections{3,iconnmax} = [0; 0; 0; 0; 0; 0; 0; -m_dry_first];

nodes = [15 15 15 15];
setup.autoscale = 'on';
setup.name = 'Launch-Vehicle-Ascent';
setup.funcs = {'launchCost','launchDae','launchEvent','launchConnect'};
setup.derivatives = 'complex';
setup.direction = 'increasing';
setup.nodes = nodes;
setup.limits = limits;
setup.connections = connections;
setup.solinit = solinit;

output = gpops(setup);
solution = output.solution;

function dae = launchDae(sol,iphase);

global CONSTANTS;
t = sol{1};
x = sol{2};
u = sol{3};
p = sol{4};
r = x(:,1:3);
v = x(:,4:6);
m = x(:,7);
zz = zeros(size(t));
rad = sqrt(sum(r.*r,2));
omega_matrix = CONSTANTS.omega_matrix;
% omega_matrix = [0 -CONSTANTS.omega 0; CONSTANTS.omega 0 0; 0 0 0];
vrel = v-r*omega_matrix.';
speedrel = sqrt(sum(vrel.*vrel,2));
altitude = rad-CONSTANTS.Re;
rho = CONSTANTS.rho0*exp(-altitude/CONSTANTS.H);
a1 = CONSTANTS.rho0*CONSTANTS.sa*CONSTANTS.cd;
a2 = a1*exp(-altitude/CONSTANTS.H);
bc = a2.*(rho./(2*m));
bc = (rho./(2*m)).*CONSTANTS.sa*CONSTANTS.cd;

bcspeedmat = [zz zz zz];
bcspeed = bc.*speedrel;
bcspeedmat = [bcspeed bcspeed bcspeed];
Drag = [zz zz zz];
Drag = -bcspeedmat.*vrel;
muoverradcubedmat = [zz zz zz];
muoverradcubed = CONSTANTS.mu./rad.^3;
muoverradcubedmat = [muoverradcubed muoverradcubed muoverradcubed];
grav = [zz zz zz];
grav = -muoverradcubedmat.*r;

if iphase==1,
    T_srb = 6*CONSTANTS.thrust_srb*ones(size(t));
    T_first = CONSTANTS.thrust_first*ones(size(t));
    T_tot = T_srb+T_first;
    m1dot = -T_srb./(CONSTANTS.g0*CONSTANTS.ISP_srb);
    m2dot = -T_first./(CONSTANTS.g0*CONSTANTS.ISP_first);
    mdot = m1dot+m2dot;
elseif iphase==2,
    T_srb = 3*CONSTANTS.thrust_srb*ones(size(t));
    T_first = CONSTANTS.thrust_first*ones(size(t));
    T_tot = T_srb+T_first;

```

```

        m1dot = -T_srb./ (CONSTANTS.g0*CONSTANTS.ISP_srb);
        m2dot = -T_first./ (CONSTANTS.g0*CONSTANTS.ISP_first);
        mdot = m1dot+m2dot;
elseif iphase==3
    T_first = CONSTANTS.thrust_first*ones(size(t));
    T_tot = T_first;
    mdot = -T_first./ (CONSTANTS.g0*CONSTANTS.ISP_first);
elseif iphase==4,
    T_second = CONSTANTS.thrust_second*ones(size(t));
    T_tot = T_second;
    mdot = -T_second./ (CONSTANTS.g0*CONSTANTS.ISP_second);
end;

path = sum(u.*u,2);
Tovermmat = [zz zz zz];
Toverm = T_tot./m;
Tovermmat = [Toverm Toverm Toverm];
thrust = Tovermmat.*u;

rdot = v;
vdot = thrust+Drag+grav;

dae(:,8) = path;
dae(:,1:7) = [rdot vdot mdot];

function [Mayer,Lagrange] = launchCost(sol,iphase);

t0 = sol{1,1};
x0 = sol{1,2};
tf = sol{1,3};
xf = sol{1,4};
t = sol{2,1};
x = sol{2,2};
u = sol{2,3};
p = sol{2,4};

Lagrange = zeros(size(t));
if iphase==4,
    Mayer = -xf(7);
else
    Mayer = zeros(size(t0));
end;

function events = launchEvent(sol,iphase);

global CONSTANTS
t0 = sol{1};
x0 = sol{2};
tf = sol{3};
xf = sol{4};
p = sol{5};

if iphase==4,
    oe = launchrv2oe(xf(1:3),xf(4:6),CONSTANTS.mu);
    events= zeros(size(x0(1:5)));
    events(1:5) = oe(1:5);
end;

```

The output of the above code from *GPOPS* is summarized in the following three plots that contain the altitude, speed, and controls.

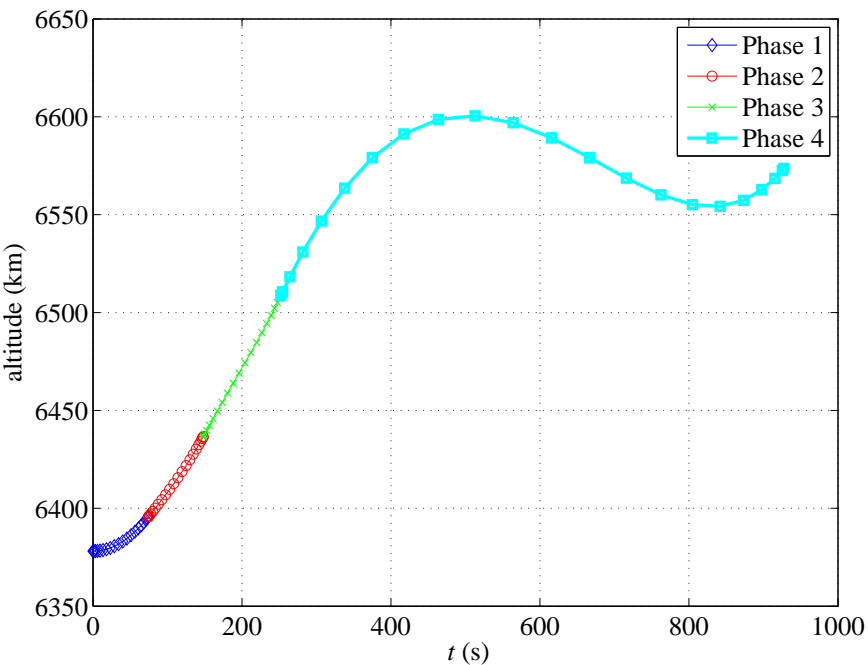


Figure 3.11 Altitude vs. time for the launch vehicle ascent problem.

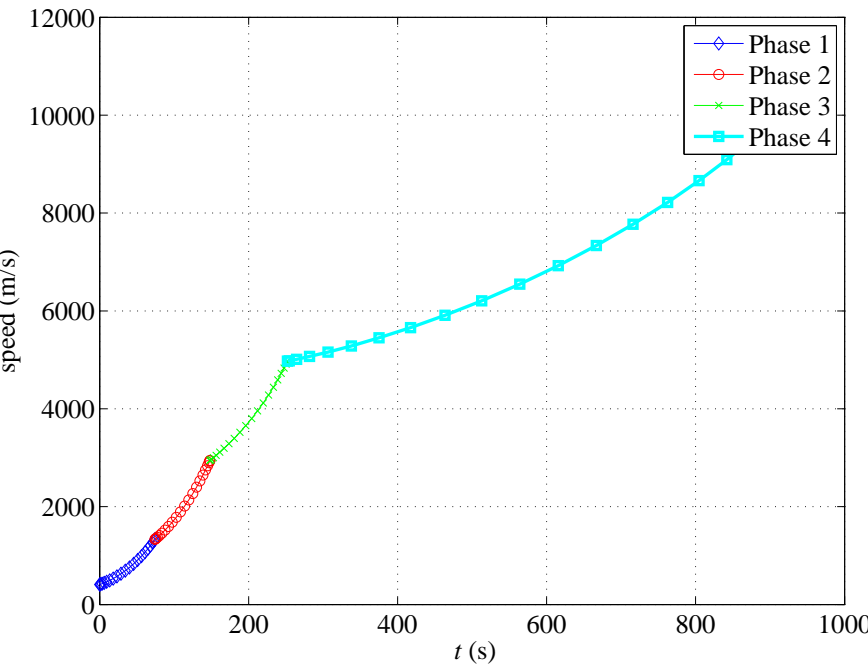


Figure 3.12 Inertial speed vs. time for the launch vehicle ascent problem.

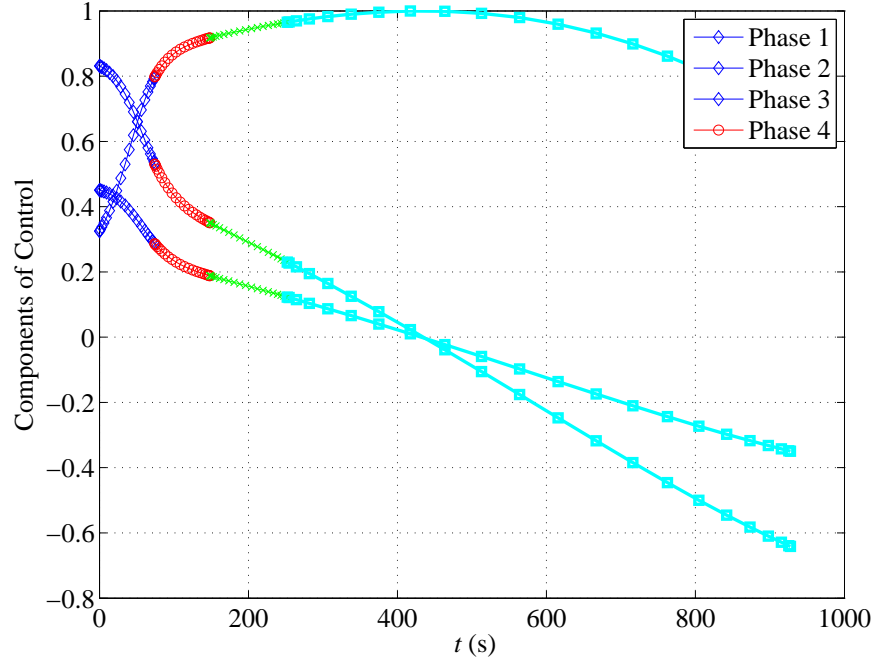


Figure 3.13 Controls vs. time for the launch vehicle ascent problem.

3.4 Minimum Time-to-Climb of a 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 Bryson, et al. (1969), but the model used in this study was taken from Betts (2001) with the exception that a linear extrapolation of the thrust data as found in Betts (2001) 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

$$J = t_f \quad (3-18)$$

subject to the dynamic constraints

$$\dot{h} = v \sin \gamma \quad (3-19)$$

$$\dot{v} = \frac{T \cos \alpha - D}{m} - \frac{\mu}{r^2} \sin \gamma \quad (3-20)$$

$$\dot{\gamma} = \frac{T \sin \alpha + L}{mv} + \cos \gamma \left(\frac{v}{r} - \frac{\mu}{vr^2} \right) \quad (3-21)$$

$$\dot{m} = -\frac{T}{g_0 I_{sp}} \quad (3-22)$$

and the boundary conditions

$$h(0) = 0 \text{ ft} \quad (3-23)$$

$$v(0) = 424.26 \text{ ft/s} \quad (3-24)$$

$$\gamma(0) = 0 \text{ rad} \quad (3-25)$$

$$m(0) = 1304.024 \text{ slug} \quad (3-26)$$

$$h(t_f) = 65600 \text{ ft} \quad (3-27)$$

$$v(t_f) = 968.148 \text{ ft/s} \quad (3-28)$$

$$\gamma(t_f) = 0 \text{ rad} \quad (3-29)$$

$$m(t_f) = \text{free} \quad (3-30)$$

where h is the altitude, v is the speed, γ is the flight path angle, m is the vehicle mass, α is the angle of attack, $r = R_e + h$ is the geocentric radius, R_e is the radius of the Earth, T is the magnitude of the thrust force, D is the magnitude of the drag force, L is the magnitude of the lift force, g_0 is the sea level acceleration due to gravity, μ is the gravitational parameter of the Earth, and I_{sp} is the engine specific impulse. The thrust and aerodynamic coefficients are computed using the thrust tables provided in (Bryson, et al., 1969), but the details are not provided here.

The MATLAB code that solves the minimum time-to-climb of a supersonic aircraft is shown below. In particular, this problem requires the specification of a function that computes the cost functional and differential equations. In addition, it is necessary to interpolate the thrust and aerodynamic data in order to obtain the solution. The code below illustrates one important feature of *GPOPS*, namely, that it is possible to use Matlab Automatic Differentiation (MAD) with table data when the “spline” option is given in the table. This feature is important because the ability to use tabular data alleviates the user from having to provide an analytic fit of the data. It is noted that some of the data is not provided in the MATLAB code below, but is given in MATLAB binary files included in the *GPOPS* distribution.

```
% Minimum Time-to-Climb Problem
% The vehicle model for this problem is taken
% from the following two references:
% Seywald, H., Cliffs, E. M., and Well, K. H.,
% "Range Optimal Trajectories for an Aircraft Flying in
% the Vertical Plane," Journal of Guidance, Control, and Dynamics,
% Vol. 17, No. 2, March-April, 1994.
%
% Rao, A. V., Extension of a Computational Singular Perturbation
% Methodology to Optimal Control Problems, Ph.D. Thesis, Dept. of
% Mechanical and Aerospace Engineering, Princeton University,
% June 1996.

clear all
clc
global CoF;

gpopsInitialize;
g = 9.80665;
m = 37000/2.2;
Coefficients;
feettometer = .3048;

h0 = 0*feettometer;
hf = 65600*feettometer;
v0 = 424.26*feettometer;
vf = 968.148*feettometer;
e0 = (v0^2/(2*g)+h0);
ef = (vf^2/(2*g)+hf);
fpa0 = 0;
fpaf = 0;

hmin = 0*feettometer;
```

```

hmax = 69000*feettometer;
vmin = 1*feettometer;
vmax = 2000*feettometer;
emin = (vmin^2/(2*g)+hmin);
emax = (vmax^2/(2*g)+hmax);
fpamin = -40/180*pi;
fpamax = -fpamin;
umin = -10;
umax = 10;
t0min = 0;
t0max = 0;
tfmin = 100;
tfmax = 350;

% Phase 1 Information
iphase = 1;
limits{imin}{iphase,itime} = [t0min tfmin];
limits{imax}{iphase,itime} = [t0max tfmax];
limits{imin}{iphase,istate}(1,:) = [h0 hmin hf];
limits{imax}{iphase,istate}(1,:) = [h0 hmax hf];
limits{imin}{iphase,istate}(2,:) = [e0 emin ef];
limits{imax}{iphase,istate}(2,:) = [e0 emax ef];
limits{imin}{iphase,istate}(3,:) = [fpa0 fpamin fpaf];
limits{imax}{iphase,istate}(3,:) = [fpa0 fpamax fpaf];
limits{imin}{iphase,icontrol} = umin;
limits{imax}{iphase,icontrol} = umax;
limits{imin}{iphase,iparameter} = [];
limits{imax}{iphase,iparameter} = [];
limits{imin}{iphase,ipath} = [];
limits{imax}{iphase,ipath} = [];
limits{imin}{iphase,ievent} = [];
limits{imax}{iphase,ievent} = [];
limits{imin}{iphase,iduration} = [];
limits{imax}{iphase,iduration} = [];
solinit{iphase,itime} = [t0min; tfmax];
solinit{iphase,istate}(:,1) = [hf/2; hf/2];
solinit{iphase,istate}(:,2) = [ef/2; ef/2];
solinit{iphase,istate}(:,3) = [fpamax/2; fpamax/2];
solinit{iphase,3} = [0; 0];
solinit{iphase,4} = []; % No parameters in Phase 1

connections = [];
setup.name = 'Minimum-Time-to-Climb-Problem';
setup.funcs{1} = 'minimumClimbCost';
setup.funcs{2} = 'minimumClimbDae';
setup.funcs{3} = 'mininumClimbEvent';
setup.funcs{4} = 'minimumClimbConnect';
setup.nodes = 50;
setup.limits = limits;
setup.solinit = solinit;
setup.connections = connections;
setup.derivatives = 'complex';
setup.direction = 'increasing';
setup.autoscale = 'on';

output = gpops(setup);

solution = output.solution;

function daeout = minimumClimbDae(soldae,iphase);
global CoF;

t = soldae{1};
s = soldae{2};
u = soldae{3};
p = soldae{4};

```

```

h = s(:,1);
E = s(:,2);
fpa = s(:,3);

g = 9.80665;
m = 37000./2.2;
S = 60;
hbar = h./1000;

%rho calculation
z = (-3.48643241e-2).*hbar+(3.50991865e-3).*hbar.^2-(8.33000535e-5).*hbar.^3+(1.15219733e-6).*hbar.^4;
r = 1.0228066.*exp(-z);
y = -0.12122693.*hbar+r-1.0228055;
rho = 1.225.*exp(y);
%rho calculation

%speed of sound%
theta = 292.1-8.87743.*hbar+0.193315.*hbar.^2+(3.72e-3).*hbar.^3;
a = 20.0468.*sqrt(theta);
%speed of sound%

%Velocity and mach
Eminush = (E-h).^2;
Eminush = sqrt(Eminush);
v = sqrt(2.*g.*Eminush);

M = v./a;
%Velocity and mach

%Who are lift and drag
q = 0.5.*rho.*v.*v.*S;
L = m.*g.*u;
numeratorCD0 = CoF(1,1).*M.^0+CoF(1,2).*M.^1+CoF(1,3).*M.^2+CoF(1,4).*M.^3+CoF(1,5).*M.^4;
denominatorCD0 = CoF(2,1).*M.^0+CoF(2,2).*M.^1+CoF(2,3).*M.^2+CoF(2,4).*M.^3+CoF(2,5).*M.^4;
Cd0 = numeratorCD0./denominatorCD0;
numeratorK = CoF(3,1).*M.^0+CoF(3,2).*M.^1+CoF(3,3).*M.^2+CoF(3,4).*M.^3+CoF(3,5).*M.^4;
denominatorK = CoF(4,1).*M.^0+CoF(4,2).*M.^1+CoF(4,3).*M.^2+CoF(4,4).*M.^3+CoF(4,5).*M.^4+CoF(4,6).*M.^5;
K = numeratorK./denominatorK;
D = q.*(Cd0+K.*(m.^2).*(g.^2)./(q.^2)).*(u.^2));
%Who are lift and drag

%Who is thrust
e0 = CoF(5,1).*M.^0+CoF(6,1).*M.^1+CoF(7,1).*M.^2+CoF(8,1).*M.^3+CoF(9,1).*M.^4+CoF(10,1).*M.^5;
e1 = CoF(5,2).*M.^0+CoF(6,2).*M.^1+CoF(7,2).*M.^2+CoF(8,2).*M.^3+CoF(9,2).*M.^4+CoF(10,2).*M.^5;
e2 = CoF(5,3).*M.^0+CoF(6,3).*M.^1+CoF(7,3).*M.^2+CoF(8,3).*M.^3+CoF(9,3).*M.^4+CoF(10,3).*M.^5;
e3 = CoF(5,4).*M.^0+CoF(6,4).*M.^1+CoF(7,4).*M.^2+CoF(8,4).*M.^3+CoF(9,4).*M.^4+CoF(10,4).*M.^5;
e4 = CoF(5,5).*M.^0+CoF(6,5).*M.^1+CoF(7,5).*M.^2+CoF(8,5).*M.^3+CoF(9,5).*M.^4+CoF(10,5).*M.^5;
e5 = CoF(5,6).*M.^0+CoF(6,6).*M.^1+CoF(7,6).*M.^2+CoF(8,6).*M.^3+CoF(9,6).*M.^4+CoF(10,6).*M.^5;

T = (e0.*hbar.^0+e1.*hbar.^1+e2.*hbar.^2+e3.*hbar.^3+e4.*hbar.^4+e5.*hbar.^5).*9.80665./2.2;
%Who is thrust

%*****

%User Input - Give me f in dot(x) = f
%*****

Edot = v./(m.*g).*(T-D);
hdot = v.*sin(fpa);
fpadot = g./v.*(L./(m.*g)-cos(fpa));

daeout = [hdot Edot fpadot];

function [Mayer,Lagrange]=minimumClimbCost(solcost,iphas);

```

```

t0 = solcost{1,1};
s0 = solcost{1,2};
tf = solcost{1,3};
sf = solcost{1,4};
t  = solcost{2,1};
s  = solcost{2,2};
u  = solcost{2,3};
p  = solcost{2,4};

Mayer = tf;
Lagrange = zeros(size(t));

```

The output of the above code from *GPOPS* is summarized in the following three plots that contain the altitude, speed, flight path angle, and angle of attack:

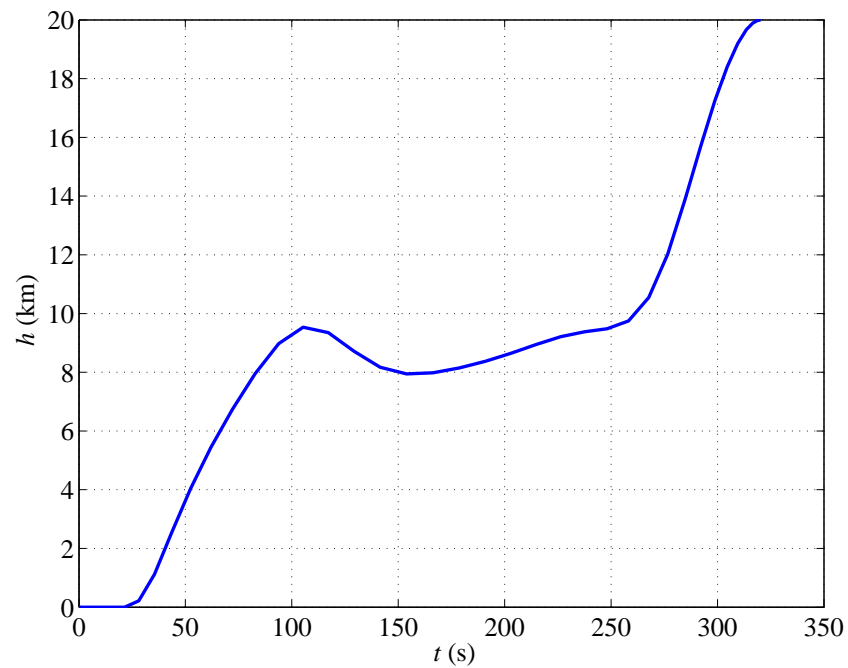


Figure 3.14 Altitude vs. Time for supersonic aircraft minimum time-to-climb.

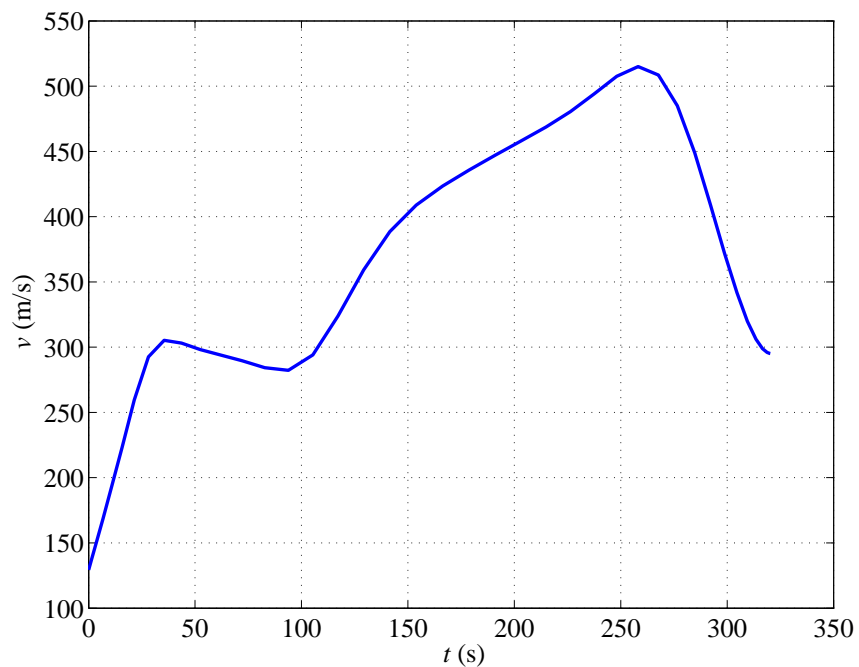


Figure 3.15 Speed vs. Time for supersonic aircraft minimum time-to-climb.

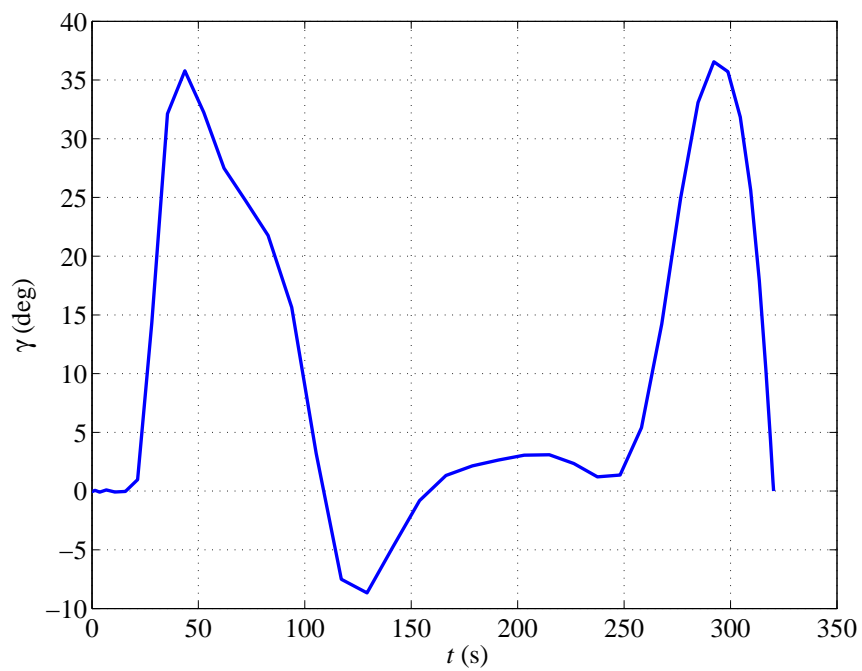


Figure 3.16 Flight path angle vs. Time for supersonic aircraft minimum time-to-climb.

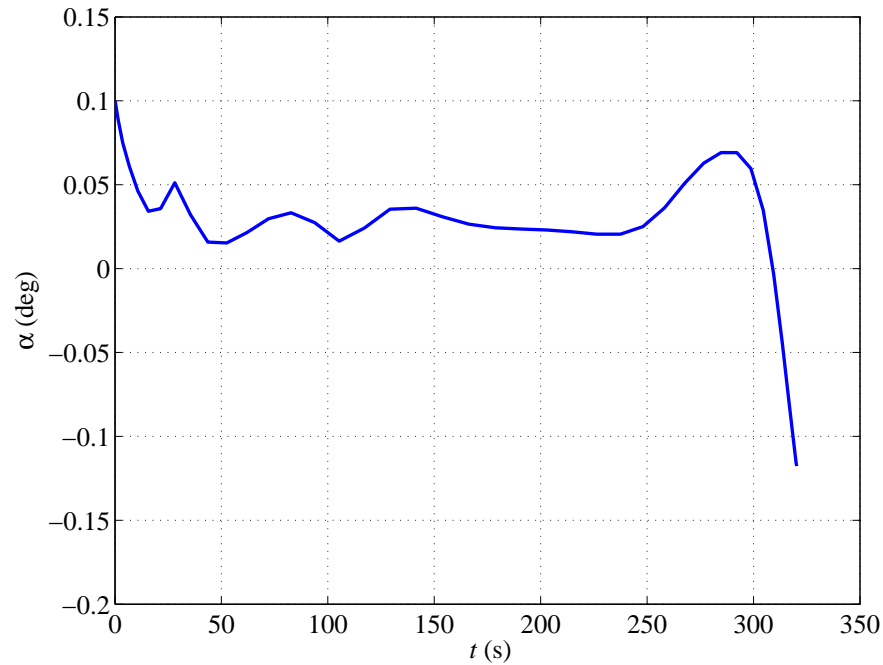


Figure 3.17 Angle of attack vs. Time for supersonic aircraft minimum time-to-climb.

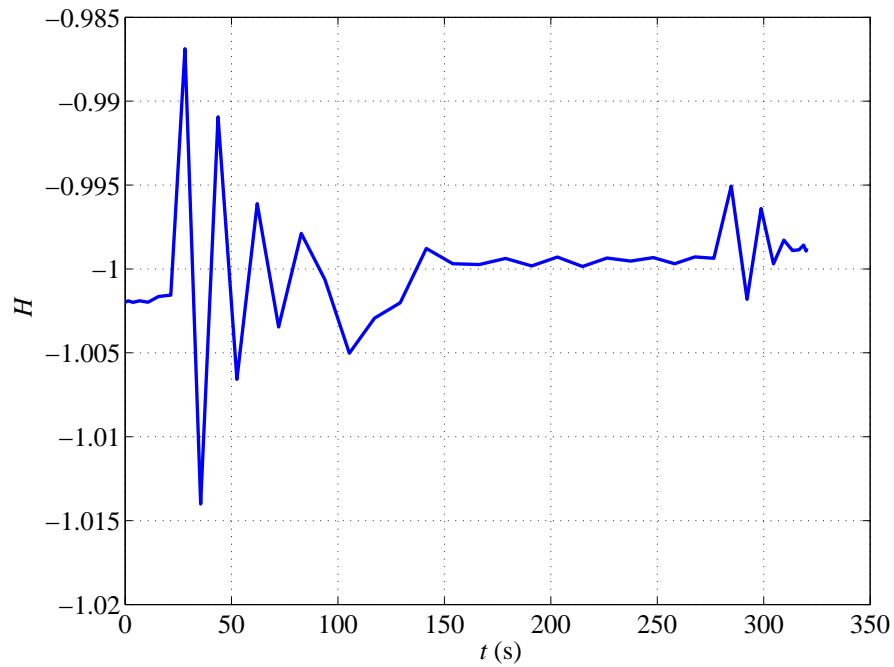


Figure 3.18 Hamiltonian vs. time for supersonic aircraft minimum time-to-climb.

3.5 Some Concluding Remarks

While *GP-OPS* has been designed to take some of the cumbersomeness out of solving an optimal control problem numerically, the user must still be wary of several aspects of computational optimal control that

will make it easier to use *GPOPS*. First, as noted earlier, it is *highly* recommended that the user scale the problem manually because the automatic scaling procedure is by no means foolproof. Second, the choice of variables to solve an optimal control problem can make all the difference in the world as to how quickly and reliably a solution is obtained. For example, atmospheric flight problems with large lift maneuvers tend to be easier to solve if spherical coordinates (where position is parameterized using radius, longitude, and latitude, while velocity is parameterized using speed, flight path angle, and heading angle) are used as compared to Cartesian coordinates whereas launch vehicle ascent problems (which have no lift) tend to be better parameterized using Cartesian coordinates. Finally, even if the optimizer returns the result that the optimality conditions have been satisfied, it is extremely important to analyze the solution to make sure that (1) the solution is the one corresponding to the problem that the user wants to solve and (2) if the solution makes sense. In short, a great deal of time in solving optimal control problems is spent in formulation and analysis.

References

- Gill, P. E., Murray, W., and Saunders, M. A., "User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming," Report, University of California, San Diego, 24 April 2007.
- Benson, D. A., *A Gauss Pseudospectral Transcription for Optimal Control*, Ph.D. Dissertation, Department of Aeronautics and Astronautics, MIT, November 2004.
- Benson, D. A., Huntington, G. T., Thorvaldsen, T. P., and Rao, A. V., "Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, November-December, 2006, pp. 1435–1440.
- Huntington, G. T. and Rao, A. V., "Optimal Spacecraft Formation Configuration Using a Gauss Pseudospectral Method," *Proceedings of the 2005 AAS/AISS Spaceflight Mechanics Meeting*, AAS Paper 05-103, Copper Mountain, Colorado, January 23–27, 2005.
- Huntington, G. T., Benson, D. A., and Rao, A. V., "Post-Optimality Evaluation and Analysis of a Formation Flying Problem via a Gauss Pseudospectral Method," *Proceedings of the 2005 AAS/AIAA Astrodynamics Specialist Conference*, AAS Paper 05-339, Lake Tahoe, California, August 7–11, 2005.
- Huntington, G. T. and Rao, A. V., "Optimal Reconfiguration of a Spacecraft Formation via a Gauss Pseudospectral Method," *Proceedings of the 2005 AAS/AIAA Astrodynamics Specialist Conference*, AAS Paper 05-338, Lake Tahoe, California, August 7–11, 2005.
- Huntington, G. T. and Rao, A. V., "Design of Optimal Tetrahedral Spacecraft Formations," *The Journal of the Astronautical Sciences*, to appear in 2007.
- Huntington, G. T., *Advancement and Analysis of a Gauss Pseudospectral Transcription for Optimal Control*, Ph.D. Dissertation, Department of Aeronautics and Astronautics, MIT, May 2007.
- Huntington, G. T., Benson, D. A., How, J. P., Kanizay, N., Darby, C. L., and Rao, A. V., "Computation of Boundary Controls Using a Gauss Pseudospectral Method," *2007 Astrodynamics Specialist Conference*, Mackinac Island, Michigan, AAS Paper 07-381, August 21–24, 2007.
- Huntington, G. T., Benson, D. A., and Rao, A. V., "A Comparison of Accuracy and Computational Efficiency of Three Pseudospectral Methods," *2007 Guidance, Navigation, and Control Conference*, Hilton Head, South Carolina, AIAA Paper 2007-6405, August 21–24, 2007.
- Elnagar, G., Kazemi, M., Razzaghi, M., "The Pseudospectral Legendre Method for Discretizing Optimal Control Problems," *IEEE Transactions on Automatic Control* Vol. 40, No. 10, October 1995.
- Elnagar, G. and Kazemi, M., "Pseudospectral Chebyshev Optimal Control of Constrained Nonlinear Dynamical Systems," *Computational Optimization and Applications*, Vol. 11, 1998, pp. 195–217.
- Davis, P., *Interpolation and Approximation*, Dover Publications, 1975.
- Pontryagin, L.S., Boltyanskii, V., Gamkrelidze, R., Mischenko, E., *The Mathematical Theory of Optimal Processes*, New York: Interscience, 1962.
- Kirk, D. E., *Optimal Control Theory*, Dover Publications, 1970.

- Betts, J. T., *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM Press, Philadelphia, 2001.
- Rao, A. V. and Mease, K. D., "Eigenvector Approximate Dichotomic Basis Method for Solving Hyper-Sensitive Optimal Control Problems," *Optimal Control Applications and Methods*, Vol. 21, No. 1, 2000, pp. 1–19.
- Bate, R. R., Mueller, D. D., and White, J. E., *Fundamentals of Astrodynamics*, Dover Publications, New York, 1971.
- Bryson, A. E., Desai, M. N., and Hoffman, W. C., "Energy-State Approximation in Performance Optimization of Supersonic Aircraft," *Journal of Aircraft*, Vol. 6, No. 6, 1969, pp. 481–488.
- Martins, J. R. R., Sturdza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262.