Chapter 4

The Optimal Control Problem

4.1 Introduction

4.1.1 Dynamic Constraints

The optimal control problem may be interpreted as an extension of the NLP problem to an infinite number of variables. For fundamental background in the associated *calculus of variations*, the reader should refer to Bliss [42]. First, let us consider a simple problem with a single phase and no path constraints. Specifically, suppose we must choose the control functions $\mathbf{u}(t)$ to minimize

$$J = \phi\left[\mathbf{y}(t_F), t_F\right] \tag{4.1}$$

subject to the state equations

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] \tag{4.2}$$

and the boundary conditions

$$\boldsymbol{\psi}[\mathbf{y}(t_F), \mathbf{u}(t_F), t_F] = \mathbf{0}, \tag{4.3}$$

where the initial conditions $\mathbf{y}(t_I) = \mathbf{y}_I$ are given at the fixed initial time t_I and the final time t_F is free. This is a very simplified version of the problem (called an *autonomous system*) that will be addressed later in the chapter. We have intentionally chosen a problem with only equality constraints. However, in contrast to the discussion in Chapters 1 and 2, we now have two types of constraints. The equality constraint (4.2) may be viewed as "continuous" since it must be satisfied over the entire interval $t_I \le t \le t_F$, whereas the equality (4.3) may be viewed as "discrete" since it is imposed at the specific time t_F. In a manner analogous to the definition of the Lagrangian function (1.55), we form an augmented performance index

$$\hat{J} = \left[\phi + \boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\psi}\right]_{t_F} - \int_{t_I}^{t_F} \boldsymbol{\lambda}^{\mathsf{T}}(t)\{\dot{\mathbf{y}} - \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)]\}\, dt. \tag{4.4}$$

Notice that, in addition to the Lagrange multipliers $\boldsymbol{\nu}$ for the discrete constraints, we also have multipliers $\boldsymbol{\lambda}(t)$, referred to as *adjoint or costate variables* for the continuous (differential equation) constraints. In the finite-dimensional case, the necessary conditions for a constrained optimum (1.56) and (1.57) were obtained by setting the first derivatives of

the Lagrangian to zero. In this setting, the analogous operation is to set the *first variation* $\delta \hat{J} = 0$. It is convenient to define the *Hamiltonian*

$$H = \boldsymbol{\lambda}^{\mathsf{T}}(t)\mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] \tag{4.5}$$

and the auxiliary function

$$\Phi = \phi + \boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\psi}. \tag{4.6}$$

The necessary conditions, referred to as the *Euler–Lagrange equations*, which result from setting the first variation to zero, in addition to (4.2) and (4.3), are

$$\dot{\boldsymbol{\lambda}} = -\mathbf{H}_y^{\mathsf{T}}, \tag{4.7}$$

called the *adjoint equations*,

$$\mathbf{0} = \mathbf{H}_u^{\mathsf{T}}, \tag{4.8}$$

called the *control equations*, and

$$\boldsymbol{\lambda}(t_F) = \boldsymbol{\Phi}_y^{\mathsf{T}}\Big|_{t=t_F}, \tag{4.9}$$

$$0 = (\Phi_t + H)|_{t=t_F}, \tag{4.10}$$

called the *transversality conditions*. In these expressions, the partial derivatives $\mathbf{H}_y$, $\mathbf{H}_u$, and $\boldsymbol{\Phi}_y$ are considered row vectors, i.e., $\mathbf{H}_y \equiv (\partial H/\partial y_1, \ldots, \partial H/\partial y_n)$. The control equations (4.8) are a simplified statement of the *Pontryagin maximum principle* [145]. A more general expression is

$$\mathbf{u} = \arg\min_{\mathbf{u}\in\mathcal{U}} H, \tag{4.11}$$

where $\mathcal{U}$ defines the domain of feasible controls. Note that the algebraic sign has been changed such that (4.11) is really a "minimum" principle in order to be consistent with the algebraic sign conventions used elsewhere, even though, in the original reference [145], Pontryagin derived a "maximum" principle. The maximum principle states that the control variable must be chosen to optimize the Hamiltonian (at every instant in time) subject to limitations on the control imposed by state and control path constraints. In essence, the maximum principle is a constrained optimization problem in the variables $\mathbf{u}(t)$ at all values of t. The complete set of necessary conditions consists of a DAE system (4.2), (4.7), and (4.8) with boundary conditions at both t_I and t_F (4.9), (4.10), and (4.3). This is often referred to as a *two-point boundary value problem*. A more extensive presentation of this material can be found in Bryson and Ho [54].

4.1.2 Algebraic Equality Constraints

Generalizing the problem in the previous section, let us assume that we impose algebraic *path constraints* of the form

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t] \tag{4.12}$$

in addition to the other conditions (4.2), (4.3). The treatment of this path constraint depends on the matrix of partial derivatives $\mathbf{g}_u$ (3.59). Two possibilities exist. If the matrix $\mathbf{g}_u$ is full rank, then the system of differential and algebraic equations (4.2), (4.12) is a DAE of

index one, and (4.12) is termed a *control variable equality constraint*. For this case, the Hamiltonian (4.5) is replaced by

$$H = \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{f} + \boldsymbol{\mu}^{\mathsf{T}}\mathbf{g}, \tag{4.13}$$

which will result in modification to both the adjoint equations (4.7) and the control equations (4.8).

The second possibility is that the matrix $\mathbf{g}_u$ is rank deficient. In this case, we can differentiate (4.12) with respect to t and reduce the index of the DAE as discussed in Section 3.5 (cf. (3.60)–(3.63)). The result is a new path-constraint function $\mathbf{g}'$, which is mathematically equivalent provided that the original constraint is imposed at some point on the path, say $0 = \mathbf{g}[\mathbf{y}(t_I), \mathbf{u}(t_I), t_I]$. For this new path function, again the matrix $\mathbf{g}'_u$ may be full rank or rank deficient. If the matrix is full rank, the original DAE system is said to have *index two* and this is referred to as a *state variable constraint* of order one. In the full-rank case, we may redefine the Hamiltonian using $\mathbf{g}'$ in place of $\mathbf{g}$. Of course, if the matrix $\mathbf{g}'_u$ is rank deficient, the process must be repeated. This is referred to as *index reduction* in the DAE literature [2, 48]. It is important to note that index reduction may be difficult to perform and imposition of a high index path constraint may be prone to numerical error. This technique is illustrated in Example 6.14 for a multibody mechanism.

4.1.3 Singular Arcs

In the preceding section we addressed the DAE system

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{u}, t], \tag{4.14}$$

$$\mathbf{0} = \mathbf{g}[\mathbf{y}, \mathbf{u}, t], \tag{4.15}$$

which can appear when path constraints are imposed on the optimal control problem. However, even in the absence of path constraints, the necessary conditions (4.2), (4.7), and (4.8) lead to the DAE system

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{u}, t], \tag{4.16}$$

$$\dot{\boldsymbol{\lambda}} = -\mathbf{H}_y^{\mathsf{T}}, \tag{4.17}$$

$$\mathbf{0} = \mathbf{H}_u^{\mathsf{T}}. \tag{4.18}$$

When viewed in this fashion, the differential variables are $(\mathbf{y}, \boldsymbol{\lambda})$ and the algebraic variables are $\mathbf{u}$. Because it is a DAE system, one expects the algebraic condition to define the algebraic variables. Thus, one expects the optimality condition $\mathbf{0} = \mathbf{H}_u^{\mathsf{T}}$ to define the control variable provided the matrix $\mathbf{H}_{uu}$ is nonsingular. However, if $\mathbf{H}_{uu}$ is a singular matrix, the control $\mathbf{u}$ is not uniquely defined by the optimality condition. This situation is referred to as a *singular arc*, and the analysis of this problem involves techniques quite similar to those discussed above for path constraints. Furthermore, singular arc problems are not just mathematical curiosities since $\mathbf{H}_{uu}$ is singular whenever $H[\mathbf{y}, \mathbf{u}, t]$ is a linear function of $\mathbf{u}$, which can occur for linear ODEs with no path constraints. The famous sounding rocket problem proposed by Robert Goddard in 1919 [165] contains a singular arc. Recent work in periodic optimal flight [160, 152] and the analysis of wind shear during landing [9] (cf. Example 6.10) involves formulations with singular arcs.

4.1.4 Algebraic Inequality Constraints

The preceding sections have addressed the treatment of equality path constraints. Let us now consider inequality path constraints of the form

$$\mathbf{0} \leq \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{4.19}$$

Unlike an equality constraint, which must be satisfied for all $t_I \leq t \leq t_F$, inequality constraints may either be active ($\mathbf{0} = \mathbf{g}$) or inactive ($\mathbf{0} < \mathbf{g}$) at each instant in time. In essence, the time domain is partitioned into constrained and unconstrained subarcs. During the unconstrained arcs, the necessary conditions are given by (4.2), (4.7), and (4.8), whereas the conditions with modified Hamiltonian (4.13) are applicable in the constrained arcs. Thus, the imposition of inequality constraints presents three major complications. First, the *number* of constrained subarcs present in the optimal solution is not known a priori. Second, the *location* of the *junction points* when the transition from constrained to unconstrained (and vice versa) occurs is unknown. Finally, at the junction points, it is possible that both the control variables $\mathbf{u}$ and the adjoint variables $\boldsymbol{\lambda}$ are discontinuous. Additional *jump conditions*, which are essentially boundary conditions imposed at the junction points, must be satisfied. Thus, what was a two-point BVP may become a multipoint BVP when inequalities are imposed. For a more complete discussion of this subject, the reader is referred to the tutorial by Pesch [139] and the text by Bryson and Ho [54].

4.2 Necessary Conditions for the Discrete Problem

To conclude the discussion, let us reemphasize the relationship between optimal control and NLP problems with a simple example. Suppose we must choose the control functions $\mathbf{u}(t)$ to minimize

$$J = \phi\left[\mathbf{y}(t_F), t_F\right] \tag{4.20}$$

subject to the state equations

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)]. \tag{4.21}$$

Let us assume that the initial and final times t_I and t_F are fixed and the initial conditions $\mathbf{y}(t_I) = \mathbf{y}_I$ are given. Let us define NLP variables

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{u}_1, \mathbf{y}_2, \mathbf{u}_2, \mathbf{y}_3, \mathbf{u}_3, \ldots, \mathbf{y}_M, \mathbf{u}_M) \tag{4.22}$$

as the values of the state and control evaluated at $t_1, t_2, \ldots, t_M$, where $t_{k+1} = t_k + h$ with $h = t_F/M$. Now

$$\dot{\mathbf{y}} \approx \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h}. \tag{4.23}$$

Let us substitute this approximation into (4.21), thereby defining the NLP (*defect*) constraints

$$c_k(\mathbf{x}) \equiv \boldsymbol{\zeta}_k = \mathbf{y}_{k+1} - \mathbf{y}_k - h\mathbf{f}(\mathbf{y}_k, \mathbf{u}_k) = 0 \tag{4.24}$$

for $k = 1, \ldots, M-1$ and NLP objective function

$$F(\mathbf{x}) = \phi(\mathbf{y}_M). \tag{4.25}$$

The problem defined by (4.22), (4.24), and (4.25) is an NLP. From (1.55), the Lagrangian is

$$L(\mathbf{x},\boldsymbol{\lambda}) = F(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{c}(\mathbf{x}) = \phi(\mathbf{y}_M) - \sum_{k=1}^{M-1} \boldsymbol{\lambda}_k^{\top}\left[\mathbf{y}_{k+1} - \mathbf{y}_k - h\mathbf{f}(\mathbf{y}_k,\mathbf{u}_k)\right]. \tag{4.26}$$

The necessary conditions for this problem follow directly from the definitions (1.58) and (1.59):

$$\frac{\partial L}{\partial \boldsymbol{\lambda}_k} = \mathbf{y}_{k+1} - \mathbf{y}_k - h\mathbf{f}(\mathbf{y}_k,\mathbf{u}_k) = 0, \tag{4.27}$$

$$\frac{\partial L}{\partial \mathbf{y}_k} = (\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k-1}) + h\boldsymbol{\lambda}_k^{\top}\frac{\partial \mathbf{f}}{\partial \mathbf{y}_k} = 0, \tag{4.28}$$

$$\frac{\partial L}{\partial \mathbf{u}_k} = h\boldsymbol{\lambda}_k^{\top}\frac{\partial \mathbf{f}}{\partial \mathbf{u}_k} = 0, \tag{4.29}$$

$$\frac{\partial L}{\partial \mathbf{y}_M} = -\boldsymbol{\lambda}_{M-1} + \frac{\partial \phi}{\partial \mathbf{y}_M} = 0. \tag{4.30}$$

Now let us consider the limiting form of this problem as $M \to \infty$ and $h \to 0$. Clearly, in the limit, equation (4.27) becomes the state equation (4.2), equation (4.28) becomes the adjoint equation (4.7), equation (4.29) becomes the control equation (4.8), and equation (4.30) becomes the transversality condition (4.9). Essentially, we have demonstrated that the KKT NLP necessary conditions approach the optimal control necessary conditions as the number of variables grows. The NLP Lagrange multipliers can be interpreted as discrete approximations to the optimal control adjoint variables which will be discussed more extensively in Section 4.11. While this discussion is of theoretical importance, it also suggests a number of ideas that are the basis of modern numerical methods. In particular, if the analysis is extended to inequality-constrained problems, it is apparent that the task of identifying the NLP active set is equivalent to defining constrained subarcs and junction points in the optimal control setting. Early results on this transcription process can be found in Canon, Cullum, and Polak [61], Polak [143], and Tabak and Kuo [164]. Since the most common type of transcription employs *collocation* [109], the terms "collocation" and "transcription" are often used synonymously. More recently, interest has focused on using alternative methods of discretization [77, 112, 168]. O. von Stryk [167] presents results using a Hermite approximation for the state variables and a linear control approximation, which leads to discrete approximations that are scalar multiples of those presented here.

Figure 4.1 illustrates the situation and Table 4.1 summarizes some of the major similarities between the discrete and continuous formulations of the problem.

4.3 Direct versus Indirect Methods

When describing methods for solving optimal control problems, a technique is often classified as either a *direct method* or an *indirect method*. The distinction between a direct method and an indirect method was first introduced in Section 1.4 when considering the minimization of a univariate function. A direct method constructs a sequence of

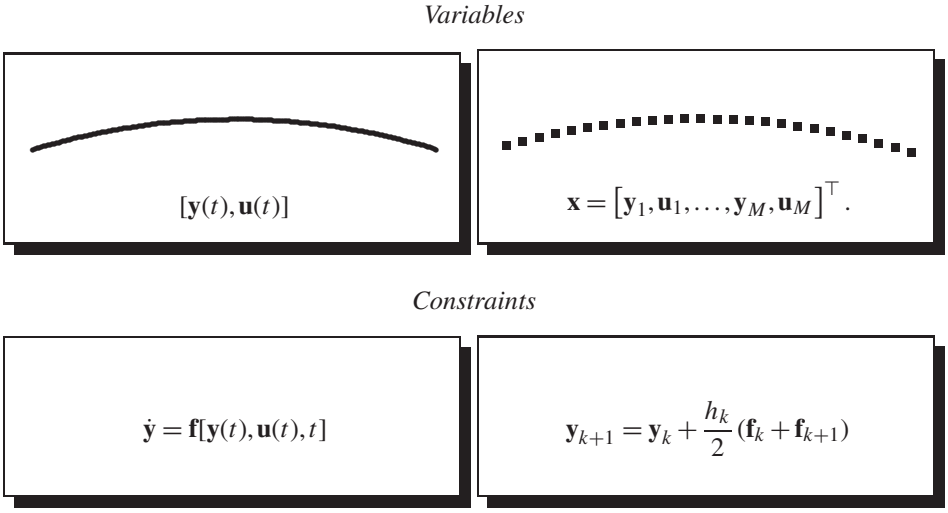


Figure 4.1. *Discretization methods.*

Table 4.1. *Discrete versus continuous formulation.*

Discrete		Continuous	
$(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M)$	NLP variables	$\mathbf{y}(t)$	State variables
$(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_M)$	NLP variables	$\mathbf{u}(t)$	Control variables
$\zeta_k = 0$	Defect constraints	$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{u}, t)$	State equations
$(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_M)$	Lagrange multipliers	$\boldsymbol{\lambda}(t)$	Adjoint variables

points $x_1, x_2, \ldots, x^*$ such that the objective function is minimized and typically $F(x_1) > F(x_2) > \cdots > F(x^*)$. An indirect method attempts to find a root of the necessary condition $F'(x) = 0$. At least in principle, the direct method need only compare values for the objective function. In contrast, the indirect method must *compute* the slope $F'(x)$ and then decide if it is sufficiently close to zero. In essence, an indirect method attempts to locate a root of the necessary conditions. A direct method attempts to find a minimum of the objective (or Lagrangian) function.

How does this categorization extend to the optimal control setting? An indirect method attempts to solve the optimal control necessary conditions (4.2), (4.3), (4.7), (4.8), (4.9), and (4.10). Thus, for an indirect method, it is necessary to explicitly derive the adjoint equations, the control equations, and all of the transversality conditions. In contrast, a direct method does not require explicit derivation and construction of the necessary conditions. A direct method does not construct the adjoint equations (4.7), the control equations (4.8), or any of the transversality (boundary) conditions (4.9)–(4.10). It is also important to emphasize that there is no correlation between the method used to solve the problem and the formulation. For example, one may consider applying a multiple shooting solution technique to either an indirect *or* a direct formulation. A survey of the more common approaches can be found in [18].

So *why not use the indirect method?* There are at least three major difficulties that detract from an indirect method in practice.

1. The user must compute the quantities $\mathbf{H}_y$, $\mathbf{H}_u$, etc., that appear in the defining equations (4.7)–(4.10). Unfortunately, this operation requires an intelligent user with at least some knowledge of optimal control theory. Furthermore, even if the user is familiar with the requisite theoretical background, it may be very difficult to construct these expressions for complicated black box applications. Finally, the approach is not flexible, since each time a new problem is posed, a new derivation of the relevant derivatives is called for!

2. If the problem description includes path inequalities (4.19), it is necessary to make an a priori estimate of the constrained-arc sequence. Unfortunately, this can be quite difficult. In fact, if the number of constrained subarcs is unknown, then the *number* of iteration variables is also unknown. Furthermore, the *sequence* of constrained/unconstrained arcs is unknown, which makes it extremely difficult to impose the correct junction conditions and, thereby, define the arc boundaries.

3. Finally, the basic method is not robust. One difficulty is that the user must guess values for the adjoint variables $\boldsymbol{\lambda}$, which, because they are not physical quantities, is very nonintuitive! Even with a reasonable guess for the adjoint variables, the numerical solution of the adjoint equations can be very ill-conditioned! The sensitivity of the indirect method has been recognized for some time. Computational experience with the technique in the late 1960s is summarized by Bryson and Ho [54, p. 214]:

   > The main difficulty with these methods is *getting started*; i.e., finding a first estimate of the unspecified conditions at one end that produces a solution reasonably close to the specified conditions at the other end. The reason for this peculiar difficulty is the extremal solutions are often *very sensitive* to small changes in the unspecified boundary conditions.... Since the system equations and the Euler–Lagrange equations are coupled together, it is not unusual for the numerical integration, with poorly guessed initial conditions, to produce "wild" trajectories in the state space. These trajectories may be so wild that values of $x(t)$ and/or $\lambda(t)$ exceed the numerical range of the computer!

Section 4.12 presents an application that dramatically illustrates many of these issues. However, because of the practical difficulties with an indirect formulation, the remainder of the book will focus on direct methods.

4.4 General Formulation

An optimal control problem can be formulated as a collection of N *phases* as described in Section 3.7. In general, the independent variable t for *phase* k is defined in the region $t_I^{(k)} \le t \le t_F^{(k)}$. For many applications, the independent variable t is *time* and the phases are sequential, that is, $t_I^{(k+1)} = t_F^{(k)}$. However, neither of these assumptions is required. Within phase k, the dynamics of the system are described by a set of *dynamic* variables

$$\mathbf{z} = \begin{bmatrix} \mathbf{y}^{(k)}(t) \\ \mathbf{u}^{(k)}(t) \end{bmatrix} \tag{4.31}$$

made up of the $n_y^{(k)}$ *state variables* and the $n_u^{(k)}$ *control variables*, respectively. In addition, the dynamics may incorporate the $n_p^{(k)}$ *parameters* $\mathbf{p}^{(k)}$ that are independent of t.

Typically, the dynamics of the system are defined by a set of ODEs written in explicit form, which are referred to as the *state equations*,

$$\dot{\mathbf{y}}^{(k)} = \mathbf{f}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t], \tag{4.32}$$

where $\mathbf{y}^{(k)}$ is the $n_y^{(k)}$ dimension state vector. In addition, the solution must satisfy algebraic *path constraints* of the form

$$\mathbf{g}_L^{(k)} \le \mathbf{g}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t] \le \mathbf{g}_U^{(k)}, \tag{4.33}$$

where $\mathbf{g}^{(k)}$ is a vector of size $n_g^{(k)}$, as well as simple bounds on the state variables

$$\mathbf{y}_L^{(k)} \le \mathbf{y}^{(k)}(t) \le \mathbf{y}_U^{(k)} \tag{4.34}$$

and control variables

$$\mathbf{u}_L^{(k)} \le \mathbf{u}^{(k)}(t) \le \mathbf{u}_U^{(k)}. \tag{4.35}$$

Note that an equality constraint can be imposed if the upper and lower bounds are equal, e.g., $[g_L^{(k)}]_j = [g_U^{(k)}]_j$ for some j. This approach is consistent with our general formulation of the NLP problem as described in Section 1.12. Note that by definition, a *state variable* is a differentiated variable that appears on the left-hand side of the differential equation (4.32). In contrast, a *control variable* is an algebraic variable. Although this terminology is convenient for most purposes, there is often some ambiguity present when constructing a mathematical model of a physical system. For example, in multibody dynamics, it is common to model some physical "states" by algebraic variables. Example 6.18 presents an application in chemical kinetics with algebraic variables that can be viewed as both states and controls.

The phases are linked by boundary conditions of the form

$$\begin{aligned}
\boldsymbol{\psi}_L \le \boldsymbol{\psi} \Big[& \mathbf{y}^{(1)}(t_I^{(1)}), \mathbf{u}^{(1)}(t_I^{(1)}), \mathbf{p}^{(1)}, t_I^{(1)}, \\
& \mathbf{y}^{(1)}(t_F^{(1)}), \mathbf{u}^{(1)}(t_F^{(1)}), \mathbf{p}^{(1)}, t_F^{(1)}, \\
& \mathbf{y}^{(2)}(t_I^{(2)}), \mathbf{u}^{(2)}(t_I^{(2)}), \mathbf{p}^{(2)}, t_I^{(2)}, \\
& \mathbf{y}^{(2)}(t_F^{(2)}), \mathbf{u}^{(2)}(t_F^{(2)}), \mathbf{p}^{(2)}, t_F^{(2)}, \\
& \cdots \\
& \mathbf{y}^{(N)}(t_I^{(N)}), \mathbf{u}^{(N)}(t_I^{(N)}), \mathbf{p}^{(N)}, t_I^{(N)}, \\
& \mathbf{y}^{(N)}(t_F^{(N)}), \mathbf{u}^{(N)}(t_F^{(N)}), \mathbf{p}^{(N)}, t_F^{(N)} \Big] \le \boldsymbol{\psi}_U.
\end{aligned} \tag{4.36}$$

In spite of its daunting appearance, (4.36) has a rather simple interpretation. Basically, the boundary conditions allow the value of the dynamic variables at the beginning and end of any phase to be related to each other. For example, we might have an expression of the form

$$0 = y_1^{(1)}\left[t_F^{(1)}\right] - \cos\left[y_2^{(3)}(t_I^{(3)})\right],$$

which requires the value of the first state variable at the end of phase 1 to equal the cosine of the second state at the beginning of phase 3.

Finally, it may be convenient to evaluate expressions of the form

$$\int_{t_I^{(k)}}^{t_F^{(k)}} \mathbf{w}^{(k)}\left[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t\right] dt, \tag{4.37}$$

which involve the *quadrature functions* $\mathbf{w}^{(k)}$. Collectively, we refer to those functions evaluated during the phase, namely

$$\mathbf{F}^{(k)}(t) = \begin{bmatrix} \mathbf{f}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t] \\ \mathbf{g}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t] \\ \mathbf{w}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t] \end{bmatrix}, \tag{4.38}$$

as the vector of *continuous functions*. Similarly, functions evaluated at specific points, such as the boundary conditions $\boldsymbol{\psi}(\cdot)$, are referred to as *point functions*.

The basic optimal control problem is to determine the $n_u^{(k)}$-dimensional control vectors $\mathbf{u}^{(k)}(t)$ and parameters $\mathbf{p}^{(k)}$ to minimize the performance index

$$\begin{aligned} J = \phi\Big[& \mathbf{y}^{(1)}(t_I^{(1)}), \mathbf{u}^{(1)}(t_I^{(1)}), \mathbf{p}^{(1)}, t_I^{(1)}, \\ & \mathbf{y}^{(1)}(t_F^{(1)}), \mathbf{u}^{(1)}(t_F^{(1)}), \mathbf{p}^{(1)}, t_F^{(1)}, \\ & \mathbf{y}^{(2)}(t_I^{(2)}), \mathbf{u}^{(2)}(t_I^{(2)}), \mathbf{p}^{(2)}, t_I^{(2)}, \\ & \mathbf{y}^{(2)}(t_F^{(2)}), \mathbf{u}^{(2)}(t_F^{(2)}), \mathbf{p}^{(2)}, t_F^{(2)}, \\ & \cdots \\ & \mathbf{y}^{(N)}(t_I^{(N)}), \mathbf{u}^{(N)}(t_I^{(N)}), \mathbf{p}^{(N)}, t_I^{(N)}, \\ & \mathbf{y}^{(N)}(t_F^{(N)}), \mathbf{u}^{(N)}(t_F^{(N)}), \mathbf{p}^{(N)}, t_F^{(N)}\Big] \\ & + \sum_{j=1}^{N} \left\{ \int_{t_I^{(j)}}^{t_F^{(j)}} w^{(j)}\left[\mathbf{y}^{(j)}(t), \mathbf{u}^{(j)}(t), \mathbf{p}^{(j)}, t\right] dt \right\}. \end{aligned} \tag{4.39}$$

Notice that, like the boundary conditions, the objective function may depend on quantities computed in each of the N phases. Furthermore, the objective function includes contributions evaluated at the phase boundaries (point functions) and over the phase (quadrature functions). As written, (4.39) is known as the *problem of Bolza*. When the function $\phi \equiv 0$ in the objective, we refer to this as the *problem of Lagrange* or, if there are no integral terms $w^{(j)} \equiv 0$, the optimization is termed the *problem of Mayer*. It is worth noting that it is relatively straightforward to pose the problem in an alternative form if desirable. For example, suppose the problem is stated in the Lagrange form with a performance index

$$J = \int_{t_I}^{t_F} w\left[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t\right] dt. \tag{4.40}$$

By introducing an additional state variable, say y_{n+1}, and the differential equation

$$\dot{y}_{n+1} = w[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \tag{4.41}$$

with initial condition $y_{n+1}(t_I) = 0$, it is possible to replace the original objective function (4.40) with

$$J = \phi[\mathbf{y}(t_F)] = y_{n+1}(t_F). \tag{4.42}$$

In fact, for notational simplicity, it is common to define the optimal control problem in the Mayer form. On the other hand, it may not be desirable to make this transformation for computational reasons because adding a state variable y_{n+1} increases the size of the NLP subproblem after discretization. For this reason, the $\mathbb{SOCS}$ [38] software implementation treats problems stated in Bolza, Lagrange, or Mayer form. Efficient treatment of quadrature equations will be discussed in Section 4.9.

For clarity, we drop the phase-dependent notation from the remaining discussion; however, it is important to remember that many complex problem descriptions require different dynamics and/or constraints within each phase, and the approach accommodates this requirement.

4.5 Direct Transcription Formulation

The fundamental ingredients required for solving the optimal control problem by direct transcription are now in place. In Section 3.5, we introduced a number of methods for solving IVPs. All approaches divide the phase duration (for a single phase) into n_s segments or intervals

$$t_I = t_1 < t_2 < \cdots < t_M = t_F, \tag{4.43}$$

where the points are referred to as node, mesh, or grid points. Define the number of mesh points as $M \equiv n_s + 1$. As before, we use $\mathbf{y}_k \equiv \mathbf{y}(t_k)$ to indicate the value of the state variable at a grid point. However, the methods for solving the IVP described in Section 3.5 did not involve algebraic (control) variables. To extend the methods to control problems, let us denote the control at a grid point by $\mathbf{u}_k \equiv \mathbf{u}(t_k)$. In addition, some discretization schemes require values for the control variable at the midpoint of an interval, and we denote this quantity by $\overline{\mathbf{u}}_k \equiv \mathbf{u}(\overline{t})$ with $\overline{t} = \frac{1}{2}(t_k + t_{k-1})$. To be consistent, we also denote $\mathbf{f}_k \equiv \mathbf{f}\big[\mathbf{y}(t_k), \mathbf{u}(t_k), \mathbf{p}, t_k\big]$. It should be emphasized that the subscript k refers to a grid point within a phase.

The basic notion is now quite simple. Let us treat the values of the state and control variables as a set of NLP variables. The differential equations will be replaced by a finite set of defect constraints. As a result of the transcription, the optimal control constraints (4.32)–(4.33) are replaced by the NLP constraints

$$\mathbf{c}_L \le \mathbf{c}(\mathbf{x}) \le \mathbf{c}_U, \tag{4.44}$$

where

$$\mathbf{c}(\mathbf{x}) = \big[\boldsymbol{\zeta}_1, \boldsymbol{\zeta}_2, \ldots, \boldsymbol{\zeta}_{M-1}, \boldsymbol{\psi}_I, \boldsymbol{\psi}_F, \mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_M\big]^\top \tag{4.45}$$

with

$$\mathbf{c}_L = \big[\mathbf{0}, \ldots, \mathbf{0}, \mathbf{g}_L, \ldots, \mathbf{g}_L\big]^\top \tag{4.46}$$

and a corresponding definition of $\mathbf{c}_U$. The first $n_y n_s$ equality constraints require that the defect vectors from each of the n_s segments be zero, thereby approximately satisfying the differential equations (4.32). The boundary conditions are enforced directly by the equality constraints on $\boldsymbol{\psi}$ and the nonlinear path constraints are imposed at the grid points.

Note that nonlinear equality path constraints are enforced by setting $\mathbf{c}_L = \mathbf{c}_U$. In a similar fashion, the state and control variable bounds (4.34) and (4.35) become simple bounds on the NLP variables. The path constraints and variable bounds are always imposed at the grid points for all discretization schemes. For the Hermite–Simpson and Runge–Kutta discretization methods, the path constraints and variable bounds are also imposed at the interval midpoints.

For the sake of reference, let us summarize the variables and constraints for each of the Runge–Kutta schemes introduced in Section 3.5.

Euler Method

Variables:

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_M, \mathbf{u}_M). \tag{4.47}$$

Defects:

$$\boldsymbol{\zeta}_k = \mathbf{y}_{k+1} - \mathbf{y}_k - h_k \mathbf{f}_k. \tag{4.48}$$

Classical Runge–Kutta Method

Variables:

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{y}_1, \mathbf{u}_1, \overline{\mathbf{u}}_2, \ldots, \overline{\mathbf{u}}_M, \mathbf{y}_M, \mathbf{u}_M). \tag{4.49}$$

Defects:

$$\boldsymbol{\zeta}_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4), \tag{4.50}$$

where

$$\mathbf{k}_1 = h_k \mathbf{f}_k, \tag{4.51}$$

$$\mathbf{k}_2 = h_k \mathbf{f}\left(\mathbf{y}_k + \frac{1}{2}\mathbf{k}_1, \overline{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right), \tag{4.52}$$

$$\mathbf{k}_3 = h_k \mathbf{f}\left(\mathbf{y}_k + \frac{1}{2}\mathbf{k}_2, \overline{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right), \tag{4.53}$$

$$\mathbf{k}_4 = h_k \mathbf{f}(\mathbf{y}_k + \mathbf{k}_3, \mathbf{u}_{k+1}, t_{k+1}). \tag{4.54}$$

Trapezoidal Method

Variables:

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_M, \mathbf{u}_M). \tag{4.55}$$

Defects:

$$\boldsymbol{\zeta}_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}(\mathbf{f}_k + \mathbf{f}_{k+1}). \tag{4.56}$$

Hermite–Simpson Method

Variables:

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{y}_1, \mathbf{u}_1, \overline{\mathbf{u}}_2, \ldots, \overline{\mathbf{u}}_M, \mathbf{y}_M, \mathbf{u}_M). \tag{4.57}$$

Defects:

$$\zeta_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6}\left(\mathbf{f}_k + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_{k+1}\right), \tag{4.58}$$

where

$$\bar{\mathbf{y}}_{k+1} = \frac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1}) + \frac{h_k}{8}(\mathbf{f}_k - \mathbf{f}_{k+1}), \tag{4.59}$$

$$\bar{\mathbf{f}}_{k+1} = \mathbf{f}\left(\bar{\mathbf{y}}_{k+1}, \bar{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right). \tag{4.60}$$

For simplicity, we have presented all of the schemes assuming that the phase duration is fixed. Of course, in general, the duration of a phase may be variable and it is worthwhile discussing how this changes the discretization. First, the set of NLP variables must be augmented to include the variable time(s) t_I and/or t_F. Second, we must alter the discretization defined by (4.43) using the transformation (3.114) such that the stepsize is

$$h_k = (\tau_{k+1} - \tau_k)(t_F - t_I) = (\tau_{k+1} - \tau_k)\Delta t = \Delta\tau_k \Delta t, \tag{4.61}$$

where $\Delta t \equiv (t_F - t_I)$ and $\Delta\tau_k \equiv (\tau_{k+1} - \tau_k)$ with constants $0 \le \tau_k \le 1$ chosen so that the grid points are located at fixed fractions of the total phase duration. Thus, in essence, as the times t_I and/or t_F change, an "accordion"-like grid-point distribution results. Observe that this approach will produce a consistent function generator as described in Section 3.8. Finally, it should also be clear that the set of NLP variables can be augmented to include the parameters p if they are part of the problem description.

4.6 NLP Considerations—Sparsity

4.6.1 Background

In the preceding section, an NLP problem was constructed by discretization of an optimal control problem. In order to solve this NLP problem using any of the methods described in Chapters 1 and 2, it will be necessary to construct the first and second derivatives of the NLP constraints and objective function. With the exception of the boundary conditions, the major portion of the Jacobian matrix is defined as

$$\mathbf{G}_{ij} = \frac{\text{Change in defect constraint on segment } i}{\text{Change in optimization variable at grid point } j}.$$

This matrix will have m rows, where m is the total number of defect constraints, and n columns, where n is the total number of optimization variables. Now as a result of the discretization process, it should be clear that changing a variable at a grid point affects only the nearby constraints. Thus, the derivatives of many of the constraints with respect to the variable are zero. Figure 4.2 illustrates the situation for a simple discretization with 11 grid points. When the variable at grid point 6 is changed, the function connecting points 5 and 6 is altered, as is the function connecting points 6 and 7. However, the remaining portion of the curve is unchanged. In fact, the multiple shooting method described in Section 3.4 was

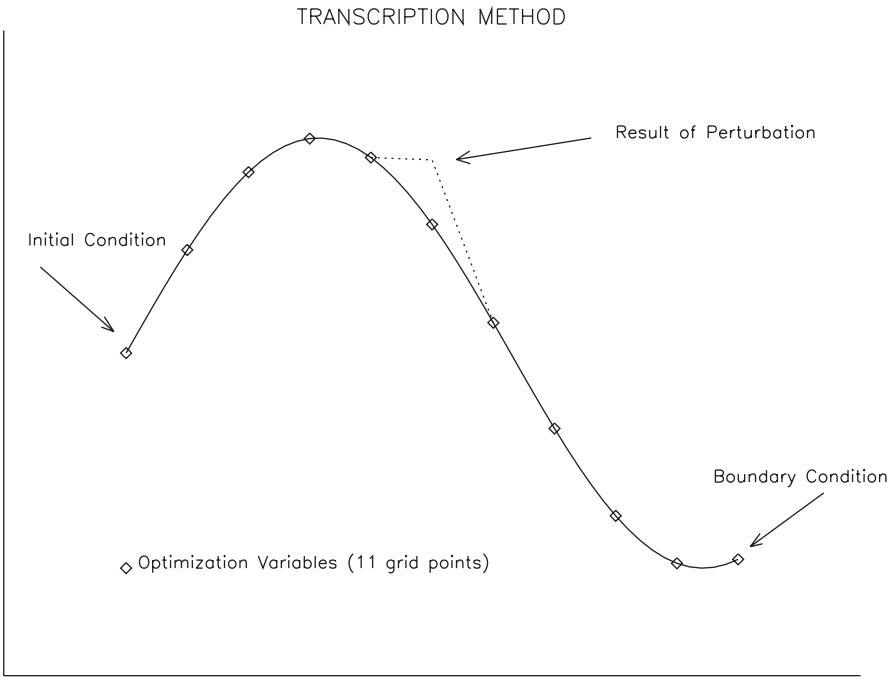


Figure 4.2. *Transcription sensitivity.*

introduced as a way to deal with the "tail wagging the dog" problem found in the simple shooting method (Section 3.3). In essence, reduced sensitivity in the BVP manifests itself mathematically as sparsity in the Jacobian matrix. The whole focus of this section is on how to construct the sparse Jacobian and Hessian matrices efficiently for the transcribed problem.

Example 4.1 LINEAR TANGENT STEERING. In order to motivate the development of the general method, it is convenient to present a simple example problem. Let us consider a problem with four states and one control described by the following state equations:

$$\dot{y}_1 = y_3, \tag{4.62}$$
$$\dot{y}_2 = y_4, \tag{4.63}$$
$$\dot{y}_3 = a\cos u, \tag{4.64}$$
$$\dot{y}_4 = a\sin u. \tag{4.65}$$

The goal is to minimize the time required for a vehicle to move from a fixed initial state to a terminal position by choosing the control $u(t)$. In this particular case, the problem has an analytic solution referred to as "linear tangent steering" [54]. This also is of considerable practical interest since it is a simplified version of the steering algorithm used by many launch vehicles, including the space shuttle.

4.6.2 Standard Approach

For the sake of comparison, let us begin with the standard direct transcription approach to this problem. The usual technique is to treat the values of the state and control at discrete times as optimization variables, thus leading to the definition

$$\mathbf{x}^{\mathsf{T}} = (t_F, \mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_M, \mathbf{u}_M) \tag{4.66}$$

of the NLP variables. The ODEs

$$\dot{\mathbf{y}} = \mathbf{f}\big[\mathbf{y}, \mathbf{u}, t\big] \tag{4.67}$$

are then approximated by the NLP constraints

$$\mathbf{0} = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}\big[\mathbf{f}_{k+1} + \mathbf{f}_k\big] \equiv \mathbf{c}(\mathbf{x}) \tag{4.68}$$

for $k = 1, \ldots, M-1$. This approximation describes a trapezoidal discretization for M grid points. The Jacobian matrix for the resulting NLP problem is then defined by

$$\mathbf{G} \equiv \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{x}}. \tag{4.69}$$

When a finite difference method is used to construct the Jacobian, it is natural to identify the constraint functions as the quantities being differentiated in (2.1). In other words, for the standard approach,

$$\mathbf{q} = \begin{bmatrix} \mathbf{c} \\ F \end{bmatrix} \tag{4.70}$$

and

$$\mathbf{D} = \begin{bmatrix} \mathbf{G} \\ (\nabla F)^{\mathsf{T}} \end{bmatrix}. \tag{4.71}$$

It is also natural to define

$$\boldsymbol{\omega}^{\mathsf{T}} = (-\lambda_1, \ldots, -\lambda_{M-1}, 1), \tag{4.72}$$

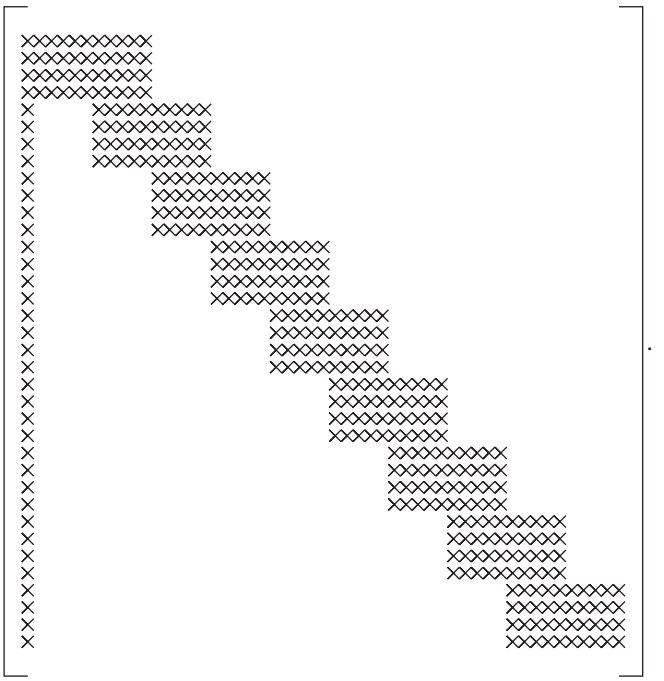
where λ_k are the Lagrange multipliers, so that (2.2)

$$\Omega(\mathbf{x}) = \sum_{i=1}^{m} \omega_i q_i(\mathbf{x}) = F - \sum_{i=1}^{M-1} \lambda_i c_i(\mathbf{x}) = L(\mathbf{x}, \boldsymbol{\lambda}) \tag{4.73}$$

is the *Lagrangian* for the NLP problem. Clearly, it follows that the Hessian of the Lagrangian $\mathbf{H} = \mathbf{E}$, where $\mathbf{E}$ is given by (2.3). For the linear tangent steering example problem, with $M = 10$ grid points, the number of NLP variables is $n = M(n_y + n_u) + 1 = 51$, where $n_y = 4$ is the number of states and $n_u = 1$ is the number of controls. The number of NLP constraints is $m = (M-1)n_y = 36$ and the number of index sets is $\gamma = 2 * (n_y + n_u) + 1 = 11$. Using the notation struct($\mathbf{G}$) to denote the structure of $\mathbf{G}$, the

resulting sparse Jacobian is of the form

$$\text{struct}(\mathbf{G}) = \left[\right. \left.\right]. \tag{4.74}$$



4.6.3 Discretization Separability

Examination of the expression for the trapezoidal defect (4.68) suggests that it may be desirable to simply group the terms by grid point, i.e.,

$$\begin{aligned}\mathbf{0} &= \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}\left[\mathbf{f}_{k+1} + \mathbf{f}_k\right] \\ &= \left[\mathbf{y}_{k+1} - \frac{h_k}{2}\mathbf{f}_{k+1}\right] + \left[-\mathbf{y}_k - \frac{h_k}{2}\mathbf{f}_k\right].\end{aligned} \tag{4.75}$$

In so doing it is possible to write the NLP constraints as

$$\mathbf{c}(\mathbf{x}) = \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.76}$$

where

$$\mathbf{B} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & & & \\ & \mathbf{I} & \mathbf{I} & & \\ & & \ddots & & \\ & & & \mathbf{I} & \mathbf{I} \end{bmatrix} \tag{4.77}$$

and

$$\mathbf{q}(\mathbf{x}) = \begin{bmatrix} -\mathbf{y}_1 - \frac{h_1}{2}\mathbf{f}_1 \\ \mathbf{y}_2 - \frac{h_1}{2}\mathbf{f}_2 \\ -\mathbf{y}_2 - \frac{h_2}{2}\mathbf{f}_2 \\ \mathbf{y}_3 - \frac{h_2}{2}\mathbf{f}_3 \\ \vdots \\ -\mathbf{y}_{M-1} - \frac{h_{M-1}}{2}\mathbf{f}_{M-1} \\ \mathbf{y}_M - \frac{h_{M-1}}{2}\mathbf{f}_M \end{bmatrix}. \tag{4.78}$$

It is then possible to construct sparse difference estimates for the matrix

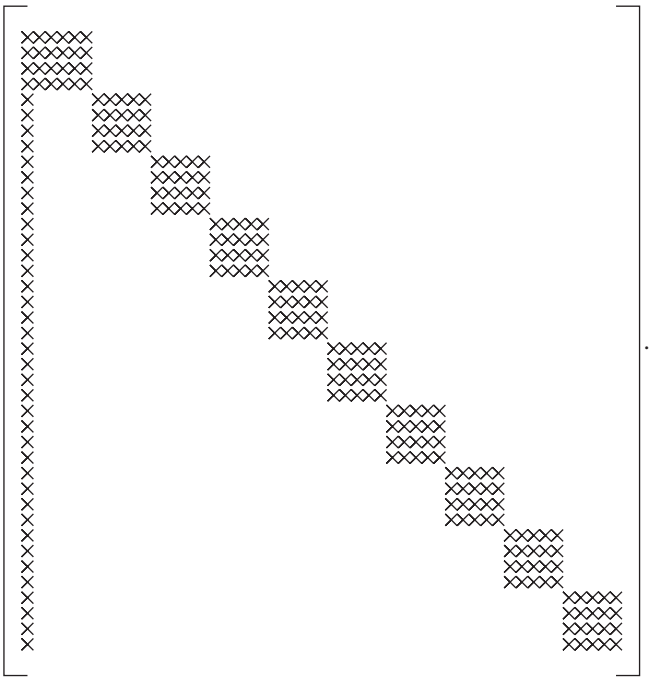
$$\mathbf{D} \equiv \frac{\partial \mathbf{q}}{\partial \mathbf{x}} \tag{4.79}$$

and then construct the NLP Jacobian using

$$\mathbf{G} \equiv \frac{\partial \mathbf{c}}{\partial \mathbf{x}} = \mathbf{BD}. \tag{4.80}$$

The reason for writing the equations in this manner becomes clear when examining the structure of the matrix $\mathbf{D}$:

$$\text{struct}(\mathbf{D}) = \tag{4.81}$$



Notice that the matrix $\mathbf{D}$ has approximately half as many nonzero elements per row as the matrix $\mathbf{G}$. Consequently, the number of index sets needed to construct $\mathbf{D}$ is $\gamma = 6$, as compared to $\gamma = 11$ for the standard approach. Thus, by exploiting separability, the Jacobian computation cost has been reduced by nearly a factor of two, and the Hessian cost is reduced by nearly a factor of three.

4.6.4 Right-Hand-Side Sparsity (Trapezoidal)

The computation savings observed by exploiting separability suggest further benefit may accrue by grouping terms and also isolating the linear terms, that is,

$$\begin{aligned}\mathbf{0} &= \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}\left[\mathbf{f}_{k+1} + \mathbf{f}_k\right] \\ &= \left[\mathbf{y}_{k+1} - \mathbf{y}_k\right] - \frac{1}{2}\Delta\tau_k[\Delta t\ \mathbf{f}_{k+1}] - \frac{1}{2}\Delta\tau_k[\Delta t\ \mathbf{f}_k],\end{aligned} \tag{4.82}$$

where, in view of transformation (3.114), $h_k = (\tau_{k+1} - \tau_k)(t_F - t_I) = (\tau_{k+1} - \tau_k)\Delta t = \Delta\tau_k \Delta t$ with $\Delta\tau_k \equiv (\tau_{k+1} - \tau_k)$ for $0 \le \tau_k \le 1$. Using this construction, the NLP constraints are

$$\mathbf{c}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.83}$$

where the constant matrices $\mathbf{A}$ and $\mathbf{B}$ are given by

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & -\mathbf{I} & \mathbf{0} & \mathbf{I} & & & \\ & & & -\mathbf{I} & \mathbf{0} & \mathbf{I} & \\ \vdots & & & & & & \ddots \end{bmatrix} \tag{4.84}$$

and

$$\mathbf{B} = -\frac{1}{2}\begin{bmatrix} \Delta\tau_1\mathbf{I} & \Delta\tau_1\mathbf{I} & & & & & \\ & & \Delta\tau_2\mathbf{I} & \Delta\tau_2\mathbf{I} & & & \\ & & & & \ddots & & \\ & & & & & \Delta\tau_{M-1}\mathbf{I} & \Delta\tau_{M-1}\mathbf{I} \end{bmatrix} \tag{4.85}$$

with the nonlinear relationships isolated in the vector

$$\mathbf{q} = \begin{bmatrix} \Delta t\,\mathbf{f}_1 \\ \Delta t\,\mathbf{f}_2 \\ \vdots \\ \Delta t\,\mathbf{f}_M \end{bmatrix}. \tag{4.86}$$

As before, we can construct sparse finite difference estimates for the matrix

$$\mathbf{D} \equiv \frac{\partial \mathbf{q}}{\partial \mathbf{x}} = \left[\begin{array}{c|ccc} \frac{\partial}{\partial t_F}(\Delta t\, \mathbf{f}_1) & \Delta t \frac{\partial \mathbf{f}_1}{\partial \mathbf{x}_1} & & \\ \frac{\partial}{\partial t_F}(\Delta t\, \mathbf{f}_2) & & \Delta t \frac{\partial \mathbf{f}_2}{\partial \mathbf{x}_2} & \\ \vdots & & & \ddots \\ \frac{\partial}{\partial t_F}(\Delta t\, \mathbf{f}_M) & & & \Delta t \frac{\partial \mathbf{f}_M}{\partial \mathbf{x}_M} \end{array}\right] \tag{4.87}$$

and then form the NLP Jacobian

$$\mathbf{G} = \mathbf{A} + \mathbf{B}\mathbf{D}. \tag{4.88}$$

An important aspect of this construction now becomes evident. Notice that the matrix $\mathbf{D}$ in (4.87) involves partial derivatives of the right-hand-side functions $\mathbf{f}$ with respect to the state and control, all evaluated at the same grid point. In particular, let us define the nonzero pattern

$$\text{struct}\left(\frac{\partial \mathbf{f}_k}{\partial \mathbf{x}_k}\right) = \text{struct}\left(\frac{\partial \mathbf{f}_k}{\partial \mathbf{y}_k} \,\middle|\, \frac{\partial \mathbf{f}_k}{\partial \mathbf{u}_k}\right) \tag{4.89}$$

as the *sparsity template*. The nonzero pattern defined by the sparsity template appears repeatedly in the matrix $\mathbf{D}$ at every grid point introduced by the discretization method. For the linear tangent steering example, the right-hand-side sparsity template is of the form

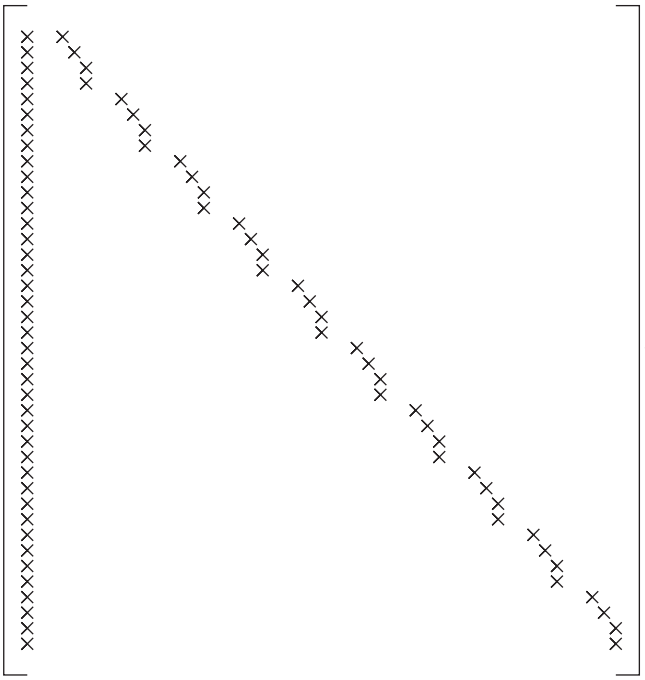
$$\text{struct}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}} \,\middle|\, \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right) = \left[\begin{array}{cccc|c} 0 & 0 & \mathsf{x} & 0 & 0 \\ 0 & 0 & 0 & \mathsf{x} & 0 \\ 0 & 0 & 0 & 0 & \mathsf{x} \\ 0 & 0 & 0 & 0 & \mathsf{x} \end{array}\right]. \tag{4.90}$$

Of course, in general, this right-hand-side sparsity template is problem dependent since it is defined by the functional form of the right-hand sides of the state equations (4.67). From a practical point of view, there are a number of alternatives for specifying the right-hand-side sparsity. One approach is to simply require the analyst to supply this information when defining the functions $\mathbf{f}$. A second alternative is to construct the information using some type of automatic differentiation of the user-provided software. The third approach, which is used in the implementation of our software [38], is to numerically construct the right-hand-side template using *random* perturbations about *random* nominal points.

Regardless of the approach used to construct the sparsity template, this information can be used to considerable advantage when constructing the matrix $\mathbf{D}$. For our linear

tangent steering example,

$$\text{struct}(\mathbf{D}) = \quad . \tag{4.91}$$



Comparing (4.81) with (4.91), it is clear that the repeated dense blocks along the diagonal in (4.81) have been replaced with repeated blocks with the right-hand-side sparsity template. The net result is that the matrix $\mathbf{D}$ can now be computed with $\gamma = 2$ index sets. To recapitulate, for this linear tangent steering example, one finds the following results:

	Dense	Sparse	Reduction (%)
Number of index sets, γ	11	2	−81.8%
Perturbations per Jacobian/Hessian	77	5	−93.5%

4.6.5 Hermite–Simpson (Compressed) (HSC)

In the previous section, the development focused on exploiting sparsity when the differential equations are approximated using a trapezoidal discretization that is second order, i.e., $\mathcal{O}(h^2)$. To accurately represent the solution, it may be desirable to use a discretization of higher order.

The discretization most widely used in the direct transcription algorithm is Hermite–Simpson, which is of order four, i.e., $\mathcal{O}(h^4)$. For this method, the defect constraints are given by

$$\mathbf{0} = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6}\left[\mathbf{f}_{k+1} + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_k\right], \tag{4.92}$$

where

$$\bar{\mathbf{f}}_{k+1} = \mathbf{f}\left[\bar{\mathbf{y}}_{k+1}, \bar{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right], \tag{4.93}$$

$$\bar{\mathbf{y}}_{k+1} = \frac{1}{2}(\mathbf{y}_{k+1} + \mathbf{y}_k) + \frac{h_k}{8}(\mathbf{f}_k - \mathbf{f}_{k+1}) \tag{4.94}$$

with $h_k = \Delta\tau_k \Delta t$. In order to emphasize the implicit nature of this method, it is instructive to substitute (4.93) and (4.94) into (4.92), yielding

$$\mathbf{0} = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6}\left\{\mathbf{f}_{k+1} + 4\mathbf{f}\left[\frac{1}{2}(\mathbf{y}_{k+1} + \mathbf{y}_k) + \frac{h_k}{8}(\mathbf{f}_k - \mathbf{f}_{k+1}), \bar{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right] + \mathbf{f}_k\right\}. \tag{4.95}$$

We shall refer to this as the *compressed* form of the Hermite–Simpson method because the Hermite interpolant (4.94) is used to locally eliminate the midpoint state. For this discretization, the NLP variables are

$$\mathbf{x}^{\mathsf{T}} = \left(t_F, \mathbf{y}_1, \mathbf{u}_1, \boxed{\bar{\mathbf{u}}_2}, \ldots, \boxed{\bar{\mathbf{u}}_M}, \mathbf{y}_M, \mathbf{u}_M\right). \tag{4.96}$$

Notice that the values for the control variable at the interval midpoints $\bar{\mathbf{u}}_k$ are introduced as NLP variables, while the corresponding values for the state are not NLP variables.

Proceeding in a manner analogous to the trapezoidal method, the NLP constraints can be written as

$$\mathbf{c}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.97}$$

where

$$\mathbf{q} \equiv \begin{bmatrix} \Delta t\left(\mathbf{f}_2 + 4\bar{\mathbf{f}}_2 + \mathbf{f}_1\right) \\ \Delta t\left(\mathbf{f}_3 + 4\bar{\mathbf{f}}_3 + \mathbf{f}_2\right) \\ \vdots \\ \Delta t\left(\mathbf{f}_M + 4\bar{\mathbf{f}}_M + \mathbf{f}_{M-1}\right) \end{bmatrix}. \tag{4.98}$$

Let us define

$$\mathbf{v}_k = \Delta t\left(\mathbf{f}_{k+1} + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_k\right). \tag{4.99}$$

The derivative matrix is then given by

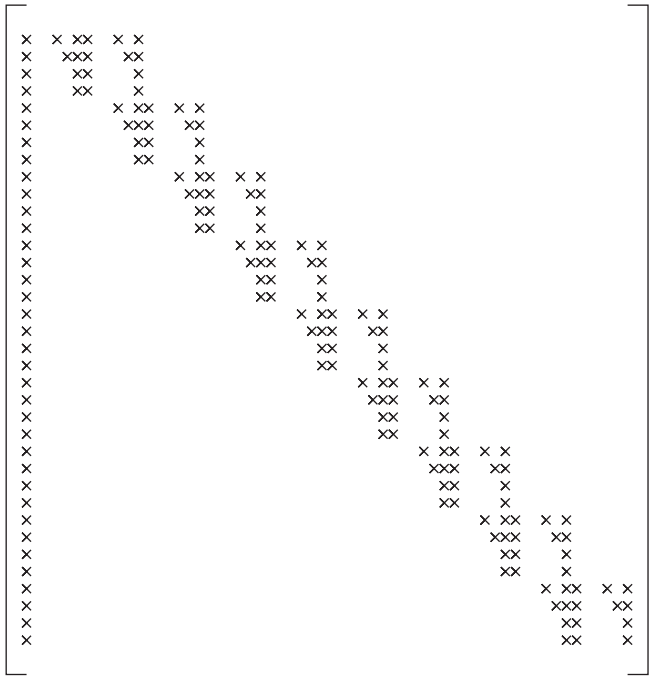
$$\mathbf{D} \equiv \frac{\partial \mathbf{q}}{\partial \mathbf{x}} = \left[\begin{array}{c|cccccccc} \frac{\partial}{\partial t_F}(\mathbf{v}_1) & \frac{\partial \mathbf{v}_1}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{v}_1}{\partial \bar{\mathbf{u}}_2} & \frac{\partial \mathbf{v}_1}{\partial \mathbf{x}_2} & & & & & \\ \frac{\partial}{\partial t_F}(\mathbf{v}_2) & & & \frac{\partial \mathbf{v}_2}{\partial \mathbf{x}_2} & \frac{\partial \mathbf{v}_2}{\partial \bar{\mathbf{u}}_3} & \frac{\partial \mathbf{v}_2}{\partial \mathbf{x}_3} & & & \\ \vdots & & & & & & \ddots & & \\ \frac{\partial}{\partial t_F}(\mathbf{v}_{M-1}) & & & & & & & & \frac{\partial \mathbf{v}_{M-1}}{\partial \mathbf{x}_M} \end{array}\right]. \tag{4.100}$$

As in the previous section, it is tempting to construct a template for the repeated sparse blocks in the matrix $\mathbf{D}$. In this case, the sparsity template is

$$\text{struct}\left(\frac{\partial \mathbf{v}_k}{\partial \mathbf{x}_k}\middle|\frac{\partial \mathbf{v}_k}{\partial \overline{\mathbf{u}}_{k+1}}\middle|\frac{\partial \mathbf{v}_k}{\partial \mathbf{x}_{k+1}}\right)=\left[\begin{array}{cccc|c|c|cccc|c}0&0&\mathsf{x}&0&\mathsf{x}&\mathsf{x}&0&0&\mathsf{x}&0&\mathsf{x}\\0&0&0&\mathsf{x}&\mathsf{x}&\mathsf{x}&0&0&0&\mathsf{x}&\mathsf{x}\\0&0&0&0&\mathsf{x}&\mathsf{x}&0&0&0&0&\mathsf{x}\\0&0&0&0&\mathsf{x}&\mathsf{x}&0&0&0&0&\mathsf{x}\end{array}\right]. \tag{4.101}$$

For the linear tangent steering example, we find that

$$\text{struct}(\mathbf{D})= \tag{4.102}$$



It is now worth comparing these results with the trapezoidal method described in the previous section. First, notice that the template (4.101) has more nonzeros than the trapezoidal template (4.90). This can be attributed to the fact that the function $\mathbf{v}$ in (4.99) is a *nonlinear* combination of the right-hand-side functions $\mathbf{f}$. Second, notice that it is not possible to uncouple the neighboring grid-point evaluations as was done when constructing (4.81). This leads to repeated blocks in (4.102) that are more than twice as wide as those in the trapezoidal method. The net result is that the HSC discretization requires $\gamma = 6$, whereas the trapezoidal method needs only $\gamma = 2$.

4.6.6 Hermite–Simpson (Separated) (HSS)

An apparent shortcoming of the compressed form of the Hermite–Simpson discretization is the fact that sparsity in the right-hand side of the differential equations is not fully exploited. Essentially, this is due to the local elimination of the state variable at the midpoint of each

interval, i.e., at $t_k + h_k/2$, which implicitly couples neighboring grid points. This difficulty can be avoided by explicitly introducing the local elimination variables and constraints. Thus, instead of (4.92), the discretization constraints *without* local compression are

$$\mathbf{0} = \overline{\mathbf{y}}_{k+1} - \frac{1}{2}(\mathbf{y}_{k+1} + \mathbf{y}_k) - \frac{h_k}{8}(\mathbf{f}_k - \mathbf{f}_{k+1}) \qquad (\textit{Hermite}), \tag{4.103}$$

$$\mathbf{0} = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6}\left[\mathbf{f}_{k+1} + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_k\right] \qquad (\textit{Simpson}). \tag{4.104}$$

The first constraint defines the Hermite interpolant for the state at the interval midpoint, while the second constraint enforces the Simpson quadrature over the interval. It is also necessary to introduce additional NLP variables, namely the state variables at the midpoint of each interval, leading to the augmented set

$$\mathbf{x}^{\mathsf{T}} = \left(t_F, \mathbf{y}_1, \mathbf{u}_1, \boxed{\overline{\mathbf{y}}_2}, \boxed{\overline{\mathbf{u}}_2}, \mathbf{y}_2, \mathbf{u}_2, \ldots, \boxed{\overline{\mathbf{y}}_M}, \boxed{\overline{\mathbf{u}}_M}, \mathbf{y}_M, \mathbf{u}_M\right). \tag{4.105}$$

As before, the NLP constraints are

$$\mathbf{c}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.106}$$

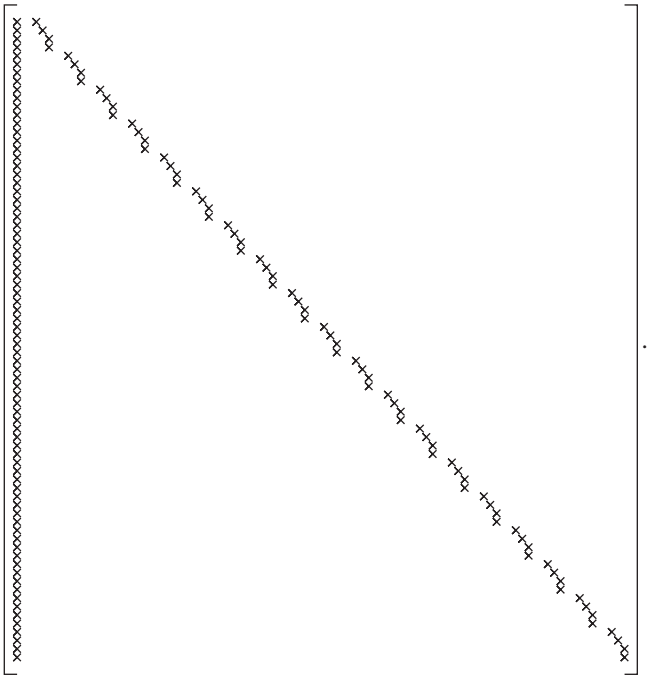
where $\mathbf{A}$ and $\mathbf{B}$ are constant matrices and

$$\mathbf{q} = \begin{bmatrix} \Delta t\, \mathbf{f}_1 \\ \Delta t\, \bar{\mathbf{f}}_2 \\ \Delta t\, \mathbf{f}_2 \\ \vdots \\ \Delta t\, \mathbf{f}_{M-1} \\ \Delta t\, \bar{\mathbf{f}}_M \\ \Delta t\, \mathbf{f}_M \end{bmatrix}. \tag{4.107}$$

Thus, by increasing the number of NLP variables and constraints, the sparsity properties of

the trapezoidal method have been retained and we find

$$\text{struct}(\mathbf{D}) = \quad . \tag{4.108}$$



It is not surprising that the separated Hermite–Simpson formulation requires $\gamma = 2$ index sets just like the trapezoidal, which is clear when comparing (4.108) with (4.91). In effect, this formulation has introduced additional variables and constraints at the interval midpoints.

4.6.7 *K*-Stage Runge–Kutta Schemes

All of the discretizations described are particular examples of K-stage Runge–Kutta schemes introduced in Section 3.5. It is worthwhile to outline how the sparsity-exploiting techniques can be extended. In general, the K-stage Runge–Kutta scheme (3.25)–(3.26) can be written as

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \sum_{j=1}^{K} \beta_j \mathbf{f}_{kj}, \tag{4.109}$$

where

$$\mathbf{y}_{kj} = \mathbf{y}_k + h_k \sum_{\ell=1}^{K} \alpha_{j\ell} \mathbf{f}_{k\ell}, \tag{4.110}$$

$$\mathbf{f}_{kj} = \mathbf{f}\left[\mathbf{y}_{kj}, \mathbf{u}_{kj}, t_{kj}\right], \tag{4.111}$$

$$\mathbf{u}_{kj} = \mathbf{u}\left(t_{kj}\right), \tag{4.112}$$

$$t_{kj} = t_k + h_k \rho_j \tag{4.113}$$

for $1 \le j \le K$. The variables at the grid points $(\mathbf{y}_k, \mathbf{u}_k)$ are termed *global* since they must be determined simultaneously over the entire interval $t_I \le t \le t_F$. The other variables,

namely $(\mathbf{y}_{kj}, \mathbf{u}_{kj})$, are *local* to the interval $t_k \le t \le t_{k+1}$. The usual approach is to eliminate the local variables—a process called *parameter condensation*. For the Hermite–Simpson method, which is a three-stage implicit Runge–Kutta scheme, the parameter-condensation process yields the NLP constraints (4.95). Unfortunately, the local elimination process is undesirable when sparsity considerations are introduced. Thus, in general, to exploit sparsity for K-stage Runge–Kutta schemes, one should introduce

1. the local variables $(\mathbf{y}_{kj}, \mathbf{u}_{kj})$ as additional NLP variables and
2. the local elimination conditions (4.110) as additional NLP constraints.

4.6.8 General Approach

In the preceding sections, it has been demonstrated that the discrete approximation to the differential equations can be formulated to exploit sparsity in the problem differential equations. Although the discussion has focused on the constraints derived from the ODEs, the concepts extend in a natural way to all of the problem functions in the NLP problem. Thus, for path-constrained optimal control problems written in the semiexplicit form

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}\big[\mathbf{y}, \mathbf{u}, \mathbf{p}, t\big], \\ \mathbf{g}_\ell \le \mathbf{g}\big[\mathbf{y}, \mathbf{u}, \mathbf{p}, t\big] &\le \mathbf{g}_u, \end{aligned}$$

the key notion is to write the complete set of transcribed NLP functions as

$$\begin{bmatrix} \mathbf{c}(\mathbf{x}) \\ F(\mathbf{x}) \end{bmatrix} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.114}$$

where $\mathbf{A}$ and $\mathbf{B}$ are constant matrices and $\mathbf{q}$ involves the nonlinear functions at grid points. Then it is necessary to construct the *sparsity template* for all of the continuous functions (4.38), that is,

$$\mathcal{T} = \text{struct}\left[\begin{array}{c|c|c} \frac{\partial \mathbf{f}}{\partial \mathbf{y}} & \frac{\partial \mathbf{f}}{\partial \mathbf{u}} & \frac{\partial \mathbf{f}}{\partial \mathbf{p}} \\ \frac{\partial \mathbf{g}}{\partial \mathbf{y}} & \frac{\partial \mathbf{g}}{\partial \mathbf{u}} & \frac{\partial \mathbf{g}}{\partial \mathbf{p}} \\ \frac{\partial \mathbf{w}}{\partial \mathbf{y}} & \frac{\partial \mathbf{w}}{\partial \mathbf{u}} & \frac{\partial \mathbf{w}}{\partial \mathbf{p}} \end{array}\right]. \tag{4.115}$$

Similar information for the nonlinear boundary functions $\boldsymbol{\psi}$ can also be incorporated. From the sparsity template information, it is possible to construct the sparsity for the matrix $\mathbf{D}$ and compute the finite difference index sets. The first derivative information needed to solve the NLP can then be computed from

$$\begin{bmatrix} \mathbf{G} \\ (\nabla F)^{\mathsf{T}} \end{bmatrix} = \mathbf{A} + \mathbf{B}\mathbf{D}. \tag{4.116}$$

It is also easy to demonstrate that the sparsity pattern for the NLP Hessian is a subset of the sparsity for the matrix $(\mathbf{B}\mathbf{D})^{\mathsf{T}}(\mathbf{B}\mathbf{D})$, which can also be constructed from the known sparsity of $\mathbf{D}$. It follows from (4.114) that the Lagrangian is

$$L(\mathbf{x}, \boldsymbol{\lambda}) = \Big[-\boldsymbol{\lambda}^{\mathsf{T}}, 1\Big]\begin{bmatrix} \mathbf{c} \\ F \end{bmatrix} = \Big[-\boldsymbol{\lambda}^{\mathsf{T}}, 1\Big]\mathbf{A}\mathbf{x} + \Big[-\boldsymbol{\lambda}^{\mathsf{T}}, 1\Big]\mathbf{B}\mathbf{q} = \boldsymbol{\sigma}^{\mathsf{T}}\mathbf{x} + \boldsymbol{\omega}^{\mathsf{T}}\mathbf{q}. \tag{4.117}$$

Since there is no second derivative contribution from the linear term $\boldsymbol{\sigma}^T\mathbf{x}$, it is then straightforward to compute the actual Hessian matrix using the sparse differencing formulas (2.6) and (2.7) with $\boldsymbol{\omega}^T = [-\boldsymbol{\lambda}^T, 1]\mathbf{B}$.

Additional computational efficiency can be achieved by exploiting separability in the problem formulation. This subject will be revisited in Section 5.4. At this point it should be emphasized that the potential benefits that accrue from exploiting sparsity in the right-hand sides cannot be achieved with methods that integrate the DAEs over multiple steps. Unfortunately, when a standard initial value method is used to integrate a coupled set of DAEs, in general the repeated blocks will be dense, not sparse. So, for example, a multiple shooting algorithm would require as many perturbations as there are state and control variables. In contrast, for the transcription method it is quite common for $\gamma \ll n_y + n_u$! More information on this subject can be found in [40].

4.6.9 Performance Issues

The preceding sections have described an approach for exploiting sparsity to reduce the cost of constructing finite difference derivatives. However, the approach does introduce an issue that can significantly affect the computational performance of the mesh-refinement process. We temporarily defer the discussion of how mesh refinement is achieved to Section 4.7. In particular, when comparing the HSC form to the HSS form, it is not readily obvious which discretization is better. Both are fourth order methods, i.e., $\mathcal{O}(h^4)$. In general, the number of index sets for the HSS form will be less than for the HSC form, i.e., $\gamma_s \leq \gamma_c$. However, in order to reduce the cost of computing derivatives, it is necessary to introduce additional NLP variables and constraints, thereby increasing the size of the NLP problem, i.e., $n_s > n_c$. Thus, it is fundamental to assess the performance penalty associated with a larger NLP versus the performance benefit associated with reduced derivative costs. To this end, let us consider the following model for *total cost per NLP iteration*:

$$\begin{aligned} T &= (\text{Finite differences}) + (\text{Linear algebra}) \\ &= \frac{1}{2} N_r \gamma(\gamma+3) T_r + cn^b \\ &\approx c_1 M \gamma(\gamma+3) T_r + c_2 M^b. \end{aligned}$$

Essentially, the total cost per iteration of an NLP is treated as the sum of two terms. The first term is attributed to the cost of computing sparse finite difference derivative approximations. As such, it depends on the number of index sets γ, the number of times the right-hand-side functions are evaluated N_r, and the corresponding right-hand-side evaluation time T_r. The second term is attributed to the operations performed by the NLP algorithm and, as such, is related to the size of the problem n. This cost model can be rewritten in terms of the common parameter M, which is the number of grid points. Clearly, this formula depends on problem-specific quantities and the discretization method. Let us denote the cost per NLP iteration for the HSS method by T_s. Similarly, let T_c denote the time for the HSC form. Now it is clear that, as the grid becomes large, $M \to \infty$ and the linear

algebra cost will dominate. Thus, for large grids, the HSC method is preferable because the problem size will be smaller. In fact, we can find a *crossover grid size* M^* such that $T_s = T_c$. Then, during the mesh-refinement process,

- use HSS when $M < M^*$ and
- use HSC when $M > M^*$.

Numerical tests on a set of 50 mesh-refinement problems suggest that the exponent $b \approx 1.9$. These performance tradeoffs are illustrated in Figure 4.3. To be more precise, the 50 problems were run with a value of $b = 1.9$ and the total solution time was recorded. Then the same set of problems was run with different values for b. The change in the total solution time with respect to the reference value at $b = 1.9$ is plotted as a * in Figure 4.3. The percentage change in the number of function evaluations (relative to the reference value at $b = 1.9$) is plotted with a solid line. Thus, by choosing $b = 1.9$, the best overall algorithm performed was obtained while keeping the number of right-hand-side evaluations small. Examples 6.13 and 6.14 describe applications with computationally expensive right-hand-side evaluations, where the HSS discretization is clearly preferable.

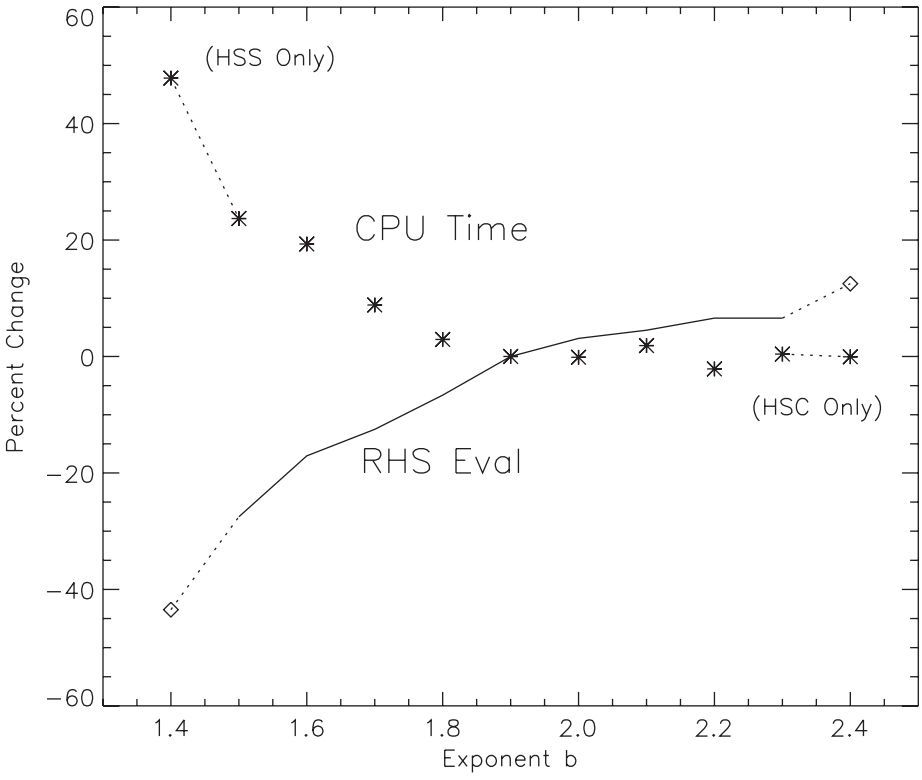


Figure 4.3. *Discretization tradeoffs.*

4.6.10 Performance Highlights

Obviously the problem sparsity will dictate how effective the method is when compared to a standard approach. This section describes how the techniques perform on a particular application that can significantly exploit these properties.

Example 4.2 HEAT EQUATION. An example describing the optimal control of a heating process is presented by Heinkenschloss [111]. It can be viewed as a simplified model for the heating of a probe in a kiln. The temperature is described by the following nonlinear parabolic partial differential equation (PDE):

$$q(x,t) = (a_1 + a_2 y)\frac{\partial y}{\partial t} - a_3\frac{\partial^2 y}{\partial x^2} - a_4\left(\frac{\partial y}{\partial x}\right)^2 - a_4 y\frac{\partial^2 y}{\partial x^2} \tag{4.118}$$

with boundary conditions given by

$$(a_3 + a_4 y)\left.\frac{\partial y}{\partial x}\right|_{x=0} = g\left[y(0,t) - u(t)\right], \tag{4.119}$$

$$(a_3 + a_4 y)\left.\frac{\partial y}{\partial x}\right|_{x=1} = 0, \tag{4.120}$$

$$y(x,0) = y_I(x). \tag{4.121}$$

The boundary $x = 1$ is the inside of the probe and $x = 0$ is the outside, where the heat source $u(t)$ is applied. The goal is to minimize the deviation of the temperature from a desired profile, as defined by the objective

$$\phi = \frac{1}{2}\int_0^T \left\{\left[y(1,t) - y_d(t)\right]^2 + \gamma u^2(t)\right\} dt, \tag{4.122}$$

by choosing the control function subject to the simple bounds

$$u_L \le u(t) \le u_U. \tag{4.123}$$

For consistency with [111], we define the specified functions

$$y_d(t) = 2 - e^{\rho t}, \tag{4.124}$$

$$y_I(x) = 2 + \cos(\pi x), \tag{4.125}$$

$$\begin{aligned} q(x,t) = {} & \left[\rho(a_1 + 2a_2) + \pi^2(a_3 + 2a_4)\right]e^{\rho t}\cos(\pi x) \\ & - a_4\pi^2 e^{2\rho t} + (2a_4\pi^2 + \rho a_2)e^{2\rho t}\cos^2(\pi x) \end{aligned} \tag{4.126}$$

and parameter values

$$\begin{aligned} &a_1 = 4, \quad a_2 = 1, \quad a_3 = 4, \quad a_4 = -1, \quad u_U = 0.1, \\ &\rho = -1, \quad T = 0.5, \quad \gamma = 10^{-3}, \quad g = 1, \quad u_L = -\infty. \end{aligned}$$

This distributed parameter optimal control problem can be cast as a lumped parameter control problem using the *method of lines*. The method of lines is an approach for

converting a system of PDEs into a system of ODEs. In so doing, the methods of this book become directly applicable. Let us consider a spatial discretization defined by

$$x_i = \frac{i-1}{N-1} \tag{4.127}$$

for $i = 0, 1, \ldots, N, N+1$, where $\delta = 1/(N-1)$. Using this discretization, the partial derivatives are approximated by

$$\frac{\partial y}{\partial x} \approx \frac{1}{2\delta}(y_{i+1} - y_{i-1}), \tag{4.128}$$

$$\frac{\partial^2 y}{\partial x^2} \approx \frac{1}{\delta^2}(y_{i+1} - 2y_i + y_{i-1}). \tag{4.129}$$

After substituting this approximation into the defining relationships and simplifying, one obtains an optimal control problem with the state vector $\mathbf{y}^{\mathsf{T}} = (y_1, \ldots, y_N)$ and the control vector $\mathbf{v}^{\mathsf{T}} = (u, y_0, y_{N+1}) = (v_1, v_2, v_3)$. The state equations are

$$\dot{y}_1 = \frac{1}{(a_1 + a_2 y_1)}\left[q_1 + \frac{1}{\delta^2}(a_3 + a_4 y_1)(y_2 - 2y_1 + v_2) + a_4\left(\frac{y_2 - v_2}{2\delta}\right)^2\right], \tag{4.130}$$

$$\dot{y}_i = \frac{1}{(a_1 + a_2 y_i)}\left[q_i + \frac{1}{\delta^2}(a_3 + a_4 y_i)(y_{i+1} - 2y_i + y_{i-1}) + a_4\left(\frac{y_{i+1} - y_{i-1}}{2\delta}\right)^2\right] \tag{4.131}$$

for $i = 2, \ldots, N-1$ and

$$\dot{y}_N = \frac{1}{(a_1 + a_2 y_N)}\left[q_N + \frac{1}{\delta^2}(a_3 + a_4 y_N)(v_3 - 2y_N + y_{N-1}) + a_4\left(\frac{v_3 - y_{N-1}}{2\delta}\right)^2\right]. \tag{4.132}$$

The boundary conditions (4.119) and (4.120) become path constraints

$$0 = g(y_1 - v_1) - \frac{1}{2\delta}(a_3 + a_4 y_1)(y_2 - v_2), \tag{4.133}$$

$$0 = \frac{1}{2\delta}(a_3 + a_4 y_N)(v_3 - y_{N-1}) \tag{4.134}$$

and the remaining condition (4.121) defines the initial condition for the states, i.e.,

$$y_i(0) = y_I(x_i). \tag{4.135}$$

Finally, the objective function is just

$$\phi = \frac{1}{2}\int_0^T \left\{\left[y_N - y_d\right]^2 + \gamma v_1^2\right\} dt. \tag{4.136}$$

For the results below, $N = 50$, and the state and path equations form a nonlinear index-one DAE system. The optimal solution is illustrated in Figures 4.4 and 4.5. These results were

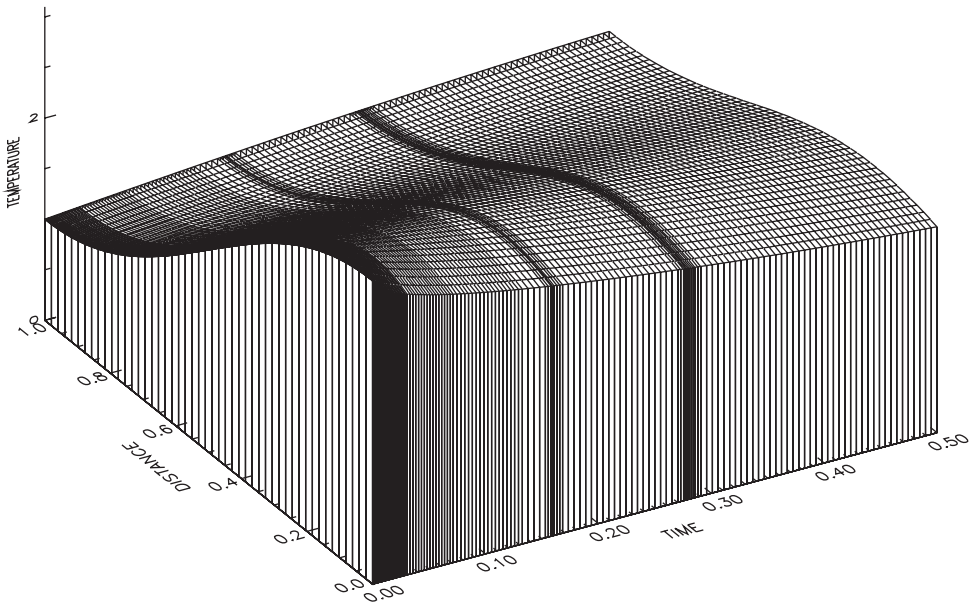


Figure 4.4. *Optimal temperature distribution.*

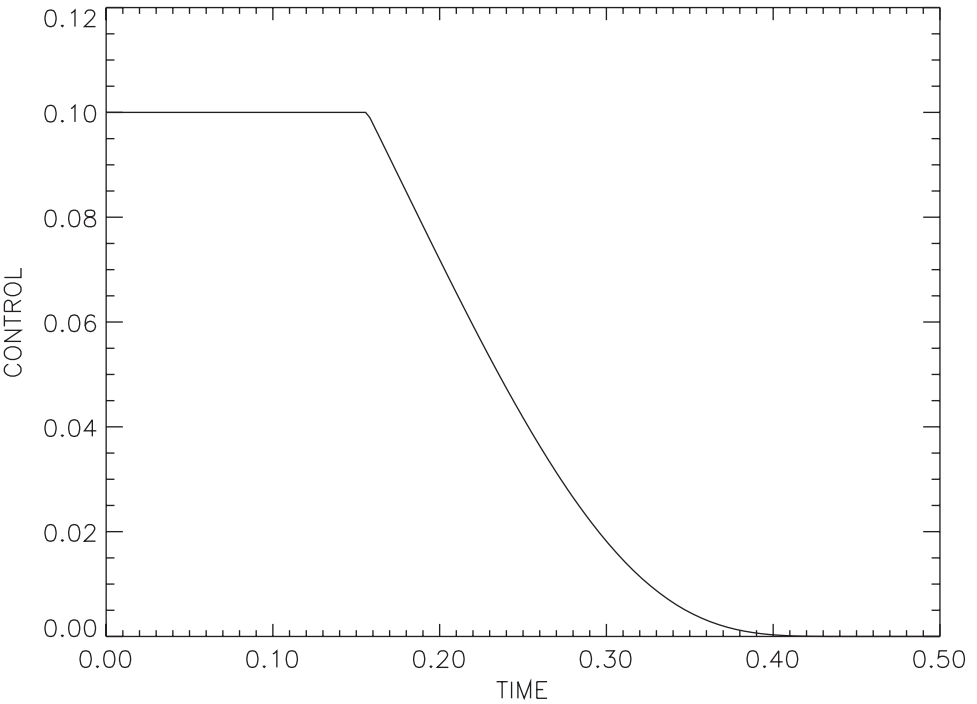


Figure 4.5. *Optimal control distribution.*

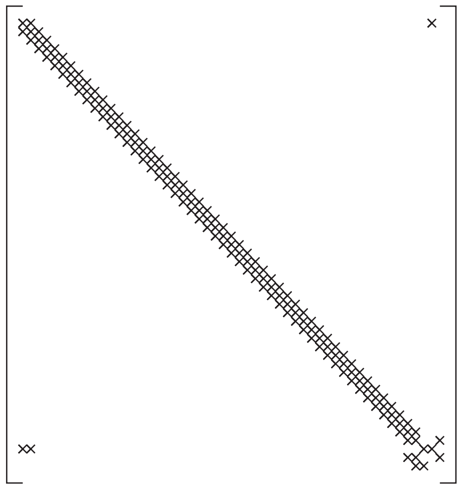
obtained after 5 mesh-refinement iterations (using the algorithm described in Section 4.7), with 164 points in the final time-dependent grid. The final NLP problem involved 9181 variables, 9025 active constraints, and 156 degrees of freedom.

The impact of sparsity on the performance of the algorithm is summarized below:

	Dense	Sparse	Reduction (%)
γ_s	53	4	−92.5%
γ_c	109	15	−86.2%
Func. eval.	42552	983	−97.7%
CPU time	1851.43	377.86	−79.6%

First, notice the dramatic reduction in the number of index sets γ for both HSS and HSC discretization forms. This is due primarily to the right-hand-side sparsity template

$$\mathcal{T} = \text{struct}\left[\begin{array}{c|c} \frac{\partial \mathbf{f}}{\partial \mathbf{y}} & \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{g}}{\partial \mathbf{y}} & \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{w}}{\partial \mathbf{y}} & \frac{\partial \mathbf{w}}{\partial \mathbf{u}} \end{array}\right] =$$



$$. \quad (4.137)$$

The comparison is even more dramatic when considering the number of function evaluations needed to solve the problem. If right-hand-side sparsity is not exploited, the number of function evaluations is 42552. In comparison, by exploiting sparsity, the problem was solved in 983 evaluations, for a reduction of 97.7%.

4.7 Mesh Refinement

To review, the *transcription method* has three fundamental steps:

Direct Transcription: *Transcribe* the optimal control problem into an NLP problem by discretization.

Sparse NLP: Solve the sparse (SQP or barrier) NLP.

Mesh Refinement: Assess the accuracy of the approximation (i.e., the finite-dimensional problem) and if necessary refine the discretization and then repeat the optimization steps.

In fact, the approach can be considered an *SNLP* or *sequential nonlinear programming algorithm*. Techniques for executing step 2 of the transcription method, that is, solving large, sparse NLP problems, are described in Chapter 2. Chapter 3 describes techniques for transcribing the original control problem into a finite-dimensional NLP problem. In the preceding sections, we described how to efficiently compute the sparse Jacobian and Hessian matrices needed by the NLP algorithm. Let us now turn our attention to step 3 in the transcription method, which is called *mesh refinement* or *grid refinement*.

4.7.1 Representing the Solution

The first step in the mesh-refinement process is to construct an approximation to the continuous solution from the information available at the solution of the NLP. Specifically, for the state variable $\mathbf{y}(t)$, let us introduce the approximation

$$\mathbf{y}(t) \approx \widetilde{\mathbf{y}}(t) = \sum_{i=1}^{n_1} \boldsymbol{\gamma}_i D_i(t), \tag{4.138}$$

where the functions $D_i(t)$ form a basis for C^1 cubic B-splines with $n_1 = 2M$, where M is the number of mesh points. The coefficients $\boldsymbol{\gamma}_i$ in the state variable representation are uniquely defined by Hermite interpolation of the discrete solution. Specifically, we require the spline approximation (4.138) to match the state at the grid points

$$\widetilde{\mathbf{y}}(t_k) = \mathbf{y}_k \tag{4.139}$$

for $k = 1, \ldots, M$. In addition, we force the derivative of the spline approximation to match the right-hand side of the differential equations, that is, from (4.32),

$$\frac{d}{dt}\widetilde{\mathbf{y}}(t_k) = \mathbf{f}_k \tag{4.140}$$

for $k = 1, \ldots, M$.

A similar technique can be used to construct an approximation for the control variables from the discrete data. Specifically, let us introduce the approximation

$$\mathbf{u}(t) \approx \widetilde{\mathbf{u}}(t) = \sum_{i=1}^{n_2} \boldsymbol{\beta}_i C_i(t). \tag{4.141}$$

When a trapezoidal discretization is used, the functions $C_i(t)$ form a basis for C^0 piecewise linear B-splines with $n_2 = M$. The coefficients $\boldsymbol{\beta}_i$ in the control variable representation are uniquely defined by interpolation of the discrete solution. Specifically, we require the spline approximation (4.141) to match the control at the grid points

$$\widetilde{\mathbf{u}}(t_k) = \mathbf{u}_k \tag{4.142}$$

for $k = 1, \ldots, M$. When a Hermite–Simpson solution is available, it is possible to use a higher-order approximation for the control. In this case, the functions $C_i(t)$ form a basis for C^0 quadratic B-splines with $n_2 = 2M - 1$. The coefficients can be defined from the

values at the grid points (4.142) as well as the values of the control at the midpoint of the interval, that is,

$$\widetilde{\mathbf{u}}\left[\frac{(t_{k+1}+t_k)}{2}\right] = \overline{\mathbf{u}}_{k+1} \tag{4.143}$$

for $k = 1, \ldots, M-1$.

It is convenient to collect the preceding results in terms of a common B-spline basis. In particular, the continuous functions can be written as

$$\begin{bmatrix} \mathbf{y}(t) \\ \mathbf{u}(t) \end{bmatrix} \approx \begin{bmatrix} \widetilde{\mathbf{y}}(t) \\ \widetilde{\mathbf{u}}(t) \end{bmatrix} = \sum_{i=1}^{N} \boldsymbol{\alpha}_i B_i(t), \tag{4.144}$$

where the functions $B_i(t)$ form a basis for C^0 cubic B-splines with $N = 3M - 2$. It is important to emphasize that, by construction, the state variables are C^1 cubics and the control will be either C^0 quadratic or linear functions, even though they are represented in the space of C^0 cubic B-splines. For a more complete discussion of B-spline approximation, the reader should consult [69]. Is there a reason to prefer a polynomial representation over some other form, such as rational function or Fourier series? Yes! Recall that an implicit Runge–Kutta scheme is a collocation method. The interpolation conditions used to construct the polynomial approximation (in B-spline form) are just the collocation conditions (3.38)–(3.39).

4.7.2 Estimating the Discretization Error

The preceding section describes an approach for representing the functions $\widetilde{\mathbf{y}}(t)$ and $\widetilde{\mathbf{u}}(t)$. The fundamental question is how well these functions approximate the true solution $\mathbf{y}(t)$ and $\mathbf{u}(t)$. To motivate the discussion, let us reconsider the simplified form of the problem introduced in Section 4.1. Suppose we must choose the control functions $\mathbf{u}(t)$ to minimize (4.1) subject to the state equations (4.2) and the boundary conditions (4.3). The initial conditions $\mathbf{y}(t_I) = \mathbf{y}_I$ are given at the fixed initial time t_I, and the final time t_F is free. As stated, solving the necessary conditions is a two-point BVP in the variables $\mathbf{y}(t)$, $\mathbf{u}(t)$, *and* $\boldsymbol{\lambda}(t)$. In fact, many of the refinement ideas we will discuss are motivated by boundary value methods (cf. [2]).

A direct transcription method does not explicitly form the necessary conditions (4.7)–(4.10). In fact, one of the major reasons direct transcription methods are popular is that it is not necessary to derive expressions for $\mathbf{H}_y$, $\mathbf{H}_u$, and $\boldsymbol{\Phi}_y$ and it is not necessary to estimate values for the adjoint variables $\boldsymbol{\lambda}(t)$. On the other hand, because the adjoint equations are not available, they cannot be used to assess the accuracy of the solution. Consequently, we choose to address a different measure of discretization error. Specifically, we *assume* $\widetilde{\mathbf{u}}(t)$ is correct (and optimal) and estimate the error between $\widetilde{\mathbf{y}}(t)$ and $\mathbf{y}(t)$. This is a subtle, but very important, distinction, for it implies that optimality of the control history $\widetilde{\mathbf{u}}(t)$ is not checked when measuring the discretization error. The functions $\widetilde{\mathbf{u}}(t)$ are constructed to interpolate the discrete values $\mathbf{u}_k$ as described in the previous section. The discrete values $\mathbf{u}_k$ are the solution of an NLP problem and satisfy the NLP (KKT) necessary conditions. However, only in the limit as $h_k \to 0$ do the KKT conditions become equivalent to the necessary conditions (4.7)–(4.10). Computational experience will be presented that tends to corroborate the validity of this approach.

Specifically, let us estimate the error in the state variables as a result of the discretization (4.56) or (4.58). We restrict this analysis to the class of controls that can be represented as C^0 quadratic B-splines. This restriction in turn implies that one can expect to accurately solve an optimal control problem provided

1. the optimal state variable $\mathbf{y}(t)$ is C^1 and
2. the optimal control variable $\mathbf{u}(t)$ is C^0

within the phase, i.e., for $t_I \leq t \leq t_F$. On the other hand, the solution to the optimal control problem as posed in (4.32)–(4.35) may in fact require discontinuities in the control and/or state derivatives. In particular, when the path constraints do not involve the control variable explicitly, the optimal solution may contain *corners*. Similarly, when the control appears linearly in the differential equations, *bang-bang* control solutions can be expected. Consequently, if the transcription method described is applied to problems of this type, some inaccuracy must be expected unless the locations of discontinuities are introduced explicitly as phase boundaries. Thus, we will be satisfied with accurately solving problems when the control is continuous and the state is differentiable. If this is not true, we will be satisfied if the method "does something reasonable."

When analyzing the behavior of an integration method, it is common to ascribe an *order of accuracy* to the algorithm (cf. [66]). Typically, the solution of an ODE is represented by an expansion of the form

$$\mathbf{y}(t,h) = \mathbf{y}(t) + \sum_{i=p}^{\infty} \mathbf{c}_i(t) h^i. \tag{4.145}$$

The *global error* at a point t_{k+1} is the difference between the computed solution $\mathbf{y}_{k+1}$ and the exact solution $\mathbf{y}(t_{k+1})$. The *local error* is the difference between the computed solution $\mathbf{y}_{k+1}$ and the solution of the differential equation that passes through the computed point $\mathbf{y}_k$. For ODEs, typically if the global error is $\mathcal{O}(h^p)$, then the local error is $\mathcal{O}(h^{p+1})$. Thus, if the order of accuracy of a method is p, the local error for the method at step k is of the form

$$\epsilon_k \approx \|\mathbf{c}_k h^{p+1}\|, \tag{4.146}$$

where the coefficients $\mathbf{c}_k$ typically depend on partial derivatives of the right-hand side $\mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t]$. The Hermite–Simpson discretization (4.58) is of order $p = 4$, while the trapezoidal discretization (4.56) is of order $p = 2$.

The standard order analysis of a system of ODEs is modified when considering a system of DAEs [48]. In particular, when one or more of the path constraints (4.33) is active, the constrained arc is characterized by a DAE of the form

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t], \tag{4.147}$$

$$0 = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{4.148}$$

The index of a DAE is one measure of how singular the DAE is. It is known that numerical methods may experience an order reduction when applied to a DAE. When a path constraint becomes active, the index of the DAE may change, and there can be a corresponding change in the order of the method. As a result, for a path-constrained problem, we must assume that (4.146) is replaced by

$$\epsilon_k \approx \|\mathbf{c}_k h^{p-r+1}\|, \tag{4.149}$$

where r is the order reduction. Thus, we expect that for some problems the index and hence the order reduction will change as a function of time t. Unfortunately, in general, the index of the DAE is unknown and, therefore, the order reduction is also not known.

There is a further problem in that the theory for IRK methods applied to DAEs usually leads to different amounts of order reduction in different variables [48]. This is true for both the trapezoidal and Hermite–Simpson methods [60, 118]. In addition, the difference between the local and global errors can sometimes be greater than one. In a complex optimization problem, the activation and deactivation of constraints cannot only change the index but can change what the index of a particular variable is. We distinguish only between the control and the state so that order reduction is always taken to be the largest order reduction in all the state variables. In contrast to most traditional methods, let us estimate the order reduction and use this information to improve the refinement process. Consequently, it is not important to determine why the order is reduced but rather by how much [25, 24].

To estimate the order reduction, we need to first estimate the discretization error on a given mesh. There are a number of ways to do so. As previously stated, in estimating the local error we assume that the computed control is correct. Consider a single interval $t_k \le t \le t_k + h_k$, that is, a single integration step. Suppose the NLP problem has produced a spline solution $\widetilde{\mathbf{y}}(t)$ and $\widetilde{\mathbf{u}}(t)$ to the ODEs (4.147). From (4.147),

$$\mathbf{y}(t_k + h_k) = \mathbf{y}(t_k) + \int_{t_k}^{t_k+h_k} \dot{\mathbf{y}}\,dt = \mathbf{y}(t_k) + \int_{t_k}^{t_k+h_k} \mathbf{f}\big[\mathbf{y}, \mathbf{u}, t\big]\,dt. \tag{4.150}$$

Observe that this expression for $\mathbf{y}(t_k + h_k)$ involves the true value for both $\mathbf{y}$ and $\mathbf{u}$, which are unknown. Consequently, we may consider the approximation

$$\widehat{\mathbf{y}}(t_k + h_k) \equiv \mathbf{y}(t_k) + \int_{t_k}^{t_k+h_k} \mathbf{f}\big[\widetilde{\mathbf{y}}(t), \widetilde{\mathbf{u}}(t), t\big]\,dt, \tag{4.151}$$

where the spline solution $\widetilde{\mathbf{y}}(t)$ and $\widetilde{\mathbf{u}}(t)$ appears in the integrand. A second alternative is the expression

$$\widehat{\mathbf{y}}(t_k + h_k) \equiv \mathbf{y}(t_k) + \int_{t_k}^{t_k+h_k} \mathbf{f}\big[\mathbf{y}(t), \widetilde{\mathbf{u}}(t), t\big]\,dt, \tag{4.152}$$

where the real solution $\mathbf{y}(t)$ and the spline approximation $\widetilde{\mathbf{u}}(t)$ appear in the integrand.

With either (4.151) or (4.152), we can define the *discretization error* on the kth mesh iteration as

$$\eta_k = \max_i \{a_i\, |\widetilde{y}_i(t_k + h_k) - \widehat{y}_i(t_k + h_k)|\} \tag{4.153}$$

for $i = 1, \ldots, n$, where the weights a_i are chosen to appropriately normalize the error.

However, our particular need for these estimates imposes certain special restrictions. First, we want them to be part of a method that will be used on a wide variety of problems, including some problems that are stiff. Second, we want to use the estimates on coarse grids. Unfortunately, an estimate computed from (4.152) based on an explicit integrator may be unstable on coarse grids. While (4.152) might be the most accurate, its computation would require an explicit integration of $\dot{\mathbf{y}} = \mathbf{f}\big[\mathbf{y}, \widetilde{\mathbf{u}}, t\big]$ on a possibly large grid with tight error control. This is particularly unfortunate since both of the primary discretization methods (trapezoidal and Hermite–Simpson) are implicit schemes with very good stability

properties. These special requirements lead us to abandon (4.152) and focus on (4.151). First, let us rewrite (4.151) as

$$\begin{aligned}\widehat{\mathbf{y}}_{k+1} &= \mathbf{y}_k + \int_{t_k}^{t_k+h_k} \widetilde{\mathbf{f}} dt \\ &= \mathbf{y}_k + \int_{t_k}^{t_k+h_k} \dot{\widetilde{\mathbf{y}}} dt - \int_{t_k}^{t_k+h_k} \dot{\widetilde{\mathbf{y}}} dt + \int_{t_k}^{t_k+h_k} \widetilde{\mathbf{f}} dt \\ &= \mathbf{y}_k + \widetilde{\mathbf{y}}_{k+1} - \widetilde{\mathbf{y}}_k - \int_{t_k}^{t_k+h_k} [\dot{\widetilde{\mathbf{y}}} - \widetilde{\mathbf{f}}] dt.\end{aligned}$$

By definition, there is no error at the beginning of the interval for a local error estimate, so we can set $\mathbf{y}_k = \widetilde{\mathbf{y}}_k$, which leads to

$$\widetilde{\mathbf{y}}_{k+1} - \widehat{\mathbf{y}}_{k+1} = \int_{t_k}^{t_k+h_k} [\dot{\widetilde{\mathbf{y}}} - \widetilde{\mathbf{f}}] dt.$$

Taking absolute values of each component, we then obtain the bound

$$\left|\widetilde{\mathbf{y}}_{i,k+1} - \widehat{\mathbf{y}}_{i,k+1}\right| = \left| \int_{t_k}^{t_k+h_k} \left[\dot{\widetilde{\mathbf{y}}}_i - \widetilde{\mathbf{f}}_i\right] dt \right| \le \int_{t_k}^{t_k+h_k} |\dot{\widetilde{\mathbf{y}}}_i - \widetilde{\mathbf{f}}_i| dt.$$

Therefore, let us define the *absolute local error* on a particular step by

$$\boldsymbol{\eta}_{i,k} = \int_{t_k}^{t_{k+1}} |\boldsymbol{\varepsilon}_i(s)| ds, \tag{4.154}$$

where

$$\boldsymbol{\varepsilon}(t) = \dot{\widetilde{\mathbf{y}}}(t) - \mathbf{f}[\widetilde{\mathbf{y}}(t), \widetilde{\mathbf{u}}(t), t] \tag{4.155}$$

defines the error in the differential equation as a function of t. Notice that the arguments of the integrand use the spline approximations (4.144) for the state and control evaluated at intermediate points in the interval. From this expression for the absolute error, we can define the *relative local error* by

$$\epsilon_k \approx \max_i \frac{\boldsymbol{\eta}_{i,k}}{(w_i + 1)}, \tag{4.156}$$

where the scale weight

$$w_i = \max_{k=1}^{M} \left[|\tilde{y}_{i,k}|, |\dot{\tilde{y}}_{i,k}| \right] \tag{4.157}$$

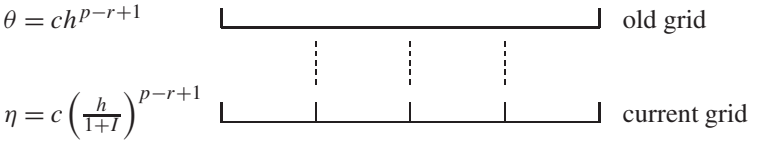
defines the maximum value for the ith state variable or its derivative over the M grid points in the phase. Notice that ϵ_k is the maximum relative error over all components i in the state equations $\dot{\mathbf{y}} - \mathbf{f}$ evaluated in the interval k.

The approximations (4.153) and (4.156) are similar; however, they differ in two respects. The weightings are quite different. Also, (4.153) emphasizes the error in prediction, which is typical with ODE integrators, while (4.156) emphasizes the error in solving the equations. Since the latter is closer to the $\mathbb{SOCS}$ termination criteria, we will use (4.156) in the ensuing algorithms.

Now let us consider how to compute an estimate for the error η_k. Since this discretization error is essential for estimating the order reduction, we choose to construct an accurate estimate for the integral (4.154). Because the spline approximations for the state and control are used, the integral (4.154) can be evaluated using a standard quadrature method. In particular, we compute the integral using a Romberg quadrature algorithm with a tolerance close to machine precision.

4.7.3 Estimating the Order Reduction

In order to use the formula (4.149), it is necessary to know the order reduction r. We propose computing this quantity by comparing the behavior on two successive mesh-refinement iterations. Specifically, assume that the current grid was obtained by subdividing the old grid; that is, the current grid has more points than the old grid. Let us focus on the behavior of a *single* variable on a *single* interval in the old grid as illustrated below:



Denote the discretization error on the old grid by θ. The error on an interval in the old grid is then

$$\theta = ch^{p-r+1}, \tag{4.158}$$

where p is the order of the discretization on the old grid and h is the stepsize for the interval. If the interval on the old grid is subdivided by adding I points, the resulting discretization error is

$$\eta = c\left(\frac{h}{1+I}\right)^{p-r+1}. \tag{4.159}$$

If we assume the order reduction r and the constant c are the same on the old and current grids, then we can solve (4.158) and (4.159) for these quantities.

Solving gives

$$\hat{r} = p + 1 - \frac{\log(\theta/\eta)}{\log(1+I)}. \tag{4.160}$$

Choosing an integer in the correct range, the estimated order reduction is given by

$$r = \max\left[0, \min\left(\text{nint}(\hat{r}), p\right)\right], \tag{4.161}$$

where nint denotes the nearest integer. As a final practical matter, we assume that the order reduction is the same for all $I+1$ subdivisions of the old interval. Thus, if an interval on the old grid was subdivided into three equal parts, we assume the order reduction is the same over all three parts. Thus, the resolution of our order-reduction estimates is dictated by the old, coarse grid.

To summarize, we compare the discretization errors for each interval on the old grid, i.e., θ_k, with the corresponding discretization errors on the current grid. Because the current grid is constructed by subdividing the old grid, we can then compute the estimated order reduction for all intervals on the current grid.

While (4.160) provides a formula for the observed order reduction, this estimate is sensitive to the computed discretization error estimates η and θ. To appreciate the sensitivity, let us assume q, p are the orders of the Hermite–Simpson discretization on the old and current grids. Generalizing the expression (4.160), define the function

$$Q(a,b) = q + 1 - \frac{\log(a/b)}{\log(1+I)}.$$

Notice that

$$Q(p_1 a, p_2 b) = Q(a,b) - \frac{\log(p_1/p_2)}{\log(1+I)}.$$

Here we are thinking of p_1/p_2 as the ratio in the computed discretization errors. Notice that $p_1/p_2 = 1.07$ with $I = 1$ gives a reduction of 0.1, while $p_1/p_2 = 1.15$ gives a reduction of 0.2. Thus, a change of 15% in the discretization error estimate could easily alter the estimated value of r if it reduced, say, from 1.65 to 1.45. In order to deal with this sensitivity in the mesh refinement, we have

1. computed the discretization errors using a very accurate quadrature method to evaluate (4.154) and
2. computed the weights (4.157) only once at the end of the first refinement iteration.

4.7.4 Constructing a New Mesh

The purpose of this section is to delineate an approach for constructing a new mesh using information about the discretization error on a current mesh. We will use the terminology "old grid" to refer to the previous refinement iteration, "current grid" to refer to the current iteration, and "new grid" when describing the *next* iteration. Certainly the primary goal is to create a new mesh with less discretization error than the current one. On the other hand, simply adding a large number of points to the current grid increases the size of the NLP problem to be solved, thereby causing a significant computational penalty. Briefly, then, the goal is to reduce the discretization error as much as possible using a *specified* number of new points.

To motivate the discussion, let us consider the simple example illustrated in Figure 4.6, which shows the discretization error and the stepsize as a function of the normalized interval τ. Suppose that the old grid has five intervals (six grid points) and the largest discretization error is in interval three. If interval three is subdivided by adding a grid point, the corresponding discretization error should be reduced as illustrated by the dark shaded region in Figure 4.6. Obviously, the process can be repeated, each time adding a point to the interval with the largest discretization error. Thus, a new mesh can be constructed by successively adding points to the intervals with the largest discretization errors.

In the preceding section, an approach was described for computing an error estimate for each segment or interval in the current grid. By equating (4.149) with (4.156), we obtain

$$\|\mathbf{c}_k\| h^{p-r_k+1} = \max_i \frac{\boldsymbol{\eta}_{i,k}}{(w_i+1)} \tag{4.162}$$

so that

$$\|\mathbf{c}_k\| = \max_i \frac{\boldsymbol{\eta}_{i,k}}{(w_i+1)h^{p-r_k+1}}. \tag{4.163}$$

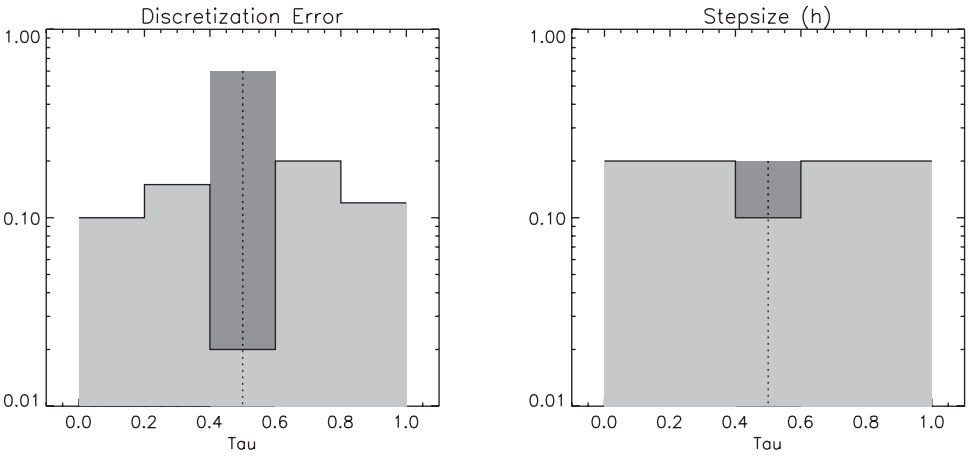


Figure 4.6. *Subdividing the grid.*

Let us suppose we are going to *subdivide* the current grid; i.e., the new grid will contain the current grid points. Let us define the integer I_k as the number of points to add to interval k, so that from (4.149) and (4.163) we may write

$$\epsilon_k \approx \|c_k\| \left(\frac{h}{1+I_k} \right)^{p-r_k+1} = \max_i \frac{\eta_{i,k}}{(w_i+1)} \left(\frac{1}{1+I_k} \right)^{p-r_k+1} \tag{4.164}$$

for integers $I_k \geq 0$. Specifically, if we add I_k points to interval k in the current mesh, this is an approximation for the error on *each* of the $1+I_k$ subintervals. Then the new mesh can be constructed by choosing the set of integers I_k to minimize

$$\phi(I_k) = \max_k \epsilon_k \tag{4.165}$$

and satisfy the constraints

$$\sum_{k=1}^{n_s} I_k \leq M - 1 \tag{4.166}$$

and

$$I_k \leq M_1 \tag{4.167}$$

for $k = 1, \ldots, n_s$. Essentially, we want to minimize the maximum error over all of the intervals in the current mesh by adding at most $M-1$ total points. In addition, the number of points that are added to a single interval is limited to M_1. This restriction is incorporated in order to avoid an excessive number of subdivisions in a single interval. In fact, in our computational implementation we terminate the mesh-refinement process when a single interval has M_1 points. Equations (4.165)–(4.167) define a nonlinear *integer programming problem*.

This approach formalizes a number of reasonable properties for a mesh-refinement procedure. When the errors on each interval of the current mesh are approximately the same, i.e.,

$$\epsilon_1 \approx \epsilon_2 \approx \cdots \approx \overline{\epsilon}, \tag{4.168}$$

we say that the error is *equidistributed*, where the average error

$$\overline{\epsilon} = \frac{1}{M}\sum_{k=1}^{M}\epsilon_k. \tag{4.169}$$

In this case, the new mesh defined by the integer programming problem (4.165)–(4.167) will simply subdivide each interval in the current mesh. On the other hand, the error for the current mesh may be dominated by the error on a single interval α, i.e.,

$$\epsilon_\alpha \gg \epsilon_k \tag{4.170}$$

for $k = 1,\ldots,n_s$ with $k \neq \alpha$. In this case, the solution to (4.165)–(4.167) will require adding as many as M_1 points into interval α. Typically, we use $M_1 = 5$.

4.7.5 The Mesh-Refinement Algorithm

Let us now summarize the procedure for mesh refinement. Denote the mesh-refinement iteration number by j_r. Assume the current grid has M points. The goal of the mesh-refinement procedure is to select the number and location of the grid points in the new mesh as well as the order of the new discretization. Typically, we begin with a low-order discretization and switch to a high-order method at some point in the process. In our software implementation, the default low/high-order pairs are trapezoidal and Hermite–Simpson, respectively. The desired error tolerance is δ, and we would like the new mesh to be constructed such that it has an error below this tolerance. In fact, when making predictions, we would like the *predicted* errors to be "safely" below, say, $\hat{\delta} = \kappa\delta$, where $0 < \kappa < 1$. Typically, we set $\kappa = 1/10$. The procedure begins with values for the discretization error on all intervals in the current mesh, i.e., ϵ_k for $k = 1,\ldots,n_s$, and we initialize $I_k = 0$.

Mesh-Refinement Algorithm

1. **Construct Continuous Representation.** Compute the cubic spline representation (4.144) from the discrete solution $\mathbf{x}^*$.

2. **Estimate Discretization Error.** Compute an estimate for the discretization error ϵ_k in each segment of the current mesh, that is, evaluate (4.154) using Romberg quadrature; compute the average error from (4.169).

3. **Select Primary Order for New Mesh.**

   (a) If the error is equidistributed for the low-order method, increase the order; i.e., if $p < 4$ and $\epsilon_\alpha \leq 2\overline{\epsilon}$, then set $p = 4$ and terminate.

   (b) Otherwise, if ($p < 4$) and $j_r > 2$, then set $p = 4$ and terminate.

4. **Estimate Order Reduction**. Compare the current and old grids to compute r_k from (4.160) and (4.161).

5. **Construct New Mesh**.

   (a) Compute the interval α with maximum error, i.e.,

   $$\epsilon_\alpha = \max_k \epsilon_k. \tag{4.171}$$

   (b) Terminate if

   - M' points have been added ($M' \geq \min[M_1, \kappa M]$)

   and

   - the *error is within tolerance:* $\epsilon_\alpha \leq \delta$ and $I_\alpha = 0$ or
   - the *predicted error is safely within tolerance:* $\epsilon_\alpha \leq \kappa\delta$ and $0 < I_\alpha < M_1$ or
   - $M - 1$ points have been added or
   - M_1 points have been added to a *single* interval.

   (c) Add a point to interval α, i.e., $I_\alpha \leftarrow I_\alpha + 1$.

   (d) Update the predicted error for interval α from (4.164).

   (e) Return to step 5(a).

Observe that early in the mesh-refinement process, when the discretization error estimates are crude, we limit the growth so that the new mesh will have at most $2M - 1$ points. The intent is to force a new NLP solution, which presumably will lead to better estimates of the error. Furthermore, in step 5(b) of the procedure, when $I_\alpha \neq 0$, the error ϵ_α is predicted (and presumably less reliable). In this case, we force it to be safely less than the tolerance before stopping. In addition, the refinement is not terminated without adding a minimum number of grid points. This is done to preclude a sequence of refinement iterations that add only one or two points.

The procedure used to modify the order and size of the mesh is somewhat heuristic and is designed to be efficient on most applications. Because the trapezoidal method is both robust and efficient when computing sparse finite difference derivatives, the intent is to solve the initial sparse NLP problem using a trapezoidal discretization. In fact, computational experience demonstrates the value of initiating the process with a coarse mesh and low-order method. On a set of 49 optimal control test problems, on average, the strategy requires 17.1% fewer evaluations of the right-hand-side functions **f** and **g** than a strategy that begins with the Hermite–Simpson discretization. On the other hand, in the software implementation (SOCS) [38], it is possible to specify the initial discretization, which may be effective when the user can provide a good initial guess for the solution. If the discretization error appears to be equidistributed, it is reasonable to switch to a higher-order method (i.e., Hermite–Simpson). However, when the error is badly distributed, at least two different discrete solutions are obtained before the order is increased. The default low-order (trapezoidal) and high-order (Hermite–Simpson) schemes are both IRK methods. The method used for constructing a new mesh always results in a subdivision of the current mesh, which has been found desirable in practice. The minimax approach to adding points is designed to emphasize equidistributing the error when only a limited number of points are included in the new grid.

Is mesh refinement necessary? The examples in Sections 6.1 and 7.1.8 demonstrate just how important it is.

4.7.6 Computational Experience

The mesh-refinement procedure was tested on a standard set of $\mathbb{SOCS}$ test problems. The collection consists of 53 optimal control problems and/or BVPs with path constraints, various degrees of nonlinearity, and computational complexity. For comparison, the same problems were solved using the old mesh-refinement strategy described in [39], which did not estimate the order reduction. The results are summarized below.

Performance Summary (Old versus New)	
Total time decrease (16601.63 versus 12956.20)	−22%
Average time change (all problems)	−0.23%
Maximum time increase (all problems)	109.31%
Minimum time decrease (all problems)	−62.20%

Examination of the results suggests that for most problems, there was little difference in the two techniques. However, since the total time for the test set was noticeably reduced, this also suggests that there was very significant improvement on at least a few of the problems. Clearly, for problems that do not exhibit any significant index reduction, little change can be expected. On the other hand, for *some* "hard" problems, there is apparently a noticeable improvement.

Example 4.3 ALP RIDER. To illustrate the method, let us consider a problem specifically constructed to be hard. The system is defined by the following DAEs:

$$\begin{aligned}
\dot{y}_1 &= -10y_1 + u_1 + u_2,\\
\dot{y}_2 &= -2y_2 + u_1 + 2u_2,\\
\dot{y}_3 &= -3y_3 + 5y_4 + u_1 - u_2,\\
\dot{y}_4 &= 5y_3 - 3y_4 + u_1 + 3u_2,\\
y_1^2 + y_2^2 + y_3^2 + y_4^2 &\geq 3p(t,3,12) + 3p(t,6,10) + 3p(t,10,6) + 8p(t,15,4) + 0.01,
\end{aligned}$$

where the exponential "peaks" $p(t,a,b) = e^{-b(t-a)^2}$. The system of differential equations is stiff with eigenvalues $\{-10,-2,-3\pm 5i\}$. Notice also that the single state variable inequality path constraint is defined in terms of the peak functions. The system has boundary conditions

$$\begin{aligned}
\mathbf{y}^{\mathsf{T}}(0) &= [2,1,2,1],\\
\mathbf{y}^{\mathsf{T}}(20) &= [2,3,1,-2]
\end{aligned}$$

and the goal is to minimize the objective function

$$J(\mathbf{y},\mathbf{u}) = \int_0^{20} 10^2(y_1^2 + y_2^2 + y_3^2 + y_4^2) + 10^{-2}(u_1^2 + u_2^2)dt.$$

We refer to this as the Alp rider problem because the minimum value for the objective function tends to force the state to ride the path constraint. Figure 4.7 illustrates the peaks for the path-constraint function for this problem.

In fact, this problem is very similar to the path encountered by a terrain-following aircraft. This is quite obvious in the solution illustrated in Figure 4.8, which demonstrates peaks at the locations $(3,6,10,15)$.

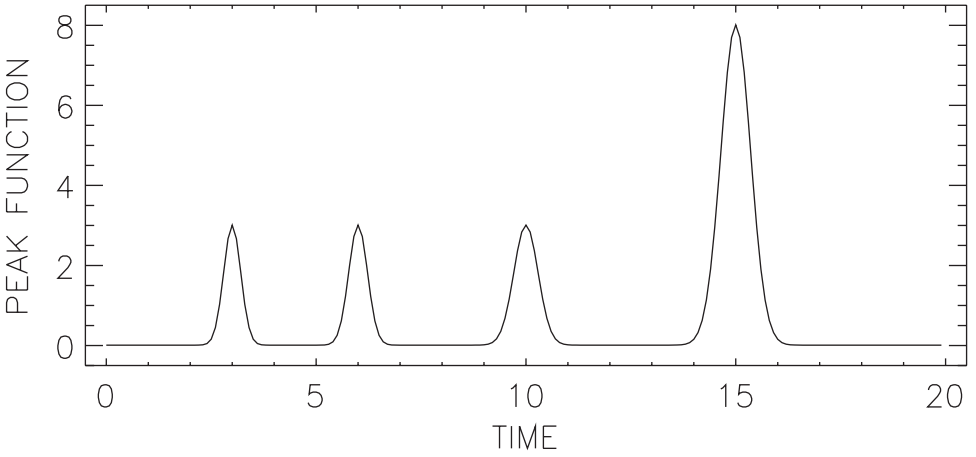


Figure 4.7. *Alp rider peaks.*

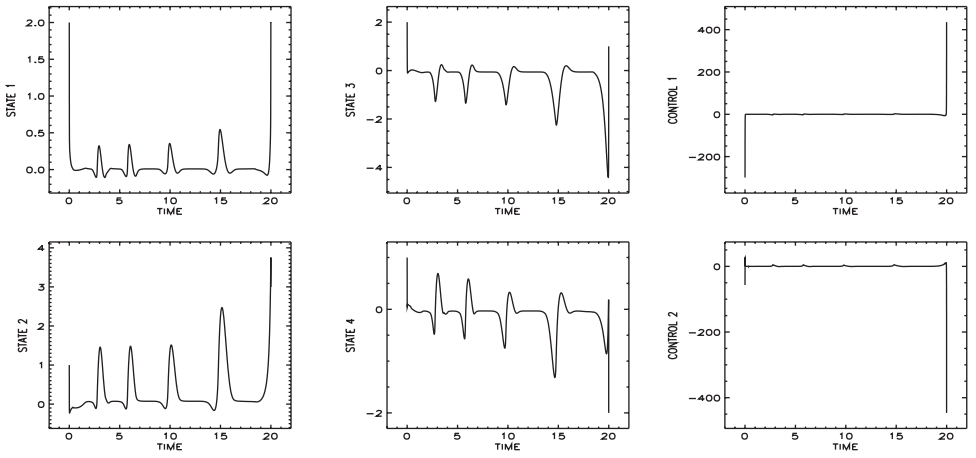


Figure 4.8. *Alp rider solution.*

The Alp rider example required 11 mesh-refinement iterations, which are tabulated in the rows of Table 4.2. For the results in this table, we initiated the algorithm with the Hermite–Simpson discretization, although results for the default method are very similar. The first iteration began with 21 equally spaced grid points (NPT) and was solved after 66 gradient evaluations (NGC) and 62 Hessian evaluations (NHC) for a total of 7529 function evaluations (NFE), including finite difference perturbations. This solution required 308689 evaluations of the right-hand sides (NRHS) of the DAEs and produced a solution with a discretization error (ERRODE) of 0.32×10^0, which was obtained in 0.22×10^2 sec. Using the solution from the first iteration as an initial guess, the second solution, using 41 grid points, was obtained after an additional 91 Hessian evaluations. Notice that it is necessary to substantially increase the size of the mesh before the solution of the NLP problem is

Table 4.2. *Alp rider performance summary.*

GRID	NPT	NGC	NHC	NFE	NRHS	ERRODE	CPU (sec)
1	21	66	62	7529	308689	0.32×10^{0}	0.22×10^{2}
2	41	93	91	10969	888489	0.29×10^{-1}	0.66×10^{2}
3	76	37	35	4244	640844	0.74×10^{-2}	0.63×10^{2}
4	84	19	16	1993	332831	0.89×10^{-3}	0.35×10^{2}
5	119	26	23	2833	671421	0.18×10^{-3}	0.65×10^{2}
6	194	18	16	1964	760068	0.31×10^{-4}	0.95×10^{2}
7	253	17	15	1844	931220	0.11×10^{-4}	0.14×10^{3}
8	304	10	8	1004	609428	0.37×10^{-5}	0.16×10^{3}
9	607	6	4	524	635612	0.35×10^{-6}	0.54×10^{3}
10	785	5	3	404	633876	0.16×10^{-6}	0.92×10^{3}
11	992	5	3	404	801132	0.29×10^{-7}	0.14×10^{4}
	992	302	276	33712	7213610		3543.81

obtained quickly; this suggests that the quadratic convergence region for this application is very small. For comparison, the old refinement method [39] required 27370.44 sec, 2494 grid points, and 10 iterations to solve this problem. This substantial difference in computation time can be attributed to the size of the final mesh. In particular, the computational cost of the NLP problem is related to the number of grid points and, since the final grid for the old method is approximately 2.5 times as large, it is not surprising that the computational cost for the old method is 7.7 times larger than for the new approach.

Figure 4.9 illustrates the behavior of the refinement algorithm on the Alp rider example. The first iteration is shown with the darkest shading and the last iteration has the lightest shading. Observe that the early refinement steps tend to cluster points near the boundaries and peaks where the calculated discretization error is largest. The final mesh-refinement iterations tend to have a more uniform distribution of error because the grid points have been clustered in the appropriate regions.

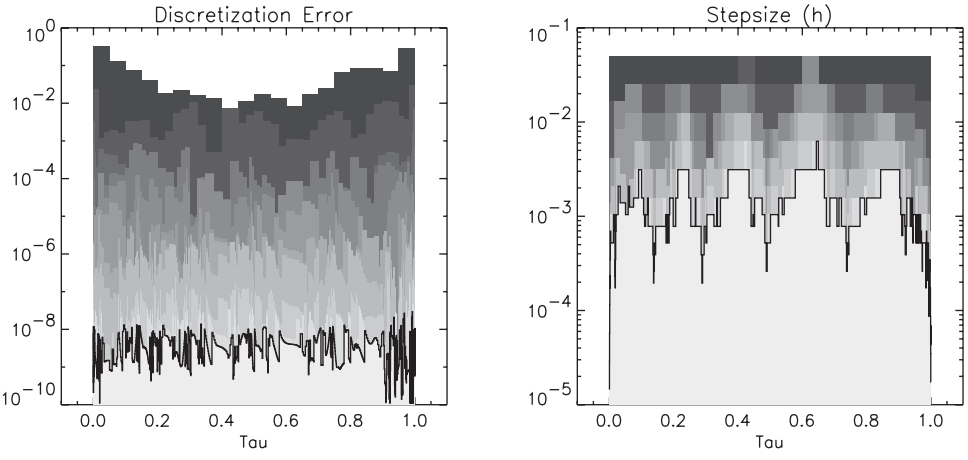


Figure 4.9. *Alp rider mesh-refinement history.*

4.8 Scaling

In Section 1.16, we discussed the importance of scaling an NLP problem in order to obtain robust and rapid convergence to a solution. The special structure of the optimal control problem can be exploited to improve the scaling of the underlying NLP subproblem. Many of the trajectory problems (e.g., Example 6.1) discussed in the next chapter are formulated using quantities with significant differences in the units for the state variables. For example, some of the state variables represent angles ranging from $-\pi$ to π, where others, like altitude, may range from 0 to 10^6 ft. Thus, scaling is necessary to make the ranges of the variables more uniform. While no scaling method is universally successful, the technique presented is often helpful.

To make the analysis physically meaningful, we would like to choose scaling in the control setting rather than the NLP setting. In other words, we would like to choose scaling for the state variables and the control variables. Thus, for the purposes of this analysis, let us assume that the scaling is the same over all grid points and temporarily drop the grid-point notation. Rewriting the NLP variable scaling (1.148) in vector form and then applying it to the control setting, we find

$$\begin{bmatrix} \widetilde{\mathbf{y}} \\ \widetilde{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_y & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_u \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{u} \end{bmatrix} + \begin{bmatrix} \mathbf{r}_y \\ \mathbf{r}_u \end{bmatrix}, \tag{4.172}$$

which relates the scaled state $\widetilde{\mathbf{y}}$ and control $\widetilde{\mathbf{u}}$ to the unscaled quantities $\mathbf{y}$ and $\mathbf{u}$. The $n_y \times n_y$ diagonal matrix $\mathbf{V}_y$ contains the state variable scale weights and the $n_u \times n_u$ diagonal matrix $\mathbf{V}_u$ contains the control variable scale weights. The corresponding variable shifts are defined by the vectors $\mathbf{r}_y$ and $\mathbf{r}_u$.

In general, we impose defect constraints $\boldsymbol{\zeta} = \mathbf{0}$ and path constraints $\mathbf{g} = \mathbf{0}$. Thus, from (1.149) we expect

$$\widetilde{\mathbf{c}} = \begin{bmatrix} \mathbf{W}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_g \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta} \\ \mathbf{g} \end{bmatrix} \tag{4.173}$$

will relate the scaled constraints $\widetilde{\mathbf{c}}$ to the unscaled constraints $\boldsymbol{\zeta}$ and $\mathbf{g}$. Here $\mathbf{W}_f$ is an $n_y \times n_y$ diagonal matrix of differential equation constraint scale weights and $\mathbf{W}_g$ is an $n_g \times n_g$ diagonal matrix of path-constraint scale weights.

The Jacobian matrix in the scaled quantities is given by

$$\widetilde{\mathbf{G}} = \begin{bmatrix} \mathbf{W}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_g \end{bmatrix} \mathbf{G} \begin{bmatrix} \mathbf{V}_y & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_u \end{bmatrix}^{-1}. \tag{4.174}$$

Now all of the transcription methods (4.48), (4.50), (4.56), and (4.58) are of the form

$$\boldsymbol{\zeta} \sim \mathbf{y} - h_k \mathbf{f}. \tag{4.175}$$

Thus, the unscaled Jacobian matrix has the form

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_f \\ \mathbf{G}_g \end{bmatrix} \sim \begin{bmatrix} \mathbf{I} - h_k \dfrac{\partial \mathbf{f}}{\partial \mathbf{y}} & -h_k \dfrac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \dfrac{\partial \mathbf{g}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{u}} \end{bmatrix}. \tag{4.176}$$

But notice that as $h_k \to 0$, the rows of the Jacobian corresponding to the differential equations $\mathbf{G}_f \sim \mathbf{I}$. This implies that the conditioning of the Jacobian improves as the stepsize is reduced. Furthermore, if we want to have the scaled Jacobian $\widetilde{\mathbf{G}}_f \sim \mathbf{I}$, we can set

$$\mathbf{W}_f = \mathbf{V}_y \tag{4.177}$$

in (4.174). In other words, if the state variable y_k is to be scaled by the weight v_k, then the constraint ζ_k should also be scaled by the same weight. To achieve similar results for the path constraints, the matrix $\mathbf{W}_g$ should be chosen so that the rows of $\mathbf{G}_g$ have norm one. In other words, it may be desirable to replace the original path constraints with a scaled expression of the form

$$\mathbf{0} = \mathbf{W}_g \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{4.178}$$

It is worth noting that the matrix $\mathbf{G}$ (4.176) is closely related to the *iteration matrix* used in the corrector iterations of many numerical integrators [48].

It now remains to choose the variable scaling to deal with the problem of units described above. Ideally, we would like to choose the variable scales $\mathbf{V}$ and shifts $\mathbf{r}$ such that the scaled quantities $\widetilde{\mathbf{y}} \sim \mathcal{O}(1)$ and lie in the interval $[-0.5, +0.5]$. When simple bounds such as (4.34) are available, it is clear that

$$v_k = \frac{1}{y_{U,k} - y_{L,k}}, \tag{4.179}$$

$$r_k = \frac{1}{2} - \frac{y_{U,k}}{y_{U,k} - y_{L,k}}. \tag{4.180}$$

When simple bounds are not part of the problem description, it is necessary to construct equivalent information about the variable ranges.

In Section 1.16, it was also suggested that the objective function scale weights (1.150) should be chosen so that the condition number of the KKT matrix (1.66) is close to one. A crude estimate for the condition number can be constructed from the *Gerschgorin bound* (2.40). Thus one can compute

$$\varpi = \max\{|\sigma_L|, |\sigma_U|\}, \tag{4.181}$$

where σ_L and σ_U are the Gerschgorin estimates for the smallest and largest eigenvalues of the Hessian $\mathbf{H}_L$. Then one can choose the objective scale in (1.150) to be

$$w_0 = \frac{1}{\varpi}. \tag{4.182}$$

This objective scaling is computed only at NLP solutions, i.e., at the end of each mesh refinement. To prevent excessive changes from one refinement iteration to the next, we use the geometric mean of the estimates from the current and previous iterates.

Ultimately the scale weights must be chosen based on a set of heuristics, and in the SOCS software, the automatic scaling procedure constructs the scale weights at the initial point based on initial gradient estimates obtained when detecting problem sparsity and then recomputes the scaling at the solution of each mesh-refinement problem. The technique implements the following scaling heuristics:

Rule 1: Scale from a *control perspective*; e.g., state variable $y_4(t)$ has the same scale for all grid points.

Rule 2: *Variable Scaling*

1. Estimate the largest/smallest variable value from a
   (a) user input upper/lower bound or if not available a
   (b) user-specified initial guess.
2. Normalize and shift the variables (4.179)–(4.180). When (a) and (b) provide no information default scaling to 1.

Rule 3: *ODE Defect Scaling*

1. Set ODE defect scaling to state variable scaling (4.177) or optionally
2. choose it to normalize the defect gradients.

Rule 4: *Algebraic and Boundary Constraint Scaling*

1. Estimate the largest/smallest constraint value from user input upper/lower bound.
2. Optionally, estimate and/or compute the Jacobian (4.176).
3. Normalize the constraint.
4. When bounds provide no information set scaling to 1.

Rule 5: *Objective Scaling*

1. Set the objective scaling to (4.182) or optionally to a
2. user-specified value.

4.9 Quadrature Equations

The general formulation of an optimal control problem may involve *quadrature functions* as introduced in (4.37). The quadrature functions may appear either in the objective function as in (4.40) or as integral constraints of the form

$$\psi_L \leq \int_{t_I}^{t_F} w\left[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t\right] dt \leq \psi_U. \tag{4.183}$$

To simplify the discussion, let us focus on an optimal control problem stated in Lagrange form, although all results are directly applicable when the integrated quantities are constraints. Recall that the Lagrange problem

$$\min_u J = \int_{t_I}^{t_F} w(\mathbf{y}, \mathbf{u}, t) dt, \tag{4.184}$$

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{u}, t), \tag{4.185}$$

$$\mathbf{y}^{\mathsf{T}}(t_I) = \boldsymbol{\psi}_0^{\mathsf{T}} \tag{4.186}$$

can be restated as a Mayer problem of the form

$$\min_u J = y_{n+1}(t_F), \tag{4.187}$$

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{u}, t), \tag{4.188}$$

$$\dot{y}_{n+1} = w(\mathbf{y}, \mathbf{u}, t), \tag{4.189}$$

$$\mathbf{y}^{\mathsf{T}}(t_I) = \boldsymbol{\psi}_0^{\mathsf{T}}, \tag{4.190}$$

$$y_{n+1}(t_I) = 0. \tag{4.191}$$

Clearly, the Mayer and Lagrange forms are mathematically equivalent, but are they computationally comparable? The answer is no!

First, the *number* of state variables for the Mayer formulation is greater than the number for the Lagrange formulation. Thus, when the state variable is discretized, the number of NLP variables will be increased by Mn_w, where n_w is the number of quadrature functions and M is the number of grid points. Since the size of the NLP is larger, one can expect some degradation in the computation time relative to the Lagrange form.

The second (less obvious) reason concerns *robustness*. When the Lagrange formulation is discretized, the original ODEs are approximated by defect constraints, $\boldsymbol{\zeta}_k = \mathbf{0}$, such as (4.56) and (4.58). In contrast, when the Mayer formulation is discretized, defect constraints are introduced for *both* the original ODEs (4.188) *and* the quadrature differential equation (4.189). Thus, it may be more difficult for the NLP algorithm to solve both the original ODE defects and the quadrature defect constraints. This effect is most noticeable when the quadrature functions are nonlinear and/or the grid is coarse. In essence, the discretized version of the Lagrange problem is more "forgiving" than the discretized version of the Mayer problem.

From a numerical standpoint, the key notion is to *implicitly* introduce the additional state variable y_{n+1} without actually having it appear explicitly in the relevant expressions. To see how this is achieved, let us construct the quadrature approximation when a trapezoidal discretization is employed:

$$\int_{t_I}^{t_F} w(\mathbf{y}, \mathbf{u}, t)dt = y_{n+1}(t_M) \tag{4.192}$$

$$= \sum_{k=1}^{M-1} \frac{\Delta\tau_k \Delta t}{2}[w_{k+1} + w_k]. \tag{4.193}$$

Expression (4.193) is obtained by recursive application of the trapezoidal discretization, e.g., (4.68), with the specified initial condition $y_{n+1}(t_I) = 0$. The final expression can be rewritten as

$$\int_{t_I}^{t_F} w(\mathbf{y}, \mathbf{u}, t)dt = \mathbf{b}^{\mathsf{T}}\mathbf{q}, \tag{4.194}$$

where the coefficient vector is

$$\mathbf{b}^{\mathsf{T}} = \frac{1}{2}\left[\Delta\tau_1, (\Delta\tau_2 + \Delta\tau_1), (\Delta\tau_3 + \Delta\tau_2), \ldots, (\Delta\tau_{M-1} + \Delta\tau_{M-2}), \Delta\tau_{M-1}\right] \tag{4.195}$$

and the vector $\mathbf{q}$ has elements

$$q_k = \Delta t\, w_k \tag{4.196}$$

for $k = 1, \ldots, M$. Observe that the elements of $\mathbf{q}$ are functions of the original state and control. Furthermore, the objective function can be computed from the original state and control—y_{n+1} does not appear explicitly in any expression. Thus, we have constructed the objective function in the form (4.114), where the last row of the matrix $\mathbf{B}$ is the vector $\mathbf{b}^{\mathsf{T}}$. In summary, all of the techniques described in Section 4.6 can be used to construct the NLP Jacobian and Hessian matrices while exploiting sparsity in the quadrature functions w. It should also be clear that the same approach can be used when the quadrature expressions are computed using a Simpson discretization.

The treatment of quadrature equations must also be addressed in the mesh-refinement process. Again, the goal is to *implicitly* form the relevant information without explicitly introducing an additional state variable. If a Mayer formulation was used and an additional state was introduced, there would be a contribution to the discretization error from (4.155) of the form

$$\varepsilon(t) = \dot{\tilde{y}}_{n+1}(t) - w[\tilde{\mathbf{y}}(t), \tilde{\mathbf{u}}(t), t], \tag{4.197}$$

which must be evaluated in order to compute the integral (4.154). To eliminate the explicit contribution of the additional state, we must construct $\dot{\tilde{y}}_{n+1}(t)$. This quantity can be easily computed by simply interpolating the local information. When a trapezoidal quadrature is used, a linear interpolant can be constructed through the values at the nearest grid points, $y_{n+1}(t_k)$ and $y_{n+1}(t_{k+1})$. Similarly, a cubic (Hermite) interpolant can be constructed by also using the values of $w_{n+1}(t_k)$ and $w_{n+1}(t_{k+1})$. In either case, this local interpolating function provides the necessary information to evaluate the discretization error.

The treatment of quadrature equations that has been described

1. does not introduce an additional state variable $y_{n+1}(t)$ and
2. does not introduce additional defect constraints
3. but does adjust the mesh as though the additional state were introduced.

In essence the technique ensures that the discretization mesh is constructed such that all continuous functions (4.38) are accurately represented. In fact we use the same approach to guarantee accuracy in the algebraic equations (4.33).

Example 4.4 HYPERSENSITIVE CONTROL. Rao and Mease [148] present an example that illustrates the importance of this approach. Rao and Mease refer to this as a "hypersensitive" problem. It is extremely difficult to solve using an indirect method, and is equally difficult when treated in the Mayer form. The problem is defined by a single differential equation and is stated in Lagrange form:

$$\min_u J = \int_0^{t_F} \left[y^2 + u^2 \right] dt, \tag{4.198}$$

$$\dot{y} = -y^3 + u, \tag{4.199}$$

where $y(0) = 1$, $y(t_F) = 1.5$, and $t_F = 10000$. The state variable history and the corresponding distribution of stepsize are illustrated in Figure 4.10. The initial constant stepsize is shown with a dotted line. Because of the dramatic change in the state variable near the initial and final times, it is necessary to have a very small stepsize in these regions. The $\mathbb{SOCS}$ iteration history, beginning with 25 equally spaced grid points, is summarized in Table 4.3. Notice that the discretization error is reduced by 12 orders of magnitude by the

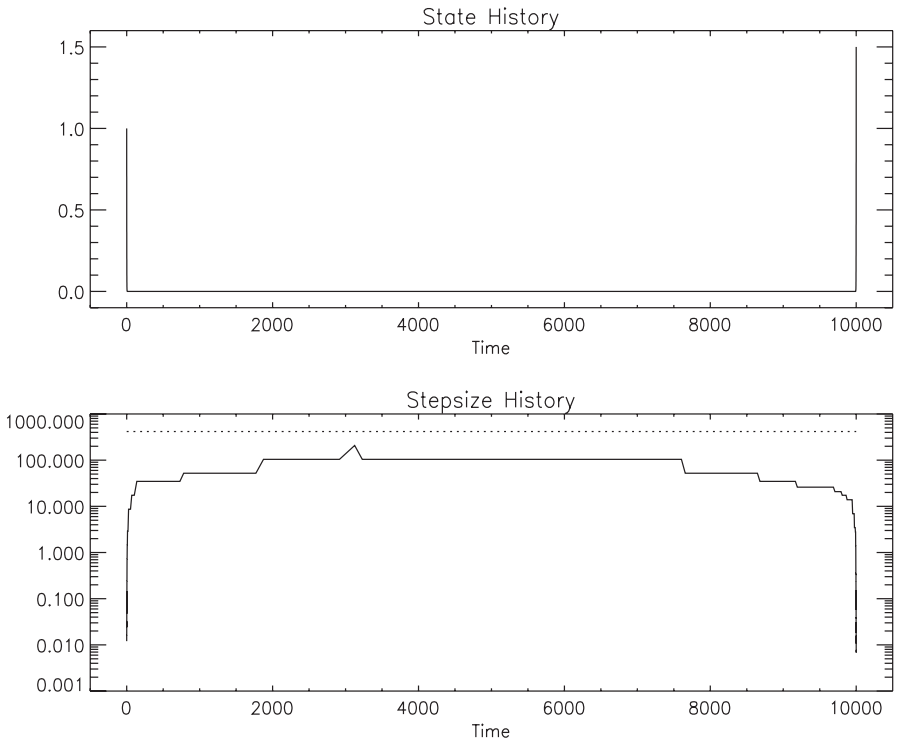


Figure 4.10. *Hypersensitive problem.*

mesh-refinement procedure. When the problem is solved in Mayer form, the solution to the very first NLP (with 25 grid points) requires over 1322 iterations and 7860 function evaluations. This is 131 times more than the 60 function evaluations reported in line 1 of Table 4.3.

Table 4.3. *Hypersensitive problem performance summary.*

GRID	NPT	NGC	NHC	NFE	NRHS	ERRODE	CPU (sec)
1	25	14	2	60	1500	0.15×10^{5}	0.62×10^{0}
2	25	21	19	124	6076	0.37×10^{3}	0.15×10^{1}
3	49	11	9	63	6111	0.28×10^{2}	0.18×10^{1}
4	97	11	8	61	11773	0.49×10^{1}	0.29×10^{1}
5	109	29	27	585	126945	0.11×10^{1}	0.58×10^{1}
6	115	10	8	185	42365	0.13×10^{0}	0.25×10^{1}
7	121	8	6	144	34704	0.12×10^{-1}	0.22×10^{1}
8	128	8	1	93	23715	0.96×10^{-3}	0.18×10^{1}
9	137	5	1	60	16380	0.19×10^{-4}	0.15×10^{1}
10	181	4	1	49	17689	0.55×10^{-6}	0.16×10^{1}
11	229	4	1	49	22393	0.31×10^{-7}	0.19×10^{1}
	229	125	83	1473	309651		24.23

4.10 Algebraic Variable Rate Constraints

For some problems it may be necessary to impose a constraint of the form

$$r_L \leq \frac{du}{dt} \leq r_U \tag{4.200}$$

over the entire phase, that is, for $t_I \leq t \leq t_F$ and $r_L < r_U$. One technique for treating such a constraint is to view the "real" control $u(t)$ as a state variable. Then a new state variable $y_{n+1}(t)$ and a new control $u_{m+1}(t)$ can be introduced. If an additional differential equation

$$\dot{y}_{n+1}(t) = u_{m+1}(t) \tag{4.201}$$

is also included, then $u_{m+1}(t)$ represents the algebraic variable rate, and (4.200) can be enforced by the simple bounds

$$r_L \leq u_{m+1}(t) \leq r_U. \tag{4.202}$$

Since y_{n+1} is just the integral of the rate u_{m+1}, the original control u can be replaced by the new state throughout the problem description. Now, just as with the treatment of quadrature equations described in Section 4.9 the transformed problem seems equivalent to the original. But are they computationally comparable? Again, the answer is no! In fact this transformation shares the same shortcomings discussed for quadrature equations. In particular the number of state variables is increased leading to a larger NLP problem after discretization. Second, by introducing a new state and control, the index of the DAE system of the transformed problem can be increased. Furthermore the new control appears linearly and thus introduces the prospect of singular arcs. Example 6.10 illustrates this behavior. Thus the transformed problem is often more difficult to solve. In order to overcome these shortcomings it is worthwhile exploiting the special form of these constraints.

When using a trapezoidal discretization the control is a linear function of time in each interval and the rate is given by

$$r_L \leq \frac{u(t_{k+1}) - u(t_k)}{h_k} \leq r_U, \tag{4.203}$$

where

$$h_k = (\tau_{k+1} - \tau_k)(t_F - t_I) = (\tau_{k+1} - \tau_k)\Delta t = \Delta\tau_k \Delta t, \tag{4.204}$$

with $\Delta t \equiv (t_F - t_I)$ and $\Delta\tau_k \equiv (\tau_{k+1} - \tau_k)$ with constants $0 \leq \tau_k \leq 1$. Now the expression

$$r_L \leq \frac{u_{k+1} - u_k}{\Delta\tau_k(t_F - t_I)} \leq r_U \tag{4.205}$$

can be written as

$$[\Delta\tau_k(t_F - t_I)]r_L \leq u_{k+1} - u_k \leq r_U[\Delta\tau_k(t_F - t_I)], \tag{4.206}$$

which for M grid points, i.e., for $k = 1, \ldots, (M-1)$, become the linear constraints

$$0 \le u_{k+1} - u_k - [\Delta\tau_k(t_F - t_I)]r_L, \tag{4.207}$$

$$0 \ge u_{k+1} - u_k - [\Delta\tau_k(t_F - t_I)]r_U \tag{4.208}$$

in the NLP variables u_k, t_F, t_I.

When a Hermite–Simpson discretization is used, the control is a quadratic function of time within each interval. Since the derivative of the interpolating quadratic is linear, the extreme values occur at the ends of each interval. Thus from Table 1.8 we have

$$r_L \le \frac{1}{h_k}\left[-3u_k + 4\bar{u}_{k+1} - u_{k+1}\right] \le r_U, \tag{4.209}$$

$$r_L \le \frac{1}{h_k}\left[u_k - 4\bar{u}_{k+1} + 3u_{k+1}\right] \le r_U, \tag{4.210}$$

where $\bar{u}_{k+1} = u(t_k + h_k/2)$. Rewriting leads to the four constraints

$$0 \le -3u_k + 4\bar{u}_{k+1} - u_{k+1} - h_k r_L, \tag{4.211}$$

$$0 \ge -3u_k + 4\bar{u}_{k+1} - u_{k+1} - h_k r_U, \tag{4.212}$$

$$0 \le u_k - 4\bar{u}_{k+1} + 3u_{k+1} - h_k r_L, \tag{4.213}$$

$$0 \ge u_k - 4\bar{u}_{k+1} + 3u_{k+1} - h_k r_U, \tag{4.214}$$

where $h_k = \Delta\tau_k(t_F - t_I)$.

For the special case $r_L = r_U$, the two inequality constraints (4.207)–(4.208) are replaced by the single equality constraint

$$0 = u_{k+1} - u_k - [\Delta\tau_k(t_F - t_I)]r_L. \tag{4.215}$$

Similarly for Hermite–Simpson the four inequality constraints (4.211)–(4.214) are replaced by the two equality constraints

$$0 = u_{k+1} - u_k - [\Delta\tau_k(t_F - t_I)]r_L, \tag{4.216}$$

$$0 = \bar{u}_{k+1} - u_k - \frac{1}{2}[\Delta\tau_k(t_F - t_I)]r_L. \tag{4.217}$$

Regardless of which discretization is used, all of the rate constraint equations are linear with respect to the optimization variables and consequently appear as an additional row of the form $\mathbf{a}^{\mathsf{T}}\mathbf{x}$ in (4.114). Furthermore, since the constraints are linear, first derivative information can be computed without any additional finite difference index sets, and there is no contribution to the Hessian matrix. Finally we should note one other important difference when the problem is transformed and the rate $u_{m+1}(t)$ is treated as a control variable. By construction the rate is continuous across all grid points, that is, $u_{m+1}(t_k^-) = u_{m+1}(t_k^+)$. In contrast, when the rates are constrained by either (4.207)–(4.208) or (4.211)–(4.214) the control rate $\dot{u}$ is not necessarily continuous across a grid point. In effect, the transformation (4.201) imposes an additional degree of continuity on the original control $u(t)$.

4.11 Estimating Adjoint Variables

A direct transcription or collocation method can be used to solve very general optimal control problems, and because a direct NLP algorithm is used there is no need to form adjoint

equations when computing the solution. Nevertheless, in Section 4.2 it was demonstrated that the Lagrange multipliers used by the NLP algorithm are related to the adjoint variables. First recall the NLP problem formulation presented in Section 1.8. Suppose that we must choose the n variables $\mathbf{x}$ to minimize

$$F(\mathbf{x}) \tag{4.218}$$

subject to the $m \leq n$ constraints

$$\mathbf{c}(\mathbf{x}) = \mathbf{0}. \tag{4.219}$$

In this setting it is common to define the Lagrangian

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) = F(\mathbf{x}) - \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{c}(\mathbf{x}) = F(\mathbf{x}) - \sum_{i=1}^{m} \alpha_i c_i(\mathbf{x}), \tag{4.220}$$

where $\boldsymbol{\alpha}$ is an m-vector of Lagrange multipliers.

In contrast for an infinite-dimensional problem suppose we must choose the control functions $\mathbf{u}(t)$ to minimize

$$J = \int_{t_I}^{t_F} L[\mathbf{y}(t), \mathbf{u}(t)]dt \tag{4.221}$$

subject to the DAE constraints

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{u}, t], \tag{4.222}$$

$$\mathbf{0} = \mathbf{g}[\mathbf{y}, \mathbf{u}, t]. \tag{4.223}$$

We form an augmented performance index in a manner analogous to the definition of the Lagrangian function (4.220),

$$\hat{J} = \int_{t_I}^{t_F} L[\mathbf{y}(t), \mathbf{u}(t)]dt - \int_{t_I}^{t_F} \boldsymbol{\lambda}^{\mathsf{T}}(t)\{\dot{\mathbf{y}} - \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)]\}\, dt + \int_{t_I}^{t_F} \boldsymbol{\mu}^{\mathsf{T}}(t)\mathbf{g}[\mathbf{y}(t), \mathbf{u}(t)]dt. \tag{4.224}$$

Observe that the differential-algebraic constraints (4.222)–(4.223) are adjoined to the objective function using the adjoint variables $\lambda(t)$ and $\mu(t)$. Necessary conditions for the NLP problem (4.218)–(4.219) are usually derived by setting the first derivative of the Lagrangian (4.220) to zero. By analogy necessary conditions for the optimal control problem (4.221)–(4.223) are derived using the calculus of variations by setting the first variation of the augmented performance index (4.224) to zero.

At this point it is worth calling attention to the different notational conventions adopted for the NLP problem (4.218)–(4.219) and the optimal control problem (4.221)–(4.223). In particular, in this section we use $\boldsymbol{\alpha}$ to denote the Lagrange multipliers in order to avoid confusion with the symbol $\boldsymbol{\lambda}$ for the adjoint variables. In (4.220) the NLP Lagrangian is denoted by the symbol $\mathcal{L}$ to avoid confusion with the integrand L in (4.221). Furthermore the definition of the Lagrangian (4.220) has a negative sign for the term $-\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{c}$. Bryson and Ho [54] write the second term of (4.224) with a positive sign by augmenting the term $\boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{f} - \dot{\mathbf{y}})$.

The relationship between the NLP Lagrange multipliers $\boldsymbol{\alpha}$ and the adjoint variables $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ has been a subject of considerable theoretical interest. If the differential equations

are approximated using a simple Euler discretization, in Section 4.2 it was demonstrated that $\boldsymbol{\lambda} \to \boldsymbol{\alpha}$ as the mesh size $h \to 0$. When a Hermite–Simpson discretization is used both von Stryk [167] and Enright and Conway [77] show that $\boldsymbol{\lambda} \approx \frac{3}{2}\boldsymbol{\alpha}$. Hager [105] investigates the rate of convergence for various approximation schemes. A feature shared by many of these analyses is that they provide estimates in the limit, that is, as $h \to 0$. We have a slightly different goal here, namely, to provide approximate values for the adjoint variables constructed from the NLP Lagrange multipliers when the discretization mesh sizes are finite and possibly unequal. Engelsone and Campbell [75] develop a number of important results concerning the accuracy of these estimates.

4.11.1 Quadrature Approximation

Let us begin by considering methods for computing numerical approximations to an integral of the form

$$\int_{t_I}^{t_F} p(t)g(t)dt. \tag{4.225}$$

Let us discretize the problem by introducing M grid points that subdivide the domain into $(M-1)$ intervals according to

$$t_I = t_1 < t_2 < \cdots < t_{M-1} < t_M = t_F \tag{4.226}$$

and denote the steplengths by

$$h_k = t_{k+1} - t_k. \tag{4.227}$$

If we denote the values $p_k = p(t_k)$ and $g_k = g(t_k)$, the trapezoidal approximation to the integral is given by

$$\int_{t_I}^{t_F} p(t)g(t)dt = \sum_{k=1}^{M-1} \frac{h_k}{2}\left[p_k g_k + p_{k+1}g_{k+1}\right] \tag{4.228}$$

$$= \left[\left(\frac{h_1}{2}\right)p_1\right]g_1 + \sum_{k=2}^{M-1}\left[\left(\frac{h_{k-1}+h_k}{2}\right)p_k\right]g_k + \left[\left(\frac{h_{M-1}}{2}\right)p_M\right]g_M. \tag{4.229}$$

If we denote the values at interval midpoints by $p_{k+\frac{1}{2}} = p(t_k + h_k/2)$ and $g_{k+\frac{1}{2}} = g(t_k + h_k/2)$, then Simpson's quadrature rule can be used to approximate the integral giving

$$\int_{t_I}^{t_F} p(t)g(t)dt = \sum_{k=1}^{M-1} \frac{h_k}{6}\left[p_k g_k + 4p_{k+\frac{1}{2}}g_{k+\frac{1}{2}} + p_{k+1}g_{k+1}\right] \tag{4.230}$$

$$= \left[\left(\frac{h_1}{6}\right)p_1\right]g_1 + \sum_{k=1}^{M-1}\left[\left(\frac{2h_k}{3}\right)p_{k+\frac{1}{2}}\right]g_{k+\frac{1}{2}} + \sum_{k=2}^{M-1}\left[\left(\frac{h_{k-1}+h_k}{6}\right)p_k\right]g_k + \left[\left(\frac{h_{M-1}}{6}\right)p_M\right]g_M. \tag{4.231}$$

4.11.2 Path Constraint Adjoints

Let us focus on the third term in (4.224). When a trapezoidal discretization is used the path constraints (4.223) are enforced by imposing the NLP constraints

$$0 = g[\mathbf{y}(t_k), \mathbf{u}(t_k), t_k] = g_k \tag{4.232}$$

for $k = 1, \ldots, M$. If a trapezoidal discretization is used, then from (4.229) and the definition of the Lagrangian (4.220) we can write

$$\begin{aligned}\int_{t_I}^{t_F} \mu(t) g(t) dt &= \left[\left(\frac{h_1}{2}\right)\mu_1\right] g_1 + \sum_{k=2}^{M-1}\left[\left(\frac{h_{k-1}+h_k}{2}\right)\mu_k\right] g_k \\ &\quad + \left[\left(\frac{h_{M-1}}{2}\right)\mu_M\right] g_M \qquad (4.233)\\ &= -\alpha_1 g_1 - \sum_{k=2}^{M-1} \alpha_k g_k - \alpha_M g_M. \qquad (4.234)\end{aligned}$$

Comparing terms in (4.233) and (4.234) we find that

$$\mu_1 = -\left(\frac{2}{h_1}\right)\alpha_1, \tag{4.235}$$

$$\mu_k = -\left(\frac{2}{h_{k-1}+h_k}\right)\alpha_k, \qquad k = 2, \ldots, M-1, \tag{4.236}$$

$$\mu_M = -\left(\frac{2}{h_{M-1}}\right)\alpha_M. \tag{4.237}$$

When a Hermite–Simpson discretization is used the path constraints (4.223) are enforced by imposing the NLP constraints at the grid points (4.232) and also at the midpoints

$$0 = g[\mathbf{y}(t_{k+\frac{1}{2}}), \mathbf{u}(t_{k+\frac{1}{2}}), t_{k+\frac{1}{2}}] = g_{k+\frac{1}{2}} \tag{4.238}$$

for $k = 1, \ldots, M-1$. If a Hermite–Simpson discretization is used, then from (4.231) and the definition of the Lagrangian (4.220) we can write

$$\begin{aligned}\int_{t_I}^{t_F} \mu(t) g(t) dt &= \left[\left(\frac{h_1}{6}\right)\mu_1\right] g_1 + \sum_{k=1}^{M-1}\left[\left(\frac{2h_k}{3}\right)\mu_{k+\frac{1}{2}}\right] g_{k+\frac{1}{2}} \\ &\quad + \sum_{k=2}^{M-1}\left[\left(\frac{h_{k-1}+h_k}{6}\right)\mu_k\right] g_k + \left[\left(\frac{h_{M-1}}{6}\right)\mu_M\right] g_M \\ &= -\alpha_1 g_1 - \sum_{k=1}^{M-1} \alpha_{k+\frac{1}{2}} g_{k+\frac{1}{2}} - \sum_{k=2}^{M-1} \alpha_k g_k - \alpha_M g_M. \qquad (4.239)\end{aligned}$$

Comparing terms we find that

$$\mu_1 = -\left(\frac{6}{h_1}\right)\alpha_1, \tag{4.240}$$

$$\mu_{k+\frac{1}{2}} = -\left(\frac{3}{2h_k}\right)\alpha_{k+\frac{1}{2}}, \qquad k = 1,\ldots,M-1, \tag{4.241}$$

$$\mu_k = -\left(\frac{6}{h_{k-1}+h_k}\right)\alpha_k, \qquad k = 2,\ldots,M-1, \tag{4.242}$$

$$\mu_M = -\left(\frac{6}{h_{M-1}}\right)\alpha_M. \tag{4.243}$$

4.11.3 Differential Constraint Adjoints

The technique used to construct adjoint approximations for the path constraints can also be applied to compute the adjoint variables associated with the differential constraints (4.222). If we replace the path constraint function $g(t)$ with the differential constraint $\dot{y} - f$ in (4.230), we obtain

$$\int_{t_I}^{t_F} \lambda(t)[\dot{y} - f]dt = \sum_{k=1}^{M-1} \frac{h_k}{6}\left\{\lambda_k[\dot{y}_k - f_k] + 4\lambda_{k+\frac{1}{2}}[\dot{y}_{k+\frac{1}{2}} - f_{k+\frac{1}{2}}] + \lambda_{k+1}[\dot{y}_{k+1} - f_{k+1}]\right\}. \tag{4.244}$$

However, since a Hermite–Simpson method is based on Hermite interpolation, by construction $\lambda_k[\dot{y}_k - f_k] = 0$ at all grid points and this expression simplifies considerably:

$$\int_{t_I}^{t_F} \lambda(t)[\dot{y} - f]dt = \sum_{k=1}^{M-1} \frac{2h_k}{3}\lambda_{k+\frac{1}{2}}[\dot{y}_{k+\frac{1}{2}} - f_{k+\frac{1}{2}}]. \tag{4.245}$$

The Hermite interpolant for the state variable can be written as

$$y(t) = (1 - 3\delta^2 + 2\delta^3)y_k + (3\delta^2 - 2\delta^3)y_{k+1} + (h_k\delta - 2h_k\delta^2 + h_k\delta^3)f_k + (-h_k\delta^2 + h_k\delta^3)f_{k+1}, \tag{4.246}$$

where $\delta = (t - t_k)/h_k$ defines the location of the evaluation time relative to the beginning of the interval. If this expression is first differentiated to obtain an expression for $\dot{y}$ and then evaluated at the midpoint $t_{k+\frac{1}{2}}$, we obtain

$$\dot{y}_{k+\frac{1}{2}} = \frac{3}{2h_k}y_{k+1} - \frac{3}{2h_k}y_k - \frac{1}{4}f_{k+1} - \frac{1}{4}f_k. \tag{4.247}$$

This expression can be substituted into (4.245) to give

$$\begin{aligned}\int_{t_I}^{t_F} \lambda(t)[\dot{y}-f]dt &= \sum_{k=1}^{M-1} \frac{2h_k}{3}\lambda_{k+\frac{1}{2}}\left[\dot{y}_{k+\frac{1}{2}} - f_{k+\frac{1}{2}}\right] \\ &= \sum_{k=1}^{M-1} \frac{2h_k}{3}\lambda_{k+\frac{1}{2}}\left[\left(\frac{3}{2h_k}y_{k+1} - \frac{3}{2h_k}y_k - \frac{1}{4}f_{k+1} - \frac{1}{4}f_k\right) - f_{k+\frac{1}{2}}\right] \\ &= \sum_{k=1}^{M-1} \lambda_{k+\frac{1}{2}}\left[y_{k+1} - y_k - \frac{h_k}{6}(f_k + 4f_{k+\frac{1}{2}} + f_{k+1})\right]. \end{aligned} \tag{4.248}$$

Since the Hermite–Simpson method approximates the differential equations by satisfying the defect constraints

$$y_{k+1} - y_k - \frac{h_k}{6}(f_k + 4f_{k+\frac{1}{2}} + f_{k+1}) = 0 \tag{4.249}$$

it follows immediately that the NLP multiplier and adjoint variable satisfy

$$\lambda_{k+\frac{1}{2}} = \alpha_{k+\frac{1}{2}}. \tag{4.250}$$

4.11.4 Numerical Comparisons

Let us now illustrate the accuracy of the adjoint variable estimates on a series of example problems. In order to verify the accuracy of the adjoint variables we need a "truth" model. Consequently each example problem will be solved using two different techniques. First the problem will be solved using the direct transcription method. This direct solution provides estimates for the adjoint variables. For comparison each problem is also solved using an *indirect transcription* method. The indirect approach of course requires explicit solution of both the state and adjoint equations in conjunction with the appropriate boundary conditions. Mesh refinement was used for both solution techniques, and the requested accuracy was 10^{-7} or approximately eight significant figures.

It is worth remarking that these examples demonstrate two philosophically distinct approaches for solving optimal control problems. In the direct transcription approach, we first discretize the differential equations and then solve a finite-dimensional NLP. We refer to this as *discretize then optimize*. In contrast, if the optimality conditions (e.g., adjoint equations) are derived first and then discretized, we refer to this as *optimize then discretize*. A more complete discussion of this topic is presented in Section 4.12.

Example 4.5 LINEAR TANGENT STEERING. The first problem has an analytic solution and was introduced in Example 4.1. The goal is to minimize the objective

$$F = \int_0^{t_F} dt = t_F \tag{4.251}$$

subject to the constraints

$$\dot{y}_1 = y_3, \tag{4.252}$$
$$\dot{y}_2 = y_4, \tag{4.253}$$
$$\dot{y}_3 = a\cos u, \tag{4.254}$$
$$\dot{y}_4 = a\sin u, \tag{4.255}$$

where $a = 100$. The boundary conditions are given by

$$y_1(0) = 0, \tag{4.256}$$
$$y_2(0) = 0, \qquad y_2(t_F) = 5, \tag{4.257}$$
$$y_3(0) = 0, \qquad y_3(t_F) = 45, \tag{4.258}$$
$$y_4(0) = 0, \qquad y_4(t_F) = 0. \tag{4.259}$$

The Hamiltonian is

$$H = 1 + \lambda_1 y_3 + \lambda_2 y_4 + a\lambda_3 \cos u + a\lambda_4 \sin u \tag{4.260}$$

with adjoint equations

$$\dot{\lambda}_1 = 0, \tag{4.261}$$
$$\dot{\lambda}_2 = 0, \tag{4.262}$$
$$\dot{\lambda}_3 = -\lambda_1, \tag{4.263}$$
$$\dot{\lambda}_4 = -\lambda_2. \tag{4.264}$$

The maximum principle defines the optimal control

$$H_u = 0 = -\lambda_3 \sin u + \lambda_4 \cos u \tag{4.265}$$

and the control variable is eliminated by virtue of the relations

$$\cos u = \frac{-\lambda_3}{\sqrt{\lambda_3^2 + \lambda_4^2}},$$
$$\sin u = \frac{-\lambda_4}{\sqrt{\lambda_3^2 + \lambda_4^2}}.$$

The optimality conditions lead to the additional boundary conditions

$$\lambda_1(t_F) = 0, \tag{4.266}$$
$$H(t_F) = 0 = 1 + \lambda_2 y_4 + a\lambda_3 \cos u + a\lambda_4 \sin u. \tag{4.267}$$

The optimal value of the objective function obtained by the direct method is $F^* = 0.5545737562$. In Figure 4.11 the state variable history obtained using the direct method is illustrated using a gray shading. The corresponding values of the state variables at the grid points obtained from the indirect method are plotted with dots. Figure 4.12 presents a comparison of the adjoint variables λ. Here we use the indirect method as a "truth model" and illustrate the solution with a shaded region. The discrete adjoint estimates obtained using the direct method described above are plotted over the shaded region as dots. In Figure 4.13 the control computed by the direct method is shown as a shaded region and the indirect control is plotted over with dots at the grid points and midpoints. It should be noted that when $\mathbb{SOCS}$ is used to compute the solution using the indirect method the values of the adjoint variables are available at the *grid points*. In contrast the discrete adjoint estimates are computed at the interval *midpoints*. Furthermore the distribution of the grid

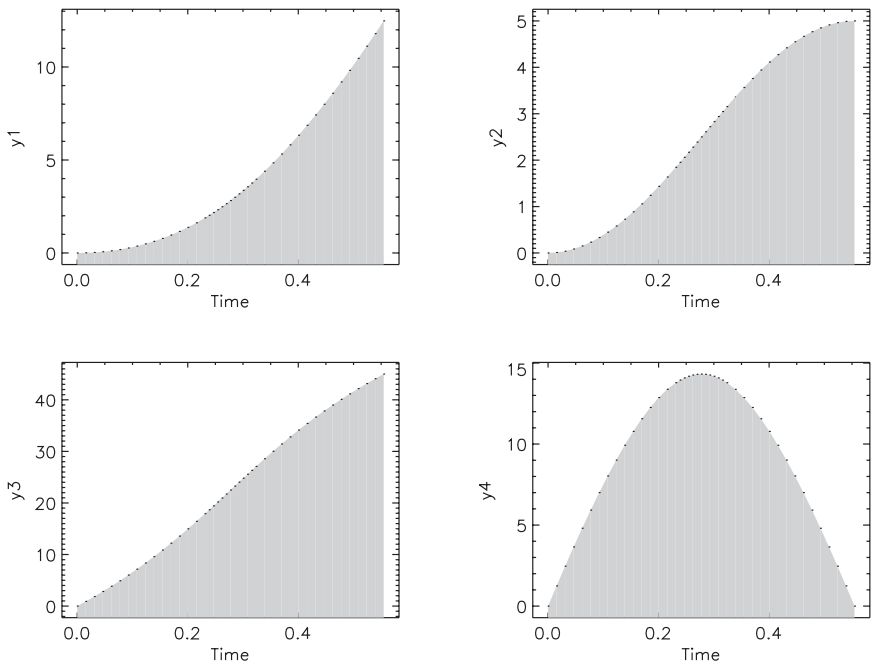


Figure 4.11. *Linear tangent steering states* (y_1, y_2, y_3, y_4).

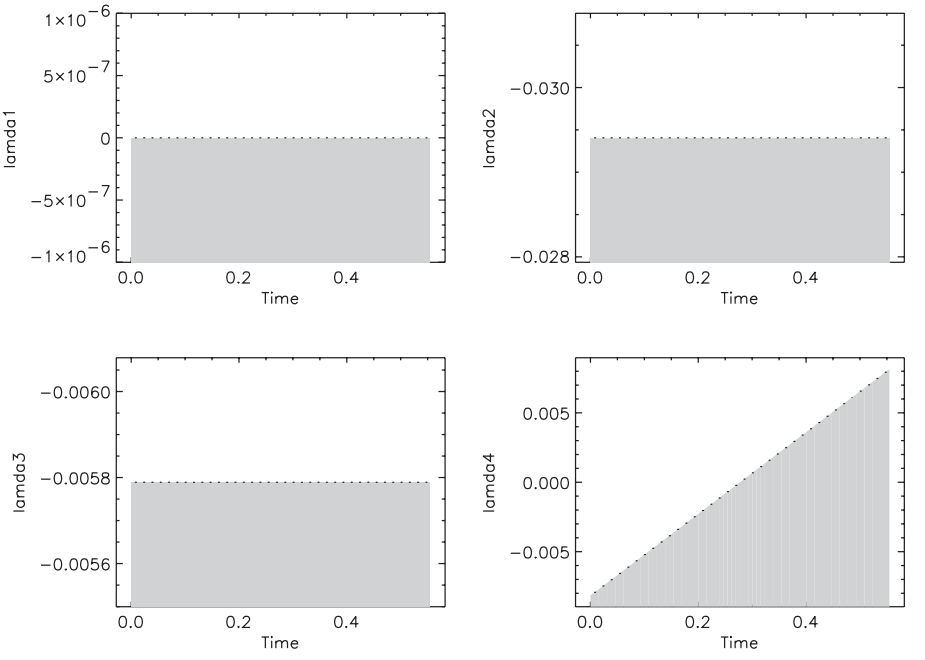


Figure 4.12. *Linear tangent steering adjoints* $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$.

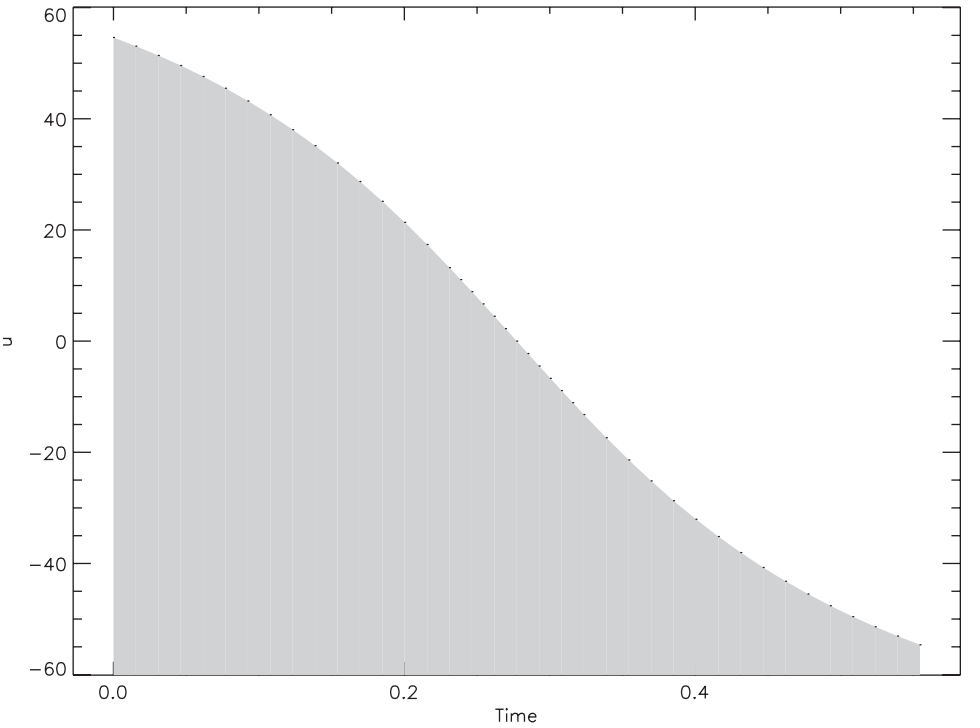


Figure 4.13. *Linear tangent steering control.*

points is different between the two solutions, the indirect method requiring $M_i = 43$ points, whereas the direct solution required $M_d = 37$ points and provided discrete adjoint estimates at 36 interval midpoints. In order to present a quantitative assessment of the accuracy we compute the error

$$\epsilon_{k+\frac{1}{2}} = \tilde{\lambda}(t_k + h_k/2) - \lambda_{k+\frac{1}{2}}. \tag{4.268}$$

In this equation the discrete adjoint estimates computed by the direct method are denoted by $\lambda_{k+\frac{1}{2}}$. For comparison, the corresponding values of the adjoint variables from the indirect solution are denoted by $\tilde{\lambda}(t_k + h_k/2)$. The latter values can be computed using Hermite cubic spline interpolation since the $\mathbb{SOCS}$ *indirect* solution is a collocation method. We also compute the maximum absolute error over all midpoints

$$\varepsilon = \max_k |\epsilon_{k+\frac{1}{2}}|. \tag{4.269}$$

Figure 4.14 illustrates the normalized error $\epsilon_{k+\frac{1}{2}}/\varepsilon$ in the discrete adjoint estimates for each adjoint variable. For this particular example the maximum error ε is quite small for all adjoint estimates.

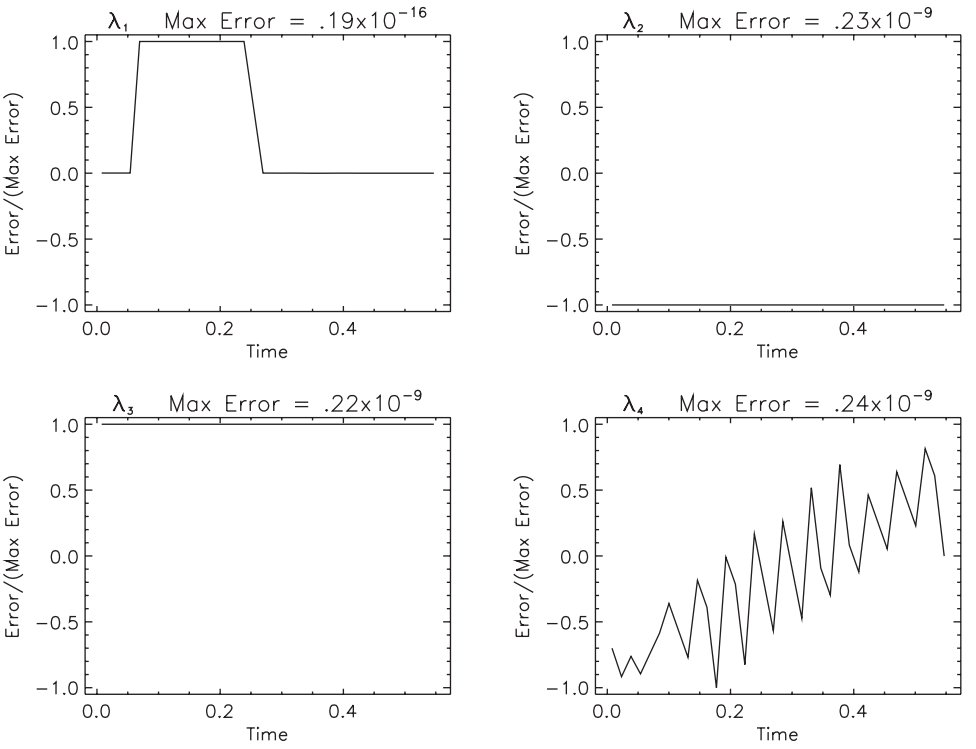


Figure 4.14. *Linear tangent steering, error in discrete adjoint estimates.*

Example 4.6 RAYLEIGH PROBLEM WITH CONTROL CONSTRAINTS. The second example, which is described by Maurer and Augustin [133], illustrates a problem with path constraints (4.223) that explicitly contain the control variables. The goal is to minimize the objective

$$F = \int_0^{t_F} (u^2 + y_1^2) dt \tag{4.270}$$

subject to the constraints

$$\dot{y}_1 = y_2, \tag{4.271}$$
$$\dot{y}_2 = -y_1 + y_2(1.4 - py_2^2) + 4u, \tag{4.272}$$
$$0 \geq u - 1, \tag{4.273}$$
$$0 \geq -u - 1 \tag{4.274}$$

with boundary conditions

$$y_1(0) = y_2(0) = -5, \qquad y_1(t_F) = y_2(t_F) = 0. \tag{4.275}$$

We fix $t_F = 4.5$ and $p = 0.14$. Observe that the constraints (4.273) and (4.274) have been written in the form $c(y,u,t) \leq 0$, which is the convention in [54]. The Hamiltonian is

$$H = u^2 + y_1^2 + \lambda_1(y_2) + \lambda_2[-y_1 + y_2(1.4 - py_2^2) + 4u] + \mu_1(u-1) + \mu_2(-u-1) \tag{4.276}$$

with adjoint equations

$$\dot{\lambda}_1 = \lambda_2 - 2y_1, \tag{4.277}$$
$$\dot{\lambda}_2 = 3p\lambda_2 y_2^2 - 1.4\lambda_2 - \lambda_1. \tag{4.278}$$

The optimal control is

$$u(t) = \begin{cases} 1, & 0 \le t \le t_1, \\ -2\lambda_2, & t_1 \le t \le t_2, \\ -1, & t_2 \le t \le t_3, \\ -2\lambda_2, & t_3 \le t \le t_F \end{cases} \tag{4.279}$$

with optimal multipliers given by

$$\mu_1(t) = -2 - 4\lambda_2(t), \qquad 0 \le t < t_1, \tag{4.280}$$
$$\mu_2(t) = -2 + 4\lambda_2(t), \qquad t_2 < t < t_3. \tag{4.281}$$

The switching structure is determined by the junction conditions

$$-2\lambda_2(t_1) = 1, \qquad -2\lambda_2(t_2) = -1, \qquad -2\lambda_2(t_3) = -1. \tag{4.282}$$

As before in Figure 4.15 we plot the direct solution states using shading and overplot the adjoint states. For the adjoints, the indirect values of λ are shaded and the discrete adjoint estimates are overplotted with dots. Figure 4.16 clearly shows the four different solution regions for the control. Because of the control constraint sign convention the optimal values for the multipliers must be nonnegative, i.e., $\mu_1(t) \ge 0$ and $\mu_2(t) \ge 0$, so we have plotted the quantity $\mu_1(t) + \mu_2(t)$ to illustrate the results. The optimal value of the objective function obtained by the direct method is $F^* = 44.72093882$. The requested accuracy of 10^{-7} was achieved using $M_d = 277$ points for the direct solution and $M_i = 292$ points (with four phases) for the indirect one. Figure 4.17 illustrates the normalized error $\epsilon_{k+\frac{1}{2}}/\varepsilon$ in the discrete adjoint estimates for each adjoint variable. For this example the discrete estimates agree to approximately three figures as suggested by the maximum error ε.

Example 4.7 RAYLEIGH PROBLEM, MIXED STATE-CONTROL CONSTRAINTS. The third example, also described by Maurer and Augustin [133], is characterized by a path constraint that has both a state and control appearing explicitly. The goal is to minimize the objective

$$F = \int_0^{t_F} (u^2 + y_1^2)dt \tag{4.283}$$

subject to the constraints

$$\dot{y}_1 = y_2, \tag{4.284}$$
$$\dot{y}_2 = -y_1 + y_2(1.4 - py_2^2) + 4u, \tag{4.285}$$
$$0 \ge u + \frac{y_1}{6} \tag{4.286}$$

with boundary conditions

$$y_1(0) = y_2(0) = -5. \tag{4.287}$$

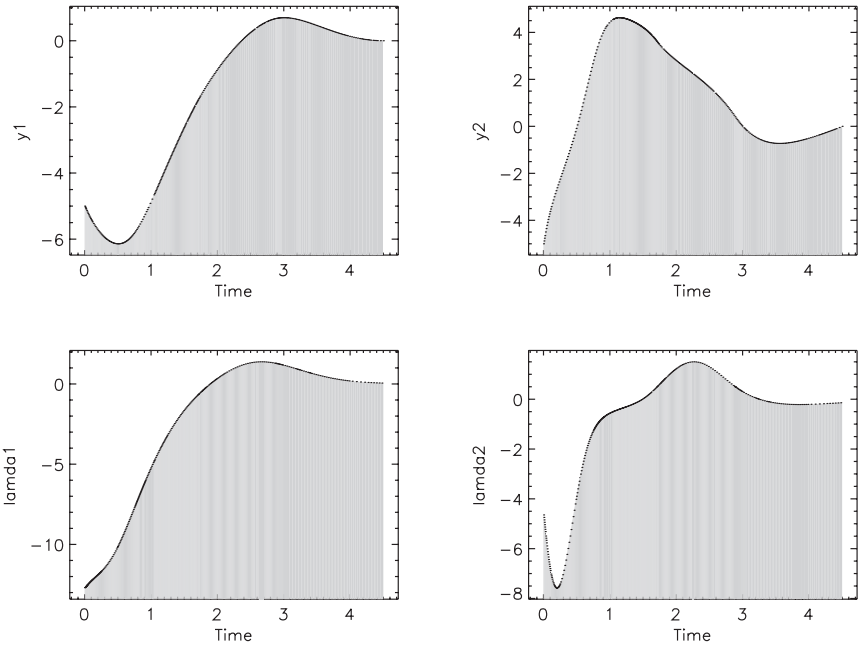


Figure 4.15. *Rayleigh problem, states* (y_1, y_2) *and adjoints* (λ_1, λ_2).

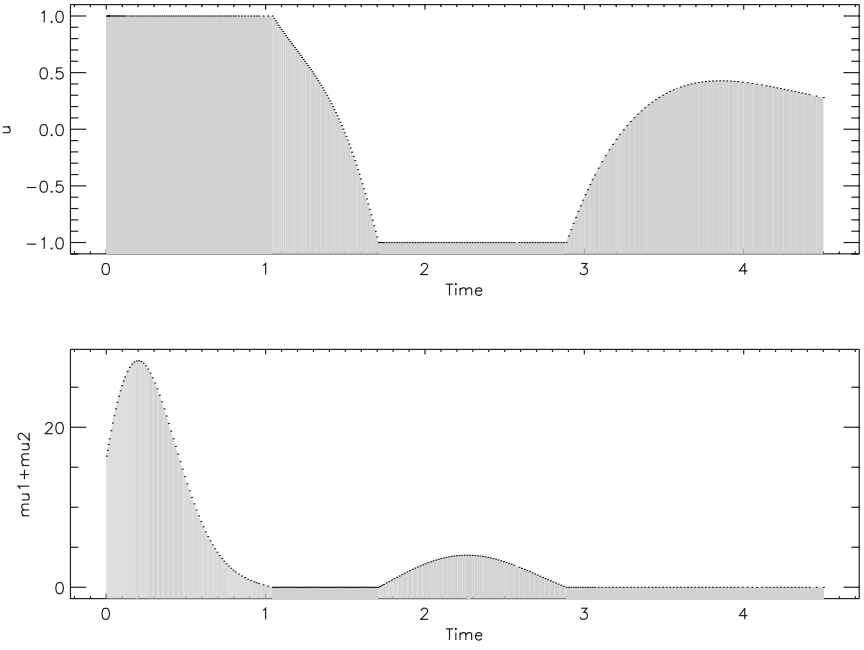


Figure 4.16. *Rayleigh problem, control and* $\mu_1 + \mu_2$.

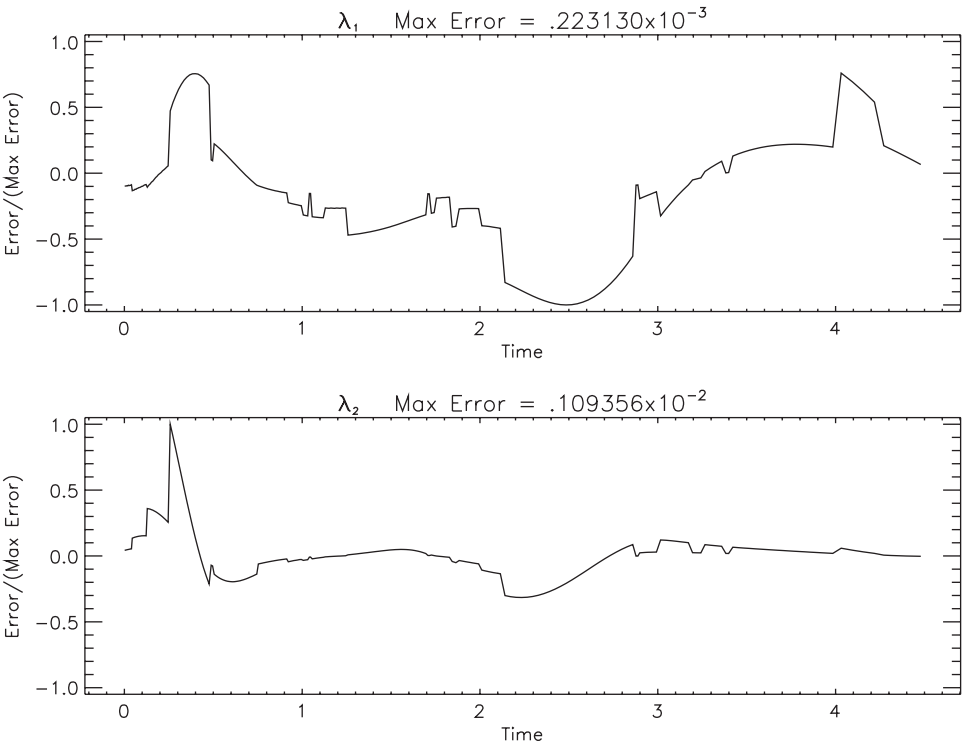


Figure 4.17. *Rayleigh problem, error in discrete adjoint estimates.*

We fix $t_F = 4.5$ and $p = 0.14$. The Hamiltonian is

$$H = u^2 + y_1^2 + \lambda_1(y_2) + \lambda_2[-y_1 + y_2(1.4 - py_2^2) + 4u] + \mu\left(u + \frac{y_1}{6}\right) \tag{4.288}$$

with adjoint equations

$$\dot{\lambda}_1 = \lambda_2 - 2y_1 - \frac{\mu}{6}, \tag{4.289}$$

$$\dot{\lambda}_2 = 3p\lambda_2 y_2^2 - 1.4\lambda_2 - \lambda_1. \tag{4.290}$$

The optimal control is

$$u(t) = \begin{cases} -y_1/6, & 0 \le t \le t_1, \\ -2\lambda_2, & t_1 \le t \le t_2, \\ -y_1/6, & t_2 \le t \le t_3, \\ -2\lambda_2, & t_3 \le t \le t_F \end{cases} \tag{4.291}$$

with optimal multiplier given by

$$\mu(t) = -2u - 4\lambda_2 = \frac{y_1}{3} - 4\lambda_2 \tag{4.292}$$

for $0 \le t \le t_1$ and $t_2 \le t \le t_3$. The switching structure is determined by the junction conditions

$$\mu(t_k) = \frac{y_1(t_k)}{3} - 4\lambda_2(t_k) = 0 \tag{4.293}$$

for $k = 1,2,3$. As before the state and differential adjoint variables are illustrated in Figure 4.18, and the control and path adjoints are shown in Figure 4.19. Again it is worth noting that we have adhered to the algebraic sign convention of Bryson and Ho [54] for the path constraint (4.286). The optimal value of the objective function obtained by the direct method is $F^* = 44.80444449$. It is interesting to note that the solution presented here has two distinct boundary arcs, whereas the solution presented in [133] has only one boundary arc. In fact our solution is slightly better (44.80444449 versus 44.80479861), which presumably can be attributed to the different switching structure. Curiously, we first solved the problem using the direct method in order to determine a good guess for the indirect method and thus were led to a switching structure with two boundary arcs! The requested accuracy of 10^{-7} was achieved using $M_d = 191$ points for the direct solution and $M_i = 271$ points (with four phases) for the indirect one. As with the previous examples the normalized error is illustrated in Figure 4.20, and it appears that the discrete adjoint estimates are accurate to approximately three figures.

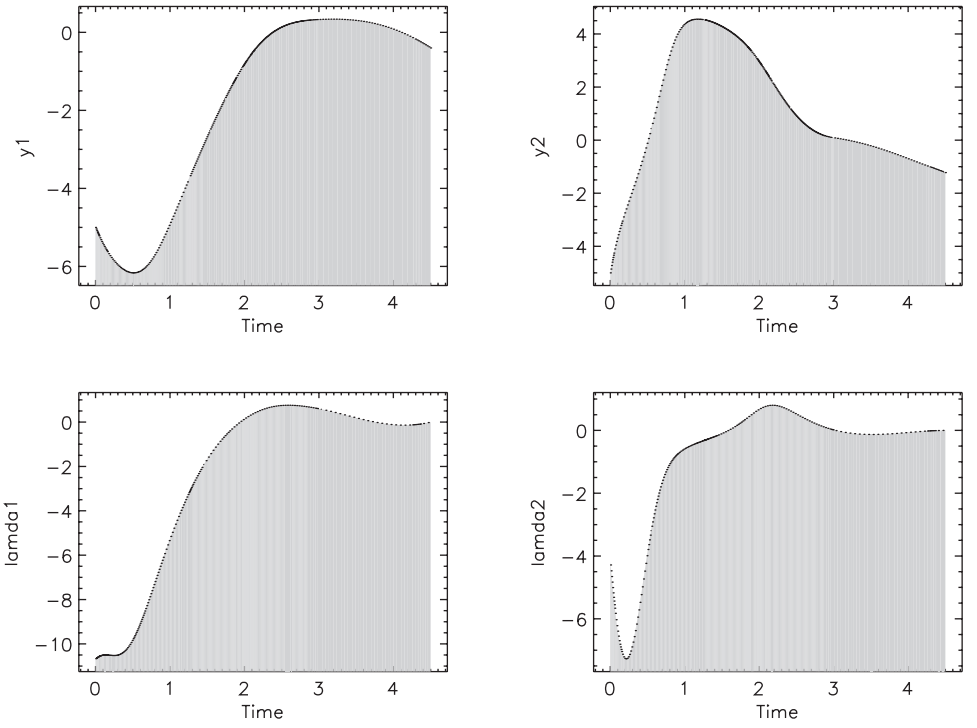


Figure 4.18. *Rayleigh mixed constraint problem, states* (y_1, y_2) *and adjoints* (λ_1, λ_2).

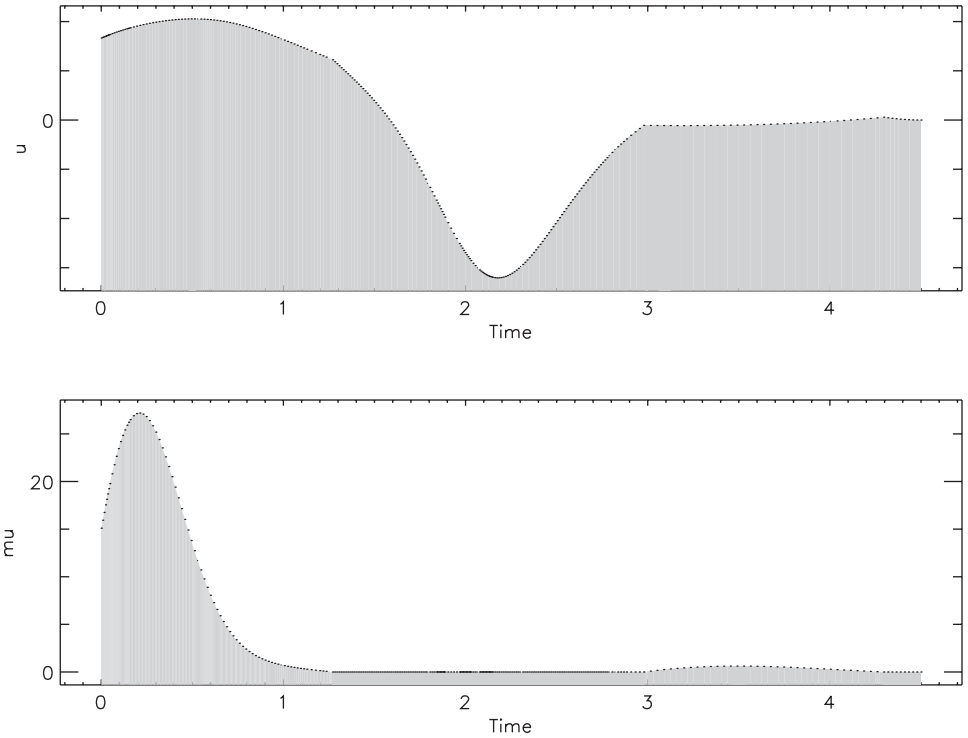


Figure 4.19. *Rayleigh mixed constraint problem, control and μ.*

Example 4.8 VAN DER POL OSCILLATOR WITH STATE CONSTRAINT. Minimize the objective

$$F = \int_0^5 (u^2 + y_1^2 + y_2^2)dt \tag{4.294}$$

subject to the constraints

$$\dot{y}_1 = y_2, \tag{4.295}$$
$$\dot{y}_2 = (1 - y_1^2)y_2 - y_1 + u, \tag{4.296}$$
$$0 \geq -y_2 + p \tag{4.297}$$

with boundary conditions

$$y_1(0) = 1, \qquad y_2(0) = 0. \tag{4.298}$$

Equation (4.296) is simply (3.49) augmented by the control variable u. We fix $p = -0.4$. Equation (4.297) is a first order state constraint, so the Hamiltonian is

$$H = u^2 + y_1^2 + y_2^2 + \lambda_1 y_2 + \lambda_2[(1 - y_1^2)y_2 - y_1 + u] + \mu[-(1 - y_1^2)y_2 + y_1 - u], \tag{4.299}$$

where the first derivative of the path constraint $\dot{s} = -\dot{y}_2 = -(1 - y_1^2)y_2 + y_1 - u$ appears in

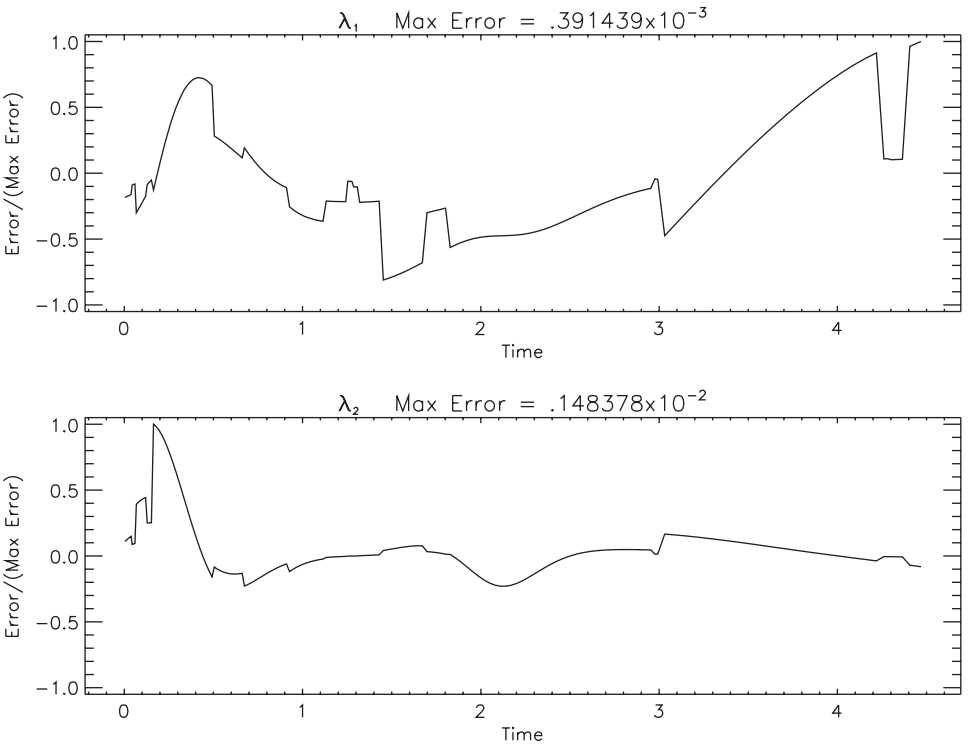


Figure 4.20. *Rayleigh mixed constraint problem, error in discrete adjoint estimates.*

H. The adjoint equations are

$$\dot{\lambda}_1 = -2y_1 + 2y_1y_2\lambda_2 + \lambda_2 - \mu(2y_1y_2 + 1), \tag{4.300}$$
$$\dot{\lambda}_2 = -2y_2 - \lambda_1 + \lambda_2(y_1^2 - 1) + \mu(1 - y_1^2). \tag{4.301}$$

The optimal control is

$$u(t) = \begin{cases} -\lambda_2/2, & 0 \le t \le t_1, \\ (y_1^2 - 1)y_2 + y_1, & t_1 \le t \le t_2, \\ -\lambda_2/2, & t_2 \le t \le 5 \end{cases} \tag{4.302}$$

with optimal multiplier given by

$$\mu(t) = \begin{cases} 0, & 0 \le t \le t_1, \\ 2(y_1^2 - 1)y_2 + 2y_1 + \lambda_2, & t_1 \le t \le t_2, \\ 0, & t_2 \le t \le 5. \end{cases} \tag{4.303}$$

The switching structure is determined by the tangency condition

$$\begin{aligned} 0 &= \dot{s} \\ &= -\dot{y_2} \\ &= -(1-y_1^2)y_2 + y_1 - u \\ &= (y_1^2 - 1)y_2 + y_1 + \lambda_2/2, \end{aligned} \tag{4.304}$$

which is enforced immediately before entering the boundary arc at $t = t_1^-$ and immediately after leaving the boundary arc at point $t = t_2^+$. The location of the exit point is also defined by enforcing the switching condition $\mu(t_2^-) = 0$. Observe that these conditions enforce continuity in the control variable across the junction points, i.e., from (4.302)

$$\begin{aligned} u(t_1^-) &= -\lambda_2/2 = (y_1^2 - 1)y_2 + y_1 = u(t_1^+), \\ u(t_2^-) &= (y_1^2 - 1)y_2 + y_1 = -\lambda_2/2 = u(t_2^+), \end{aligned} \tag{4.305}$$

and consequently lead to a jump discontinuity in the adjoint variable λ_2 at the entry to the boundary arc. The solutions are illustrated in Figures 4.21 and 4.22. First notice that the state and control variable histories obtained by the direct and indirect methods agree quite closely. The optimal value of the objective function obtained by the direct method

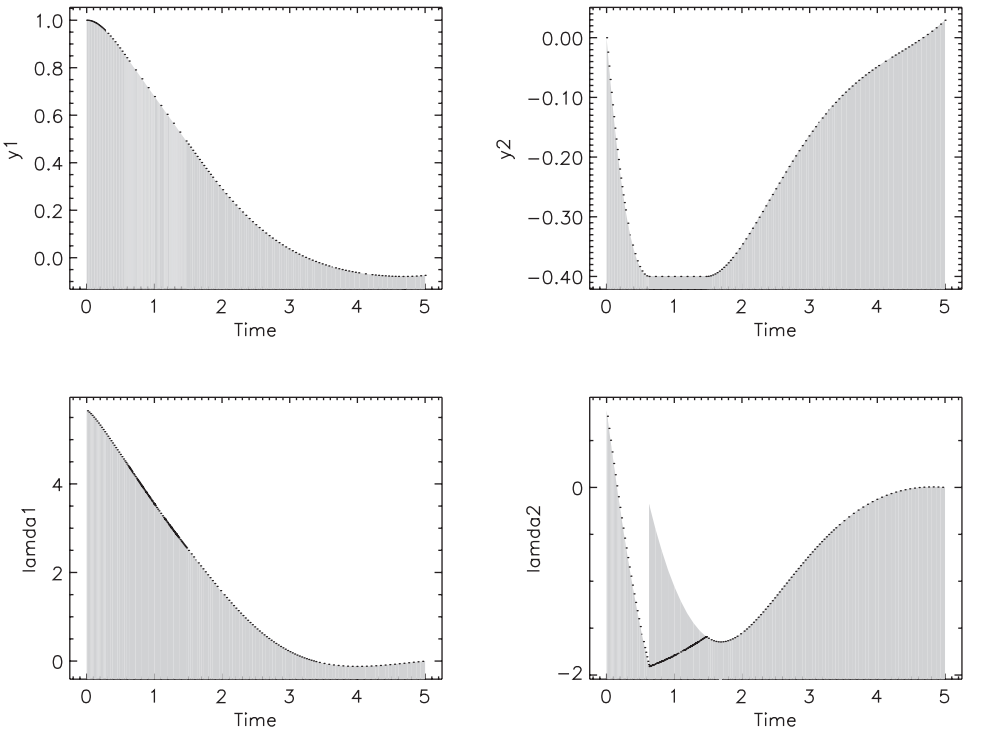


Figure 4.21. *Van der Pol problem, states* (y_1, y_2) *and adjoints* (λ_1, λ_2).

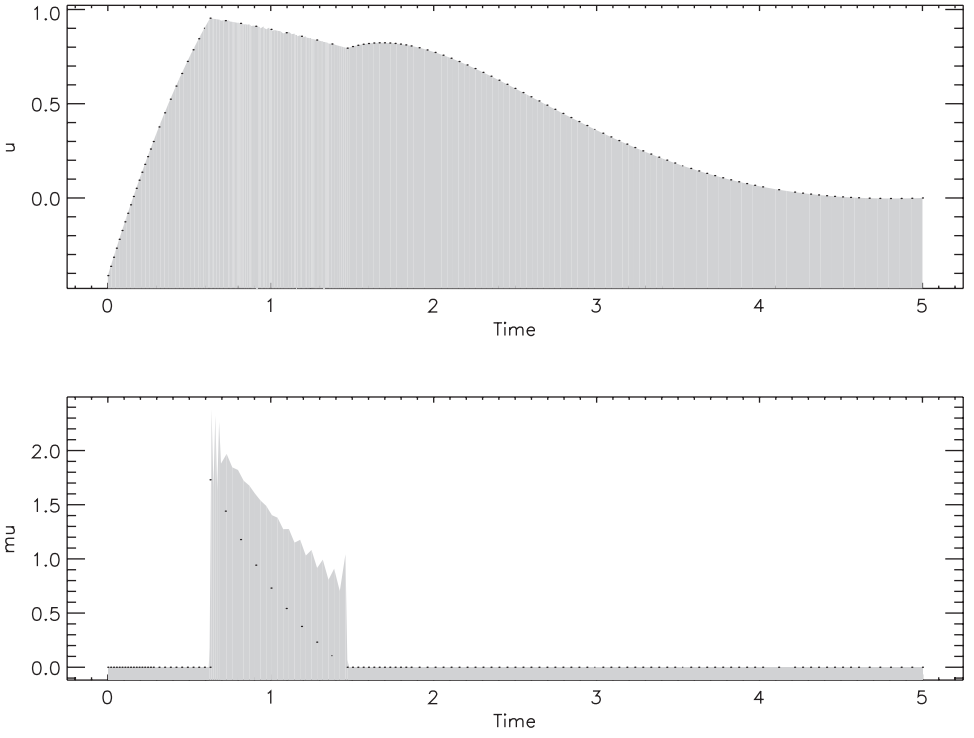


Figure 4.22. *Van der Pol problem, control and* μ.

is $F^* = 2.953733191$, and obviously both methods yield the same value. The requested accuracy of 10^{-7} was achieved using $M_d = 180$ points for the direct solution and $M_i = 111$ points (with three phases) for the indirect one. Furthermore, the adjoint variables λ agree when the state constraint is not active. However, along the boundary arc it is clear that the true value of λ_2 differs significantly from the discrete adjoint estimate. Furthermore, the true value of μ differs significantly from the discrete estimate. Nevertheless this behavior can be easily explained. First, it is clear that on the boundary arc when $t_1 \le t \le t_2$ the adjoint λ_2 has no influence on either the optimal control u or the states. In this region λ_2 can take any value and not change the objective! Similarly the value of the multiplier $\mu(t)$ does not appear in the optimal solution. The direct method can determine the solution entirely from the state and control variables provided the algebraic sign of the adjoint $\mu(t) > 0$ and this information is consistent between the two methods! Thus even though the *value* of the adjoint μ is wrong, the arithmetic sign is correct.

Although this ambiguity is somewhat disconcerting from a computational point of view, it is true in general. In their original paper [52], Bryson, Denham, and Dreyfus include an appendix entitled "Nonuniqueness of Influence Functions on a State Variable Inequality Constraint Boundary." In essence, they point out that the adjoint variables are not uniquely defined when a state constraint is active. By *convention*, they impose continuity in the adjoints at the *exit* from the state boundary. We have adhered to this convention when

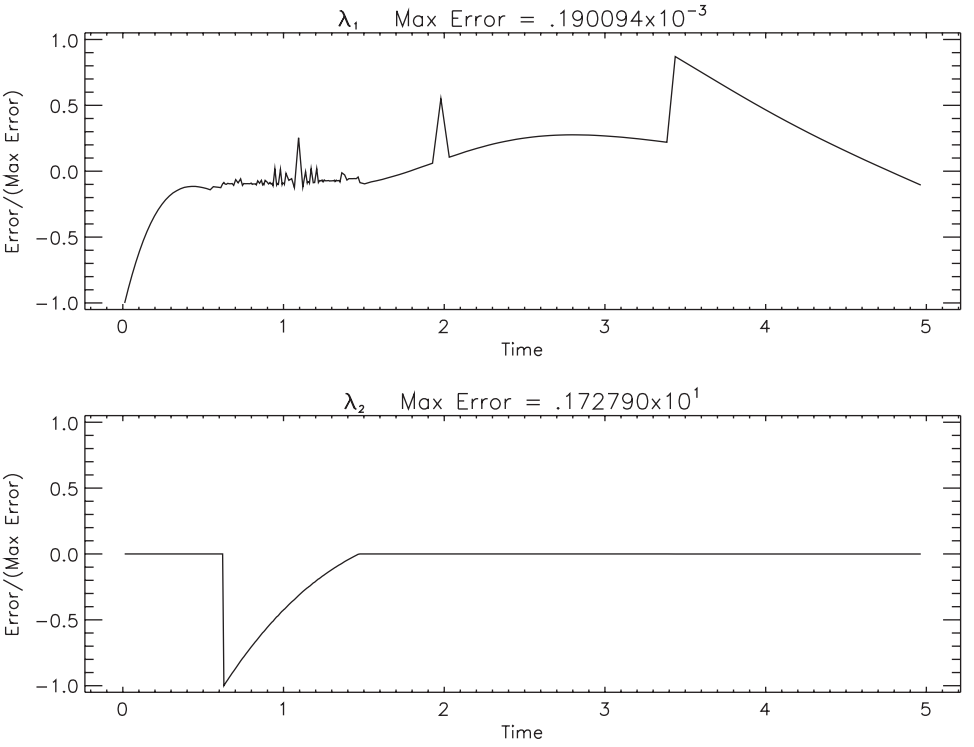


Figure 4.23. *Van der Pol problem, error in discrete adjoint estimates.*

we define our "truth" model. However, it is equally valid to impose continuity at the entry point! In short the true adjoint variables are not unique along the state boundary.

The apparent discrepancy is compounded by another factor. Recall that the discrete estimates follow from construction of the integral

$$\int_{t_I}^{t_F} \mu(t)s(t)dt \tag{4.306}$$

which appears when forming the augmented objective function (4.224). However, for a state constraint the optimality conditions are not derived by adjoining (4.306). Instead the augmented objective function is constructed by adjoining the term

$$\int_{t_I}^{t_F} \mu(t)s^q(t)dt, \tag{4.307}$$

where $s^q(t) = d^q s/dt^q$ is the qth time derivative of the state constraint. By definition the order q is constructed such that $s^q(t)$ is an explicit function of the control u. In essence the indirect method imposes the constraint $s^q(t) = 0$, whereas the direct method enforces $s(t) = 0$, and consequently the discrete adjoint estimate reflects this discrepancy.

Thus it is not surprising that the normalized error for λ_2 illustrated in Figure 4.23 is large when the state constraint is active. Elsewhere, it appears that the discrete adjoint estimates are accurate to approximately three figures.

4.12 Discretize Then Optimize

Computational techniques for solving optimal control problems typically require combining a discretization technique with an optimization method. One possibility is to *discretize then optimize*, that is, first discretize the differential equations and then apply an optimization algorithm to solve the resulting finite-dimensional problem. Conversely one can *optimize then discretize*, that is, write the continuous optimality conditions first and then discretize them. While many of the issues were discussed in Section 4.3 it is worthwhile to compare these alternatives on a specific example. What are the relative merits of each?

4.12.1 High Index Partial Differential-Algebraic Equation

Let us focus on a particular example proposed by Steve Campbell (see [26]) with additional discussion in [27, 28]. Heat transfer can be described by the PDE

$$\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2}, \tag{4.308}$$

where the spatial domain is $0 \le x \le \pi$ and the time domain is $0 \le t \le 5$. Conditions are imposed on three boundaries of this domain:

$$y(x,0) = y_0(x) = 0, \tag{4.309}$$

$$y(0,t) = u_0(t), \tag{4.310}$$

$$y(\pi,t) = u_\pi(t). \tag{4.311}$$

The input temperatures $u_0(t)$ and $u_\pi(t)$ are viewed as (algebraic) control variables. In addition the temperature over the domain is bounded according to

$$g(x,t) - y(x,t) \le 0, \tag{4.312}$$

where

$$g(x,t) = c\left[\sin x \sin\left(\frac{\pi t}{5}\right) - a\right] - b \tag{4.313}$$

is a prescribed function with $a = .5$, $b = .2$, and $c = 1$. Finally, we would like to choose the controls $u_0(t)$ and $u_\pi(t)$ to minimize

$$\phi = \int_0^5 \int_0^\pi y^2(x,t)dxdt + \int_0^5 \left[q_1 u_0^2(t) + q_2 u_\pi^2(t)\right] dt. \tag{4.314}$$

For our example we set the constants $q_1 = q_2 = 10^{-3}$.

One way to solve this problem is to introduce a discretization in the spatial direction, i.e., $x_k = k\delta = k\frac{\pi}{n}$ for $k = 0,\ldots,n$. If we denote $y(x_k,t) = y_k(t)$, then we can approximate the PDE (4.308) by the following system of ODEs:

$$\dot{y}_1 = \frac{1}{\delta^2}(y_2 - 2y_1 + u_0), \tag{4.315}$$

$$\dot{y}_k = \frac{1}{\delta^2}(y_{k+1} - 2y_k + y_{k-1}), \qquad k = 2,\ldots,n-2, \tag{4.316}$$

$$\dot{y}_{n-1} = \frac{1}{\delta^2}(u_\pi - 2y_{n-1} + y_{n-2}). \tag{4.317}$$

This *method of lines* approximation is obtained by using a central difference approximation (cf. Table 1.7) to $\frac{\partial^2 y}{\partial x^2}$ and incorporating the boundary conditions (4.310) and (4.311) to replace $y_0 = u_0$ and $y_n = u_\pi$. As a consequence of the discretization the single constraint (4.312) is replaced by the set of constraints

$$g(x_k,t) - y_k \le 0, \qquad k = 0,\ldots,n. \tag{4.318}$$

The boundary conditions (4.309) are imposed by setting $y_k(0) = 0$. Furthermore if we use a trapezoidal approximation to the integral in the x-direction, the objective function (4.314) becomes

$$\phi = \int_0^5 \left[\frac{1}{2}\delta + q_1\right] u_0^2(t)dt + \delta \sum_{k=1}^{n-1} \int_0^5 y_k^2(t)dt + \int_0^5 \left[\frac{1}{2}\delta + q_2\right] u_\pi^2(t)dt. \tag{4.319}$$

4.12.2 State Vector Formulation

Let us introduce a normalized time

$$t = \delta^2 \tau \tag{4.320}$$

and use "$\prime$" to denote differentiation with respect to τ, in which case

$$\mathbf{y}' = \frac{d\mathbf{y}}{d\tau} = \frac{d\mathbf{y}}{dt}\frac{dt}{d\tau} = \delta^2 \dot{\mathbf{y}}. \tag{4.321}$$

Using this notation let us rewrite the differential equations (4.315)–(4.317) as

$$\mathbf{y}' = \mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u} = \mathbf{f}(\mathbf{y},\mathbf{u}), \tag{4.322}$$

where the state vector is $\mathbf{y}^\mathsf{T} = (y_1, y_2, \ldots, y_{n-1})$, and the control vector is $\mathbf{u}^\mathsf{T} = (u_0, u_\pi)$. The symmetric, tridiagonal, $(n-1)\times(n-1)$ matrix $\mathbf{F}$ has elements

$$F_{\imath\jmath} = \begin{cases} -2 & \text{if } \jmath = \imath \text{ for } \imath = 1,2,\ldots,(n-1), \\ 1 & \text{if } \jmath = \imath - 1 \text{ for } \imath = 2,3,\ldots,(n-1), \\ 1 & \text{if } \imath = \jmath - 1 \text{ for } \jmath = 2,3,\ldots,(n-1), \\ 0 & \text{otherwise.} \end{cases} \tag{4.323}$$

The rectangular $(n-1)\times 2$ matrix $\mathbf{G}$ is defined by

$$G_{\imath\jmath} = \begin{cases} 1 & \text{if } \imath = 1 \text{ for } \jmath = 1, \\ 1 & \text{if } \imath = n-1 \text{ for } \jmath = 2, \\ 0 & \text{otherwise.} \end{cases} \tag{4.324}$$

We can also write the objective function (4.319)

$$\phi = \frac{1}{2}\int_0^{5\delta^{-2}} \left(\mathbf{y}^{\mathsf{T}}\mathbf{A}\mathbf{y} + \mathbf{u}^{\mathsf{T}}\mathbf{B}\mathbf{u}\right) d\tau, \tag{4.325}$$

where the $(n-1)\times(n-1)$ diagonal matrix

$$A_{\imath\jmath} = \begin{cases} 2\delta^3 & \text{if } \jmath = \imath \text{ for } \imath = 1,2,\ldots,(n-1), \\ 0 & \text{otherwise} \end{cases} \tag{4.326}$$

and the 2×2 diagonal matrix

$$\mathbf{B} = \begin{bmatrix} \delta^2(\delta + 2q_1) & 0 \\ 0 & \delta^2(\delta + 2q_2) \end{bmatrix}. \tag{4.327}$$

4.12.3 Direct Transcription Results

The direct transcription method utilizes a *discretize then optimize* philosophy. Specifically, the differential equations (4.322) and objective function (4.325) are replaced by discrete approximations leading to a large, sparse NLP problem. After solving the NLP the discretization (in time) is refined until a sufficiently accurate solution is obtained. If we set $n = 20$, the solution obtained using the direct method is illustrated in Figure 4.24, and Figure 4.25 illustrates the behavior of the solution with respect to the state variable constraint.

4.12.4 The Indirect Approach

The *optimize then discretize* philosophy requires explicit calculation of the optimality conditions. In order to simplify further analysis let us assume that n is even, and then define $m = n/2$. Computational experience suggests that at the solution the only active inequality constraint of the set (4.318) is at the midpoint of the spatial discretization as illustrated in Figure 4.26. Although the visual examination suggests a single constrained arc, there are results in the literature stating when touch points [156] and constraint arcs [117] cannot exist. Thus it may also be important to determine the isolated points (touch points) where the constraints are active. Thus we ignore all but the single inequality

$$s(\mathbf{y},\tau) = g_m(\delta^2\tau) - y_m = g_m(\delta^2\tau) - \mathbf{e}^{\mathsf{T}}\mathbf{y} \le 0, \tag{4.328}$$

where the $n-1$ vector

$$e_\imath = \begin{cases} 1 & \text{if } \imath = m \text{ for } \imath = 1,2,\ldots,(n-1), \\ 0 & \text{otherwise.} \end{cases} \tag{4.329}$$

Denote the kth derivative of $s(\mathbf{y},\tau)$ by $d^k s/d\tau^k = s^{(k)}(\tau)$. Then

$$\begin{aligned} s^{(1)} &= g_m^{(1)} - \mathbf{e}^{\mathsf{T}}\mathbf{y}' \\ &= g_m^{(1)} - \mathbf{e}^{\mathsf{T}}(\mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u}) \\ &= g_m^{(1)} - \mathbf{e}^{\mathsf{T}}\mathbf{F}\mathbf{y} \end{aligned} \tag{4.330}$$

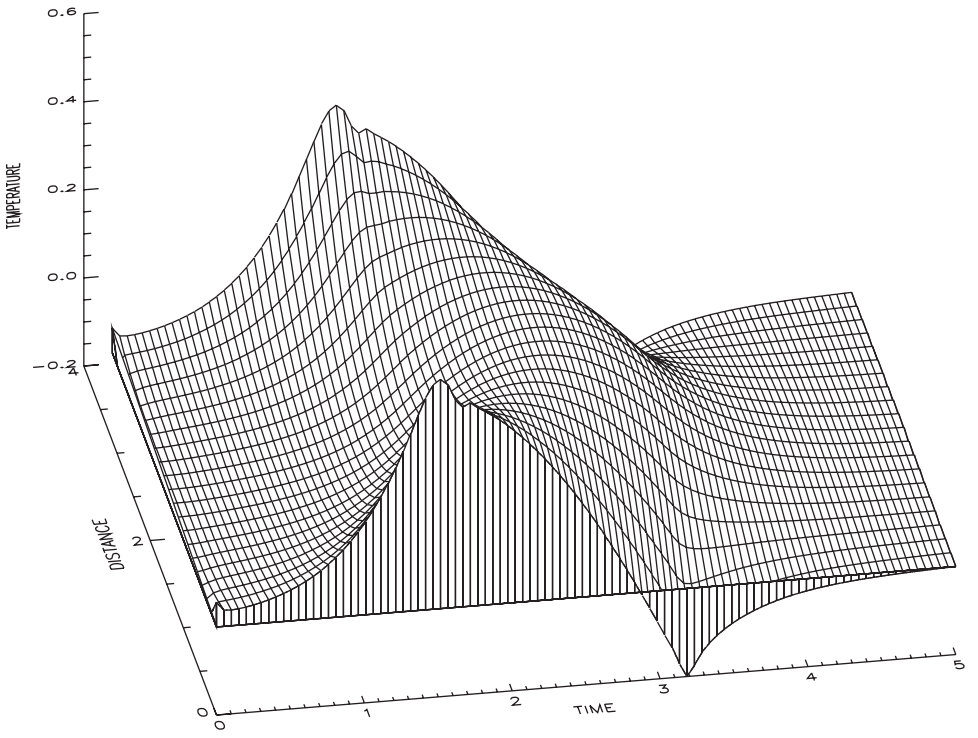


Figure 4.24. $y(x,t)$.

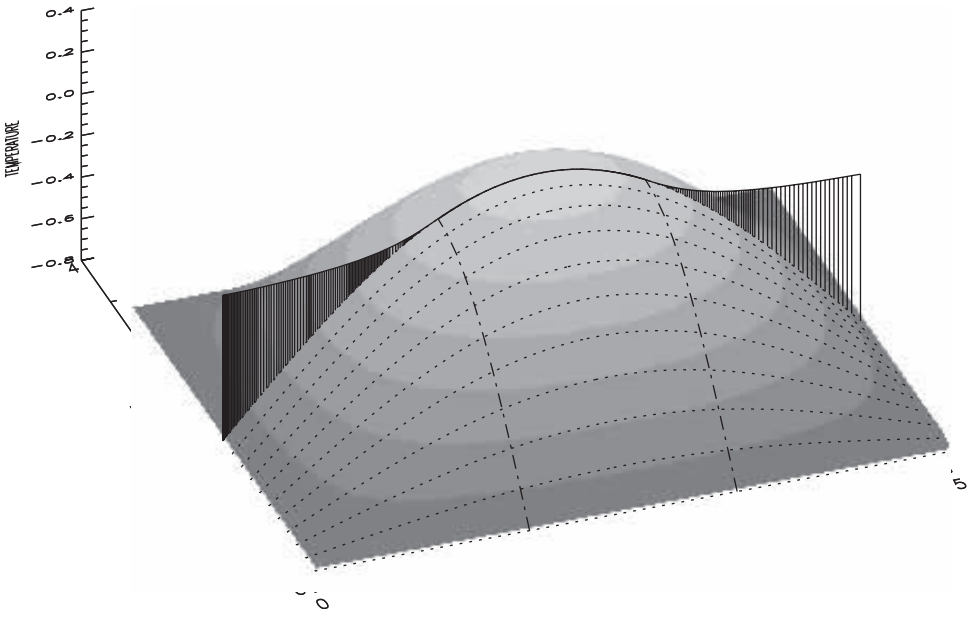


Figure 4.25. *Inequality* $y(x_{10},t) \geq g(x_{10},t)$.

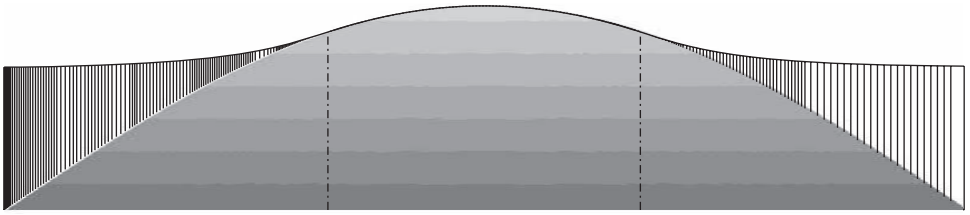


Figure 4.26. *Solution regions.*

since $\mathbf{e}^{\mathsf{T}}\mathbf{G}\mathbf{u} = 0$, where the definition of $\mathbf{y}'$ from (4.322) is used to simplify the expression. The process of differentiation followed by substitution can be repeated leading to the expression

$$s^{(k)} = g_m^{(k)} - \mathbf{e}^{\mathsf{T}} \left(\prod_{i=1}^{k} \mathbf{F} \right) \mathbf{y} \tag{4.331}$$

when $k < m$ and

$$s^{(m)} = g_m^{(m)} - \mathbf{e}^{\mathsf{T}} \left(\prod_{i=1}^{m-1} \mathbf{F} \right) (\mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u}). \tag{4.332}$$

The kth derivative of the constraint function is given by

$$g_m^{(k)} = \frac{d^k}{d\tau^k} \Big[g(x_m, \delta^2 \tau) \Big] = \delta^{2k} \frac{d^k}{dt^k} \big[g(x_m, t) \big], \tag{4.333}$$

which can be computed from the defining expression (4.313).

Let us define

$$\widehat{\mathbf{e}}_k^{\mathsf{T}} = \mathbf{e}^{\mathsf{T}} \left(\prod_{i=1}^{k} \mathbf{F} \right) \tag{4.334}$$

for $k = 0, 1, \ldots, m-1$. Thus for $0 \le k < m$, (4.331) becomes

$$s^{(k)} = g_m^{(k)} - \widehat{\mathbf{e}}_k^{\mathsf{T}} \mathbf{y} \tag{4.335}$$

and (4.332) is just

$$s^{(m)} = g_m^{(m)} - \widehat{\mathbf{e}}_{m-1}^{\mathsf{T}} (\mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u}). \tag{4.336}$$

4.12.5 Optimality Conditions

Unconstrained Arcs ($s < 0$)

When the state constraint is strictly satisfied, the *Hamiltonian* is given by

$$\begin{aligned} H &= L(\mathbf{y}, \mathbf{u}, t) + \boldsymbol{\lambda}^{\mathsf{T}} \mathbf{f} \\ &= \frac{1}{2} \mathbf{y}^{\mathsf{T}} \mathbf{A} \mathbf{y} + \frac{1}{2} \mathbf{u}^{\mathsf{T}} \mathbf{B} \mathbf{u} + \boldsymbol{\lambda}^{\mathsf{T}} (\mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u}). \end{aligned} \tag{4.337}$$

Recalling that $\mathbf{F}^\mathsf{T} = \mathbf{F}$ for our problem, it then follows that the *adjoint equations* (4.7) are

$$\boldsymbol{\lambda}' = -\mathbf{H}_y^\mathsf{T} = -\mathbf{A}\mathbf{y} - \mathbf{F}\boldsymbol{\lambda}. \tag{4.338}$$

The *maximum principle* (4.8) yields

$$\mathbf{0} = \mathbf{H}_u^\mathsf{T} = \mathbf{B}\mathbf{u} + \mathbf{G}^\mathsf{T}\boldsymbol{\lambda}, \tag{4.339}$$

which can be solved analytically to give an expression for the optimal control

$$\mathbf{u} = -\mathbf{B}^{-1}\mathbf{G}^\mathsf{T}\boldsymbol{\lambda}. \tag{4.340}$$

Constrained Arcs ($s = 0$)

When the solution lies on the state constraint the *Hamiltonian* is

$$\begin{aligned} H &= L(\mathbf{y},\mathbf{u},t) + \boldsymbol{\lambda}^\mathsf{T}\mathbf{f} + \mu s^{(m)} \\ &= \frac{1}{2}\mathbf{y}^\mathsf{T}\mathbf{A}\mathbf{y} + \frac{1}{2}\mathbf{u}^\mathsf{T}\mathbf{B}\mathbf{u} + \boldsymbol{\lambda}^\mathsf{T}(\mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u}) + \mu\left[g_m^{(m)} - \widehat{\mathbf{e}}_{m-1}^\mathsf{T}(\mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u})\right] \\ &= \frac{1}{2}\mathbf{y}^\mathsf{T}\mathbf{A}\mathbf{y} + \frac{1}{2}\mathbf{u}^\mathsf{T}\mathbf{B}\mathbf{u} + \mu g_m^{(m)} + (\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1})^\mathsf{T}(\mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u}). \end{aligned} \tag{4.341}$$

In this case the *adjoint equations* are

$$\boldsymbol{\lambda}' = -\mathbf{H}_y^\mathsf{T} = -\mathbf{A}\mathbf{y} - \mathbf{F}\boldsymbol{\lambda} + \mu\widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{F} = -\mathbf{A}\mathbf{y} - \mathbf{F}(\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1}). \tag{4.342}$$

The *maximum principle* yields

$$\mathbf{0} = \mathbf{H}_u^\mathsf{T} = \mathbf{B}\mathbf{u} + \mathbf{G}^\mathsf{T}\boldsymbol{\lambda} - \mu\mathbf{G}^\mathsf{T}\widehat{\mathbf{e}}_{m-1} = \mathbf{B}\mathbf{u} + \mathbf{G}^\mathsf{T}(\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1}), \tag{4.343}$$

which can be solved analytically to give an expression for the optimal control

$$\mathbf{u} = -\mathbf{B}^{-1}\mathbf{G}^\mathsf{T}(\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1}). \tag{4.344}$$

On the constrained arc we require $s^{(m)} = 0$, so if we substitute (4.344) into (4.336) and rearrange, we obtain

$$\begin{aligned} 0 &= s^{(m)} \\ &= g_m^{(m)} - \widehat{\mathbf{e}}_{m-1}^\mathsf{T}(\mathbf{F}\mathbf{y} + \mathbf{G}\mathbf{u}) \\ &= g_m^{(m)} - \widehat{\mathbf{e}}_{m-1}^\mathsf{T}\left[\mathbf{F}\mathbf{y} - \mathbf{G}\mathbf{B}^{-1}\mathbf{G}^\mathsf{T}(\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1})\right] \\ &= g_m^{(m)} - \widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{F}\mathbf{y} + \widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{G}\mathbf{B}^{-1}\mathbf{G}^\mathsf{T}(\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1}) \\ &= g_m^{(m)} - \widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{F}\mathbf{y} + \widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{G}\mathbf{B}^{-1}\mathbf{G}^\mathsf{T}\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{G}\mathbf{B}^{-1}\mathbf{G}^\mathsf{T}\widehat{\mathbf{e}}_{m-1}. \end{aligned} \tag{4.345}$$

Solving (4.345) we obtain the following expression:

$$\mu = \frac{g_m^{(m)} - \widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{F}\mathbf{y} + \widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{G}\mathbf{B}^{-1}\mathbf{G}^\mathsf{T}\boldsymbol{\lambda}}{\widehat{\mathbf{e}}_{m-1}^\mathsf{T}\mathbf{G}\mathbf{B}^{-1}\mathbf{G}^\mathsf{T}\widehat{\mathbf{e}}_{m-1}}. \tag{4.346}$$

Boundary Conditions

Inspection of the direct solution (Figure 4.26) suggests that there is one constrained region. Consequently let us assume that the optimal solution is defined on three distinct regions

$$0 \le \tau_1 \le \tau_2 \le 5\delta^{-2}, \tag{4.347}$$

where the constrained arc occurs for $\tau_1 \le \tau \le \tau_2$ and the other regions are unconstrained. Obviously from (4.309) we must have

$$\mathbf{y}(0) = \mathbf{0}. \tag{4.348}$$

If we define the vector

$$\mathbf{N}(\mathbf{y}, \tau) = \begin{bmatrix} s^{(0)} \\ s^{(1)} \\ \vdots \\ s^{(m-1)} \end{bmatrix}, \tag{4.349}$$

then at the beginning of the constrained arc ($\tau = \tau_1$) we must satisfy the *tangency conditions*

$$\mathbf{N}(\mathbf{y}, \tau_1^+) = \mathbf{0}. \tag{4.350}$$

Note that it is equally valid to impose these conditions at the other boundary ($\tau = \tau_2$); however, we retain the convention established in [54] and [139]. The adjoint variables at the boundary must satisfy

$$\boldsymbol{\lambda}(\tau_1^-) = \boldsymbol{\lambda}(\tau_1^+) + \mathbf{N}_y^{\mathsf{T}} \boldsymbol{\pi}, \tag{4.351}$$

where $\boldsymbol{\pi}$ is an m-vector and from (4.335) and (4.349) we have

$$\mathbf{N}_y^{\mathsf{T}} = -\left[\widehat{\mathbf{e}}_0, \widehat{\mathbf{e}}_1, \ldots, \widehat{\mathbf{e}}_{m-1}\right]. \tag{4.352}$$

If we define the vector

$$\mathbf{N}_\tau = \begin{bmatrix} g^{(1)} \\ g^{(2)} \\ \vdots \\ g^{(m)} \end{bmatrix}, \tag{4.353}$$

then we must also have

$$H(\tau_1^-) = H(\tau_1^+) - \boldsymbol{\pi}^{\mathsf{T}} \mathbf{N}_\tau. \tag{4.354}$$

The following simple continuity conditions must be imposed:

$$\boldsymbol{\lambda}(\tau_2^-) = \boldsymbol{\lambda}(\tau_2^+), \tag{4.355}$$

$$\mathbf{y}(\tau_1^-) = \mathbf{y}(\tau_1^+), \tag{4.356}$$

and

$$\mathbf{y}(\tau_2^-) = \mathbf{y}(\tau_2^+). \tag{4.357}$$

Finally, we must impose the terminal condition

$$\boldsymbol{\lambda}(5\delta^{-2}) = \mathbf{0}. \tag{4.358}$$

Optimality Conditions: Summary

Summarizing the optimality conditions leads to the statement of a multipoint BVP. Specifically, when the solution is unconstrained we must satisfy the differential equations

$$\begin{aligned} \mathbf{y}' &= \mathbf{F}\mathbf{y} - \mathbf{G}\mathbf{B}^{-1}\mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda}, \\ \boldsymbol{\lambda}' &= -\mathbf{A}\mathbf{y} - \mathbf{F}\boldsymbol{\lambda}, \end{aligned} \tag{4.359}$$

and the constrained arcs satisfy the differential equations

$$\begin{aligned} \mathbf{y}' &= \mathbf{F}\mathbf{y} - \mathbf{G}\mathbf{B}^{-1}\mathbf{G}^{\mathsf{T}}(\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1}), \\ \boldsymbol{\lambda}' &= -\mathbf{A}\mathbf{y} - \mathbf{F}(\boldsymbol{\lambda} - \mu\widehat{\mathbf{e}}_{m-1}) \end{aligned} \tag{4.360}$$

with boundary conditions (4.348), (4.350), (4.351), (4.354), (4.355), (4.356), (4.357), and (4.358).

4.12.6 Computational Comparison—Direct versus Indirect

Direct Method

The $\mathbb{SOCS}$ software was used to obtain solutions for a number of different spatial discretizations n. Table 4.4 presents a summary of the computational results. The first column of the table gives the value of the spatial discretization. Column two gives the number of mesh-refinement iterations required to achieve a discretization error of 10^{-7} or approximately eight significant figures, and column three shows the number of grid points required to achieve this accuracy. The *discretize then optimize* approach implemented in $\mathbb{SOCS}$ requires solving a sequence of large, sparse NLP problems, and the next three columns summarize information about the NLP performance. Column four gives the total number of gradient/Jacobian evaluations required to solve all of the NLP problems. The number of Hessian evaluations is given in the next column, followed by the total number of function evaluations including those needed for finite difference perturbations. The final column gives the total CPU time to compute the solution on a SUN Blade 150 workstation. Default tolerances were used for all of the NLP subproblems which guarantee that all discrete constraints were satisfied to a tolerance of $\epsilon = \sqrt{\varepsilon_m} \approx 10^{-8}$, where ε_m is the machine precision. Since the direct method does not require special treatment for constrained arcs these results were obtained using a single phase formulation.

Table 4.4. *Direct method (one phase).*

n	Refn	Grid	Grad	Hesn	Func	CPU
4	8	172	32	8	959	7.84
6	8	317	32	9	908	26.4
20	13	1063	50	14	1617	474
40	16	2105	61	18	2057	3685

Indirect Method

In contrast to the direct method, which can be formulated using a single phase, the indirect method must be modeled using more than one phase because the optimality conditions are

different on constrained and unconstrained arcs. If one assumes there are three phases as illustrated by Figure 4.26, then the $\mathbb{SOCS}$ software can also be applied to the boundary value problem summarized in section 4.12.5. The numerical results for this approach are presented in Table 4.5.

Table 4.5. *Indirect method (three phases).*

n	Refn	Grid	Grad	Hesn	Func	CPU
4	5	183	29	0	419	2.46
4†	5	183	51	21	2777	8.61
6†	6	229	150	52	908	62.8
20			*Fails to Converge*			
40			*Fails to Converge*			

The first row of the table corresponding to the case $n = 4$ demonstrates the expected behavior for the method. The mesh was refined 5 times, leading to a final grid with 183 points. The solution of the BVP required 29 gradient evaluations and 419 total function evaluations and was computed in 2.46 seconds. Since there were no degrees of freedom (and no objective function), no Hessian evaluations were required. For all other cases tested it was either difficult or impossible to compute a converged solution. In an attempt to improve robustness, the BVP was formulated as a constrained optimization problem by introducing slack variables. Specifically, instead of treating boundary conditions such as (4.354) as equality constraints of the form

$$\psi(\mathbf{x}) = 0$$

we formulated conditions of the form

$$\psi(\mathbf{x}) + s^+ - s^- = 0,$$

where the slack variables $s^+ \geq 0$ and $s^- \geq 0$, and then minimized an objective $F(\mathbf{x}) = \sum(s^+ + s^-)$. The second and third rows in Table 4.5 present results obtained using the slack variable formulation (denoted by †). In both cases solutions were obtained with converged values for the slacks $s \leq \epsilon$; however, the rate of convergence was very poor, as indicated by the number of Hessian evaluations. It was not possible to obtain a converged solution using any technique when $n > 6$.

4.12.7 Analysis of Results

The Quandary

The numerical results presented in Section 4.12.6 pose a number of controversial issues. It appears that the direct method solves the problem in a rather straightforward manner for all values of the spatial discretization n. Yet when the inequality constraint is active, $s(\mathbf{y}, \tau) = 0$, the combined DAE has index $m = n/2 + 1$. Thus, referring to Table 4.4, it appears that the direct method has solved a DAE system of very high index (11 or 21). However, usually the only way to solve a high index DAE is to use *index reduction* [48] since no known discretization converges for index 21 DAEs. In fact, the default discretization in

SOCS, like most discretizations, fails to converge if the index is greater than three [60]. In short, it appears that the direct method works, even though it shouldn't!

In contrast, the index reduction procedure has been performed for the indirect method. Specifically, the necessary conditions given by (4.345) explicitly involve the mth time derivative of the constraint $s(\mathbf{y}, \tau) = 0$; i.e., index reduction has been performed. In fact, *consistent initial conditions* for the high index DAE are imposed via the tangency conditions (4.350). In short, it appears that the indirect method does not work, even though it should!

In order to explain this quandary, we first suspected a bug in the code. To address this issue we compare the direct and indirect solutions for the cases $n = 4$ and $n = 6$. Figures 4.27 and 4.28 plot the difference between the direct and indirect solutions. At least visually the solutions for the state variable (temperature) appear to agree quite closely. There is a slight discrepancy between the control variable solutions; however, this can possibly be attributed to differences in the formulation. Specifically, the direct method has a single phase, and the control approximation is continuous by construction. In contrast, the indirect method has three phases and consequently can introduce a discontinuity in the control at the phase boundary. This still does not explain why the direct method works and the indirect method fails.

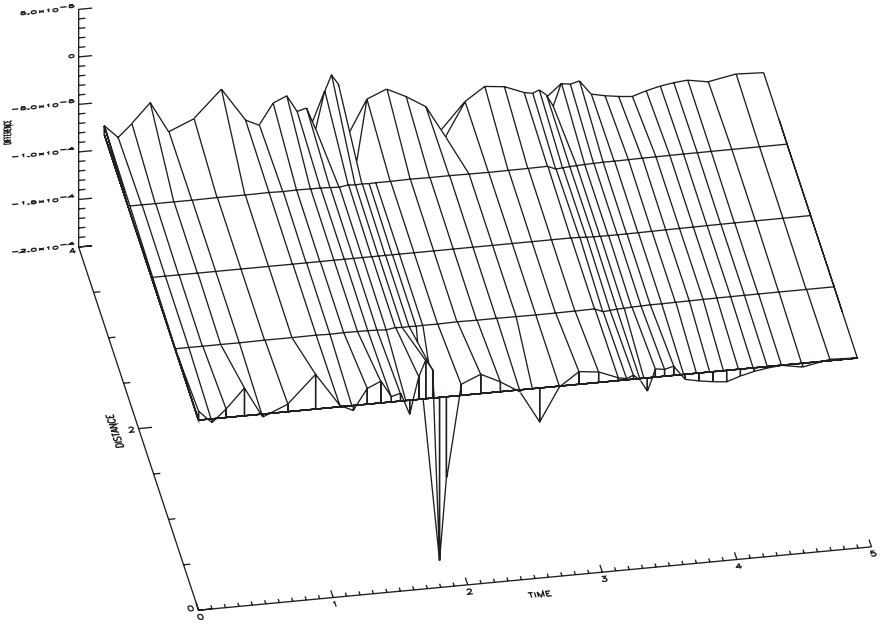


Figure 4.27. *Difference between direct and indirect solutions* $n = 4$.

The Explanation

Let us focus on the direct method. In particular let us take a microscopic look at the results obtained by the direct method. Figure 4.29 plots the quantity $s(\mathbf{y}, \tau)/\epsilon$, where $\epsilon = \sqrt{\varepsilon_m} \approx 10^{-8}$ and ε_m is the machine precision. Since ϵ is the NLP constraint tolerance, we

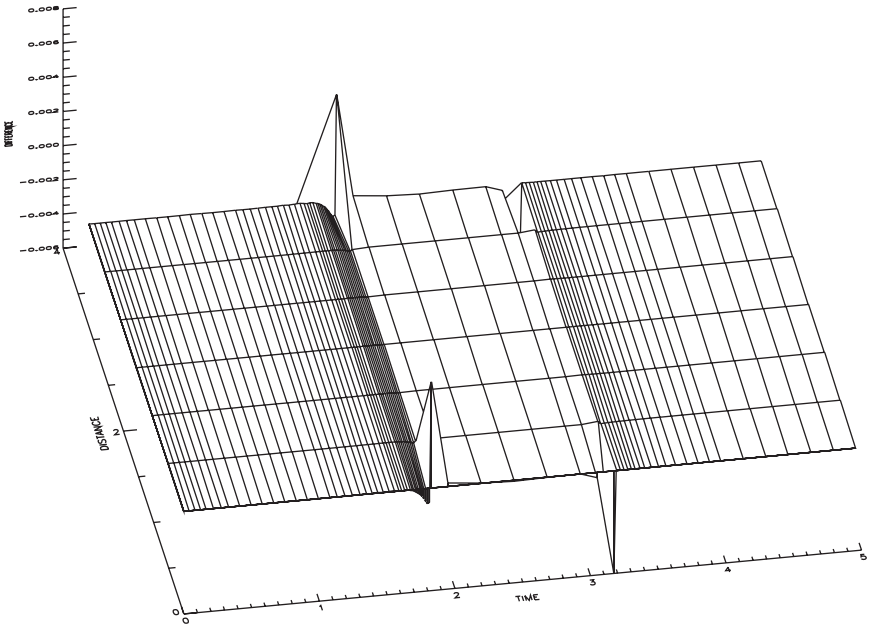


Figure 4.28. *Difference between direct and indirect solutions* $n = 6$.

can view $s(\mathbf{y}, \tau)/\epsilon$ as a *normalized path constraint error*. More precisely, the constraint $|s(\mathbf{y}, \tau)| \leq \epsilon$ when $-1 \leq s(\mathbf{y}, \tau)/\epsilon \leq 1$. Thus in Figure 4.29 we are plotting the regions when the path constraint is within convergence tolerance. Conversely when $s(\mathbf{y}, \tau)/\epsilon < -1$ the mathematical constraint (4.328) is strictly satisfied. Regions corresponding to strict satisfaction of the path constraint are shaded gray.

From the figure it would appear that when $n = 4$ there is exactly one region where the path constraint is within tolerance. However, for $n = 6$ it appears there are three distinct regions where the path constraint is within tolerance. And for $n = 20$ and $n = 40$ there may be as many as five regions where the path constraint is within tolerance. At least graphically, this suggests that the *number* of constrained arcs is not always one! It may be three, or five, or perhaps some other value. It also suggests that the number of constrained arcs may change with the spatial discretization n.

How many constrained arcs are there? Even though a graphical analysis cannot answer the question we immediately have an explanation for the apparent failure of the *indirect* method. In order to explicitly state the necessary conditions (4.359)–(4.360) with boundary conditions (4.348), (4.350), (4.351), (4.354), (4.355), (4.356), (4.357), and (4.358), we assumed there was only *one* constrained arc. Clearly, if the *number* of constrained arcs is wrong, the optimality conditions are wrong! Successful use of the indirect method requires a priori knowledge of the *number* of constrained arcs!

At this point, it is important to note the result from [117] which says that for problems of the type we consider here, there cannot be any constraint arcs at all if $n/2$ is an odd integer greater than 2. There can be only touch points. The computational complexity of the problem as n increases prevents a more careful examination of this behavior. In

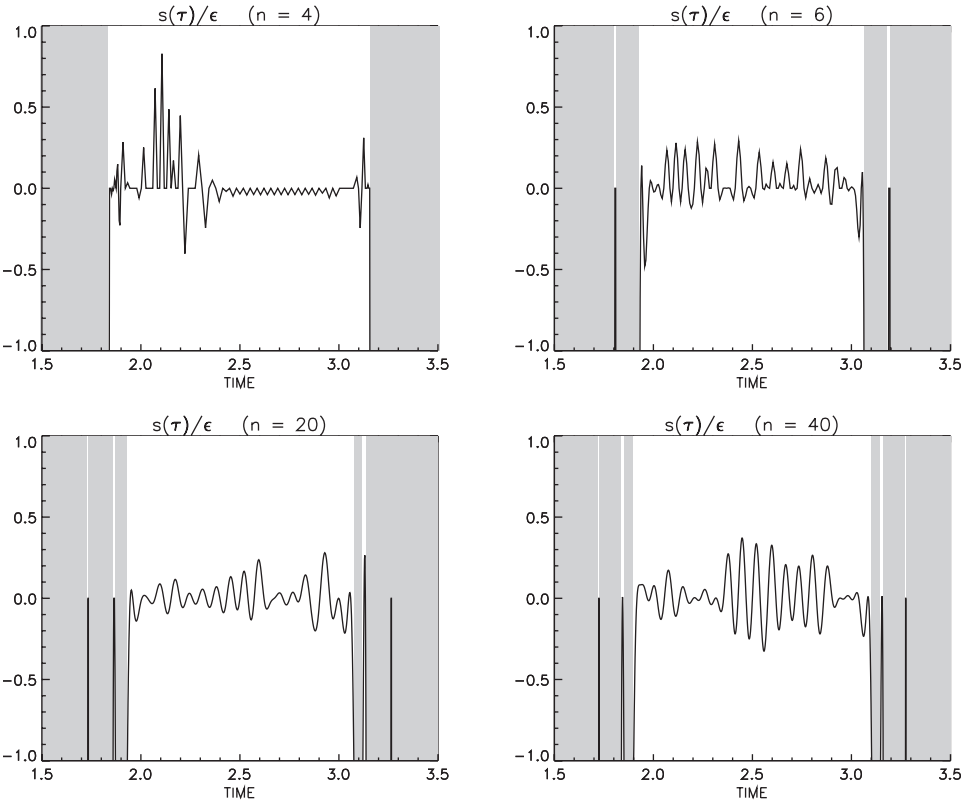


Figure 4.29. *Inequality constraint error.*

[28] a parameterized fixed-dimensional problem is developed with a similar structure for which we are able to compute the solutions much more accurately. What we see there is an increasing number of touch points except that the deviation from the constraint was several orders less than 10^{-7}. For this problem the theory held but was numerically meaningless since the constraint deviation was below normal error bounds.

Academic problems are often "simple." In contrast, real world applications often contain very complicated constraints. For these applications it is seldom possible to predict when the constraints will be active or compute the requisite necessary conditions. Also, as illustrated above and in [28], the theoretically predicted behavior may occur at or below the computational tolerances used to solve the problem, leading to an extremely ill-conditioned indirect solution. Thus the difficulties exhibited by this example can occur in real world problems where it is even harder to predict what will happen.

To fully understand how the direct method successfully obtained a solution it is helpful to review how the sparse NLP algorithm treats the discrete problem. During the final mesh-refinement iterations an implicit Runge–Kutta (Hermite–Simpson) discretization is used, and for this method the NLP variables from (4.57) are

$$\mathbf{x} = \left[\mathbf{y}_1, \mathbf{u}_1, \overline{\mathbf{u}}_2, \mathbf{y}_2, \mathbf{u}_2, \overline{\mathbf{u}}_3, \ldots, \mathbf{y}_M, \mathbf{u}_M\right]^\top. \tag{4.361}$$

Specifically, the NLP variable set consists of the state and control variables at M grid points $\mathbf{y}_j \equiv \mathbf{y}(\tau_j)$, $\mathbf{u}_j \equiv \mathbf{u}(\tau_j)$, and the control variables at the midpoints $\bar{\tau}_j \equiv (\tau_j + \tau_{j-1})/2$ of the $M-1$ discretization intervals $\overline{\mathbf{u}}_j \equiv \mathbf{u}[\bar{\tau}_j]$. The differential equations (4.322) are approximated by the defect constraints (4.58)

$$\boldsymbol{\zeta}_j = \mathbf{y}_{j+1} - \mathbf{y}_j - \frac{h_j}{6}\left[\mathbf{f}_{j+1} + 4\bar{\mathbf{f}}_{j+1} + \mathbf{f}_j\right] = 0, \qquad j = 1,\ldots,(M-1). \tag{4.362}$$

Of particular relevance to this discussion is the treatment of the (continuous) state inequality constraint (4.328), which is approximated by enforcing it at the grid points

$$s(\mathbf{y}_j, \tau_j) \le 0, \qquad j = 1,\ldots,M, \tag{4.363}$$

and the midpoints

$$s[\overline{\mathbf{y}}_j, \bar{\tau}_j] \le 0, \qquad j = 1,\ldots,(M-1). \tag{4.364}$$

Taken collectively, the set of $(2M-1)$ constraints (4.363)–(4.364) are treated as inequality constraints by the NLP algorithm.

Table 4.6 summarizes this information for the four spatial discretization cases. The first row gives the total number of discrete inequality constraints $(2M-1)$ that are enforced by the NLP algorithm. The second row of the table gives the number of inequality constraints that are within tolerance, which corresponds to the information displayed in Figure 4.29. Row three tabulates the number of constraints treated as "active" by the NLP algorithm. Row four gives the percentage of constraints within tolerance that are also considered active by the NLP.

Table 4.6. *The NLP active set.*

Spatial Discretization, n	4	6	20	40
Inequalities s	343	633	2125	4209
Inequalities within tolerance, $\lvert s\rvert < \epsilon$	102	149	475	1083
Active Inequalities	59	52	38	40
Percent Active	58	35	8	4

Clearly this data suggests that only a small fraction of the discrete constraints within convergence tolerance are actually treated as "active" inequalities by the NLP algorithm. Now, the active inequality constraints at the NLP solution satisfy a *constraint qualification test*. More precisely the underlying QP algorithm constructs the set of active inequality constraints such that the gradients are linearly independent (cf. Section 1.12). In essence the active set is constructed to satisfy the so-called *linear independent constraint qualification* (LIQC). Conversely, we suspect that if all of the inequalities within tolerance had been included in the active set, the resulting Jacobian matrix would be rank deficient. However, the solution does satisfy the more general *Mangasarian–Fromovitz constraint qualification* (MFQC) as demonstrated recently by Kameswaran and Biegler [120].

Can the direct approach be used to solve a high index (greater than three) DAE? Suppose the high index path constraint had been treated as an *equality* constraint. This would require treating every grid point as an equality constraint, thereby leading to a singular Jacobian matrix. In addition, as noted earlier, the discretization does not converge for high

index DAEs. The direct approach will fail if the path constraint is treated as an equality. Indeed, the solutions presented exploit the fact that the path constraints are *inequalities*, not *equalities*. In other words the direct approach solves the high index DAE conditions by directly enforcing a small subset of the discrete conditions and trivially satisfying the others. This explains why the direct approach works when it shouldn't!

Since the NLP active set is determined using the Lagrange multipliers for the discrete problem, the behavior of the direct method also demonstrates a theoretical issue of some interest. In [23], we prove for a class of higher index state constrained control problems that the discrete solution converges to the "continuous" optimal control using the theory of consistent approximations [144]. Although the proof is applicable only for a trapezoidal discretization, it does not rely on the convergence of the discrete NLP Lagrange multipliers to the continuous adjoint variables. In fact, it is shown in [23] that the multipliers fail to converge to the adjoint variables. In [23], it was also shown that for problems with an active constraint arc that the direct method could stabilize an unstable discretization by using small perturbations off the constraint arc. We suspect that the consistent approximation ideas will apply here, but as yet we have not extended the results in [23] to this situation.

To summarize, this example presents a comparison of two techniques for solving optimal control problems, namely the direct versus the indirect method. For the purpose of demonstration we focused on a model problem that requires solving a PDE subject to an inequality constraint on the state. After discretization using a method of lines, the problem can be stated as an optimal control problem with linear ODEs subject to a high index path inequality constraint. A preliminary examination suggests that the model problem has a single constrained arc, and the appropriate optimality conditions are derived for this case. However, we also demonstrate that the assumed arc structure is incorrect. The example illustrates that to successfully use the indirect method it is necessary to guess

(a) the number of constrained arcs,
(b) the location of the constrained arcs, and
(c) the correct index of each constrained arc.

In general it is extremely difficult to determine this information a priori, and failure to do so will cause the indirect method to fail. In contrast the direct method works because the underlying NLP algorithm determines the number, location, and correct active grid points for a linearly independent active set. In short, the direct method provides a robust method for solving optimal control problems especially when state inequality constraints are present.

4.13 Questions of Efficiency

> Traditionally, there are two major parts of a successful optimal control or optimal estimation solution technique. The first part is the "optimization" method. The second part is the "differential equation" method. When faced with an optimal control or estimation problem it is tempting to simply "paste" together packages for optimization and numerical integration. While naive approaches such as this may be moderately successful, the goal of this book is to suggest that there is a better way! The methods used to solve the differential equations and optimize the functions are intimately related (cf. the preface).

Algorithms for solving the NLP problem are described in Chapters 1 and 2. Chapters 3 and 4 present approaches for transcribing the continuous problem into an NLP. What

remains is to intelligently select an NLP algorithm that is appropriate for this class of problems. But how does one quantify a "good algorithm?" Ideally, a "good algorithm" will solve "most" problems "faster" than a "bad algorithm." Typically computer time is used to measure algorithm speed; however, when this is done it is imperative that all testing be done using the same hardware, compiler options, and operating system. The number of function evaluations can be used instead of (or in addition to) computer time, but then one must carefully define a "function evaluation." Furthermore to make a fair comparison between algorithms it is important to

(a) test a large suite of problems
(b) using the same initial guess and
(c) the same convergence criteria.

When comparing one NLP to another, it is common to perform benchmark tests using a suite of standard problems, e.g., Hock and Schittkowski [114] or CUTE [45]. While this provides useful information, it is not appropriate for testing an optimal control algorithm. In particular we would like to choose the best algorithm for solving a *sequence* of NLP problems. Typically at a step in the sequence the NLP problem will have

(a) more variables than the previous NLP,
(b) more constraints than the previous NLP,
(c) and possibly different nonlinearity (because of a different discretization).

The initial guess for each NLP is determined by the solution of the previous (coarse grid) solution. Convergence criteria involve the *sequence* of NLP problems, rather than just a single NLP. Since each NLP is derived by transcription of an optimal control problem, the nonlinearity is dictated by the differential equations. Finally all problems exhibit matrix sparsity that is associated with the discretization technique. In short, choosing an algorithm to efficiently solve a *sequence* of NLPs is not the same as choosing an algorithm for a single NLP. One cannot simply "paste" together packages for optimization and numerical integration!

Unfortunately, there is no easy way to prove which algorithm is best suited for this environment. Nevertheless, considerable insight can be gained by examining the behavior of the sparse SQP algorithm on a typical problem. Let us revisit an example first presented in Table 2.2 (p. 54) of reference [21]. This space shuttle reentry problem has five state and two control variables and is discussed in Example 6.2 as well as in [36]. To illustrate the behavior, the problem was solved using a trapezoidal discretization with M equally spaced grid points. Ten different cases were solved and Table 4.7 summarizes the significant performance parameters. The number of grid points shown in column one ranged from 50 to 25600. Columns two, three, and four present the dimensions of the resulting NLP problem, i.e., the number of variables n, active constraints $\hat{m}$, and degrees of freedom $n_d = n - \hat{m}$. Column five of the table (NFE) shows the total number of function evaluations (including finite difference perturbations) needed to solve the problem. Column six contains the number of Jacobian (gradient) evaluations (NGE) needed to converge, and column seven presents the number of Hessian evaluations (NHE). Column eight of the table presents the CPU time (seconds) to obtain a solution on a Dell M90 laptop computer. The discretization error ϵ_α is presented in the last column. It should be emphasized that the results in Table 4.7 *do not* use the mesh-refinement algorithm described in Section 4.7. Instead, each case begins with an initial guess constructed by linearly interpolating the state and control variables between the initial and final times.

Table 4.7. *Shuttle reentry example.*

M	n	$\hat{m}$	n_d	NFE	NGE	NHE	T(sec)	ϵ_α
50	351	253	98	405	16	7	0.120000	1.19×10^{-2}
100	701	503	198	494	18	9	0.260000	2.25×10^{-3}
200	1401	1003	398	494	18	9	0.570000	3.64×10^{-4}
400	2801	2003	798	405	16	7	1.06000	5.26×10^{-5}
800	5601	4003	1598	440	16	8	2.75000	7.08×10^{-6}
1600	11201	8003	3198	367	15	6	6.83000	9.18×10^{-7}
3200	22401	16003	6398	413	16	7	21.7700	1.17×10^{-7}
6400	44801	32003	12798	502	18	9	92.7300	1.47×10^{-8}
12800	89601	64003	25598	457	17	8	356.660	1.84×10^{-9}
25600	179201	128003	51198	503	18	9	4588.33	6.64×10^{-10}

A second set of data was gathered by repeating the 10 cases using a Hermite–Simpson compressed (HSC) discretization instead of trapezoidal, and the combined results are illustrated using a logarithmic scale in Figure 4.30. The number of variables n from Table 4.7 are plotted with a solid line and range from 351 to 179201. The number of constraints is plotted using a dotted line, and the number of degrees of freedom are plotted using a dashed line. The shaded regions (light, medium, and dark, respectively) present the corresponding information for the HSC discretization. Obviously the NLP dimensions grow linearly with the mesh size M.

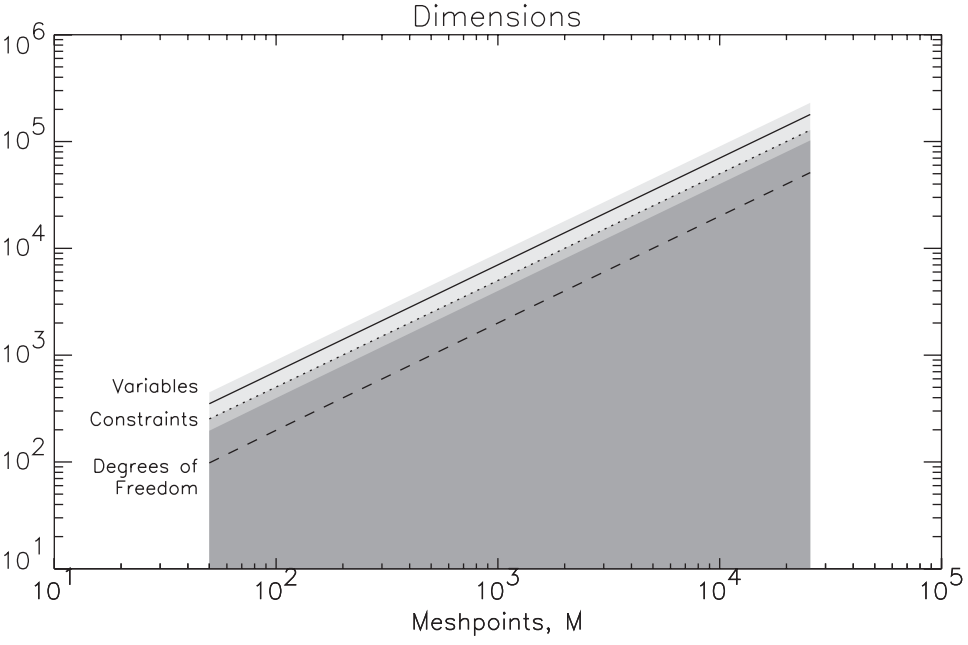


Figure 4.30. *NLP dimensions increase with mesh.*

Figure 4.31 illustrates the quantities that *do not change* with the mesh size. Specifically the number of function evaluations, the number of gradient evaluations, and the number of Hessian evaluations are essentially the same regardless of whether the NLP problem has 351 variables or 179201 variables. There are two reasons for this:

1. *sparse differences*—first and second derivatives are computed using sparse finite differences, and the number of index sets remains unchanged by mesh size, and
2. *quadratic convergence*—a second order (Newton) NLP algorithm is used, and the number of iterations is not altered by problem size.

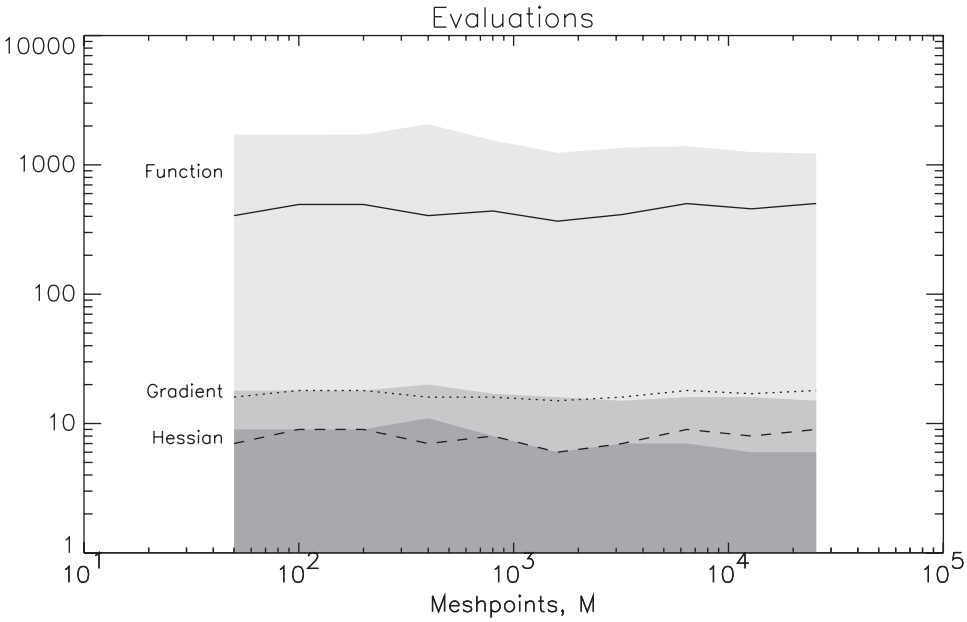


Figure 4.31. *NLP evaluations do not increase with mesh.*

Figure 4.32 illustrates how the accuracy of the solution as measured by the discretization error is related to the size of the grid. The discretization error for a trapezoidal method is displayed by the shaded region, whereas the Hermite–Simpson error is plotted with a solid line. As expected, the higher-order Hermite–Simpson method requires fewer (equally spaced) grid points to achieve the same accuracy as a lower-order trapezoidal scheme. It is also interesting to note that the error for the HSC method actually increases as the grid becomes very fine, presumably because of roundoff error effects. For comparison, the behavior of the mesh-refinement algorithm with a requested tolerance of $\epsilon_\alpha \leq \delta = 10^{-7}$ is also displayed in the figure, using a dashed line. Note that after the first two refinements using a trapezoidal method, the third refinement changes to a (higher-order) HSC scheme, without changing the number of grid points. This behavior is obvious because of the vertical line with $M = 99$.

Of course the most important figure of merit when assessing an algorithm is how fast it is possible to obtain a solution to a requested accuracy. Figure 4.33 illustrates how

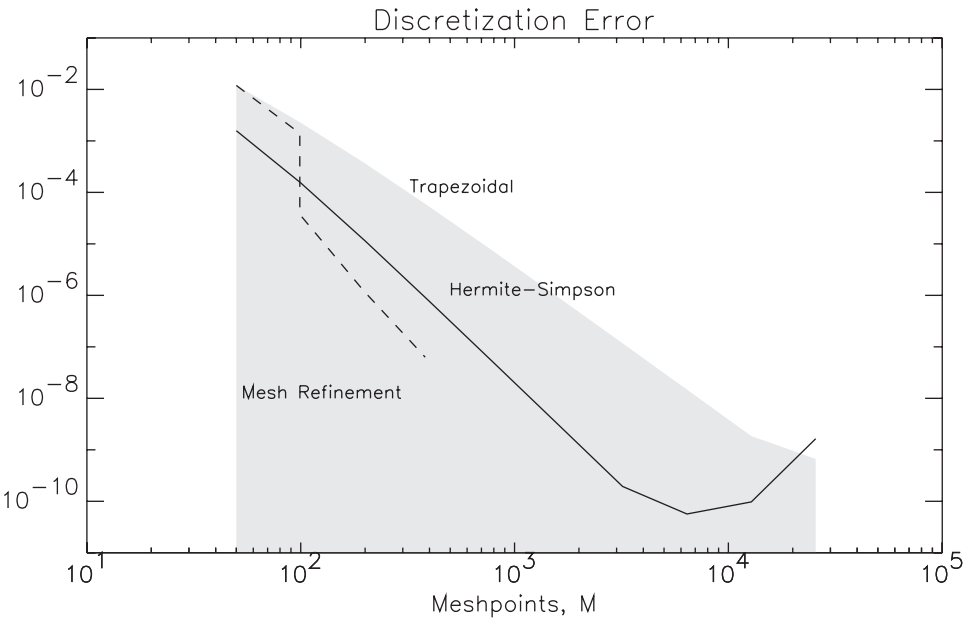


Figure 4.32. *Discretization error decreases with mesh.*

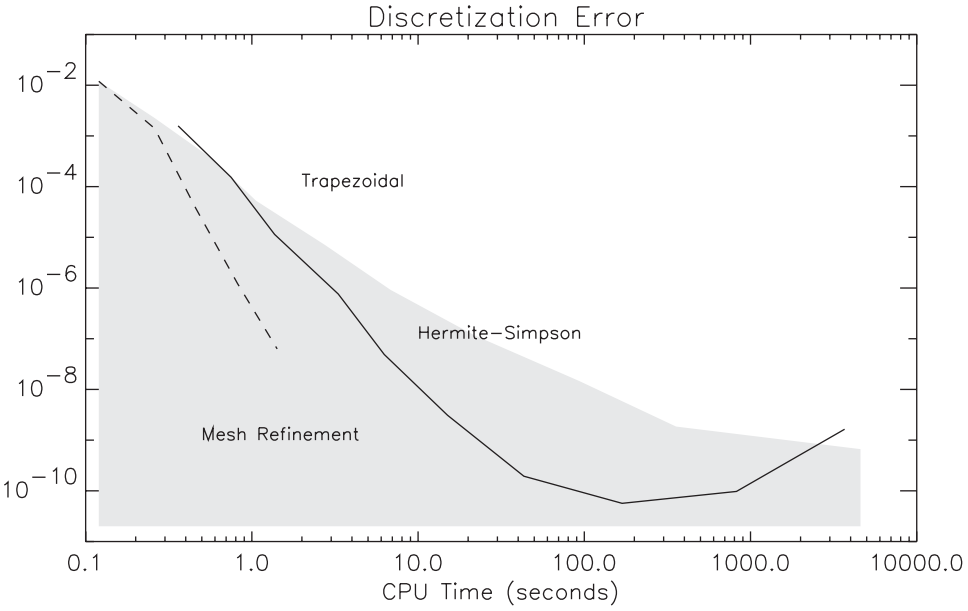


Figure 4.33. *Discretization error versus CPU time.*

the computation time (CPU seconds) is related to the discretization error. The shaded region illustrates how the CPU time increases as the discretization error decreases when using an equidistributed trapezoidal grid. The solid line illustrates the same information using an equidistributed Hermite–Simpson scheme. As expected a specified accuracy can be obtained much faster using the HSC method. Again the best algorithmic performance is demonstrated when mesh refinement is used. Finally, notice that the CPU time grows *linearly* with the size of the problem! This cost can be attributed to the fact that the computational complexity of the sparse linear algebra is $\mathcal{O}(\kappa n)$, where κ is a constant defined by the nonzero percentage of the matrix rather than $\mathcal{O}(n^3)$ for dense linear algebra.

Question: Newton or Quasi-Newton Hessian?

Historically, quasi-Newton Hessian approximations were developed for problems with no constraints. Thus when extending the technique to constrained problems one can utilize a quasi-Newton approximation to the *full* Hessian. However, exploiting matrix sparsity when using a quasi-Newton update has to date been computationally unsuccessful. Furthermore, for optimal control applications, the problem dimension $n \sim (n_y + n_u)M$ can be quite large. One common alternative is to apply a quasi-Newton update to the *projected or reduced* Hessian (1.63), which is dense. For example this technique is used by Gill, Murray, and Saunders [93] in the software SNOPT. Unfortunately, as they state in [94] the method is "...best suited for problems with a moderate number of degrees of freedom (say, up to 2000)." As a practical matter the computational linear algebra costs associated with operations on the $(n_d \times n_d)$ *dense* reduced Hessian $\mathbf{Z}^{\mathsf{T}}\mathbf{H}_L\mathbf{Z}$ become prohibitive, in addition to any computer storage issues. Byrd, Hribar, and Nocedal [59] use a *limited memory update* in the software KNITRO in an attempt to deal with the storage requirements of the dense projected Hessian matrix. Unfortunately like all quasi-Newton methods, limited memory updates do not demonstrate quadratic convergence. Thus each NLP problem requires repeated solution of a large dense linear system. Furthermore, as the mesh size increases, the process must be repeated on another, larger NLP and consequently becomes prohibitively expensive for problems with many degrees of freedom. Since $n_d \sim n_u M$ if a quasi-Newton NLP is used in an SNLP algorithm, the number of controls and/or mesh size are severely limited. In general a quasi-Newton algorithm could not exhibit the behavior illustrated in Figure 4.31. Generally, algorithms that fully exploit Hessian sparsity have demonstrated superior computational performance for large-scale optimization. Full Hessian sparsity is exploited by the SQP and barrier methods in SOCS as well as the LOQO algorithm of Vanderbei and Shanno [166].

Answer: Newton.

Question: Barrier or SQP Algorithm?

A second, more serious performance issue occurs when comparing NLP algorithms for use within the context of an SNLP. Is an SQP better than a barrier method for sequential nonlinear programming? An answer can be deduced by examining the algorithm behavior on a collection of optimal control test problems. Two algorithms were compared: the Schur-complement sparse SQP, and the primal-dual barrier method described in Chapter 2. The test suite consists of 99 problems drawn from many different applications, including every problem in this book. All problems were solved on the same computer, using the

same compiler and operating system. All problems were solved to the same discretization accuracy $\epsilon_\alpha \leq \delta = 10^{-7}$ given by (4.171), using the same mesh-refinement procedure. Both the SQP and the barrier algorithm utilize gradient and Hessian information constructed using the sparse finite difference techniques discussed, with an NLP convergence tolerance of $\sqrt{\epsilon_m}$, where ϵ_m is the machine precision. Finally, the same initial guess was used to initiate the SNLP, with no difference in problem scaling.

Table 4.8 summarizes the results of the numerical comparison. All 99 problems were solved correctly when the SQP algorithm was used. In contrast the barrier algorithm solved only 83 (84%) of the problems. Using the SQP results as a reference, the table presents a number of comparisons. The average increase in run time for the barrier algorithm was 106.24%, and the median increase was 30.39%. Of the 83 problems successfully solved, the worst case was 1554.41% more expensive than the SQP, and the best case was 64.77% better than the SQP. Consistent trends are observed when comparing the number of function evaluations.

Table 4.8. *Percentage change for barrier versus SQP.*

83 of 99 Solved	Average	Median	Max	Min
Run Time Change (%)	106.24	30.39	1554.41	−64.77
Function Evaluations (%)	204.07	170.25	874.38	−49.74

There are many factors that affect the behavior of a barrier algorithm for solving a single NLP.

1. The barrier transformation embodied in (2.89) is nonlinear. As such, one might expect the iterations to be more "nonlinear" and therefore slower, when compared to an SQP.

2. When posing a general NLP in barrier format the number of variables (2.74) and constraints (2.75), (2.76) is larger than the SQP formulation. This may increase the cost of the linear algebra.

3. Problem scaling may interact with the nonlinearity of the barrier transformation.

While these factors are applicable to the solution of a single NLP, far more serious difficulties are encountered when many NLP problems must be solved within the context of an SNLP. Typically the NLP problem size grows as the discretization mesh is refined. In this context it is required that one efficiently solve a *sequence* of NLPs. The obvious way to achieve this goal is to use coarse grid information to "hot start" the fine grid NLP. In fact one can use high-order polynomial interpolation of the coarse grid solution to construct a very good initial guess for the NLP problem that must be solved on a fine grid. Thus, as the mesh is refined the initial guess for the NLP subproblem becomes better and better. An SQP algorithm can exploit this. In contrast for an interior-point algorithm, the iterates must be *strictly feasible*. Constructing a feasible initial iterate by perturbing the user-supplied initial guess is a straightforward process performed by computational software as described in Section 2.11.9. However, in so doing, the very first iterate for each barrier NLP is inconsistent with the coarse grid interpolation. Furthermore, an initial estimate must be supplied for the barrier parameter μ in (2.89), and a poor choice can dramatically degrade the performance of a barrier algorithm. In short, a *barrier algorithm cannot exploit a*

good guess! This fundamental shortcoming of an interior-point method is discussed by Forsgren [86].

The SNLP process presents one final difficulty for any NLP algorithm (barrier or SQP). Each NLP subproblem is nonlinear, and computational algorithms are designed to yield local, but not global, solutions. As the mesh is refined, in principle, the sequence of NLP subproblems should not change regardless of what NLP algorithm is used. However, computational experience suggests otherwise—as illustrated by the results summarized in Tables 7.2 and 7.3. In particular for some problems in the test suite, the barrier algorithm found different local solutions than the SQP, ultimately leading to failure. To date, this behavior has been observed only with the barrier NLP, and not when using an SQP.

Answer: SQP.

4.14 What Can Go Wrong

In Section 4.3, we spent some time discussing what's wrong with the indirect method. We are now in a position to consider *what's good about the direct method*.

1. Since the adjoint equations are not formed explicitly, analytic derivatives are not required. Instead, equivalent information can be computed using sparse finite differences. Consequently, a user with minimal knowledge of optimal control theory can use the technique! Complex black box applications can be handled easily. The approach is flexible and new formulations are handled readily. Finally, sparsity in the right-hand side of the dynamic equations (4.32) and (4.33) can be treated automatically.

2. Path inequalities (4.33) do not require an a priori estimate of the constrained-arc sequence because the NLP active set procedure automatically determines the arc sequence.

3. The method is very robust since the user must guess only the problem variables $\mathbf{y}, \mathbf{u}$. Furthermore, the NLP globalization strategy, which is designed to improve a *merit function*, has a much larger region of convergence than finding a root of the gradient of the Lagrangian, $\nabla L = 0$, which is the approach used by an indirect method.

For most applications, the direct method is quite powerful and eliminates the deficiencies of an indirect approach. Nevertheless, there are some situations that are problematic, and it is worthwhile to address them.

4.14.1 Singular Arcs

Normally, the optimal control function is uniquely defined by the optimality condition (4.11). On the other hand, a *singular arc* is characterized by having both $\mathbf{H}_u = \mathbf{0}$ *and* a singular matrix $\mathbf{H}_{uu}$. When this situation appears, the corresponding sparse NLP problem that arises as a part of the direct transcription method is singular. In particular, the projected Hessian matrix is not positive definite. The standard remedial action used by most NLP algorithms is to modify the Hessian matrix used within the NLP problem, for example, using the Levenberg parameter (2.39) described in Section 2.5. While this approach does "fix" the NLP subproblem, it does not address the real difficulty. Conversely, one can attempt

to correct the real problem by imposing the necessary condition appropriate for a singular arc, namely

$$\frac{d^2}{dt^2}(\mathbf{H}_u) = \mathbf{0}. \tag{4.365}$$

If this mixed approach is used with general-purpose software such as $\mathbb{SOCS}$, it is possible to obtain the correct solution. However, this hybrid technique has all of the drawbacks of an indirect method since the analytic necessary conditions must be derived and the arc sequence guessed. Finally, the proper formulation may avoid the appearance of a singular arc as illustrated in Example 6.10.

Example 4.9 GODDARD ROCKET PROBLEM. To illustrate this situation, consider a simple version of the classical Goddard rocket problem [54]:

$$\begin{aligned} \dot{h} &= v, \\ \dot{v} &= \frac{1}{m}\left[T(t) - \sigma v^2 \exp[-h/h_0]\right] - g, \\ \dot{m} &= -T(t)/c. \end{aligned} \tag{4.366}$$

It is required to find the thrust history $0 \le T(t) \le T_m$ such that the final altitude $h(t_F)$ is maximized for given initial conditions on altitude h, velocity v, and mass m. The problem definition is completed by the following parameters: $T_m = 200$, $g = 32.174$, $\sigma = 5.4915 \times 10^{-5}$, $c = 1580.9425$, $h_0 = 23800.$, $h(0) = v(0) = 0$, and $m(0) = 3$.

It can be demonstrated that the optimal solution consists of three subarcs—the first at maximum thrust $T(t) = T_m$, followed by a singular arc, with a final arc at minimum thrust $T(t) = 0$. On the singular arc, application of (4.365) leads to the nonlinear algebraic constraint

$$\begin{aligned} 0 = {} & T(t) - \sigma v^2 \exp[-h/h_0] - mg \\ & - \frac{mg}{1 + 4(c/v) + 2(c^2/v^2)}\left[\frac{c^2}{h_0 g}\left(1 + \frac{v}{c}\right) - 1 - 2\frac{c}{v}\right], \end{aligned} \tag{4.367}$$

which uniquely defines the control $T(t)$. Thus, we can formulate the application as a three-phase problem. All three phases must satisfy the differential equations (4.366); however, in phase 2 it is also necessary to satisfy the (index-one) path constraint (4.367). Furthermore, when entering the singular arc at the beginning of phase 2 it is necessary to impose the boundary condition

$$0 = mg - \left(1 + \frac{v}{c}\right)\sigma v^2 \exp[-h/h_0]. \tag{4.368}$$

The final times for all three phases are also introduced as NLP variables in the transcribed formulation. By incorporating knowledge of the singular arc, the $\mathbb{SOCS}$ software can efficiently and accurately compute the solution. Table 4.9 summarizes the computational performance, and the solution is illustrated by the shaded region in Figure 4.34. Notice that three mesh-refinement iterations were required to reduce the error in the differential equations (ERRODE) by increasing the number of grid points M from 60 to 91. Furthermore, the solution to each NLP subproblem was computed efficiently in terms of the number of function evaluations (NFE) and the CPU time.

Table 4.9. *Singular arc using hybrid method.*

Iter.	M	NFE	ERRODE	CPU (sec)
1	60	30	0.67×10^{-4}	0.15×10^{1}
2	79	26	0.55×10^{-5}	0.12×10^{1}
3	91	14	0.70×10^{-7}	0.96×10^{0}
		70		3.64

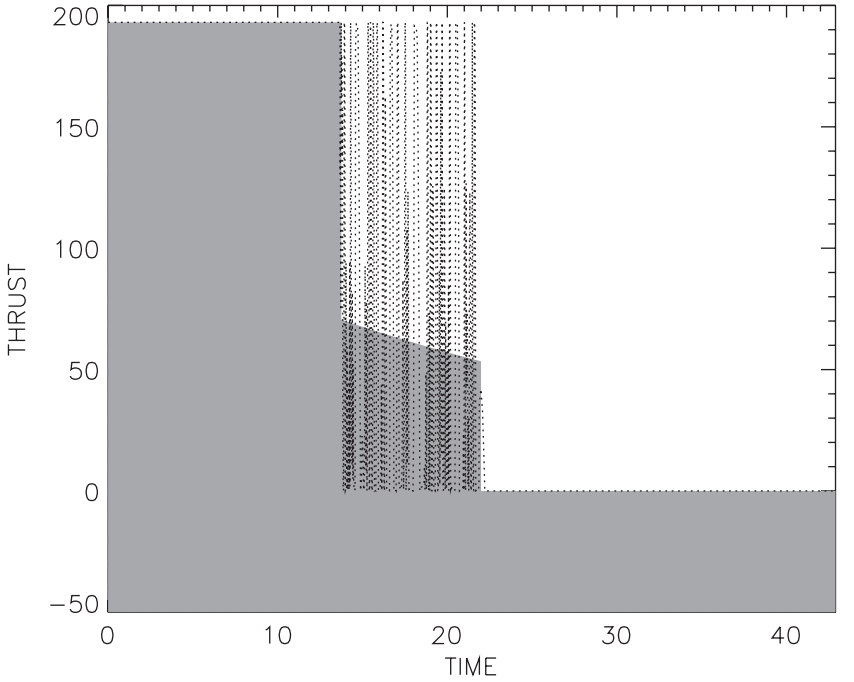


Figure 4.34. *Singular arc solution.*

In contrast, when no knowledge of the singular arc is incorporated and a standard one-phase approach is used, the computational performance is much less efficient, as indicated by the results in Table 4.10. Furthermore, even though the final solution had 450 grid points, the control history during the singular arc is quite oscillatory, as illustrated by the dotted line in Figure 4.34. The oscillatory control is, of course, a direct consequence of the fact that the control is not uniquely determined on the singular arc unless the higher-order conditions (4.365) are imposed. The mesh refinement attempts to correct the perceived inaccuracy on the singular arc by adding grid points to this region. It is interesting to note, however, that the computed optimal objective function value $h^* = 18550.6$ is quite close to the true value computed using the hybrid approach ($h^* = 18550.9$).

Table 4.10. *Singular arc using standard method.*

Iter.	M	NFE	ERRODE	CPU (sec)
1	50	189	0.52×10^{-3}	0.46×10^{1}
2	96	416	0.72×10^{-3}	0.17×10^{2}
3	103	2078	0.13×10^{-3}	0.58×10^{2}
4	188	3398	0.64×10^{-4}	0.34×10^{3}
5	375	230	0.26×10^{-6}	0.19×10^{3}
6	450	230	0.84×10^{-7}	0.17×10^{3}
		6541		775.30

4.14.2 State Constraints

A standard approach to a problem with path constraint(s)

$$\mathbf{0} \leq \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]$$

is well behaved provided the matrix $\mathbf{g}_u$ is full rank. However, for *state constraints* $\mathbf{g}[\mathbf{y}, \mathbf{u}, t] \equiv \mathbf{g}[\mathbf{y}, t]$, it is clear that the matrix $\mathbf{g}_u$ is *not* full rank. In this case, after discretization, the sparse NLP Jacobian matrix does not have full row rank and, consequently, violates the NLP *constraint-qualification test*! Furthermore, numerically detecting the rank of a matrix is difficult.

On the other hand, if the analyst recognizes this situation, it is possible to use *index reduction* to construct a well-posed discretized subproblem. Essentially the user must *analytically* differentiate $\mathbf{g}$ and then substitute the state equations into the resulting expression. This process can be repeated q times until the control $\mathbf{u}$ appears explicitly. Then, during the constrained arcs, one can impose the derived conditions, e.g.,

$$\frac{d^q}{dt^q}(\mathbf{g}) = \mathbf{0}.$$

Additional endpoint conditions

$$\frac{d^{q-k}}{dt^{q-k}}(\mathbf{g})\,|_{t=t_a} = \mathbf{0}$$

for $k = 1, \ldots, q$ must be imposed at one end of the constrained arc (where $t = t_a$). For indirect formulations, the adjoint variables also require additional "jump conditions" at the end of the arc. Although this technique can be used to obtain the correct solution, unfortunately it is not "automatic" and, as such, has all of the drawbacks of the indirect method as discussed in Section 4.12.

Example 4.10 BRACHISTOCHRONE. To illustrate the situation, consider the classical Brachistochrone problem with a state variable inequality constraint:

Minimize t_F subject to

$$\begin{aligned} \dot{x} &= v \cos u, \\ \dot{y} &= v \sin u, \\ \dot{v} &= g \sin u, \\ 0 &\geq y - x/2 - h \end{aligned}$$

with boundary conditions $x_0 = y_0 = v_0 = 0$ and $x_F = 1$. Now for $h = 0.1$, the behavior of the standard approach summarized in Table 4.11 efficiently and accurately computes the solution illustrated in Figure 4.35. Unfortunately, using the same technique with $h = 0$ will fail because the initial conditions $x_0 = 0$, $y_0 = 0$, together with the path constraint $y_0 - x_0/2 = 0$, constitute a rank-deficient set of constraints, and the Jacobian is singular!

Table 4.11. *State-constrained solution using standard method.*

Iter.	M	NFE	ERRODE	CPU
1	10	149	0.16×10^{-2}	0.13×10^{1}
2	19	35	0.19×10^{-3}	0.54×10^{0}
3	21	140	0.13×10^{-4}	0.10×10^{1}
4	41	119	0.55×10^{-6}	0.21×10^{1}
5	69	119	0.53×10^{-7}	0.72×10^{1}
		562		12.16

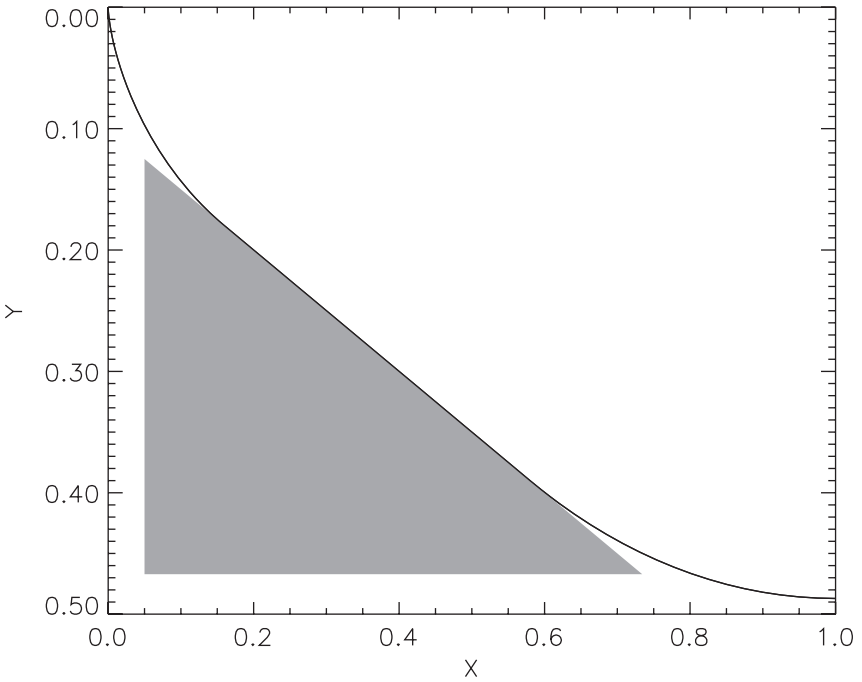


Figure 4.35. *State-constrained solution.*

4.14.3 Discontinuous Control

Example 4.11 BANG-BANG CONTROL. As a final example of the limitations present in general-purpose software such as SOCS, it is worth reviewing the behavior on a problem

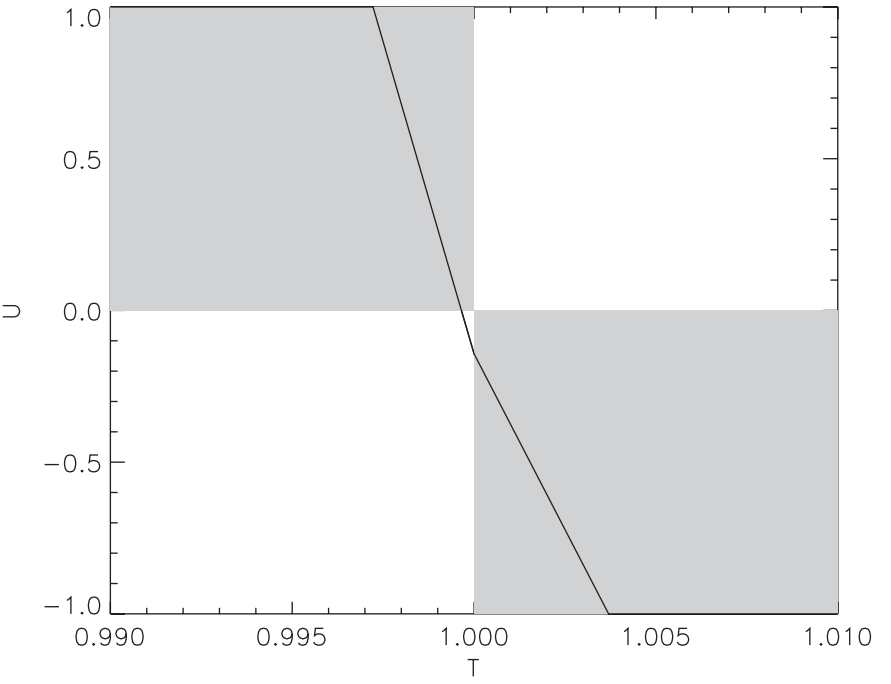


Figure 4.36. *Bang-bang control approximation.*

with discontinuous optimal control. This situation can occur whenever the differential equations are linear, and is readily illustrated with the solution to the following problem: Minimize t_F subject to

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= u, \\ -1 \quad &\leq u \leq \quad 1 \end{aligned}$$

with boundary conditions $x_0 = y_0 = 0$, $x_F = 1$, and $y_F = 0$. The exact optimal solution is bang-bang with

$$u^*(t) = \begin{cases} 1 & \text{if } t < 1, \\ -1 & \text{if } t > 1. \end{cases}$$

The continuous control function $u(t)$ approximation to this solution is illustrated in Figure 4.36. While this is a reasonably good approximation (notice the independent variable scale is magnified), it nevertheless is still an approximation. Furthermore, an alternative parameterization of the control that permits discontinuities would readily construct the exact answer. Unfortunately, this again is not "automatic" and, as such, represents a limitation in the current methodology.