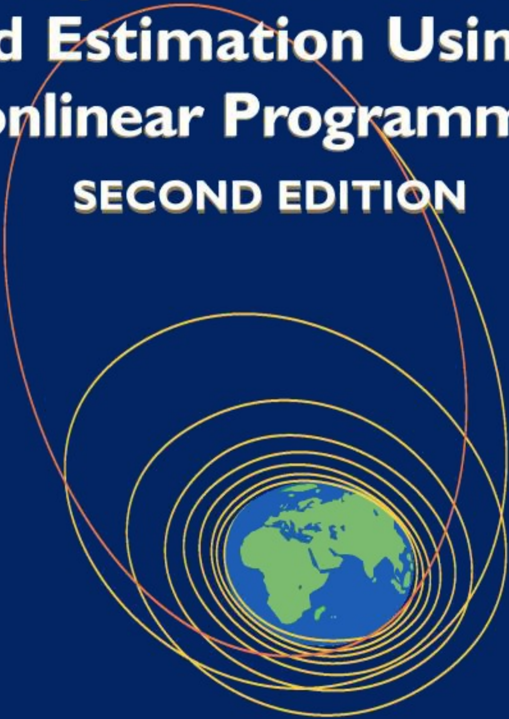
# Practical Methods for Optimal Control and Estimation Using Nonlinear Programming

SECOND EDITION



John T. Betts

Advances in Design and Control

siam

# Practical Methods for Optimal Control and Estimation Using Nonlinear Programming

# Practical Methods for Optimal Control and Estimation Using Nonlinear Programming

SECOND EDITION

John T. Betts

Society for Industrial and Applied Mathematics
Philadelphia

**Library of Congress Cataloging-in-Publication Data**
Betts, John T. 1943-
Practical methods for optimal control and estimation using nonlinear programming / John T. Betts. — 2nd ed.
p. cm. — (Advances in design and control)
Includes bibliographical references and index.
ISBN 978-0-898716-88-7
1. Control theory. 2. Mathematical optimization. 3. Nonlinear programming. I. Title.
QA402.3.B47 2009
629.8'312—dc22

2009025106

For Theon and Dorothy

He Inspired Creativity
She Cherished Education

# Contents

# Preface

Solving an optimal control or estimation problem is not easy. Pieces of the puzzle are found scattered throughout many different disciplines. Furthermore, the focus of this book is on *practical methods*, that is, methods that I have found actually work! In fact everything described in this book has been implemented in production software and used to solve real optimal control problems. Although the reader should be proficient in advanced mathematics, no theorems are presented.

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.

The first two chapters of this book focus on the optimization part of the problem. In Chapter 1 the important concepts of nonlinear programming for small dense applications are introduced. Chapter 2 extends the presentation to problems which are both large and sparse. Chapters 3 and 4 address the differential equation part of the problem. Chapter 3 introduces relevant material in the numerical solution of differential (and differential-algebraic) equations. Methods for solving the optimal control problem are treated in some detail in Chapter 4. Throughout the book the interaction between optimization and integration is emphasized. Chapter 5 describes how to solve optimal estimation problems. Chapter 6 presents a collection of examples that illustrate the various concepts and techniques. Real world problems often require solving a sequence of optimal control and/or optimization problems, and Chapter 7 describes a collection of these "advanced applications."

While the book incorporates a great deal of new material not covered in *Practical Methods for Optimal Control Using Nonlinear Programming* [21], it does not cover everything. Many important topics are simply not discussed in order to keep the overall presentation concise and focused. The discussion is general and presents a unified approach to solving optimal estimation and control problems. Most of the examples are drawn from my experience in the aerospace industry. Examples have been solved using a particular implementation called $\mathbb{SOCS}$. I have tried to adhere to notational conventions from both optimization and control theory whenever possible. Also, I have attempted to use consistent notation throughout the book.

The material presented here represents the collective contributions of many people. The nonlinear programming material draws heavily on the work of John Dennis, Roger Fletcher, Phillip Gill, Sven Leyffer, Walter Murray, Michael Saunders, and Margaret Wright. The material on differential-algebraic equations (DAEs) is drawn from the work of Uri Ascher, Kathy Brenan, and Linda Petzold. Ray Spiteri graciously shared his classroom notes on DAEs. I was introduced to optimal control by Stephen Citron, and I routinely refer to the text by Bryson and Ho [54]. Over the past 20 years I have been fortunate to participate in workshops at Oberwolfach, Munich, Minneapolis, Victoria, Banff, Lausanne, Griefswald, Stockholm, and Fraser Island. I've benefited immensely simply by talking with Larry Biegler, Hans Georg Bock, Roland Bulirsch, Rainer Callies, Kurt Chudej, Tim Kelley, Bernd Kugelmann, Helmut Maurer, Rainer Mehlhorn, Angelo Miele, Hans Josef Pesch, Ekkehard Sachs, Gottfried Sachs, Roger Sargent, Volker Schulz, Mark Steinbach, Oskar von Stryk, and Klaus Well.

Three colleagues deserve special thanks. Interaction with Steve Campbell and his students has inspired many new results and interesting topics. Paul Frank has played a major role in the implementation and testing of the large, sparse nonlinear programming methods described. Bill Huffman, my coauthor for many publications and the $\mathbb{SOCS}$ software, has been an invaluable sounding board over the last two decades. Finally, I thank Jennifer for her patience and understanding during the preparation of this book.

*John T. Betts*

Chapter 1

# Introduction to Nonlinear Programming

## 1.1 Preliminaries

This book concentrates on numerical methods for solving the optimal control problem. The fundamental principle of all effective numerical optimization methods is to solve a difficult problem by solving a sequence of simpler subproblems. In particular, the solution of an optimal control problem will require the solution of one or more finite-dimensional subproblems. As a prelude to our discussions on optimal control, this chapter will focus on the nonlinear programming (NLP) problem. The NLP problem requires finding a finite number of variables such that an *objective function* or *performance index* is optimized without violating a set of *constraints*. The NLP problem is often referred to as *parameter optimization*. Important special cases of the NLP problem include *linear programming* (LP), *quadratic programming* (QP), and *least squares* problems.

Before proceeding further, it is worthwhile to establish the notational conventions used throughout the book. This is especially important since the subject matter covers a number of different disciplines, each with its own notational conventions. Our goal is to present a unified treatment of all these fields. As a rule, scalar quantities will be denoted by lowercase letters (e.g., $\alpha$). Vectors will be denoted by boldface lowercase letters and will usually be considered column vectors, as in

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \tag{1.1}$$

where the individual components of the vector are $x_k$ for $k = 1, \ldots, n$. To save space, it will often be convenient to define the *transpose*, as in

$$\mathbf{x}^\mathsf{T} = (x_1, x_2, \ldots, x_n). \tag{1.2}$$

A *sequence* of vectors will often be denoted as $\mathbf{x}_k, \mathbf{x}_{k+1}, \ldots$. Matrices will be denoted by boldface capital letters, as in

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}. \tag{1.3}$$

## 1.2 Newton's Method in One Variable

The fundamental approach to most iterative schemes was suggested over 300 years ago by Newton. In fact, Newton's method is the basis for all of the algorithms we will describe. We begin with the simplest form of Newton's method and then in subsequent sections generalize the discussion until we have presented one of the most widely used NLP algorithms, namely the *sequential quadratic programming* (SQP) method.

Suppose it is required to find the value of the variable $x$ such that the *constraint* function

$$c(x) = 0. \tag{1.4}$$

Let us denote the solution by $x^*$ and let us assume $x$ is a guess for the solution. The basic idea of Newton's method is to approximate the nonlinear function $c(x)$ by the first two terms in a Taylor series expansion about the current point $x$. This yields a linear approximation for the constraint function at the new point $\bar{x}$, which is given by

$$c(\bar{x}) = c(x) + c'(x)(\bar{x} - x), \tag{1.5}$$

where $c'(x) = dc/dx$ is the slope of the constraint at $x$. Using this linear approximation, it is reasonable to compute $\bar{x}$, a new estimate for the root, by solving (1.5) such that $c(\bar{x}) = 0$, i.e.,

$$\bar{x} = x - [c'(x)]^{-1} c(x). \tag{1.6}$$

Typically, we denote $p \equiv \bar{x} - x$ and rewrite (1.6) as

$$\bar{x} = x + p, \tag{1.7}$$

where

$$p = -[c'(x)]^{-1} c(x). \tag{1.8}$$

Of course, in general, $c(x)$ is not a linear function of $x$, and consequently we cannot expect that $c(\bar{x}) = 0$. However, we might hope that $\bar{x}$ is a better estimate for the root $x^*$ than the original guess $x$; in other words we might expect that

$$|\bar{x} - x^*| \le |x - x^*| \tag{1.9}$$

and

$$|c(\bar{x})| \le |c(x)|. \tag{1.10}$$

If the new point is an improvement, then it makes sense to repeat the process, thereby defining a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \dots$ with point $(k+1)$ in the sequence given by

$$x^{(k+1)} = x^{(k)} - [c'(x^{(k)})]^{-1} c(x^{(k)}). \tag{1.11}$$

For notational convenience, it usually suffices to present a single step of the algorithm, as in (1.6), instead of explicitly labeling the information at step $k$ using the superscript notation $x^{(k)}$. Nevertheless, it should be understood that the algorithm defines a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$. The sequence is said to *converge* to $x^*$ if

$$\lim_{k\to\infty} |x^{(k)} - x^*| = 0. \tag{1.12}$$

In practice, of course, we are not interested in letting $k \to \infty$. Instead we are satisfied with terminating the sequence when the computed solution is "close" to the answer. Furthermore, the *rate of convergence* is of paramount importance when measuring the computational efficiency of an algorithm. For Newton's method, the rate of convergence is said to be *quadratic* or, more precisely, *q-quadratic* (cf. [71]). The impact of quadratic convergence can be dramatic. Loosely speaking, it implies that each successive estimate of the solution will *double* the number of significant digits!

**Example 1.1** NEWTON'S METHOD—ROOT FINDING. To demonstrate, let us suppose we want to solve the constraint

$$c(x) = a_1 + a_2 x + a_3 x^2 = 0, \tag{1.13}$$

where the coefficients $a_1, a_2, a_3$ are chosen such that $c(0.1) = -0.05$, $c(0.25) = 0$, and $c(0.9) = 0.9$. Table 1.1 presents the Newton iteration sequence beginning from the initial guess $x = 0.85$ and proceeding to the solution at $x^* = 0.25$. Figure 1.1 illustrates the first three iterations. Notice in Table 1.1 that the error between the computed solution and the true value, which is tabulated in the third column, exhibits the expected doubling in significant figures from the fourth iteration to convergence.

So what is wrong with Newton's method? Clearly, quadratic convergence is a very desirable property for an algorithm to possess. Unfortunately, if the initial guess is not sufficiently close to the solution, i.e., within the *region of convergence*, Newton's method may diverge. As a simple example, Dennis and Schnabel [71] suggest applying Newton's method to solve $c(x) = \arctan(x) = 0$. This will diverge when the initial guess $|x^{(0)}| > a$, converge when $|x^{(0)}| < a$, and cycle indefinitely if $|x^{(0)}| = a$, where $a = 1.3917452002707$. In essence, Newton's method behaves well near the solution (*locally*) but lacks something permitting it to converge *globally*. So-called globalization techniques, aimed at correcting this deficiency, will be discussed in subsequent sections. A second difficulty occurs when

**Table 1.1.** *Newton's method for root finding.*

| Iter. | $c(x)$ | $x$ | $\|x - x^*\|$ |
|---|---|---|---|
| 1 | 0.79134615384615 | 0.85000000000000 | 0.60000000000000 |
| 2 | 0.18530192382759 | 0.47448669201521 | 0.22448669201521 |
| 3 | $3.5942428588261\times10^{-2}$ | 0.30910437279376 | $5.9104372793756\times10^{-2}$ |
| 4 | $3.6096528286200\times10^{-3}$ | 0.25669389900972 | $6.6938990097217\times10^{-3}$ |
| 5 | $5.7007630268141\times10^{-5}$ | 0.25010744198003 | $1.0744198002549\times10^{-4}$ |
| 6 | $1.5161639596584\times10^{-8}$ | 0.25000002858267 | $2.8582665845267\times10^{-8}$ |
| 7 | $1.0547118733939\times10^{-15}$ | 0.25000000000000 | $1.8873791418628\times10^{-15}$ |

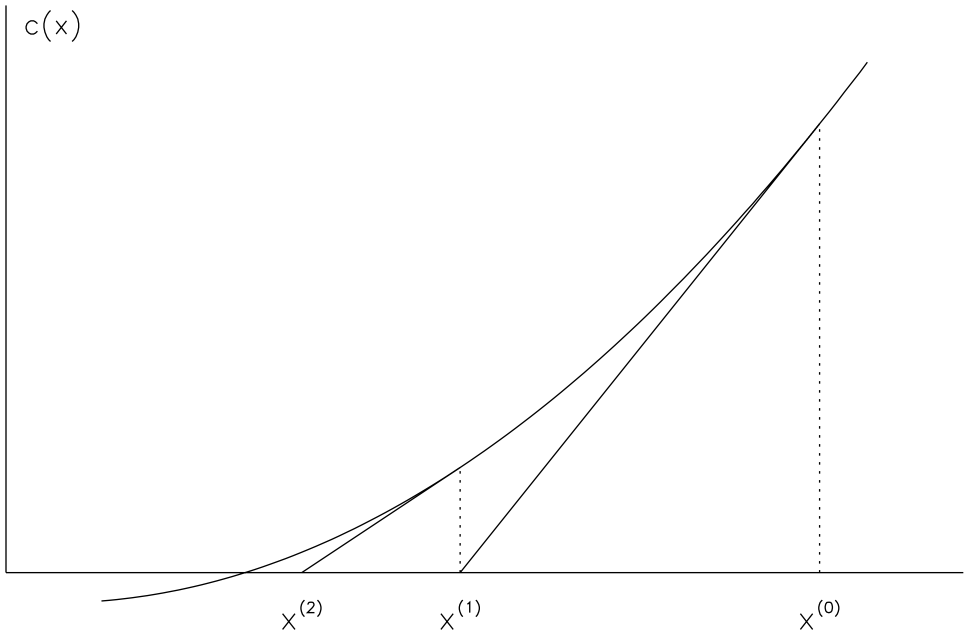


**Figure 1.1.** *Newton's method for root finding.*

the slope $c'(x) = 0$. Clearly, the correction defined by (1.6) is not well defined in this case. In fact, Newton's method loses its quadratic convergence property if the slope is zero at the solution, i.e., $c'(x^*) = 0$. Finally, Newton's method requires that the slope $c'(x)$ can be computed at every iteration. This may be difficult and/or costly, especially when the function $c(x)$ is complicated.

## 1.3 Secant Method in One Variable

Motivated by a desire to eliminate the explicit calculation of the slope, one can consider approximating it at $x^k$ by the secant

$$c'(x^k) \approx B = \frac{c(x^k) - c(x^{k-1})}{x^k - x^{k-1}} \equiv \frac{\Delta c}{\Delta x}. \tag{1.14}$$

Notice that this approximation is constructed using two previous iterations but requires values only for the constraint function $c(x)$. This expression can be rewritten to give the so-called secant condition

$$B\Delta x = \Delta c, \tag{1.15}$$

where $B$ is the (scalar) secant approximation to the slope. Using this approximation, it then follows that the Newton iteration (1.6) is replaced by the secant iteration

$$\bar{x} = x - B^{-1}c(x) = x + p, \tag{1.16}$$

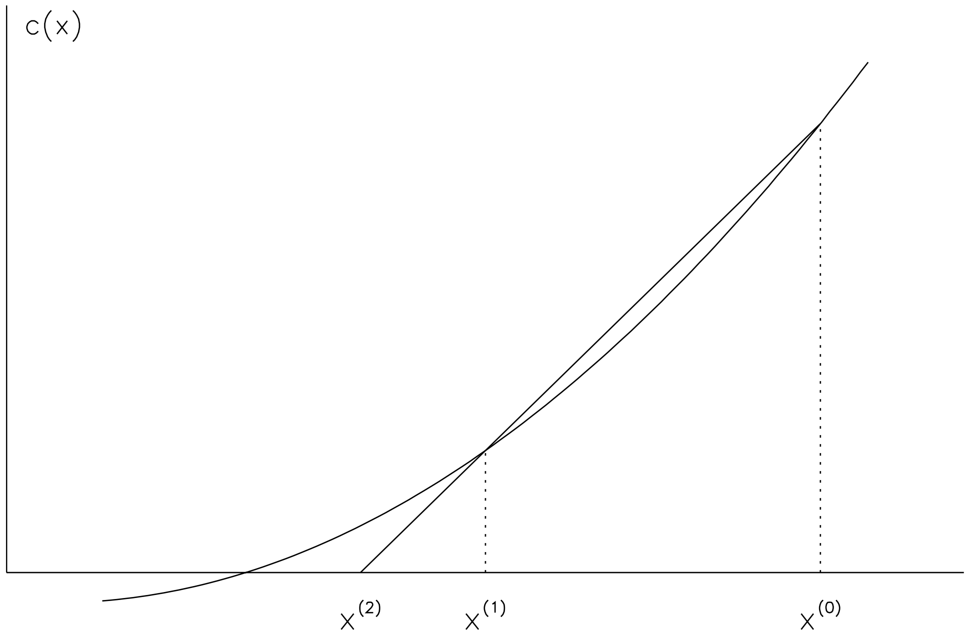


**Figure 1.2.** *Secant method for root finding.*

which is often written as

$$x^{k+1} = x^k - \frac{x^k - x^{k-1}}{c(x^k) - c(x^{k-1})} c(x^k). \tag{1.17}$$

Figure 1.2 illustrates a secant iteration applied to Example 1.1 described in the previous section.

Clearly, the virtue of the secant method is that it does not require calculation of the slope $c'(x^k)$. While this may be advantageous when derivatives are difficult to compute, there is a downside! The secant method is *superlinearly* convergent, which, in general, is not as fast as the quadratically convergent Newton algorithm. Thus, we can expect convergence will require more iterations, even though the cost per iteration is less. A distinguishing feature of the secant method is that the slope is approximated using information from previous iterates in lieu of a direct evaluation. This is the simplest example of a so-called quasi-Newton method.

## 1.4 Newton's Method for Minimization in One Variable

Now let us suppose we want to compute the value $x^*$ such that the nonlinear *objective function* $F(x^*)$ is a minimum. The basic notion of Newton's method for root finding is to approximate the nonlinear constraint function $c(x)$ by a simpler model (i.e., linear) and then compute the root for the linear model. If we are to extend this philosophy to optimization, we must construct an approximate model of the objective function. Just as in the development of (1.5), let us approximate $F(x)$ by the first three terms in a Taylor series expansion about the current point $x$:

$$F(\bar{x}) = F(x) + F'(x)(\bar{x} - x) + \frac{1}{2}(\bar{x} - x)F''(x)(\bar{x} - x). \tag{1.18}$$

Notice that we cannot use a linear model for the objective because a linear function does not have a finite minimum point. In contrast, a quadratic approximation to $F(x)$ is the simplest approximation that does have a minimum. Now for $\bar{x}$ to be a minimum of the quadratic (1.18), we must have

$$\frac{dF}{d\bar{x}} \equiv F'(\bar{x}) = 0 = F'(x) + F''(x)(\bar{x} - x). \tag{1.19}$$

Solving for the new point yields

$$\bar{x} = x - [F''(x)]^{-1} F'(x). \tag{1.20}$$

The derivation has been motivated by minimizing $F(x)$. Is this equivalent to solving the slope condition $F'(x) = 0$? It would appear that the iterative *optimization* sequence defined by (1.20) is the same as the iterative *root-finding* sequence defined by (1.6), provided we replace $c(x)$ by $F'(x)$. Clearly, a quadratic model for the objective function (1.18) produces a linear model for the slope $F'(x)$. However, the condition $F'(x) = 0$ defines only a *stationary point*, which can be a minimum, a maximum, or a point of inflection. Apparently what is missing is information about the *curvature* of the function, which would determine whether it is concave up, concave down, or neither.

Figure 1.3 illustrates a typical situation. In the illustration, there are two points with zero slopes; however, there is only one minimum point. The minimum point is dis-

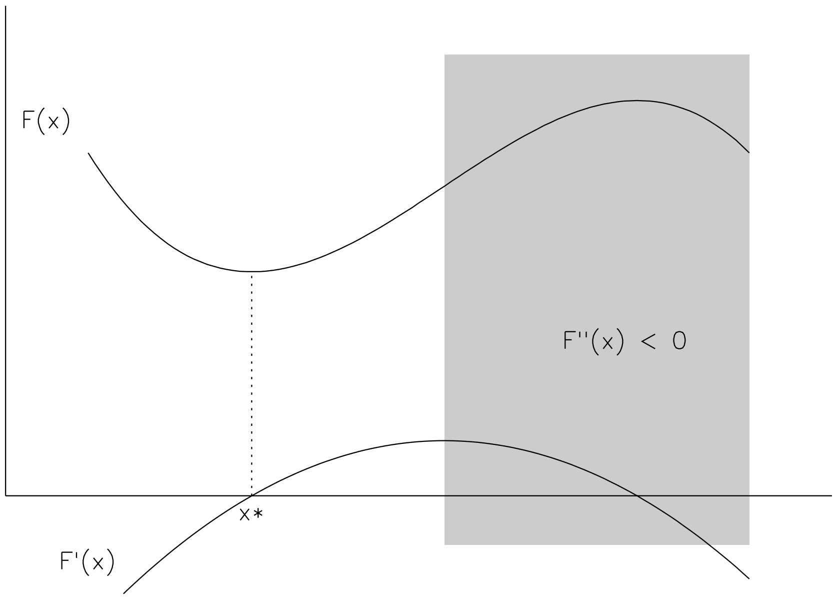


**Figure 1.3.** *Minimization in one variable.*

tinguished from the maximum by the algebraic sign of the second derivative $F''(x)$. Formally, we have

*Necessary Conditions:*

$$F'(x^*) = 0, \tag{1.21}$$

$$F''(x^*) \geq 0; \tag{1.22}$$

*Sufficient Conditions:*

$$F'(x^*) = 0, \tag{1.23}$$

$$F''(x^*) > 0. \tag{1.24}$$

Note that the sufficient conditions require that $F''(x^*) > 0$, defining a *strong local minimizer* in contrast to a *weak local minimizer*, which may have $F''(x^*) = 0$. It is also important to observe that these conditions define a *local* rather than a *global* minimizer.

## 1.5 Newton's Method in Several Variables

The preceding sections have addressed problems involving a single variable. In this section, let us consider generalizing the discussion to functions of many variables. In particular, let us consider how to find the $n$-vector $\mathbf{x}^\mathsf{T} = (x_1, \ldots, x_n)$ such that

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} c_1(\mathbf{x}) \\ \vdots \\ c_m(\mathbf{x}) \end{bmatrix} = \mathbf{0}. \tag{1.25}$$

For the present, let us assume that the number of constraints and variables is the same, i.e., $m = n$. Just as in one variable, a linear approximation to the constraint functions analogous to (1.5) is given by

$$\mathbf{c}(\overline{\mathbf{x}}) = \mathbf{c}(\mathbf{x}) + \mathbf{G}(\overline{\mathbf{x}} - \mathbf{x}), \tag{1.26}$$

where the *Jacobian matrix* $\mathbf{G}$ is defined by

$$\mathbf{G} \equiv \frac{\partial \mathbf{c}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_1}{\partial x_2} & \cdots & \frac{\partial c_1}{\partial x_n} \\ \frac{\partial c_2}{\partial x_1} & \frac{\partial c_2}{\partial x_2} & \cdots & \frac{\partial c_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial c_m}{\partial x_1} & \frac{\partial c_m}{\partial x_2} & \cdots & \frac{\partial c_m}{\partial x_n} \end{bmatrix}. \tag{1.27}$$

By convention, the $m$ rows of $\mathbf{G}$ correspond to constraints and the $n$ columns to variables. As in one variable, if we require that $\mathbf{c}(\overline{\mathbf{x}}) = \mathbf{0}$ in (1.26), we can solve the linear system

$$\mathbf{G}\mathbf{p} = -\mathbf{c} \tag{1.28}$$

for the *search direction* $\mathbf{p}$, which leads to an iteration of the form

$$\overline{\mathbf{x}} = \mathbf{x} + \mathbf{p}. \tag{1.29}$$

Thus, each Newton iteration requires a linear approximation to the nonlinear constraints $\mathbf{c}$, followed by a step from $\mathbf{x}$ to the solution of the linearized constraints at $\overline{\mathbf{x}}$. Figure 1.4 illustrates a typical situation when $n = m = 2$. It is important to remark that the multidimensional version of Newton's method shares all of the properties of its one-dimensional counterpart. Specifically, the method is quadratically convergent provided it is within a region of convergence, and it may diverge unless appropriate globalization strategies are employed. Furthermore, in order to solve (1.28) it is necessary that the Jacobian $\mathbf{G}$ be *nonsingular*, which is analogous to requiring that $c'(x) \neq 0$ in the univariate case. And, finally, it is necessary to actually compute $\mathbf{G}$, which can be costly.

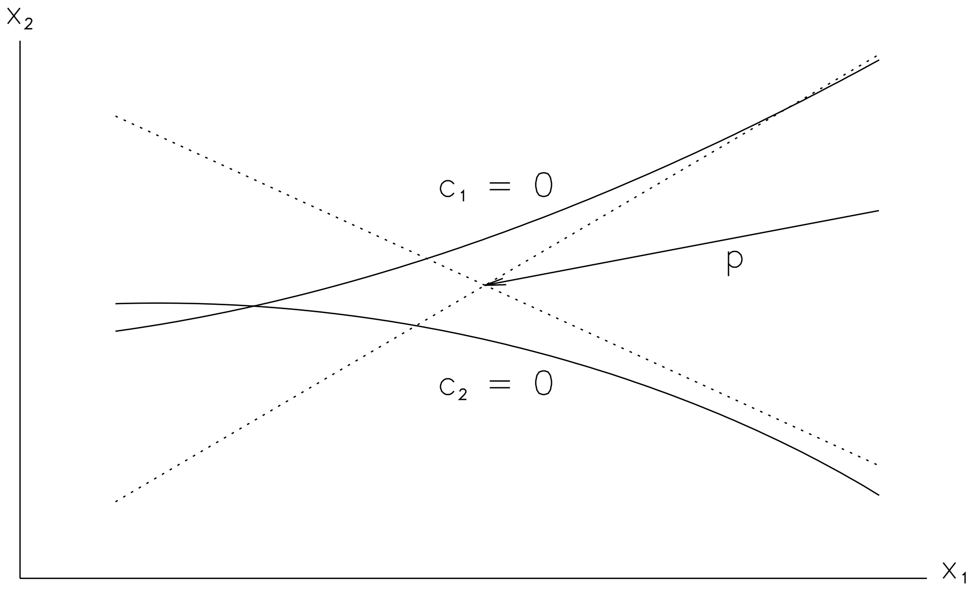


**Figure 1.4.** *Newton's method in two variables.*

## 1.6 Unconstrained Optimization

Let us now consider the multidimensional unconstrained minimization problem. Suppose we want to find the $n$-vector $\mathbf{x}^\mathsf{T} = (x_1, \ldots, x_n)$ such that the function $F(\mathbf{x}) = F(x_1, \ldots, x_n)$ is a minimum. Just as in the univariate case (1.18), let us approximate $F(\mathbf{x})$ by the first three terms in a Taylor series expansion about the point $\mathbf{x}$:

$$F(\overline{\mathbf{x}}) = F(\mathbf{x}) + \mathbf{g}^\mathsf{T}(\mathbf{x})(\overline{\mathbf{x}} - \mathbf{x}) + \frac{1}{2}(\overline{\mathbf{x}} - \mathbf{x})^\mathsf{T}\mathbf{H}(\mathbf{x})(\overline{\mathbf{x}} - \mathbf{x}). \tag{1.30}$$

The Taylor series expansion involves the $n$-dimensional *gradient* vector

$$\mathbf{g}(\mathbf{x}) \equiv \nabla_x F = \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{bmatrix} \tag{1.31}$$

and the symmetric $n \times n$ *Hessian matrix*

$$\mathbf{H} \equiv \nabla^2_{xx} F = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2^2} & \cdots & \frac{\partial^2 F}{\partial x_2 \partial x_n} \\ \vdots & & & \\ \frac{\partial^2 F}{\partial x_n \partial x_1} & \frac{\partial^2 F}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_n^2} \end{bmatrix}. \tag{1.32}$$

It is common to define the *search direction* $\mathbf{p} = \overline{\mathbf{x}} - \mathbf{x}$ and then rewrite (1.30) as

$$F(\overline{\mathbf{x}}) = F(\mathbf{x}) + \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}. \tag{1.33}$$

The scalar term $\mathbf{g}^{\mathsf{T}}\mathbf{p}$ is referred to as the *directional derivative* along $\mathbf{p}$ and the scalar term $\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}$ is called the *curvature* or *second directional derivative* in the direction $\mathbf{p}$.

It is instructive to examine the behavior of the series (1.33). First, let us suppose that the expansion is about the minimum point $\mathbf{x}^*$. Now if $\mathbf{x}^*$ is a local minimum, then the objective function must be larger at all neighboring points, that is, $F(\overline{\mathbf{x}}) > F(\mathbf{x}^*)$. In order for this to be true, the slope in all directions must be zero, that is, $(\mathbf{g}^*)^{\mathsf{T}}\mathbf{p} = \mathbf{0}$, which implies we must have

$$\mathbf{g}(\mathbf{x}^*) = \begin{bmatrix} g_1(\mathbf{x}^*) \\ \vdots \\ g_n(\mathbf{x}^*) \end{bmatrix} = \mathbf{0}. \tag{1.34}$$

This is just the multidimensional analogue of the condition (1.21). Furthermore, if the function curves up in all directions, the point $\mathbf{x}^*$ is called a strong local minimum and the third term in the expansion (1.33) must be positive:

$$\mathbf{p}^{\mathsf{T}}\mathbf{H}^*\mathbf{p} > 0. \tag{1.35}$$

A matrix[1] that satisfies this condition is said to be *positive definite*. If there are some directions with zero curvature, i.e., $\mathbf{p}^{\mathsf{T}}\mathbf{H}^*\mathbf{p} \geq 0$, then $\mathbf{H}^*$ is said to be *positive semidefinite*. If there are directions with both positive and negative curvature, the matrix is called *indefinite*. In summary, we have

[1] $\mathbf{H}^* \equiv \mathbf{H}(\mathbf{x}^*)$ (not the conjugate transpose, as in some texts).

*Necessary Conditions:*

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{0}, \tag{1.36}$$
$$\mathbf{p}^\mathsf{T}\mathbf{H}^*\mathbf{p} \geq 0; \tag{1.37}$$

*Sufficient Conditions:*

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{0}, \tag{1.38}$$
$$\mathbf{p}^\mathsf{T}\mathbf{H}^*\mathbf{p} > 0. \tag{1.39}$$

The preceding discussion was motivated by an examination of the Taylor series about the minimum point $\mathbf{x}^*$. Let us now consider the same quadratic model about an arbitrary point $\mathbf{x}$. Then it makes sense to choose a new point $\overline{\mathbf{x}}$ such that the gradient at $\overline{\mathbf{x}}$ is zero. The resulting linear approximation to the gradient is just

$$\overline{\mathbf{g}} = \mathbf{0} = \mathbf{g} + \mathbf{H}\mathbf{p}, \tag{1.40}$$

which can be solved to yield the Newton search direction

$$\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}. \tag{1.41}$$

Just as before, the Newton iteration is defined by (1.29). Since this iteration is based on finding a zero of the gradient vector, there is no guarantee that the step will move toward a local minimum rather than a stationary point or maximum. To preclude this, we must insist that the step be downhill, which requires satisfying the so-called descent condition

$$\mathbf{g}^\mathsf{T}\mathbf{p} < 0. \tag{1.42}$$

It is interesting to note that, if we use the Newton direction (1.41), the descent condition becomes

$$\mathbf{g}^\mathsf{T}\mathbf{p} = -\mathbf{g}^\mathsf{T}\mathbf{H}^{-1}\mathbf{g} < 0, \tag{1.43}$$

which can be true only if the Hessian is positive definite, i.e., (1.35) holds.

## 1.7 Recursive Updates

Regardless of whether Newton's method is used for solving nonlinear equations, as in Section 1.5, or for optimization, as described in Section 1.6, it is necessary to compute derivative information. In particular, one must compute either the Jacobian matrix (1.27) or the Hessian matrix (1.32). For many applications, this can be a costly computational burden. Quasi-Newton methods attempt to construct this information recursively. A brief overview of the most important recursive updates is included, although a more complete discussion can be found in [71], [99], and [82].

The basic idea of a recursive update is to construct a new estimate of the Jacobian or Hessian using information from previous iterates. Most well-known recursive updates are of the form

$$\overline{\mathbf{B}} = \mathbf{B} + \mathcal{R}(\Delta\mathbf{c}, \Delta\mathbf{x}), \tag{1.44}$$

where the new estimate $\overline{\mathbf{B}}$ is computed from the old estimate $\mathbf{B}$. Typically, this calculation involves a low-rank modification $\mathcal{R}(\Delta\mathbf{c}, \Delta\mathbf{x})$ that can be computed from the previous step:

$$\Delta\mathbf{c} = \mathbf{c}^k - \mathbf{c}^{k-1}, \tag{1.45}$$

$$\Delta\mathbf{x} = \mathbf{x}^k - \mathbf{x}^{k-1}. \tag{1.46}$$

The usual way to construct the update is to insist that the secant condition

$$\overline{\mathbf{B}}\Delta\mathbf{x} = \Delta\mathbf{c} \tag{1.47}$$

hold and then construct an approximation $\overline{\mathbf{B}}$ that is "close" to the previous estimate $\mathbf{B}$. In Section 1.3, the simplest form of this condition (1.15) led to the secant method. In fact, the generalization of this formula, proposed in 1965 by Broyden [50], is

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{c} - \mathbf{B}\Delta\mathbf{x})(\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}, \tag{1.48}$$

which is referred to as the *secant* or *Broyden update*. The recursive formula constructs a rank-one modification that satisfies the secant condition and minimizes the Frobenius norm between the estimates.

When a quasi-Newton method is used to approximate the Hessian matrix, as required for minimization, one cannot simply replace $\Delta\mathbf{c}$ with $\Delta\mathbf{g}$ in the secant update. In particular, the matrix $\overline{\mathbf{B}}$ constructed using (1.48) is not symmetric. However, there is a rank-one update that does maintain symmetry, known as the *symmetric rank-one* (SR1) update:

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}, \tag{1.49}$$

where $\Delta\mathbf{g} \equiv \mathbf{g}^k - \mathbf{g}^{k-1}$. While the SR1 update does preserve symmetry, it does not necessarily maintain a positive definite approximation. In contrast, the update

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{\Delta\mathbf{g}(\Delta\mathbf{g})^{\mathsf{T}}}{(\Delta\mathbf{g})^{\mathsf{T}}\Delta\mathbf{x}} - \frac{\mathbf{B}\Delta\mathbf{x}(\Delta\mathbf{x})^{\mathsf{T}}\mathbf{B}}{(\Delta\mathbf{x})^{\mathsf{T}}\mathbf{B}\Delta\mathbf{x}} \tag{1.50}$$

is a rank-two positive definite secant update provided $(\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{g} > 0$ is enforced at each iteration. This update was discovered independently by Broyden [51], Fletcher [81], Goldfarb [103], and Shanno [159] in 1970 and is known as the *BFGS update*.

The effective computational implementation of a quasi-Newton update introduces a number of additional considerations. When solving nonlinear equations, the search direction from (1.28) is $\mathbf{p} = -\mathbf{G}^{-1}\mathbf{c}$, and for optimization problems the search direction given by (1.41) is $\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}$. Since the search direction calculation involves the matrix inverse (either $\mathbf{G}^{-1}$ or $\mathbf{H}^{-1}$), one apparent simplification is to apply the recursive update directly to the inverse. In this case, the search direction can be computed simply by computing the matrix-vector product. This approach was proposed by Broyden for nonlinear equations, but has been considerably less successful in practice than the update given by (1.48), and is known as "Broyden's bad update." For unconstrained minimization, let us make the substitutions $\Delta\mathbf{x} \to \Delta\mathbf{g}$, $\Delta\mathbf{g} \to \Delta\mathbf{x}$, and $\mathbf{B} \to \mathbf{B}^{-1}$ in (1.50). By computing the inverse of the resulting expression, one obtains

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})(\Delta\mathbf{g})^{\mathsf{T}} + \Delta\mathbf{g}(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{g})^{\mathsf{T}}\Delta\mathbf{x}} - \sigma\Delta\mathbf{g}(\Delta\mathbf{g})^{\mathsf{T}}, \tag{1.51}$$

where

$$\sigma = \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}{(\Delta\mathbf{g}^{\mathsf{T}}\Delta\mathbf{x})^2}.$$

This so-called inverse positive definite secant update is referred to as the *DFP update* for its discoverers Davidon [68] and Fletcher and Powell [85].

Even though many recursive updates can be applied directly to the inverse matrices, most practical implementations do not use this approach. When the matrices $\mathbf{G}$ and/or $\mathbf{H}$ are singular, the inverse matrices do not exist. Consequently, it is usually preferable to work directly with $\mathbf{G}$ and $\mathbf{H}$. There are at least three issues that must be addressed by an effective implementation, namely efficiency, numerical conditioning, and storage. The solution of a dense linear system, such as (1.28) or (1.40), requires $\mathcal{O}(n^3)$ operations, compared with the $\mathcal{O}(n^2)$ operations needed to compute the matrix-vector product (1.41). However, this penalty can be avoided by implementing the update in "factored form." For example, the (positive definite) Hessian matrix can be written in terms of its Cholesky factorization as $\mathbf{H} = \mathbf{R}\mathbf{R}^{\mathsf{T}}$. Since the recursive update formulas represent low-rank modifications to $\mathbf{H}$, it is possible to derive low-rank modifications to the factors $\mathbf{R}$. By updating the matrices in factored form, the cost of computing the search direction can be reduced to $\mathcal{O}(n^2)$ operations, just as when the inverse is recurred directly. Furthermore, when the matrix factorizations are available, it is also possible to deal with rank deficiencies in a more direct fashion. Finally, when storage is an issue, the matrix at iteration $k$ can be represented as the sum of $L$ quasi-Newton updates to the original estimate $\mathbf{B}_0$ in the form

$$\mathbf{B}_k = \mathbf{B}_0 + \sum_{i=1}^{L} \mathbf{u}_i\mathbf{u}_i^{\mathsf{T}} - \sum_{i=1}^{L} \mathbf{v}_i\mathbf{v}_i^{\mathsf{T}}, \tag{1.52}$$

where the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ denote information from iteration $i$. If the initial estimate $\mathbf{B}_0$ requires relatively little storage (e.g., is diagonal), then all operations involving the matrix $\mathbf{B}_k$ at iteration $k$ can be performed without explicitly forming the (dense) matrix $\mathbf{B}_k$. This technique, called a *limited memory update*, requires only storing the vectors $\mathbf{u}$ and $\mathbf{v}$ over the previous $L$ iterations.

We have motivated the use of a recursive update as a way to construct Jacobian and/or Hessian information. However, we have not discussed how fast an iterative sequence will converge when the recursive update is used instead of the exact information. All of the methods that use a recursive update exhibit *superlinear* convergence provided the matrices are nonsingular. In general, superlinear convergence is not as fast as quadratic convergence. One way to measure the rate of convergence is to compare the behavior of a Newton method and a quasi-Newton method on a quadratic function of $n$ variables. Newton's method will terminate in one step, assuming finite-precision arithmetic errors are negligible. In contrast, a quasi-Newton method will terminate in at most $n$ steps, provided the steplength $\alpha$ is chosen at each iteration to minimize the value of the objective function at the new point $F(\overline{\mathbf{x}}) = F(\mathbf{x} + \alpha\mathbf{p})$. The process of adjusting $\alpha$ is called a *line search* and will be discussed in Section 1.11.

## 1.8 Equality-Constrained Optimization

The preceding sections describe how Newton's method can be applied either to optimize an objective function $F(\mathbf{x})$ *or* to satisfy a set of constraints $\mathbf{c}(\mathbf{x}) = \mathbf{0}$. Suppose now that we want to do both, that is, choose the variables $\mathbf{x}$ to minimize

$$F(\mathbf{x}) \tag{1.53}$$

subject to the $m \le n$ constraints

$$\mathbf{c}(\mathbf{x}) = \mathbf{0}. \tag{1.54}$$

The classical approach is to define the *Lagrangian*

$$L(\mathbf{x},\boldsymbol{\lambda}) = F(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{c}(\mathbf{x}) = F(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i c_i(\mathbf{x}), \tag{1.55}$$

where $\boldsymbol{\lambda}$ is an $m$-vector of *Lagrange multipliers*.[2]

In a manner analogous to the unconstrained case, optimality requires that derivatives with respect to both $\mathbf{x}$ and $\boldsymbol{\lambda}$ be zero. More precisely, necessary conditions for the point $(\mathbf{x}^*,\boldsymbol{\lambda}^*)$ to be an optimum are

$$\nabla_x L(\mathbf{x}^*,\boldsymbol{\lambda}^*) = \mathbf{0}, \tag{1.56}$$

$$\nabla_\lambda L(\mathbf{x}^*,\boldsymbol{\lambda}^*) = \mathbf{0}. \tag{1.57}$$

The gradient of $L$ with respect to $\mathbf{x}$ is

$$\nabla_x L = \mathbf{g} - \mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda} = \nabla F - \sum_{i=1}^{m} \lambda_i \nabla c_i \tag{1.58}$$

and the gradient of $L$ with respect to $\boldsymbol{\lambda}$ is

$$\nabla_\lambda L = -\mathbf{c}(\mathbf{x}). \tag{1.59}$$

Just as in the unconstrained case, these conditions do not distinguish between a point that is a minimum, a maximum, or simply a stationary point. As before, we require conditions on the curvature of the objective. Let us define the *Hessian of the Lagrangian* to be

$$\mathbf{H}_L = \nabla^2_{xx} L = \nabla^2_{xx} F - \sum_{i=1}^{m} \lambda_i \nabla^2_{xx} c_i. \tag{1.60}$$

Then a sufficient condition is that

$$\mathbf{v}^{\mathsf{T}}\mathbf{H}_L\mathbf{v} > \mathbf{0} \tag{1.61}$$

for any vector $\mathbf{v}$ in the constraint tangent space. If one compares (1.35) with (1.61), an important difference emerges. For the unconstrained case, we require that the curvature be positive in *all* directions $\mathbf{p}$. However, (1.61) applies only to directions $\mathbf{v}$ in the constraint tangent space.

[2]Some authors define $L = F + \boldsymbol{\omega}^{\mathsf{T}}\mathbf{c}$, where $\boldsymbol{\omega} = -\boldsymbol{\lambda}$, in which case corresponding changes must be made when interpreting the arithmetic sign of the multipliers.

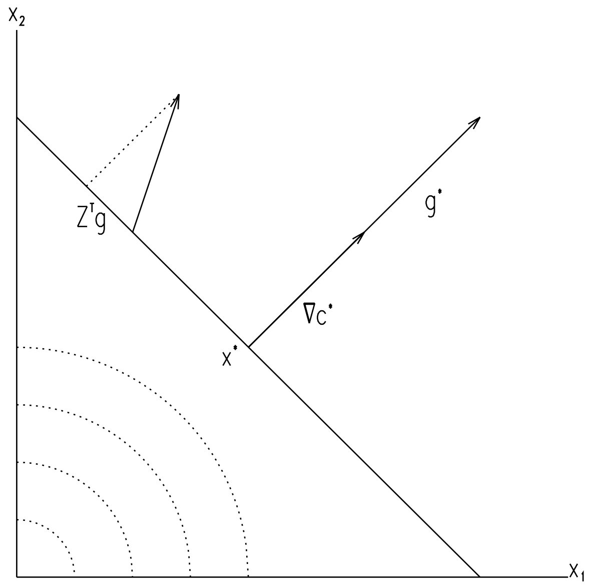


**Figure 1.5.** *Equality-constrained example.*

**Example 1.2** EQUALITY CONSTRAINED MINIMIZATION. To fully appreciate the meaning of these conditions, consider the simple example problem with two variables and one constraint illustrated in Figure 1.5. Let us minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraint

$$c(\mathbf{x}) = x_1 + x_2 - 2 = 0.$$

The solution is at $\mathbf{x}^* = (1,1)$. Now for this example, the Jacobian is just $\mathbf{G} = \nabla c^\mathsf{T} = (1,1)$, which is a vector orthogonal to the constraint. Consequently, if we choose $\mathbf{v}^\mathsf{T} = (-a,a)$ for some constant $a \neq 0$, the vector $\mathbf{v}$ is tangent to the constraint, which can be readily verified since $\mathbf{G}\mathbf{v} = (1)(-a) + (1)(a) = 0$. At $\mathbf{x}^*$, the gradient is a linear combination of the constraint gradients; i.e., (1.58) becomes

$$\mathbf{g} - \mathbf{G}^\mathsf{T}\boldsymbol{\lambda} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} 2 = \mathbf{0}.$$

Furthermore, from (1.61), the curvature

$$\mathbf{v}^\mathsf{T}\mathbf{H}_L\mathbf{v} = [-a, a] \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -a \\ a \end{bmatrix} = 4a^2$$

is clearly positive.

Notice that, at the optimal solution, the gradient vector is orthogonal to the constraint surface, or equivalently that the projection of the gradient vector onto the constraint surface is zero. A second point is also illustrated in Figure 1.5, demonstrating that the projection of the gradient is nonzero at the suboptimal point. Apparently, then, there is a matrix $\mathbf{Z}$ that projects the gradient onto the constraint surface. This implies that an equivalent form of the necessary condition (1.56) is to require that the *projected gradient* be zero, i.e.,

$$\mathbf{Z}^{\mathsf{T}}\mathbf{g} = \mathbf{0}. \tag{1.62}$$

In a similar fashion, one can define an equivalent form of the sufficient condition (1.61) in terms of the *projected Hessian* matrix

$$\mathbf{Z}^{\mathsf{T}}\mathbf{H}_L\mathbf{Z}. \tag{1.63}$$

In other words, we require the projected Hessian matrix (1.63) to be positive definite. Notice that the projection matrix $\mathbf{Z}$ has $n$ rows and $n_d = n - m$ columns. We refer to the quantity $n_d$ as the *number of degrees of freedom.* Observe also that the projected gradient (1.62) is a vector of length $n_d$ and the projected Hessian (1.63) is $n_d \times n_d$. In general, the choice of $\mathbf{Z}$ is not unique, although for the example problem one can choose any scalar multiple of $\mathbf{Z}^{\mathsf{T}} = (-1, 1)$.

## 1.8.1 Newton's Method

Let us now apply Newton's method to find the values of $(\mathbf{x}, \boldsymbol{\lambda})$ such that the necessary conditions (1.56) and (1.57) are satisfied. Proceeding formally to construct a Taylor series expansion analogous to (1.26), the expansion about $(\mathbf{x}, \boldsymbol{\lambda})$ for the functions (1.58) and (1.59) is just

$$\mathbf{0} = \mathbf{g} - \mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{H}_L(\overline{\mathbf{x}} - \mathbf{x}) - \mathbf{G}^{\mathsf{T}}(\overline{\boldsymbol{\lambda}} - \boldsymbol{\lambda}), \tag{1.64}$$

$$\mathbf{0} = -\mathbf{c} - \mathbf{G}(\overline{\mathbf{x}} - \mathbf{x}). \tag{1.65}$$

After simplification, these equations lead to the linear system analogous to that given by (1.28), which is called the Kuhn–Tucker (KT) or Karush–Kuhn–Tucker (KKT) system:

$$\begin{bmatrix} \mathbf{H}_L & \mathbf{G}^{\mathsf{T}} \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ \overline{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \mathbf{g} \\ \mathbf{c} \end{bmatrix}, \tag{1.66}$$

where $\mathbf{p}$ is the search direction for a step $\overline{\mathbf{x}} = \mathbf{x} + \mathbf{p}$ and $\overline{\boldsymbol{\lambda}}$ is the vector of Lagrange multipliers at the new point. Notice that the system is written in terms of the change in the variables, i.e., $\mathbf{p} = \overline{\mathbf{x}} - \mathbf{x}$, but does not involve the change in the multipliers, i.e., $\overline{\boldsymbol{\lambda}} - \boldsymbol{\lambda}$. Instead, it is preferable to explicitly eliminate the term $\mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda}$, which appears in the Taylor series approximation (1.64). Thus, in this instance, Newton's method is based on a linear approximation to the constraints *and* a linear approximation to the gradients. As in the unconstrained optimization case, a linear approximation to the gradient is equivalent to making a quadratic model for the quantity being optimized. It is important to note that the quadratic approximation is made to the Lagrangian (1.55) and not just the objective function $F$. Because of the underlying quadratic-linear model, Newton's method will converge in one step for a quadratic objective with linear equality constraints.

Although the derivation of the KKT system (1.66) was motivated by a Taylor series expansion, an alternative motivation is to choose $\mathbf{p}$ to minimize the quadratic objective

$$\mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} \tag{1.67}$$

subject to the linear constraints

$$\mathbf{G}\mathbf{p} = -\mathbf{c}. \tag{1.68}$$

It is straightforward to demonstrate that the optimality conditions for this quadratic-linear optimization subproblem are given by the KKT system (1.66).

## 1.9 Inequality-Constrained Optimization

Section 1.8 introduced the equality-constrained optimization problem. In this section, we consider a problem with inequality constraints. Suppose that we want to choose the variables $\mathbf{x}$ to minimize

$$F(\mathbf{x}) \tag{1.69}$$

subject to the $m$ inequality constraints

$$\mathbf{c}(\mathbf{x}) \geq \mathbf{0}. \tag{1.70}$$

In contrast to equality-constrained problems, which require $m \leq n$, the number of inequality constraints can exceed the number of variables. A point that satisfies the constraints is said to be *feasible* and the collection of all feasible points is called the *feasible region*. Conversely, points in the infeasible region violate one or more of the constraints.

Inequality-constrained problems are characterized by a fundamental concept. Specifically, at the solution $\mathbf{x}^*$,

1. some of the constraints will be satisfied as equalities, that is,

$$c_i(\mathbf{x}^*) = 0 \qquad \text{for} \qquad i \in \mathcal{A}, \tag{1.71}$$

where $\mathcal{A}$ is called the *active set*, and

2. some constraints will be strictly satisfied, that is,

$$c_i(\mathbf{x}^*) > 0 \qquad \text{for} \qquad i \in \mathcal{A}', \tag{1.72}$$

where $\mathcal{A}'$ is called the *inactive set*.

Thus, the total set of constraints is partitioned into two subsets, namely the active and inactive constraints. Clearly, the active constraints can be treated using the methods described in Section 1.8. Obviously, an inequality-constrained optimization algorithm needs some mechanism to identify the active constraints. This mechanism is referred to as an *active set strategy*. Fortunately, there is an additional necessary condition for inequality-constrained problems that is essential to active set strategies. Specifically, at the solution, the algebraic sign of the Lagrange multipliers must be correct; that is, we must have

$$\lambda_i^* \geq 0 \qquad \text{for} \qquad i \in \mathcal{A}. \tag{1.73}$$

To illustrate the impact of inequality constraints, let us consider two examples that are modifications of Example 1.2 presented in Section 1.8.

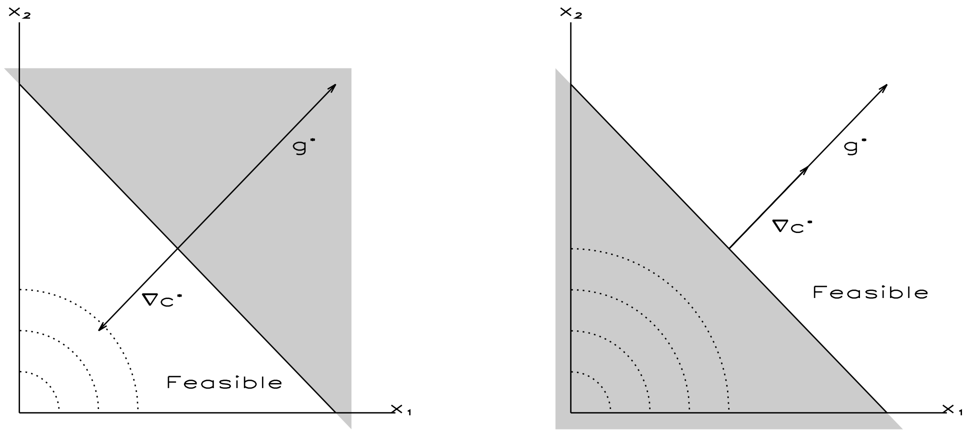


**Figure 1.6.** *Inequality-constrained examples.*

**Example 1.3** INEQUALITY MINIMIZATION—INACTIVE CONSTRAINT. The left side of Figure 1.6 illustrates the following problem: Minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraint

$$c(\mathbf{x}) = 2 - x_1 - x_2 \geq 0.$$

The gradient and Jacobian are given by

$$\mathbf{g} = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}, \qquad \mathbf{G} = \begin{bmatrix} -1 & -1 \end{bmatrix}.$$

Now at the point $\mathbf{x}^\mathsf{T} = (1,1)$, the optimality conditions are

$$\mathbf{0} = \mathbf{g} - \mathbf{G}^\mathsf{T}\boldsymbol{\lambda} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix} [-2]$$

and, since $\lambda = -2 < 0$, the constraint should be deleted from the active set. The solution is $\mathbf{x}^\mathsf{T} = (0,0)$.

**Example 1.4** INEQUALITY MINIMIZATION—ACTIVE CONSTRAINT. On the other hand, if the sense of the inequality is reversed, we must minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the inequality

$$c(\mathbf{x}) = x_1 + x_2 - 2 \geq 0.$$

This problem is illustrated on the right side of Figure 1.6. At the point $\mathbf{x}^\mathsf{T} = (1,1)$, the Lagrange multiplier $\lambda = 2 > 0$. Consequently, the constraint is *active* and cannot be deleted. The solution is $\mathbf{x}^\mathsf{T} = (1,1)$.

## 1.10 Quadratic Programming

An important special form of optimization problem is referred to as the quadratic programming (QP) problem. A QP problem is characterized by

- a quadratic objective

$$F(\mathbf{x}) = \mathbf{g}^{\mathsf{T}}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{H}\mathbf{x} \tag{1.74}$$

- and linear constraints

$$\mathbf{A}\mathbf{x} = \mathbf{a}, \tag{1.75}$$

$$\mathbf{B}\mathbf{x} \geq \mathbf{b}, \tag{1.76}$$

where $\mathbf{H}$ is the positive definite Hessian matrix.

Let us now outline an *active set method* for solving this QP problem using the concepts introduced in Sections 1.8 and 1.9. Assume that an estimate of the active set $\mathcal{A}^0$ is given in addition to a feasible point $\mathbf{x}^0$. A QP algorithm proceeds as follows:

1. Compute the minimum of the quadratic objective subject to the constraints in the active set estimate $\mathcal{A}$ treated as *equalities*, i.e., solve the KKT system (1.66).

2. Take the largest possible step in the direction $\mathbf{p}$ that does not violate any *inactive* inequalities, that is,

$$\overline{\mathbf{x}} = \mathbf{x} + \alpha\mathbf{p}, \tag{1.77}$$

   where the steplength $0 \leq \alpha \leq 1$ is chosen to maintain feasibility with respect to the inactive inequality constraints.

3. If the step is restricted, i.e., $\alpha < 1$, then
    - *add* the limiting inequality to the active set $\mathcal{A}$ and return to step 1;
    - otherwise, take the full step ($\alpha = 1$) and check the sign of the Lagrange multipliers:
        - if all of the inequalities have positive multipliers, terminate the algorithm;
        - otherwise, *delete* the inequality with the most negative $\lambda$ from the active set and return to step 1.

Notice that step 1 requires the solution of the KKT system (1.66) corresponding to the set of active constraints defined by $\mathcal{A}$. In particular, (1.66) becomes

$$\begin{bmatrix} \mathbf{H} & \mathbf{A}^{\mathsf{T}} & \widetilde{\mathbf{B}}^{\mathsf{T}} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \widetilde{\mathbf{B}} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ \overline{\eta} \\ \overline{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{g} \\ \mathbf{a} \\ \widetilde{\mathbf{b}} \end{bmatrix}, \tag{1.78}$$

where the vector $\widetilde{\mathbf{b}}$ denotes the subset of the vector $\mathbf{b}$ corresponding to the *active* inequality constraints and $\widetilde{\mathbf{B}}$ is the corresponding Jacobian matrix for the constraints in $\mathcal{A}$. The Lagrange multipliers corresponding to the equality constraints are denoted by $\eta$. For the purposes of this discussion, it suffices to say that any reasonable method for solving (1.78) can be used. However, in practice it is extremely important that this solution be computed in an efficient and numerically stable way. This is especially true because every time the active set changes in step 3, it is necessary to solve a different KKT system to compute the new step. This is because each time the active set changes, the matrix $\widetilde{\mathbf{B}}$ is altered by adding or deleting a row. Most efficient QP implementations compute the new step by modifying the previously computed solution and the associated matrix factorizations. In general, the computational expense of the QP problem is dominated by the linear algebra and one should *not* use a "brute force" solution to the KKT system! While space precludes a more detailed discussion of this subject, the interested reader should consult the book by Gill, Murray, and Wright [100]. An approach that is particularly efficient for large, sparse applications will be described in Section 2.3.

It should also be apparent that the number of QP iterations is determined by the initial active set $\mathcal{A}^0$. If the initial active set is correct, i.e., $\mathcal{A}^0 = \mathcal{A}^*$, then the QP algorithm will compute the solution in one iteration. Conversely, many QP iterations may be required if the initial active set differs substantially from the final one. Additional complications can arise when the Hessian matrix $\mathbf{H}$ is indefinite because the possibility of an unbounded solution exists. Furthermore, when a multiplier $\lambda_k \approx 0$, considerable care must be exercised when deleting a constraint in step 3.

As stated, the QP algorithm requires a feasible initial point $\mathbf{x}^0$, which may require a special "phase-1" procedure in the software. It is also worth noting that when $\mathbf{H} = \mathbf{0}$, the objective is linear and the optimization problem is referred to as a *linear programming* or *LP* problem. In this case, the active set algorithm described is equivalent to the *simplex method* proposed by Dantzig [67] in 1947. However, it is important to note that a *unique* solution to a linear program will have $n$ active constraints, whereas there may be many degrees of freedom at the solution of a quadratic program.

**Example 1.5** QUADRATIC PROGRAM. To illustrate how a QP algorithm works, let us consider minimizing

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraints

$$c_1(\mathbf{x}) = x_1 + x_2 - 2 \geq 0,$$

$$c_2(\mathbf{x}) = 4 - x_1 - \frac{2}{3}x_2 \geq 0.$$

This problem is illustrated in Figure 1.7. Table 1.2 summarizes the QP steps assuming the iteration begins at $\mathbf{x}^\mathsf{T} = (4,0)$ with $\mathcal{A}^0 = \{c_2\}$. The first step requires solving the KKT system (1.78) evaluated at the point $\mathbf{x}^\mathsf{T} = (4,0)$, that is,

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -\frac{2}{3} \\ -1 & -\frac{2}{3} & 0 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \\ \bar{\lambda}_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 0 \\ 0 \end{bmatrix}.$$

After this step, the new point is at $\bar{x}_1 = x_1 + p_1 = 2.76923$, and $\bar{x}_2 = x_2 + p_2 = 1.84615$. Since the Lagrange multiplier is negative at the new point, constraint $c_2$ can be deleted from the active set.

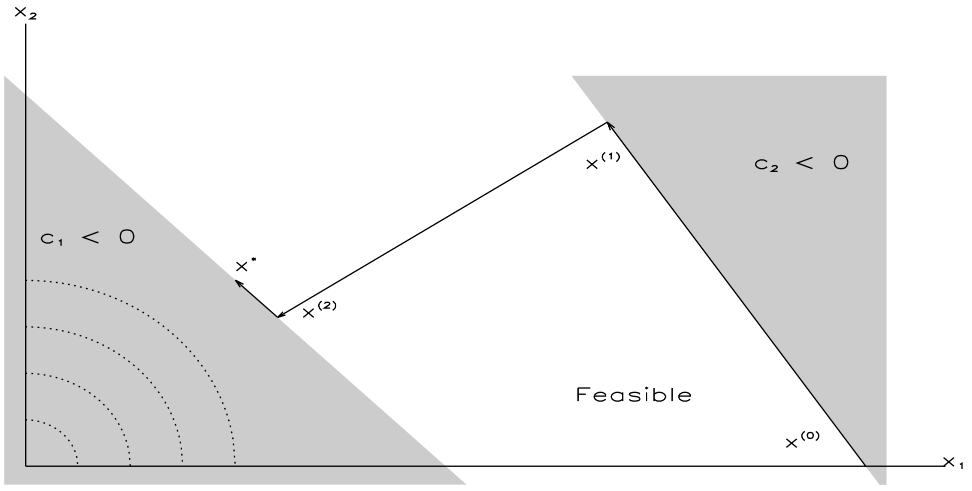


**Figure 1.7.** *QP example.*

**Table 1.2.** *QP iterations.*

| Iteration | $x_1$ | $x_2$ | Active Set $\mathcal{A}$ |
|---|---|---|---|
| 0 | 4 | 0 | $\mathcal{A}^0 = \{c_2\}$ |
| 1 | 2.76923 | 1.84615 | Delete $c_2$ ($\lambda_2 = -5.53846$). |
| 2 | 1.2 | 0.8 | Add $c_1$ ($\alpha = 0.566667$). |
| 3 | 1 | 1 | $\mathcal{A}^* = \{c_1\}$ |

The second QP step begins at $\mathbf{x}^\mathsf{T} = (2.76923, 1.84615)$ with no constraints active, i.e., $\mathcal{A}^1 = \{\emptyset\}$. The new step is computed from the KKT system

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \end{bmatrix} = \begin{bmatrix} 5.53846 \\ 3.69230 \end{bmatrix}.$$

However, in this case, it is not possible to take a full step because the first constraint would be violated. Instead, one finds $\bar{\mathbf{x}} = \mathbf{x} + \alpha\mathbf{p}$ with $\alpha = 0.566667$, to give

$$\bar{\mathbf{x}} = \begin{bmatrix} 1.2 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 2.76923 \\ 1.84615 \end{bmatrix} - 0.566667 \begin{bmatrix} 2.76923 \\ 1.84615 \end{bmatrix}.$$

In general, the length of the step is given by the simple linear prediction

$$\alpha = \min \left[ \frac{-c_i(\mathbf{x})}{\mathbf{p}^\mathsf{T} \nabla c_i} \right] > 0 \qquad \text{for} \qquad i \in \mathcal{A}'. \tag{1.79}$$

Essentially, one must compute the largest step that will not violate any of the (currently) inactive inequality constraints.

Since the second QP iteration terminates by encountering the first constraint $c_1$, it must be added to the active set so that $\mathcal{A}^2 = \{c_1\}$. Once again the new step is computed from the KKT system, which in this case is just

$$\begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \\ \bar{\lambda}_1 \end{bmatrix} = \begin{bmatrix} 2.4 \\ 1.6 \\ 0 \end{bmatrix}.$$

After computing the new point, the solution is given by $\mathbf{x}^* = (1,1)$, and the final active set is $\mathcal{A}^* = \{c_1\}$, with optimal Lagrange multipliers $\boldsymbol{\lambda}^* = (2,0)$.

## 1.11 Globalization Strategies

### 1.11.1 Merit Functions

To this point, the development has stressed the application of Newton's method. However, even for the simplest one-dimensional applications described in Section 1.2, Newton's method has deficiencies. Methods for correcting the difficulties with Newton's method are referred to as *globalization strategies*. There are two primary roles performed by a globalization strategy, namely,

1. detecting a problem with the (unmodified) Newton's method and

2. correcting the deficiency.

It should be emphasized that the goal of all globalization strategies is to do *nothing!* Clearly, near a solution it is desirable to retain the quadratic convergence of a Newton step and, consequently, it is imperative that modifications introduced by the globalization process not impede the ultimate behavior.

Referring to (1.28) and (1.29), all Newton iterations are based on linearizations of the form

$$\mathbf{Gp} = -\mathbf{c} \tag{1.80}$$

for the search direction $\mathbf{p}$, which leads to an iteration of the form

$$\bar{\mathbf{x}} = \mathbf{x} + \mathbf{p}. \tag{1.81}$$

If Newton's method is working properly, the sequence of iterates $\{\mathbf{x}^{(k)}\}$ should converge to the solution $\mathbf{x}^*$. In practice, of course, the solution $\mathbf{x}^*$ is unknown, and we must detect whether the sequence is converging properly using information that is available. The most common way to measure progress is to assign some *merit function*, $M$, to each iterate $\mathbf{x}^{(k)}$ and then insist that

$$M(\mathbf{x}^{(k+1)}) < M(\mathbf{x}^{(k)}). \tag{1.82}$$

This is one approach for detecting when the Newton sequence is working. When Newton's method is used for unconstrained optimization, an obvious merit function is just $M(\mathbf{x}) = F(\mathbf{x})$. When Newton's method is used to find the root of many functions, as in (1.80), assigning a single value to quantify "progress" is no longer quite so obvious. The most commonly used merit function for nonlinear equations is

$$M(\mathbf{x}) = \frac{1}{2}\mathbf{c}^\mathsf{T}(\mathbf{x})\mathbf{c}(\mathbf{x}). \tag{1.83}$$

If the only purpose for constructing a merit function is to quantify progress in the iterative sequence, then any other norm is also suitable; e.g., we could use

$$M(\mathbf{x}) = \|\mathbf{c}\|_1 = \sum_{i=1}^{m} |c_i|,$$

$$M(\mathbf{x}) = \|\mathbf{c}\|_2 = \left(\sum_{i=1}^{m} c_i^2\right)^{\frac{1}{2}},$$

or

$$M(\mathbf{x}) = \|\mathbf{c}\|_\infty = \max_{i=1}^{m} |c_i|.$$

Choosing a merit function for constrained optimization is still more problematic, for it is necessary to somehow balance the (often) conflicting goals of reducing the objective function while satisfying the constraints. To this point, we have motivated the discussion of a merit function as an artifice to "fix" Newton's method when it is not converging. An alternative approach is to construct some other function $P$, whose *unconstrained* minimum either is the desired constrained solution $\mathbf{x}^*$ or is related to it in a known way. Thus, we might consider solving a sequence of unconstrained problems whose solutions approach the desired constrained optimum. This point of view, which is fundamental to so-called penalty function methods, will be discussed in Section 1.14. One possible candidate is the quadratic penalty function

$$P(\mathbf{x},\rho) = F(\mathbf{x}) + \frac{\rho}{2}\mathbf{c}^\mathsf{T}(\mathbf{x})\mathbf{c}(\mathbf{x}), \tag{1.84}$$

where $\rho$ is called the *penalty weight* or *penalty parameter*. When this penalty function is minimized for successively larger values of the penalty weight $\rho$, it can be shown that the unconstrained minimizers approach the constrained solution. Unfortunately, from a computational viewpoint, this is unattractive since, as the parameter $\rho \to \infty$, the successive unconstrained optimization problems become harder and harder to solve. An alternative, which avoids this ill-conditioning, is the so-called *augmented Lagrangian function* formed by combining the Lagrangian (1.55) with a constraint penalty of the form (1.83) to yield

$$P(\mathbf{x},\boldsymbol{\lambda},\rho) = L(\mathbf{x},\boldsymbol{\lambda}) + \frac{\rho}{2}\mathbf{c}^\mathsf{T}(\mathbf{x})\mathbf{c}(\mathbf{x}) = F(\mathbf{x}) - \boldsymbol{\lambda}^\mathsf{T}\mathbf{c}(\mathbf{x}) + \frac{\rho}{2}\mathbf{c}^\mathsf{T}(\mathbf{x})\mathbf{c}(\mathbf{x}). \tag{1.85}$$

It can be shown that the unconstrained minimum of this function is equal to the constrained minimum $\mathbf{x}^*$ for a *finite* value of the parameter $\rho$. Unfortunately, in practice, choosing a "good" value for the penalty weight is nontrivial. If $\rho$ is too large, the unconstrained optimization problem is ill-conditioned and, consequently, very difficult to solve. If $\rho$ is too small, the unconstrained minimum of the augmented Lagrangian may be unbounded and/or the problem may be ill-conditioned. On the other hand, some of these drawbacks are not as critical when the augmented Lagrangian is used only to measure progress as part of a globalization strategy. Thus, we are led to the notion of computing the Newton step **p** in the usual way and then measuring progress by insisting that

$$M(\mathbf{x}^{(k+1)},\boldsymbol{\lambda}^{(k+1)},\rho) < M(\mathbf{x}^{(k)},\boldsymbol{\lambda}^{(k)},\rho), \tag{1.86}$$

where the merit function $M(\mathbf{x},\boldsymbol{\lambda},\rho) = P(\mathbf{x},\boldsymbol{\lambda},\rho)$. In summary, a merit function can be used to quantitatively decide whether or not Newton's method is working.

## 1.11.2 Line-Search Methods

Assuming for the moment that a merit function is used as an indicator of progress, how does one alter Newton's method when necessary? One approach is to alter the *magnitude* of the step using a *line-search* method. A second approach is to change both the magnitude and the *direction* using a *trust-region* method. The basic notion of a line-search method is to replace the Newton step (1.81) with

$$\overline{\mathbf{x}} = \mathbf{x} + \alpha\mathbf{p}. \tag{1.87}$$

Using this expression, it then follows that the merit function can be written as a function of the single variable $\alpha$:

$$M(\overline{\mathbf{x}}) = M(\mathbf{x}+\alpha\mathbf{p}) = M(\alpha). \tag{1.88}$$

Furthermore, it is reasonable to choose the steplength $\alpha$ such that $M(\alpha)$ is approximately minimized. Most modern line-search implementations begin with the full Newton step, i.e., $\alpha^{(0)} = 1$. Then estimates are computed until a steplength $\alpha^{(k)}$ is found that satisfies a *sufficient decrease* condition given by the *Goldstein–Armijo* condition

$$0 < -\kappa_1\alpha^{(k)}M'(0) \le M(0) - M(\alpha^{(k)}) \le -\kappa_2\alpha^{(k)}M'(0). \tag{1.89}$$

$M'(0) = (\nabla M(0))^{\mathrm{T}}\mathbf{p}$ is the direction derivative at $\alpha = 0$, with the constants $\kappa_1$ and $\kappa_2$ satisfying $0 < \kappa_1 \le \kappa_2 < 1$. Typical values for the constants are $\kappa_1 = 10^{-4}$ and $\kappa_2 = 0.9$, and do *not* require an "accurate" minimization of $M$. Nevertheless, most line-search implementations are quite sophisticated and use quadratic and/or cubic polynomial interpolation in conjunction with some type of safeguarding procedure. Furthermore, special-purpose line-search procedures are often employed for different merit functions; e.g., see Murray and Wright [136]. In fact, we have already described two applications that may be viewed as special-purpose line-search algorithms. First, if a quasi-Newton update is used to construct the Hessian, an "exact" line search will produce termination after $n$ steps on a quadratic function. Second, when solving a quadratic program, the steplength given by (1.79) is actually the minimum of the quadratic function restricted such that inactive inequalities are not violated.

**Example 1.6** NEWTON METHOD WITH LINE SEARCH. To illustrate how Newton's method works when a line-search strategy is incorporated, let us consider the simple nonlinear system

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} 1 - x_1 \\ 10(x_2 - x_1^2) \end{bmatrix} = \mathbf{0}.$$

**Table 1.3.** *Newton's method with line search.*

| Iter. | $x_1$ | $x_2$ | $\alpha$ | $M(\mathbf{x})$ |
|---|---|---|---|---|
| 0 | $-1.2000$ | $1.0000$ | $--$ | $12.100$ |
| 1 | $-1.0743$ | $0.72336$ | $0.57157 \times 10^{-1}$ | $11.425$ |
| 2 | $-0.94553$ | $0.47352$ | $0.62059 \times 10^{-1}$ | $10.734$ |
| 3 | $-0.81340$ | $0.25221$ | $0.67917 \times 10^{-1}$ | $10.025$ |
| 4 | $-0.67732$ | $0.61558 \times 10^{-1}$ | $0.75040 \times 10^{-1}$ | $9.2952$ |
| 5 | $-0.53662$ | $-0.95719 \times 10^{-1}$ | $0.83883 \times 10^{-1}$ | $8.5411$ |
| 6 | $-0.39041$ | $-0.21613$ | $0.95147 \times 10^{-1}$ | $7.7580$ |
| 7 | $-0.23751$ | $-0.29499$ | $0.10997$ | $6.9398$ |
| 8 | $-0.76248 \times 10^{-1}$ | $-0.32580$ | $0.13031$ | $6.0775$ |
| 9 | $0.66751 \times 10^{-1}$ | $-0.30355$ | $0.13287$ | $5.1787$ |
| 10 | $0.22101$ | $-0.23204$ | $0.16529$ | $4.2483$ |
| 11 | $0.36706$ | $-0.11482$ | $0.18748$ | $3.3142$ |
| 12 | $0.55206$ | $0.93929 \times 10^{-1}$ | $0.29229$ | $2.3230$ |
| 13 | $1.0000$ | $0.79935$ | $1.0000$ | $2.0131$ |
| 14 | $1.0000$ | $1.0000$ | $1.0000$ | $0.12658 \times 10^{-18}$ |

**Table 1.4.** *Newton's method without line search.*

| Iter. | $x_1$ | $x_2$ | $\alpha$ | $M(\mathbf{x})$ |
|---|---|---|---|---|
| 0 | $-1.2000$ | $1.0000$ | $--$ | $12.100$ |
| 1 | $1.0000$ | $-3.8400$ | $1.0000$ | $1171.3$ |
| 2 | $1.0000$ | $1.0000$ | $1.0000$ | $0.85927 \times 10^{-14}$ |

If the merit function $M(\mathbf{x})$ (1.83) is used to monitor progress and the steplength $\alpha$ is adjusted using a line search with polynomial interpolation, one obtains the iteration history summarized in Table 1.3. Notice that every iteration produces a reduction in the merit function as it must. However, in order to achieve this monotonic behavior, it is necessary to modify the steplength on nearly all iterations. Quadratic convergence is observed only during the final few iterations.

It is interesting to compare the behavior of Newton's method, which has a line-search globalization procedure, to that of an unmodified Newton algorithm. Table 1.4 summarizes the iterations when Newton's method is applied *without* a line search. Notice that all steps have $\alpha = 1$. Furthermore, observe that the first step produced a large increase in the constraint error as measured by the merit function $M$. Nevertheless, in this case, the unmodified Newton method was significantly more efficient than the approach with a line search. For this example, the line search is not only unnecessary but also inefficient. This suggests two questions. First, do we need a globalization strategy? Clearly, the answer is yes. Second, is a line search with merit function the most efficient strategy? Perhaps not. The numerical results suggest that there is considerable room for improvement provided a mechanism can be devised that is sufficiently robust.

### 1.11.3 Trust-Region Methods

A line-search globalization strategy computes the search direction $\mathbf{p}$ and then adjusts the steplength parameter $\alpha$ in order to define the iteration (1.87). A second alternative is to adjust both the magnitude and direction of the vector $\mathbf{p}$. Treating the current point $\mathbf{x}$ as fixed, suppose that we want to choose the variables $\mathbf{p}$ to minimize

$$F(\mathbf{x}+\mathbf{p}) \approx F(\mathbf{x}) + \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} \tag{1.90}$$

subject to the constraint

$$\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{p} \le \delta^2. \tag{1.91}$$

We assume that we can *trust* the prediction $F(\mathbf{x}+\mathbf{p})$ as long as the points lie within the region defined by the *trust radius* $\delta$. Proceeding formally to define the Lagrangian for this problem, one obtains

$$L_T(\mathbf{p},\tau) = F(\mathbf{x}) + \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} - \tau\left[\delta^2 - \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{p}\right], \tag{1.92}$$

where $\tau$ is the Lagrange multiplier associated with the trust-region inequality constraint. Now the corresponding necessary condition is just

$$\nabla_p L_T(\mathbf{p},\tau) = \mathbf{g} + \mathbf{H}\mathbf{p} + \tau\mathbf{p} = \mathbf{g} + [\mathbf{H} + \tau\mathbf{I}]\mathbf{p} = \mathbf{0}. \tag{1.93}$$

There are two possible solutions to this problem. If the trust-radius constraint is inactive, then $\tau = 0$ and the search direction $\mathbf{p}$ is just the unmodified Newton direction. On the other hand, if the trust-radius constraint is active, $\|\mathbf{p}\| = \delta$ and the multiplier $\tau > 0$. In fact, the trust radius $\delta$ and the multiplier $\tau$ are implicitly (and inversely) related to each other—when $\delta$ is large, $\tau$ is small, and vice versa. Traditional trust-region methods maintain a current estimate of the trust radius $\delta$ and adjust the parameter $\tau$ such that the constraint $\|\mathbf{p}\| = \delta$ is approximately satisfied. An alternative is to view the parameter $\tau$ as a way to construct a modified Hessian matrix $\mathbf{H} + \tau\mathbf{I}$. This interpretation was suggested by Levenberg [131] for nonlinear least squares (NLS) problems, and we shall often refer to $\tau$ as the *Levenberg parameter*. Notice that as $\tau$ becomes large, the search direction approaches the gradient direction. Table 1.5 summarizes the limiting behavior of these quantities.

**Table 1.5.** *Trust-region behavior.*

| $\tau \to \infty$ | $\tau \to 0$ |
|---|---|
| $\delta \to 0$ | $\delta \to \infty$ |
| $\mathbf{p} \propto -\mathbf{g}$ | $\mathbf{p} \to -\mathbf{H}^{-1}\mathbf{g}$ |
| $\|\mathbf{p}\| \to 0$ | $\|\mathbf{p}\| \to \|\mathbf{H}^{-1}\mathbf{g}\|$ |

Although the trust-region approach has been introduced as a globalization technique for an unconstrained optimization problem, the basic idea is far more general. The trust-region concepts can be extended to constrained optimization problems as well as, obviously, to least squares and root-solving applications. Other definitions of the trust region itself using alternative norms have been suggested. Furthermore, constrained applications have been formulated with more than one trust region to reflect the conflicting aims of constraint satisfaction and objective function reduction. Finally, it should be emphasized that trust-region concepts are often used in conjunction with other globalization ideas. For example, a common way to modify the trust radius from one iteration to the next is to compare the values of the predicted and actual reduction in a suitable merit function.

### 1.11.4 Filters

When solving a constrained optimization problem, it is necessary to introduce some quantitative method for deciding that a Newton iterate is working. One approach for achieving this is to introduce a merit function that combines the conflicting goals of constraint satisfaction and objective function reduction. However, merit functions must explicitly assign some relative weight to each conflicting goal, and choosing the correct penalty weight is often difficult. This dilemma was illustrated by the results summarized in Tables 1.3 and 1.4. Recently, Fletcher and Leyffer [84] have introduced an approach that will accept a Newton step if *either* the objective function *or* the constraint violation is decreased. The filter approach recognizes that there are two conflicting aims in nonlinear programming. The first is to minimize the objective function, that is, choose $\mathbf{x}$ to minimize

$$F(\mathbf{x}). \tag{1.94}$$

The second is to choose $\mathbf{x}$ to minimize the constraint violation

$$v[\mathbf{c}(\mathbf{x})], \tag{1.95}$$

where the *violation* can be measured using any suitable norm. Fletcher and Leyffer define $v\,[\mathbf{c}(\mathbf{x})] \equiv \|\tilde{\mathbf{c}}\|_1$, where $\tilde{c}_k = \min(0, c_k)$, for inequalities of the form $c_k \geq 0$. The basic idea is to compare information from the current iteration to information from previous iterates and then "filter" out the bad iterates.

To formalize this, denote the values of the objective and constraint violation at the point $\mathbf{x}^{(k)}$ by

$$\{F^{(k)}, v^{(k)}\} \equiv \{F(\mathbf{x}^{(k)}), v[\mathbf{c}(\mathbf{x}^{(k)})]\}. \tag{1.96}$$

When comparing the information at two different points $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(j)}$, a pair $\{F^{(k)}, v^{(k)}\}$ is said to *dominate* another pair $\{F^{(j)}, v^{(j)}\}$ if and only if both $F^{(k)} \leq F^{(j)}$ and $v^{(k)} \leq v^{(j)}$. Using this definition, we can then define a *filter* as a list of pairs

$$\mathcal{F} = \begin{bmatrix} F^{(1)}, v^{(1)} \\ F^{(2)}, v^{(2)} \\ \vdots \\ F^{(K)}, v^{(K)} \end{bmatrix} \tag{1.97}$$

such that no pair dominates any other. A new point $\{F^{(\ell)}, v^{(\ell)}\}$ is said to be acceptable for inclusion in the filter if it is not dominated by any point in the filter.

**Example 1.7** FILTER GLOBALIZATION. To illustrate how this procedure works, let us revisit Example 1.6, posed as the following optimization problem: Minimize

$$F(\mathbf{x}) = (1 - x_1)^2$$

subject to

$$\mathbf{c}(\mathbf{x}) = 10(x_2 - x_1^2) = 0.$$

Table 1.6 summarizes the behavior of a filter algorithm using the same sequence of iterates as in Table 1.4. The problem is illustrated in Figure 1.8. The iteration begins at $\mathbf{x}^{\mathsf{T}} = (-1.2, 1)$, and the first filter entry is $(F^{(1)}, v^{(1)}) = (4.84, 4.4)$. The dark shaded region in Figure 1.8 defines points that would be rejected by the filter because both the objective function and constraint violation would be worse than the first filter entry values. The second iteration at the point $\mathbf{x}^{\mathsf{T}} = (1, -3.84)$ is *accepted* by the filter because the objective function is better even though the constraint violation is worse. This point is added to the filter, thereby defining a new "unacceptable" region, which is the union of the regions defined by each point in the filter (i.e., the dark and light shaded regions). The final iterate does not violate the regions defined by either of the filter entries and is accepted as the solution.

Observe that the filter provides only a mechanism for accepting or rejecting an iterate. It does not suggest how to correct the step if a point is rejected by the filter. Fletcher

**Table 1.6.** *Filter iteration summary.*

| Iter. | $x_1$ | $x_2$ | $F(\mathbf{x})$ | $v[\mathbf{c}(\mathbf{x})]$ |
|---|---|---|---|---|
| 0 | −1.2000 | 1.0000 | 4.84 | 4.4 |
| 1 | 1.0000 | −3.8400 | 0 | 48.4 |
| 2 | 1.0000 | 1.0000 | 0 | 0 |

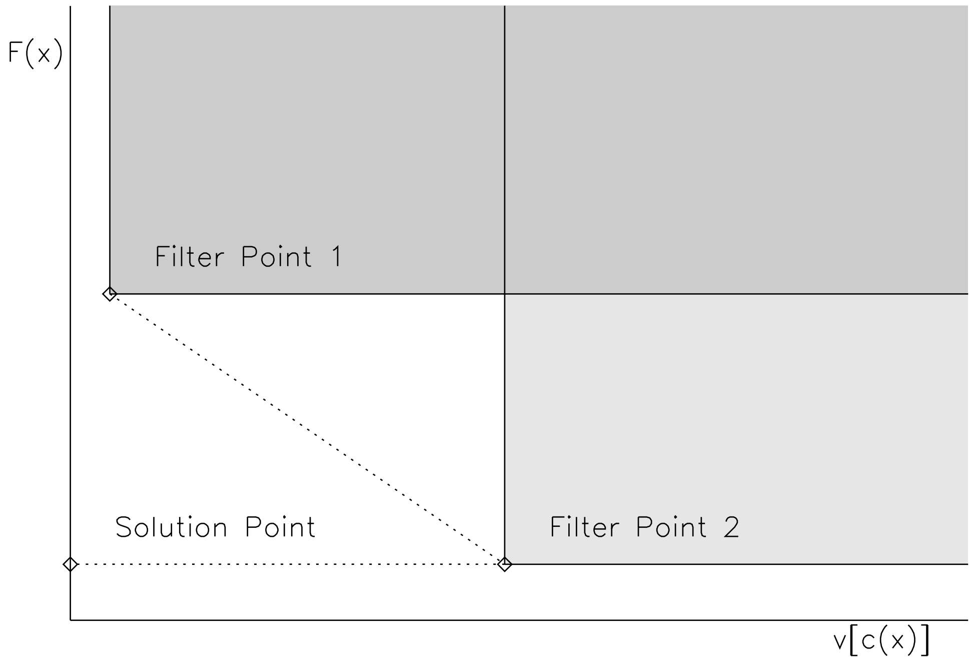


**Figure 1.8.** *NLP filter.*

and Leyffer use the filter in conjunction with a trust-region approach. On the other hand, a line-search technique that simply reduces the steplength is also an acceptable method for correcting the iterate. Practical implementation of the filter mechanism also must preclude a sequence of points that becomes unbounded in either $F(\mathbf{x})$ or $v[\mathbf{c}(\mathbf{x})]$. A generous overestimate of the upper bound on $F(\mathbf{x})$ can be included as an additional "northwest corner" entry in the filter. Similarly, an absolute upper limit on the constraint violation can be included as a "southeast corner" entry in the filter. Computation of these extreme values is readily accommodated by including the value of the largest Lagrange multiplier for each iterate in the list of saved information. The amount of information saved in the filter list is usually rather small because, when a new entry $\{F^{(\ell)}, v^{(\ell)}\}$ is added, all points dominated by the new entry are deleted from the filter.

## 1.12 Nonlinear Programming

The general nonlinear programming (NLP) problem can be stated as follows: Find the $n$-vector $\mathbf{x}^\mathsf{T} = (x_1, \dots, x_n)$ to *minimize* the scalar objective function

$$F(\mathbf{x}) \tag{1.98}$$

subject to the $m$ constraints

$$\mathbf{c}_L \le \mathbf{c}(\mathbf{x}) \le \mathbf{c}_U \tag{1.99}$$

and the simple bounds

$$\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U. \tag{1.100}$$

Equality constraints can be imposed by setting $\mathbf{c}_L = \mathbf{c}_U$.

The KKT necessary conditions for $\mathbf{x}^*$ to be an optimal point require that

- $\mathbf{x}^*$ be feasible, i.e., (1.99) and (1.100) are satisfied;
- the Lagrange multipliers $\boldsymbol{\lambda}$ corresponding to (1.99) and $\boldsymbol{\nu}$ corresponding to (1.100) satisfy
$$\mathbf{g} = \mathbf{G}^\mathsf{T}\boldsymbol{\lambda} + \boldsymbol{\nu}; \tag{1.101}$$
- the Lagrange multipliers for the inequalities be
$$\begin{cases} \text{nonpositive} & \text{for active upper bounds,} \\ \text{zero} & \text{for strictly satisfied constraints,} \\ \text{nonnegative} & \text{for active lower bounds;} \end{cases}$$
- the Jacobian $\widetilde{\mathbf{G}}$ corresponding to the active constraints have full row rank (the *constraint qualification test*).

The constraint qualification test implies that the gradients of the active constraints are linearly independent, and as such is often referred to as *linear independence constraint qualification* and abbreviated as LIQC. A weaker condition than LIQC is referred to as *Mangasarian–Fromovitz constraint qualification* or MFQC. In the simplest setting with only inequalities $\mathbf{c}(\mathbf{x}) \ge \mathbf{0}$, it ensures the existence of a direction along which, to first order, all active constraints strictly increase, i.e. there exists a vector $\mathbf{p}$ such that $\widetilde{\mathbf{G}}\mathbf{p} > \mathbf{0}$.

## 1.13 An SQP Algorithm

Let us now outline the basic steps of a sequential quadratic programming or SQP method for solving the general NLP problem. SQP methods are among the most widely used algorithms for solving general NLPs, and there are many variations of the basic approach. The method described is implemented in the $\mathbb{SOCS}$ [38] software and is similar to the algorithm employed by the NPSOL [96] software. For additional information on SQP methods, see, for example, Fletcher [82] and Gill, Murray, and Wright [99].

The basic approach described in Section 1.8 was to introduce a quadratic approximation to the Lagrangian and a linear approximation to the constraints. This development led to (1.67) and (1.68). An identical approach is followed here. Assume that the iteration begins at the point $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$. The fundamental idea is to solve the following QP subproblem:

Compute $\mathbf{p}$ to minimize a quadratic approximation to the Lagrangian

$$\mathbf{g}^\mathsf{T}\mathbf{p} + \frac{1}{2}\mathbf{p}^\mathsf{T}\mathbf{H}\mathbf{p} \tag{1.102}$$

subject to the linear approximation to the constraints

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{G}\mathbf{p} \\ \mathbf{p} \end{bmatrix} \leq \mathbf{b}_U \tag{1.103}$$

with bound vectors defined by

$$\mathbf{b}_L = \begin{bmatrix} \mathbf{c}_L - \mathbf{c} \\ \mathbf{x}_L - \mathbf{x} \end{bmatrix}, \qquad \mathbf{b}_U = \begin{bmatrix} \mathbf{c}_U - \mathbf{c} \\ \mathbf{x}_U - \mathbf{x} \end{bmatrix}. \tag{1.104}$$

The solution of this QP subproblem defines more than just the search direction $\mathbf{p}$. In particular, the quadratic program determines an estimate of the active set of constraints and an estimate of the corresponding Lagrange multipliers $\widehat{\boldsymbol{\lambda}}$ and $\widehat{\boldsymbol{\nu}}$. Thus, it is possible to define search directions for the multipliers

$$\Delta\boldsymbol{\lambda} \equiv \widehat{\boldsymbol{\lambda}} - \boldsymbol{\lambda}, \tag{1.105}$$

$$\Delta\boldsymbol{\nu} \equiv \widehat{\boldsymbol{\nu}} - \boldsymbol{\nu}. \tag{1.106}$$

Finally, the QP solution can be used to construct information needed to compute a merit function. A linear prediction for the value of the constraints is given by

$$\bar{\mathbf{s}} \equiv \mathbf{G}\mathbf{p} + \mathbf{c}. \tag{1.107}$$

Assuming we begin the iteration with an estimate for the *slack variables* $\mathbf{s}$, then it follows that

$$\Delta\mathbf{s} \equiv \bar{\mathbf{s}} - \mathbf{s} = \mathbf{G}\mathbf{p} + (\mathbf{c} - \mathbf{s}). \tag{1.108}$$

The term $(\mathbf{c} - \mathbf{s})$ has intentionally been isolated in this expression because it measures the "deviation from linearity" in the constraints $\mathbf{c}$. Notice that if the constraints $\mathbf{c}(\mathbf{x})$ are linear functions of the variables $\mathbf{x}$, then the term $(\mathbf{c} - \mathbf{s}) = \mathbf{0}$. Now the augmented search direction formed by collecting these expressions is given by

$$\begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\boldsymbol{\lambda}} \\ \bar{\boldsymbol{\nu}} \\ \bar{\mathbf{s}} \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \\ \boldsymbol{\nu} \\ \mathbf{s} \end{bmatrix} + \alpha \begin{bmatrix} \mathbf{p} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{\nu} \\ \Delta\mathbf{s} \end{bmatrix}. \tag{1.109}$$

As before, the scalar $\alpha$ defines the steplength. Observe that when a full Newton step is taken ($\alpha = 1$), the new NLP Lagrange multipliers are just the QP multipliers, i.e., $\overline{\boldsymbol{\lambda}} = \widehat{\boldsymbol{\lambda}}$ and $\overline{\boldsymbol{\nu}} = \widehat{\boldsymbol{\nu}}$.

Like all Newton methods, it is necessary to incorporate some type of globalization strategy. To this end, Gill et al. [95] suggest a modified form of the augmented Lagrangian merit function (1.85) given by

$$M(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) = F - \boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{c} - \mathbf{s}) + \frac{1}{2}(\mathbf{c} - \mathbf{s})^{\mathsf{T}} \boldsymbol{\Theta}(\mathbf{c} - \mathbf{s}), \tag{1.110}$$

where $\boldsymbol{\Theta}$ is a diagonal matrix of penalty weights. They also prove convergence for an SQP algorithm using this merit function, provided the SQP subproblem has a solution. This requires bounds on the derivatives and condition number of the Hessian matrix. Thus, the steplength $\alpha$ must be chosen to satisfy a sufficient decrease condition of the form (1.89). Most robust implementations incorporate a safeguarded line-search algorithm using quadratic and/or cubic polynomial interpolation. In addition, it is necessary that the penalty weights $\boldsymbol{\Theta}$ be chosen such that

$$M'(0) \le -\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}. \tag{1.111}$$

As a practical matter, this single condition does not uniquely define all of the penalty weights in the matrix $\boldsymbol{\Theta}$, and so a minimum norm estimate is used. Estimates for the slack variables $\mathbf{s}$ are also needed to construct the slack direction (1.108), and these quantities can be computed by minimizing $M$ as a function of $\mathbf{s}$ before taking the step. We postpone details of these calculations to the next chapter.

**Example 1.8** SQP METHOD. To illustrate the behavior of an SQP method, let us minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2 + \log(x_1 x_2) \tag{1.112}$$

subject to the constraint

$$c_1(\mathbf{x}) = x_1 x_2 \ge 1 \tag{1.113}$$

with bounds

$$0 \le x_1 \le 10, \tag{1.114}$$

$$0 \le x_2 \le 10. \tag{1.115}$$

Assume that $\mathbf{x}^{(0)} = (0.5, 2)^{\mathsf{T}}$ is an initial guess for the solution.

Figure 1.9 illustrates the iteration history when the SQP algorithm implemented in $\mathbb{SOCS}$ is applied to this problem. Notice that it is not possible to evaluate the objective function at the first predicted point (1,0). This is considered a “function error” by the $\mathbb{SOCS}$ software, and the response in the line search is to reduce the steplength. In this case, $\alpha = 9.04543273 \times 10^{-2}$ produces the point $(0.545227, 1.81909)$. Subsequent steps proceed to the solution at $(1,1)$.

## 1.14 Interior-Point Methods

Section 1.11 described using a penalty function as part of globalization strategy for Newton's method. The fundamental idea is to construct a function whose *unconstrained* minimum either is the desired constrained solution $\mathbf{x}^*$ or is related to it in a known way. Let

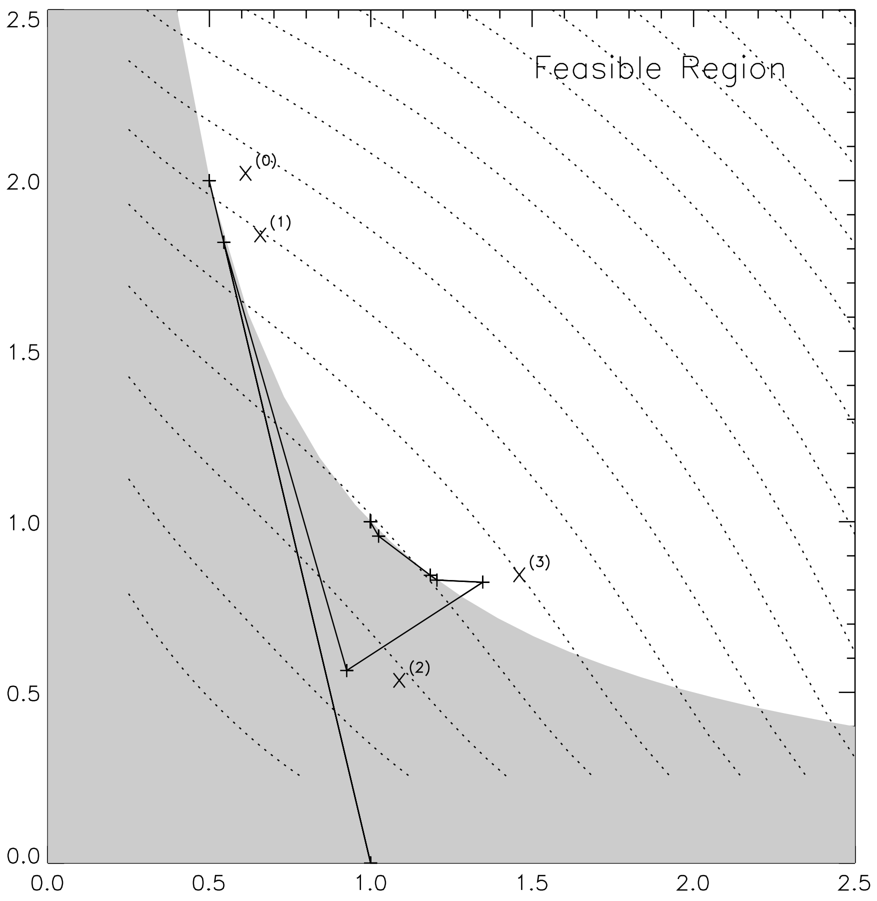


**Figure 1.9.** *NLP example.*

us focus on a particular penalty function method referred to interchangeably as a *barrier method* or *interior-point method*.

In order to illustrate the primary features let us first consider a simple example proposed by Powell [146]. For this example in two variables $(x_1, x_2)$, the objective is to minimize $x_2$ subject to the constraints

$$x_1 \cos\left(\frac{2k\pi}{m}\right) + x_2 \sin\left(\frac{2k\pi}{m}\right) \geq -1 \tag{1.116}$$

for $k = 1, 2, \ldots, m$. A solution is at $(0, -1)$ and the suggested initial guess is $\mathbf{x}^{(0)} = (.8, .5)$. This is a linear programming problem since both the objective function and the constraints are linear functions of the variables.

To understand how an optimization algorithm would behave on a problem like this it is instructive to consider a physical analogy. One can view the constraints as "fences," and the objective function as a plane surface in two dimensions. The path followed by a

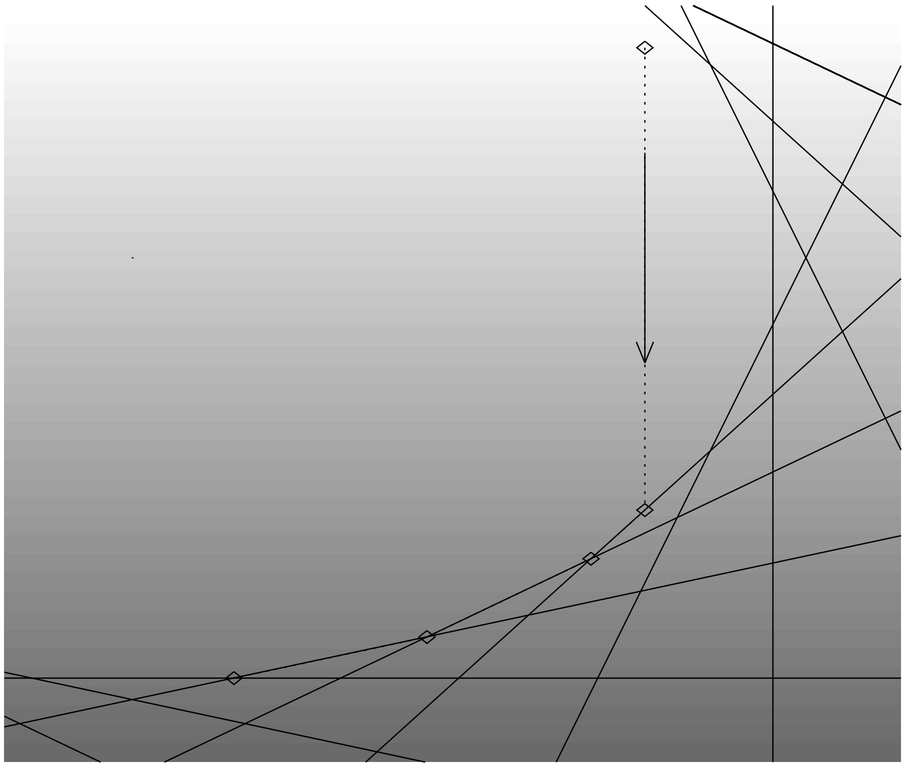


**Figure 1.10.** *Ball rolling downhill inside fences.*

ball rolling downhill inside the fenced region is illustrated in Figure 1.10. It is clear that the ball will roll downhill until it encounters a fence, and then it will follow one fence until it encounters another. Ultimately the ball will stop at the lowest point within the fenced region and will lie on one of the boundaries. Viewed mathematically, at the solution some of the constraints will be strictly feasible, i.e.,

$$x_1 \cos\left(\frac{2k\pi}{m}\right) + x_2 \sin\left(\frac{2k\pi}{m}\right) > -1,$$

and the remainder will be satisfied as equalities, i.e.,

$$x_1 \cos\left(\frac{2k\pi}{m}\right) + x_2 \sin\left(\frac{2k\pi}{m}\right) = -1.$$

Constraints that are strictly feasible are said to be *inactive* (cf. (1.72)) and the others are said to be *active* (cf. (1.72)). An obvious computational hurdle is to correctly identify which constraints are active and which are inactive. In this physical example at each fence "corner" the active set changes. Clearly the number of constraints $m$ will alter the difficulty of this linear programming problem.

The *active set method* discussed in Section 1.10 solves a sequence of equality constrained problems. In contrast, penalty function methods replace the constrained optimization problem by a *sequence* of unconstrained problems, and as such were referred to as *sequential unconstrained minimization techniques* (SUMT) by Fiacco and McCormick [79]. Thus, we can minimize the function

$$\beta(\mathbf{x},\mu) = x_2 - \mu \sum_{k=1}^{m} \ln b_k(\mathbf{x}), \tag{1.117}$$

where

$$b_k(\mathbf{x}) = x_1 \cos\left(\frac{2k\pi}{m}\right) + x_2 \sin\left(\frac{2k\pi}{m}\right) + 1. \tag{1.118}$$

Notice that the term $\ln b_k(\mathbf{x})$ becomes large if the constraints $b_k(\mathbf{x})$ are near zero, and consequently serves as a "barrier" to the minimization. Thus, when minimizing $\beta(\mathbf{x},\mu)$ iterates will be driven away from the constraint boundary at $b_k(\mathbf{x}) = 0$. The influence of the barrier term is controlled by the parameter $\mu$ called the *barrier parameter*, and the function $\beta(\mathbf{x},\mu)$ is called the log-barrier or simply *barrier function*.

How should the barrier parameter be chosen? Since we would like to approximate the original constrained solution we would like to have the barrier parameter $\mu$ be "small." Figure 1.11 illustrates the active set method when compared to the barrier method when the value of $\mu = 10^{-3}$. There are two important observations here. First, the path for the barrier method and the active set method are nearly the same. Thus, by starting with a small value for $\mu$ we will replicate the undesirable properties of an active set method. Second, while the "downhill" direction is well defined for the barrier objective, there is little information horizontally. In fact, since the contours are nearly parallel to the $x_1$-axis almost any value for $x_1$ will yield the same objective. In other words, the unconstrained barrier objective is nearly singular!

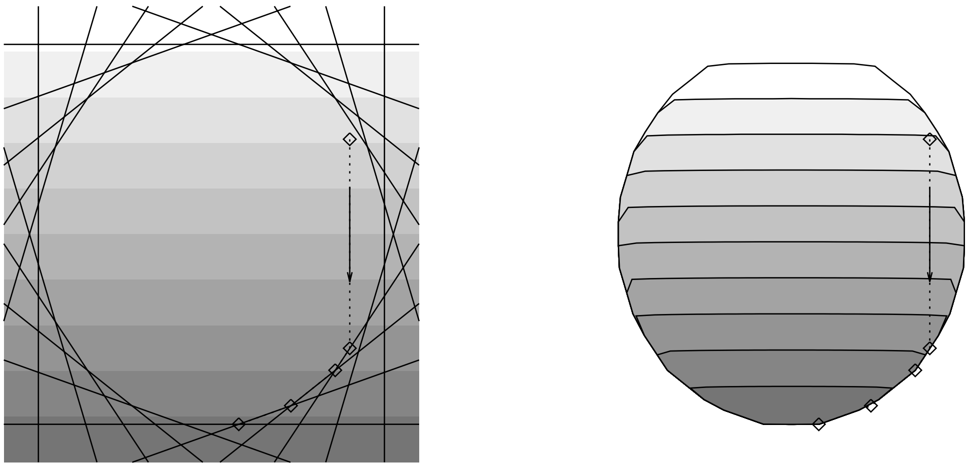


**Figure 1.11.** *Active set versus barrier method with small $\mu$.*

On the other hand suppose we choose a "large" value for $\mu$ and then minimize the barrier function. The solution to this problem serves as a good initial guess for a second

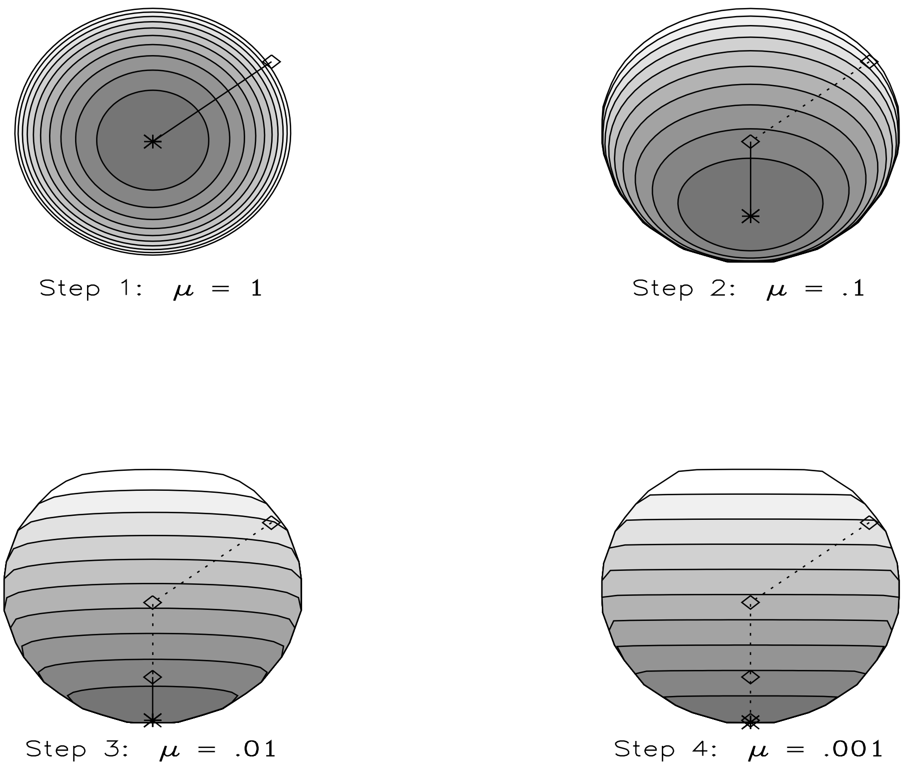


**Figure 1.12.** *Barrier method steps.*

minimization with a smaller value of $\mu$. Figure 1.12 illustrates the contours of the log-barrier function for this example when $m = 20$. The contours are plotted for four different values of the barrier parameter, namely $\mu = 1, .1, .01, .001$. The figure also illustrates the steps taken as we progress from one barrier minimizer to another. Notice when the barrier parameter is large ($\mu = 1$) the contours of the log-barrier function are almost circular about the minimum point, whereas they become progressively more severe as $\mu$ is reduced. What is more important is that the sequence of iterates tends to approach the solution along a path that is "orthogonal" to the constraints instead of "tangential" to them. This observation illustrates the barrier parameter reduction strategy. We first minimize the barrier function with a "large" value of $\mu$. The solution to this problem then serves as an initial guess for another barrier minimization with a smaller value of $\mu$. In fact, by solving a sequence of unconstrained problems, it can be shown that the unconstrained solution points approach the constrained solution as $\mu$ goes to zero. Furthermore, notice that the unconstrained minimizers in Figure 1.12 all lie inside the feasible region; hence the method is called an *interior-point algorithm*. For obvious reasons the trajectory followed by the iterates is referred to as the *central path* and is illustrated in Figure 1.13.

It should be noted that the logarithmic barrier function (1.117) is not the only choice for a penalty function. For example, Fiacco and McCormick [79] also suggest another

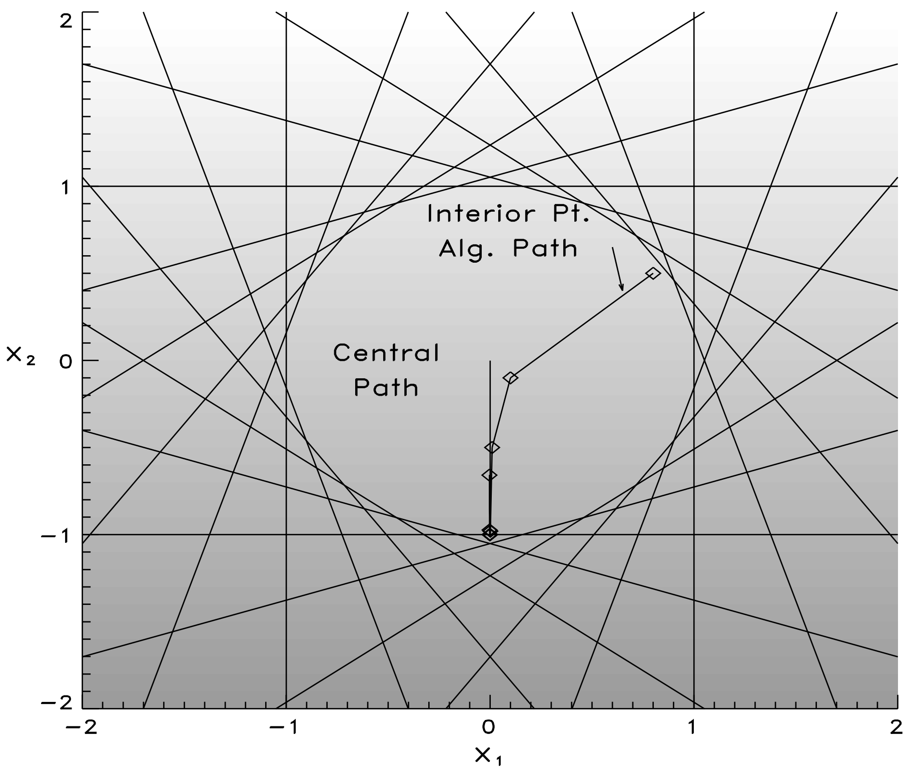


**Figure 1.13.** *Interior-point algorithm.*

interior-point penalty function

$$P(\mathbf{x},\mu) = F(\mathbf{x}) + \mu^2 \sum_{k=1}^{m} \frac{1}{c_k(\mathbf{x})} \tag{1.119}$$

for treating inequality constraints $c_k(\mathbf{x}) \geq 0$. Furthermore, the quadratic penalty function introduced in (1.84) can be extended to inequality constraints

$$P(\mathbf{x},\mu) = F(\mathbf{x}) + \frac{1}{2\mu} \sum_{k=1}^{m} \{\min[0, c_k(\mathbf{x})]\}^2. \tag{1.120}$$

When (1.120) is minimized for a sequence of penalty weights $\mu \to 0$, the unconstrained minimizers approach the solution from outside the feasible region, and thus it is referred to as an *exterior point penalty function.* Unfortunately, the Hessian matrix is both discontinuous and ill-conditioned as $\mu \to 0$, and consequently the computational effectiveness is significantly degraded.

Early implementations of interior-point methods applied standard unconstrained minimization techniques to the barrier objective (1.117). Unfortunately, there are a number of subtle issues that can significantly degrade the computational performance of a naive barrier implementation. In particular, some care must be exercised when computing the iterates to avoid ill-conditioning for small values of $\mu$, and an approach that accurately solves the underlying linear equations will be presented in Section 2.11. For efficiency we do not accurately minimize the barrier function before reducing $\mu$ and robustness is enhanced by a number of globalization techniques.

## 1.15 Mathematical Program with Complementarity Conditions

Many practical problems require modeling disjunctive behavior. Suppose, we have a model that is represented by three sets of variables $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$. Furthermore suppose some of the variables ($\mathbf{x}$ and $\mathbf{y}$) have a complementarity relationship such that either one or both must be at its bound. Formally, we can state this complementarity relationship by requiring that

$$\begin{aligned} x_k &= 0 \qquad OR \qquad y_k = 0, \\ \mathbf{x} &\geq \mathbf{0}, \qquad\qquad \mathbf{y} \geq \mathbf{0} \end{aligned}$$

for all $k$ variables in the set. This complementarity relationship is usually denoted

$$\mathbf{0} \leq \mathbf{x} \perp \mathbf{y} \geq \mathbf{0}. \tag{1.121}$$

There are several equivalent ways to impose the complementarity relationship:

$$\mathbf{x}^{\mathsf{T}}\mathbf{y} = 0, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}, \tag{1.122a}$$

$$x_k y_k = 0 \quad \forall k, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}, \tag{1.122b}$$

$$x_k y_k \leq 0 \quad \forall k, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}. \tag{1.122c}$$

A *mathematical program with complementarity conditions (MPCC)* requires finding the variables $(\mathbf{x},\mathbf{y},\mathbf{z})$ to minimize

$$F(\mathbf{x},\mathbf{y},\mathbf{z}) \tag{1.123a}$$

subject to the constraints

$$\mathbf{b}(\mathbf{x},\mathbf{y},\mathbf{z}) = \mathbf{0}, \tag{1.123b}$$

$$\mathbf{c}(\mathbf{x},\mathbf{y},\mathbf{z}) \geq \mathbf{0}, \tag{1.123c}$$

$$\mathbf{0} \leq \mathbf{x} \perp \mathbf{y} \geq \mathbf{0}. \tag{1.123d}$$

Now, an important property of the MPCC (1.123a)–(1.123d) is that it can be viewed as a *bilevel optimization problem*, that is, an optimization problem within an optimization problem. Specifically the solution to the MPCC (1.123a)–(1.123d) is equivalent to finding the variables $(\mathbf{x},\mathbf{y},\mathbf{z})$ to minimize

$$F(\mathbf{x},\mathbf{y},\mathbf{z}) \tag{1.124a}$$

subject to the constraints

$$\mathbf{b}(\mathbf{x},\mathbf{y},\mathbf{z}) = \mathbf{0}, \tag{1.124b}$$

$$\mathbf{c}(\mathbf{x},\mathbf{y},\mathbf{z}) \geq \mathbf{0}, \tag{1.124c}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{1.124d}$$

$$\mathbf{y} = \arg\left\{ \min_{\widehat{\mathbf{y}}} \mathbf{x}^{\mathsf{T}}\widehat{\mathbf{y}} \quad : \quad \widehat{\mathbf{y}} \geq \mathbf{0} \right\}. \tag{1.124e}$$

Observe that (1.124e) requires solving the inner level optimization problem for the variables $\widehat{\mathbf{y}}$ that minimize

$$\mathbf{x}^{\mathsf{T}}\widehat{\mathbf{y}} \tag{1.125a}$$

subject to the constraints

$$\widehat{\mathbf{y}} \geq \mathbf{0} \tag{1.125b}$$

treating $\mathbf{x}$ as fixed. In other words the variables $\widehat{\mathbf{y}}$ are constrained to satisfy the optimality conditions for the inner level optimization problem. When viewed in this manner the KKT optimality conditions for the inner level problem are referred to as equilibrium conditions, and the overall problem is referred to as a *mathematical program with equilibrium constraints (MPEC)*. Observe that the inner level problem (1.125a)–(1.125b) in this development is just a linear programming problem in the variables $\widehat{\mathbf{y}}$. More general forms for the inner level problem are possible leading to different equilibrium conditions. Consequently these problems are often referred to as mathematical programs with *variational inequalities*.

## 1.15.1 The Signum or Sign Operator

Suppose the problem functions involve the signum or sign function given by

$$sgn[f(x)] = \begin{cases} 1 & \text{if } f(x) > 0, \\ -1 & \text{if } f(x) < 0, \\ 0 & \text{if } f(x) = 0. \end{cases} \tag{1.126}$$

This disjunctive behavior can be modeled by introducing an additional variable $y$, where

$$y = sgn[f(x)]. \tag{1.127}$$

It then follows that for a fixed value of $x$ one can compute $y$ to minimize the objective

$$-f(x)y \tag{1.128a}$$

subject to the constraints

$$-1 \leq y \leq 1. \tag{1.128b}$$

Since (1.128a)–(1.128b) is just an NLP in the single variable $y$, the Lagrangian (1.55) is

$$L(y) = -f(x)y - p(y+1) - q(1-y), \tag{1.129a}$$

where $p$ and $q$ are the Lagrange multipliers. It then follows that the necessary conditions for optimality are just

$$-f(x) - p + q = 0, \tag{1.129b}$$

$$p(y+1) = 0, \tag{1.129c}$$

$$q(1-y) = 0, \tag{1.129d}$$

$$p \geq 0, \tag{1.129e}$$

$$q \geq 0. \tag{1.129f}$$

The first optimality condition (1.129b) defines a stationary point of the Lagrangian $\nabla_y L = 0$, and the complementarity conditions are given by (1.129c) and (1.129d). Observe that the complementarity conditions (1.129c) and (1.129d), which correspond to (1.122b), can also be written as

$$0 \le p \perp (y+1) \ge 0, \tag{1.130}$$

$$0 \le q \perp (1-y) \ge 0. \tag{1.131}$$

### 1.15.2 The Absolute Value Operator

Suppose the problem functions involve the absolute value operation $|f(x)|$. To model this discontinuous behavior one can introduce two variables where the variable

$$z = |f(x)| = f(x)y \tag{1.132}$$

and the second variable $y = sgn[f(x)]$ is chosen as before to minimize the objective

$$-f(x)y \tag{1.133a}$$

subject to the constraints

$$-1 \le y \le 1. \tag{1.133b}$$

After using (1.129b)–(1.129f) and eliminating the variable $y$ one obtains

$$z = p+q, \tag{1.133c}$$

$$f(x) = p-q, \tag{1.133d}$$

$$0 \le p \perp q \ge 0. \tag{1.133e}$$

### 1.15.3 The Maximum Value Operator

When the problem functions involve the maximum value operation $\max[f(x), a]$ for some constant $a$, the discontinuous behavior can be modeled by defining the variable

$$z = \max[f(x), a] = f(x) + [a - f(x)]y, \tag{1.134}$$

where the second variable $y$ is chosen to minimize the objective

$$[f(x) - a]y \tag{1.135a}$$

subject to the constraints

$$0 \le y \le 1. \tag{1.135b}$$

As before (1.135a)–(1.135b) is an NLP in the single variable $y$, with the Lagrangian given by

$$L(y) = [f(x) - a]y - py - q(1-y), \tag{1.136a}$$

where $p$ and $q$ are the Lagrange multipliers. It then follows that the necessary conditions for optimality are just

$$[f(x) - a] - p + q = 0, \tag{1.136b}$$

$$py = 0, \tag{1.136c}$$

$$q(1-y) = 0, \tag{1.136d}$$

$$p \ge 0, \tag{1.136e}$$

$$q \ge 0. \tag{1.136f}$$

Here the complementarity conditions can be written as

$$0 \le p \perp y \ge 0,$$
$$0 \le q \perp (1-y) \ge 0.$$

### 1.15.4 The Minimum Value Operator

Similarly when problem functions involve the minimum value operation $\min[f(x),a]$ for some constant $a$, the discontinuous behavior can be modeled by writing the variable

$$z = \min[f(x),a] = f(x) + [f(x)-a]y, \tag{1.137}$$

where the second variable $y$ is chosen to minimize the objective

$$[f(x)-a]y \tag{1.138a}$$

subject to the constraints

$$-1 \le y \le 0. \tag{1.138b}$$

In this case the Lagrangian is given by

$$L(y) = [f(x)-a]y - p(y+1) - q(-y), \tag{1.139a}$$

where $p$ and $q$ are the Lagrange multipliers. It then follows that the necessary conditions for optimality are just

$$[f(x)-a] - p + q = 0, \tag{1.139b}$$
$$p(y+1) = 0, \tag{1.139c}$$
$$q(-y) = 0, \tag{1.139d}$$
$$p \ge 0, \tag{1.139e}$$
$$q \ge 0. \tag{1.139f}$$

Here the complementarity conditions can be written as

$$0 \le p \perp (1+y) \ge 0,$$
$$0 \le q \perp (-y) \ge 0.$$

### 1.15.5 Solving an MPEC

The examples presented in the previous sections are all directed toward a single goal—the formulation of an MPEC as a standard NLP. The key notion is to explicitly specify the necessary conditions for the lower level optimization problem as constraints for the "outer" NLP. For example when dealing with the signum function, three new variables were introduced, namely $(y,p,q)$, in addition to the constraints (1.129b)–(1.129f). At first, it might appear that with appropriate reformulation an MPEC can be treated as a standard NLP, using any standard algorithm. Unfortunately, the NLP constraints violate the linear independence constraint qualification (LIQC) test, as well as the weaker Mangasarian–Fromovitz constraint qualification (MFQC). A number of different strategies have been investigated to deal with this lack of regularity (cf. [7]). Since the constraint Jacobian is rank deficient and detecting rank deficiency is problematic, the solution schemes share a common theme, which is to solve a "slightly" perturbed problem. Thus one can modify the form of (1.122a) to $\mathbf{x}^\mathsf{T}\mathbf{y} = \epsilon$, or replace (1.122b) with $x_k y_k = \epsilon$, and similarly for (1.122b) enforce $x_k y_k \le \epsilon$. An effective alternative is to move the complementarity condition from the constraints into the NLP objective using an exact penalty function.

## 1.16 What Can Go Wrong

The material in this chapter has been presented to give the reader an understanding of how a method *should* work in practice. The discussion is designed to help users efficiently use high-quality optimization software to solve practical problems. However, in practice, things do go wrong! In this section, we describe a number of common difficulties that can be encountered and suggest remedial action to correct the deficiencies. For ease of presentation, it is convenient to discuss the difficulties individually. Of course, real applications may involve more than a single difficulty and the user must be prepared to correct all problems before obtaining satisfactory performance from optimization software.

### 1.16.1 Infeasible Constraints

One of the most common difficulties encountered occurs when the NLP problem has infeasible constraints. To be more precise, infeasible constraints have *no* solution.

**Example 1.9** INFEASIBLE CONSTRAINTS. For example, the following constraints have no feasible region:

$$\begin{aligned} c_1(\mathbf{x}) &= x_1^2 x_2 - 1 &\ge 0, \\ c_2(\mathbf{x}) &= x_2 &\le -\tfrac{1}{10}. \end{aligned} \tag{1.140}$$

When general-purpose NLP software is applied to such a problem, it is likely that one or more of the following symptoms will be observed:

- the QP subproblem has no solution, which can occur if the linearized constraints have no solution;
- many NLP iterations produce very little progress;
- the penalty parameters become very large, or the barrier parameter becomes small;
- the condition number of the projected Hessian matrix becomes large; or
- the Lagrange multipliers become large.

Although most robust software will attempt to "gracefully" detect this situation, ultimately the only remedy is to reformulate the problem!

### 1.16.2 Rank-Deficient Constraints

In contrast to the previous situation, it is possible that the constraints are consistent; that is, they do have a solution. However, at the solution, the Jacobian matrix may be either ill-conditioned or rank deficient.

**Example 1.10** RANK-DEFICIENT CONSTRAINTS. For an example originally suggested by Kuhn and Tucker [126], consider minimizing

$$F(\mathbf{x}) = (x_1 - 2)^2 + x_2^2 \tag{1.141}$$

subject to the constraints

$$\begin{array}{rclcl} c_1(\mathbf{x}) & = & x_1 & \geq & 0, \\ c_2(\mathbf{x}) & = & x_2 & \geq & 0, \\ c_3(\mathbf{x}) & = & (1-x_1)^3 - x_2 & \geq & 0. \end{array} \tag{1.142}$$

The solution to this problem is $\mathbf{x}^* = (1,0)^{\mathsf{T}}$ and, clearly, $\mathbf{c}(\mathbf{x}^*) = \mathbf{0}$. However, at the solution, the active set is $\mathcal{A}^* = \{c_2, c_3\}$ and the Jacobian matrix corresponding to these constraints is rank deficient, i.e.,

$$\widetilde{\mathbf{G}} = \begin{bmatrix} 0 & 1 \\ -3(1-x_1)^2 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}. \tag{1.143}$$

This example violates the linear independent constraint qualification test.

When general-purpose NLP software is applied to such a problem, it is likely that one or more of the following symptoms will be observed:

- many NLP iterations produce very little progress;
- the penalty parameters become very large or the barrier parameter becomes very small;
- the Lagrange multipliers become large; or
- the rank deficiency in $\widetilde{\mathbf{G}}$ is detected.

Unfortunately, detecting rank deficiency in the Jacobian is not a straightforward numerical task! Consequently, it is quite common to confuse this difficulty with inconsistent constraints. Again, the remedy is to reformulate the problem!

### 1.16.3 Constraint Redundancy

A third type of difficulty can occur when the problem formulation contains constraints that are redundant. Thus, although the constraints have a solution, they may be unnecessary to the problem formulation. The following two examples illustrate the situation.

**Example 1.11** REDUNDANCY (FULL-RANK). Minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2 \tag{1.144}$$

subject to the constraint

$$c_1(\mathbf{x}) = x_1 - x_2 = 0. \tag{1.145}$$

In this case, the unconstrained solution is $\mathbf{x}^* = (0,0)^{\mathsf{T}}$, and the constraint is trivially satisfied. Constraint $c_1$ can be eliminated from the formulation without changing the answer.

**Example 1.12** REDUNDANCY (RANK-DEFICIENT). Minimize

$$F(\mathbf{x}) = (x_1 - 2)^2 + x_2^2 + x_3^2 \tag{1.146}$$

subject to the constraints

$$\begin{aligned} c_1(\mathbf{x}) &= \quad x_1 + x_2 - 1 &= 0, \\ c_2(\mathbf{x}) &= 2x_1 + 2x_2 - 2 &= 0. \end{aligned} \tag{1.147}$$

In this case, constraint $c_2 = 2c_1$ and, clearly, one of the constraints is redundant. Note also that the Jacobian matrix $\widetilde{\mathbf{G}}$ is rank deficient.

Symptoms indicating redundancy such as in Example 1.11 include

- Lagrange multipliers near zero and
- difficulty detecting the active set.

On the other hand, constraint redundancy such as in Example 1.12 is likely to exhibit symptoms similar to the rank-deficient cases discussed previously. Obviously, the remedy is to reformulate the problem and eliminate the redundant constraints!

### 1.16.4 Discontinuities

Perhaps the single biggest obstacle encountered in the practical application of NLP methods is the presence of discontinuous behavior. All of the numerical methods described assume that the objective and constraint functions are continuous and differentiable. When discontinuities are present in the problem functions, the standard quadratic/linear models that form the basis for most NLP methods are no longer appropriate. In fact, the KKT necessary conditions do not apply!

Unfortunately, in practice, there are many common examples of discontinuous behavior. Typical sources include

- branching caused by IF tests in code;
- absolute value, max, and min functions;
- linear interpolation of tabular data; and
- "internal" iteration loops, e.g., adaptive quadrature, root finding.

The most common symptoms of discontinuous functions are

- slow convergence or divergence,
- small steps ($\alpha \approx 0$) in the line search, and
- possible ill-conditioning of the Hessian matrix.

**Example 1.13** TREATING ABSOLUTE VALUES.

| *Bad Formulation* | *Good Formulation* |
|---|---|
| Find $(x,y)$ to minimize $$\lvert x\rvert$$ subject to $$y - x^2 = 1.$$ | Find $(x_1,x_2,y)$ to minimize $$x_1 + x_2$$ subject to $$\begin{aligned} y-(x_1-x_2)^2 &= 1,\\ x_1 &\geq 0,\\ x_2 &\geq 0.\end{aligned}$$ |

The "trick" motivated by the MPEC formulation (1.133c)–(1.133e) is to replace $x \to (x_1 - x_2)$ and then observe that $|x| \to (x_1 + x_2)$. The optimal solution for the preferred formulation is $\mathbf{x}^* = (0,0,1)$. As expected, the preferred formulation is solved by $\mathbb{SOCS}$ in 47 evaluations. In contrast, the original formulation requires 117 function evaluations and terminates with a small step warning. A more realistic illustration of this difficulty will be presented in Example 6.17. Although complementarity is not explicitly enforced in this example in general it may be necessary to augment the objective with a term $\rho x_1 x_2$ to ensure that $0 \leq x_1 \perp x_2 \geq 0$.

**Example 1.14** MINIMAX PROBLEMS. Minimizing maximum values (*minimax problems*).

| *Bad Formulation* | *Good Formulation* |
|---|---|
| Find $(x_1,x_2)$ to minimize $$\max_k \lvert y_k - d_k\rvert$$ for $k = 1,2,3$, where $$\begin{aligned} y_k &= x_1 + x_2 k,\\ d &= (1,1.5,1.2).\end{aligned}$$ | Find $(x_1,x_2,s)$ to minimize $$s$$ for $k = 1,2,3$, where $$\begin{aligned} y_k - d_k - s &\leq 0,\\ y_k - d_k + s &\geq 0.\end{aligned}$$ |

The trick here is to introduce the slack variable $s$ as an absolute bound on the quantities being minimized, i.e., $|y_k - d_k| \leq s$. The absolute value function is then eliminated since $-s \leq y_k - d_k \leq s$ can be rewritten as two inequality constraints. The optimal solution for the preferred formulation is $\mathbf{x}^* = (1.1,0.1,0.2)$. As expected, the preferred formulation is solved by $\mathbb{SOCS}$ in 30 evaluations. In contrast, the original formulation terminates with a small step warning after 271 evaluations at the point $\mathbf{x}^* = (1.09963,0.100148)$.

In subsequent chapters, a great deal of attention will be given to the correct formulation of optimal control applications such that discontinuous behavior can be avoided. In fact, Section 4.12 is devoted entirely to one such application. Nevertheless, it is worth

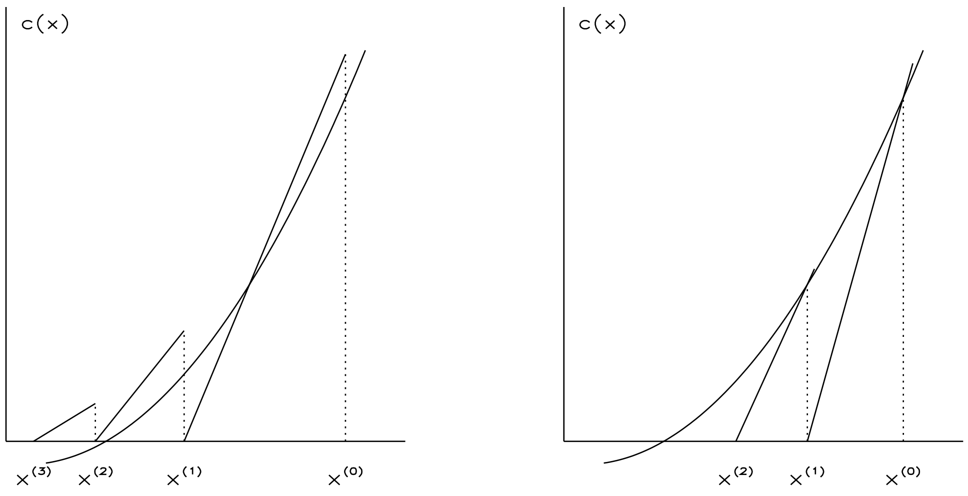


**Figure 1.14.** *The effects of noise.*

mentioning that specific remedial action for these problems includes

- recoding and/or reformulating to eliminate branching, absolute value, max, and min problems (cf. Examples 1.13 and 1.14);
- interpolation or approximation of tabular data using functions with continuity through second derivatives (cf. Example 6.4);
- avoiding the use of variable-step, variable-order integration when optimizing; and
- reformulation of “internal” iterations (e.g., Kepler’s equation) using additional “external” NLP constraints (cf. Example 6.6).

Because discontinuous behavior plays such a major role in the success or failure of an NLP application, it is worthwhile to understand the effects of “noise” on the Newton iteration. Figure 1.14 illustrates a typical Newton iteration in the presence of two types of noise. Let us assume that we are trying to solve a constraint $c(x) = 0$. However, because of errors in the function evaluation, what the Newton method “sees” is the contaminated result $c(x) + \epsilon$. Because the intent is to drive the constraint to zero, ultimately the value will be “swamped” by the noise $\epsilon$ and we expect that the iteration will *not* converge! This situation is shown in the left portion of Figure 1.14. On the other hand, suppose that the value of the constraint is correct but the slope is contaminated by noise as illustrated in the right portion of Figure 1.14. Thus, instead of computing the true slope $c'(x)$, the Newton iterate actually uses the value $c'(x) + \epsilon$. In this case, we can expect the iteration to locate a solution, but at a degraded *rate* of convergence. We can carry this analysis further by viewing optimization as equivalent to solving $\nabla_x L = 0$. Then, by analogy, if the gradient is contaminated by noise, $\nabla_x L + \epsilon$, the iteration will not converge, because ultimately the true gradient will be dominated by the noise. On the other hand, if the Hessian information is noisy, $\nabla^2_{xx} L + \epsilon$, we expect the *rate* of convergence to be degraded. It is worth noting that an approximation to the slope (e.g., secant or quasi-Newton) in the absence of noise changes the Newton iterates in a similar fashion.

### 1.16.5 Scaling

Scaling affects everything! Poor scaling can make a good algorithm bad. Scaling changes the convergence rate, termination tests, and numerical conditioning. The most common way to scale a problem is to introduce variable scaling of the form

$$\tilde{x}_k = u_k x_k + r_k \tag{1.148}$$

for $k = 1,\ldots,n$. In this expression, the scaled variables $\tilde{x}_k$ are related to the unscaled variables by the variable scale weights $u_k$ and shifts $r_k$. In like fashion, the objective and constraints are commonly scaled using

$$\tilde{c}_k = w_k c_k, \tag{1.149}$$

$$\tilde{F} = w_0 F \tag{1.150}$$

for $k = 1,\ldots,m$. The intent is to let the optimization algorithm work with the "well-scaled" quantities $\tilde{x}$, $\tilde{c}$, and $\tilde{F}$ in order to improve the performance of the algorithm. Unfortunately, it is not at all clear what it means to be well scaled!

Nevertheless, conventional wisdom suggests that some attempt should be made to construct a well-scaled problem and, consequently, we give the following *hints* (*not rules*) for good scaling:

- normalize the independent variables to have the same "range," e.g.,

$$0 \le \tilde{x}_k \le 1;$$

- normalize the dependent functions to have the same magnitude, i.e.,

$$F \approx c_1 \approx c_2 \approx \cdots \approx c_m \approx 1;$$

- normalize the rows and columns of the Jacobian to be of the same magnitude;
- scale the dependent functions so that the Lagrange multipliers are one:

$$|\lambda_1| \approx |\lambda_2| \approx \cdots \approx |\lambda_m| \approx 1;$$

- scale the problem so that the condition number of the projected Hessian matrix is close to one;
- scale the problem so that the condition number of the KKT matrix (1.66) is close to one.

Constructing an automatic computational procedure that will meet all of these goals is probably not possible. Furthermore, even if a strategy could be devised that produces "good scaling" at one point, say $\mathbf{x}$ for nonlinear functions, this may be "bad scaling" at another point $\bar{\mathbf{x}}$. As a practical matter in the $\mathbb{SOCS}$ software, we attempt to produce a well-scaled Jacobian at the user-supplied initial point, but also allow the user to override this scaling by input. Details of the approach are deferred to Section 4.8. For more helpful discussion on this subject, the reader is referred to [99, Chapter 8].

### 1.16.6 Nonunique Solution

For most of the difficulties presented an optimization algorithm will either fail to produce a solution and/or struggle to find one. But what if the problem formulation has too many solutions?

**Example 1.15** NONUNIQUE SOLUTION. Minimize

$$F(\mathbf{x}) = (x_1 - x_2)^2. \tag{1.151}$$

In this case, there are an infinite number of solutions corresponding to points on the line $x_1 = x_2$. Everywhere on this line the objective function $F = 0$ and the gradient $\mathbf{g} = \mathbf{0}$; however, clearly the sufficient condition (1.61) is violated.

The projected Hessian matrix must be positive definite only in the neighborhood of the solution, and even a well-posed problem may encounter an indefinite or negative definite projected Hessian during intermediate iterations. Consequently most computational algorithms try to "fix" the problem by making the Hessian approximation positive definite. Using a positive definite quasi-Newton update such as the BFGS (1.50) and/or a trust-region strategy (1.93) are two such "fixes." Unfortunately when the algorithm uses this strategy it may converge with no indication of trouble. However, if the iterations are repeated with a different initial guess, it is quite likely that a different solution will be computed. In fact, there may be no obvious symptoms of difficulty reported by the software unless the user happens to notice nonrepeatability in the solution! Detecting the difficulty and then correcting it requires user insight. Example 6.11 illustrates this situation.

## 1.17 Derivative Approximation by Finite Differences

Derivative information is an important ingredient in all optimization algorithms, and consequently it is useful to review one of the most prevalent methods for computing these quantities. Consider the Taylor series expansion about the point $x$

$$f(x+\delta) = f(x) + f'(x)\delta + \frac{1}{2} f''(x)\delta^2 + \cdots, \tag{1.152}$$

where the perturbation size $\delta > 0$. Dividing through by $\delta$ yields

$$\begin{aligned} f'(x) &= \frac{f(x+\delta) - f(x)}{\delta} - \frac{\delta}{2} f''(x) \ldots \\ &= \frac{f(x+\delta) - f(x)}{\delta} + \mathcal{O}(\delta). \end{aligned} \tag{1.153}$$

The terms denoted by $\mathcal{O}(\delta)$ are said to be of *order* $\delta$. If the $\mathcal{O}(\delta)$ terms are ignored, one obtains the *forward difference approximation*

$$f'(x) \approx \frac{f(x+\delta) - f(x)}{\delta}. \tag{1.154}$$

Because the Taylor series expansion is truncated after the first term, it is common to refer to

$$\eta_T = \mathcal{O}(\delta) \approx \frac{\delta}{2} |f''(x)| \tag{1.155}$$

as the *truncation error*. An estimate for the truncation error is provided by the magnitude of the first ignored term in the expansion, $\frac{\delta}{2}|f''(x)|$. This is considered a *first order* approximation, because $\delta$ appears to the first power, and in general an approximation is said to have order $k$ if $\delta^k$ appears in the truncation error. On a digital computer the difference approximation (1.154) must be evaluated using finite precision arithmetic, which introduces a second source of error. It is referred to as roundoff or *cancellation error* and can be approximated by

$$\eta'_1 = \frac{2\epsilon_a}{\delta}, \tag{1.156}$$

where $\epsilon_a$ is an estimate of the absolute function precision. For the sake of reference a number of common finite difference approximations are summarized in Table 1.7.

**Table 1.7.** *Finite difference derivative approximations.*

First Order (Forward Difference) First Derivative: ($\eta'_1 = \frac{2\epsilon_a}{\delta}$)

$$f' = \frac{1}{\delta}\big[f(x+\delta) - f(x)\big]. \tag{1.157}$$

Second Order (Central Difference) First Derivative: ($\eta'_2 = \frac{\epsilon_a}{\delta}$)

$$f' = \frac{1}{2\delta}\big[f(x+\delta) - f(x-\delta)\big]. \tag{1.158}$$

Fourth Order First Derivative: ($\eta'_4 = \frac{3\epsilon_a}{2\delta}$)

$$f' = \frac{1}{12\delta}\big[-f(x+2\delta) + 8f(x+\delta) - 8f(x-\delta) + f(x-2\delta)\big]. \tag{1.159}$$

Second Order Second Derivative: ($\eta''_2 = \frac{4\epsilon_a}{\delta^2}$)

$$f'' = \frac{1}{\delta^2}\big[f(x+\delta) - 2f(x) + f(x-\delta)\big]. \tag{1.160}$$

Fourth Order Second Derivative: ($\eta''_4 = \frac{16\epsilon_a}{3\delta^2}$)

$$f'' = \frac{1}{12\delta^2}\big[-f(x+2\delta) + 16f(x+\delta) - 30f(x) + 16f(x-\delta) - f(x-2\delta)\big]. \tag{1.161}$$

Second Order Third Derivative: ($\eta'''_2 = \frac{3\epsilon_a}{\delta^3}$)

$$f''' = \frac{1}{2\delta^3}\big[-f(x-2\delta) + 2f(x-\delta) - 2f(x+\delta) + f(x+2\delta)\big]. \tag{1.162}$$

A reasonable choice for the perturbation size $\delta$ can be computed by using the finite difference error estimates. An estimate for the total finite difference error in the case of a forward difference first derivative is

$$\eta = \eta_T + \eta'_1 = \frac{\delta}{2}|f''(x)| + \frac{2\epsilon_a}{\delta}. \tag{1.163}$$

This error is minimized by choosing

$$\delta_1^* = 2\sqrt{\frac{\epsilon_a}{|f''(x)|}}. \tag{1.164}$$

In a similar fashion the error in a central difference first derivative is approximated by

$$\eta = \frac{\delta^2}{6}|f'''(x)| + \frac{\epsilon_a}{\delta}, \tag{1.165}$$

which can be minimized by setting

$$\delta_2^* = \left(\frac{3\epsilon_a}{|f'''(x)|}\right)^{\frac{1}{3}}. \tag{1.166}$$

At first blush, making use of these expressions in a computational algorithm appears problematic. If the goal is to estimate a first derivative using the forward difference formula (1.157), according to (1.164) an estimate for the second derivative $f''(x)$ is also needed! Similarly if the goal is to construct a central difference first derivative using (1.158), one needs a third derivative in order to minimize the error according to (1.166). Furthermore these estimates change as the point $x$ changes. Fortunately for most practical applications a "good" estimate for the perturbation size usually suffices. It is reasonable to compute the optimal perturbation size only at a "typical" value for $x$ rather than at every optimization iterate. In the $\mathbb{SOCS}$ software, the perturbation size is estimated at the initial iterate $x^{(0)}$, and thereafter held fixed for subsequent iterations unless there is some other indication of suspicious behavior.

Suppose we are trying to compute a central difference first derivative using (1.158). How does one compute the third derivative $f'''(x)$ needed in (1.166)? Fortunately a finite difference approximation to $f'''(x)$ is given by (1.162) for some perturbation size, say $h$. This estimate requires evaluating the function at four perturbed points in addition to the nominal. Now, the third derivative approximation has error in it which is related to the trial perturbation $h$. So, after evaluating $f'''(x)$ using the trial perturbation, a new estimate for the trial perturbation can be computed. Typically, this process is repeated only once or twice, before an acceptable accuracy is obtained in the finite difference third derivative $f'''(x)$. Finally, the third derivative can be used to construct the optimal central difference perturbation from (1.166). Gill, Murray, and Wright [99, Chapter 8] describe an algorithm for constructing forward difference perturbation size estimates, which can be extended to accommodate central difference estimates. It should be emphasized that one first computes the third derivative using a sequence of trial perturbations, and only then is an estimate for the first derivative perturbation (1.166) constructed. As described, the procedure is applicable for computing derivatives of a single function with respect to a single variable. Extending the approach to many variables, that is, for constructing partial derivatives, is trivial and will be discussed in Section 2.2. In contrast, when differentiating many functions using the same finite difference perturbation, there may well be some loss of accuracy in the approximate derivatives.

### 1.17.1 Difference Estimates in Differential Equations

In addition to their use within a nonlinear optimization algorithm, finite difference derivative estimates may appear in another context. Suppose the function $f(x)$ represents a physical quantity defined in the region $x_L \le x \le x_U$. The mathematical description for a physical process typically involves derivatives, e.g., $f'(x)$. One way to treat this situation is to

**Table 1.8.** *Difference estimates in differential equations.*

*Second Order (Three Point) First Derivative*

$$\begin{array}{lcll}
f_1' & = & \frac{1}{2\delta}\left[-3f_1+4f_2-f_3\right], & \\
f_k' & = & \frac{1}{2\delta}\left[f_{k+1}-f_{k-1}\right], & k=2,\ldots,(n-1), \\
f_n' & = & \frac{1}{2\delta}\left[f_{n-2}-4f_{n-1}+3f_n\right]. &
\end{array}$$

*Fourth Order (Five Point) First Derivative*

$$\begin{array}{lcll}
f_1' & = & \frac{1}{12\delta}\left[-25f_1+48f_2-36f_3+16f_4-3f_5\right], & \\
f_2' & = & \frac{1}{12\delta}\left[-3f_1-10f_2+18f_3-6f_4+f_5\right], & \\
f_k' & = & \frac{1}{12\delta}\left[-f_{k+2}+8f_{k+1}-8f_{k-1}+f_{k-2}\right], & k=3,\ldots,(n-2), \\
f_{n-1}' & = & \frac{1}{12\delta}\left[-f_{n-4}+6f_{n-3}-18f_{n-2}+10f_{n-1}+3f_n\right], & \\
f_n' & = & \frac{1}{12\delta}\left[+3f_{n-4}-16f_{n-3}+36f_{n-2}-48f_{n-1}+25f_n\right]. &
\end{array}$$

*Second Order (Three Point) Second Derivative*

$$\begin{array}{lcll}
f_1'' & = & \frac{1}{\delta^2}\left[f_1-2f_2+f_3\right], & \\
f_k'' & = & \frac{1}{\delta^2}\left[f_{k-1}-2f_k+f_{k+1}\right], & k=2,\ldots,(n-1), \\
f_n'' & = & \frac{1}{\delta^2}\left[f_{n-2}-2f_{n-1}+f_n\right]. &
\end{array}$$

*Fourth Order (Five Point) Second Derivative*

$$\begin{array}{lcll}
f_1'' & = & \frac{1}{12\delta^2}\left[45f_1-154f_2+214f_3-156f_4+61f_5-10f_6\right], & \\
f_2'' & = & \frac{1}{12\delta^2}\left[10f_1-15f_2-4f_3+14f_4-6f_5+f_6\right], & \\
f_k'' & = & \frac{1}{12\delta^2}\left[-f_{k-2}+16f_{k-1}-30f_k+16f_{k+1}-f_{k+2}\right], & k=3,\ldots,(n-2), \\
f_{n-1}'' & = & \frac{1}{12\delta^2}\left[f_{n-5}-6f_{n-4}+14f_{n-3}-4f_{n-2}-15f_{n-1}+10f_n\right], & \\
f_n'' & = & \frac{1}{12\delta^2}\left[-10f_{n-5}+61f_{n-4}-156f_{n-3}+214f_{n-2}-154f_{n-1}+45f_n\right]. &
\end{array}$$

introduce a spatial discretization with $n$ "lines," i.e.,

$$x_k=(k-1)\delta+x_L, \qquad \delta=\frac{x_U-x_L}{n-1}, \qquad k=1,\ldots,n. \tag{1.167}$$

If we denote the function values at these points by $f_k=f(x_k)$, then it is tempting to use the difference estimates given in Table 1.7 with one slight modification. Specifically one must construct derivative estimates at the endpoints without utilizing values outside of the region. For example a naive application of the central difference approximation

$$f'(x_1)=\frac{1}{2\delta}\left[f_2-f_0\right] \tag{1.168}$$

requires the value $f_0=f(x_L-\delta)$. This is not desirable since the physical process may not be defined outside the region. To account for this, the difference approximations must be altered to utilize information strictly interior to the region $x_L\le x\le x_U$. It is straightforward to construct a polynomial interpolant for the nearest *interior* points $x_L\le x_k\le x_U$ and then differentiate the interpolating polynomial. For the sake of reference Table 1.8 summarizes the more commonly used formulas.

Chapter 2

# Large, Sparse Nonlinear Programming

## 2.1 Overview: Large, Sparse NLP Issues

Chapter 1 presents background on nonlinear programming (NLP) methods that are applicable to most problems. In this chapter, we will focus on NLP methods that are appropriate for problems that are both *large* and *sparse*. To set the stage for what follows, it is useful to define what we mean by large and sparse. The definition of "large" is closely tied with the tools available for solving linear systems of equations. For our purposes, we will consider a problem "small" if the underlying linear systems can be solved using a dense, direct matrix factorization. Solving a dense linear system using a direct factorization requires $\mathcal{O}(n^3)$ arithmetic operations. Thus, for today's computers, this suggests that the problem size is probably limited to matrices of order $n \approx 1000$. If the linear equations are solved using methods that exploit sparsity in the matrices but still use direct matrix factorizations, then with current computer hardware, the upper limit on the size of the problem is $n \approx 10^6$. This problem is considered "large" and is the focus of methods in this book. Although some of the NLP methods can be extended, as a rule we will not address "huge" problems for which $n > 10^6$. Typically, linear systems of this size are solved by iterative (as opposed to direct) methods. The second discriminator of interest concerns matrix sparsity. A matrix is said to be "sparse" when many of the elements are zero. For most applications of interest, the number of nonzero elements in the Hessian matrix $\mathbf{H}$ and Jacobian matrix $\mathbf{G}$ is less than $1\%$.

Many of the techniques described in Chapter 1 extend, without change, to large, sparse problems. On the other hand, there are some techniques that cannot be used for large, sparse applications. In this chapter, we will focus on those issues that are unique to the large, sparse NLP problem.

The first item concerns how to calculate Jacobian and Hessian information for large, sparse problems. All of the Newton-based methods described in Chapter 1 rely on the availability of first and second derivative information. For most practical applications, first derivatives are computed using finite difference approximations. Unfortunately, a single finite difference gradient evaluation can require $n$ additional function evaluations, and when $n$ is large, this computational cost can be excessive. Furthermore, computing second derivatives by naive extensions of the methods described in Chapter 1 becomes unattractive for a number of reasons. First, maintaining a sparse quasi-Newton approximation that is also positive definite appears unpromising. Second, even if we accept an approximation that is indefinite, one can expect to spend $\mathcal{O}(n)$ iterations before a "good" Hessian approximation is constructed—and if $n$ is large, the number of iterations can be excessive! To overcome these drawbacks, current research has focused on *pointwise quasi-Newton updates* [124], [125] and *limited memory updates*. Another alternative is to abandon the quasi-Newton approach altogether and construct this information using sparse finite differences. This technique will be explained in the next section.

The second major item addressed in this chapter concerns how to efficiently construct a Newton step when the underlying matrices are large and sparse. We first concentrate on a sparse QP algorithm and present a detailed description of the method. We then address how the method can be extended to the important sparse nonlinear least squares problem. We conclude with a discussion of a sparse interior-point (barrier) algorithm.

## 2.2 Sparse Finite Differences

### 2.2.1 Background

In general, let us define the first derivatives of a set of $\upsilon$ functions $q_i(\mathbf{x})$ with respect to $n$ variables $\mathbf{x}$ by the $\upsilon \times n$ matrix

$$\mathbf{D} \equiv \begin{bmatrix} (\nabla q_1)^\top \\ (\nabla q_2)^\top \\ \vdots \\ (\nabla q_\upsilon)^\top \end{bmatrix} = \frac{\partial \mathbf{q}}{\partial \mathbf{x}}. \tag{2.1}$$

It will also be of interest to compute second derivatives of a linear combination of the functions. In particular, we define the second derivatives of the function

$$\Omega(\mathbf{x}) = \sum_{i=1}^{\upsilon} \omega_i q_i(\mathbf{x}) \tag{2.2}$$

with respect to $n$ variables $\mathbf{x}$ by the $n \times n$ matrix

$$\mathbf{E} \equiv \nabla^2 \Omega(\mathbf{x}) = \sum_{i=1}^{\upsilon} \omega_i \nabla^2 q_i(\mathbf{x}). \tag{2.3}$$

The notion of sparse finite differencing was introduced by Curtis, Powell, and Reid [65]. They proposed that the columns of $\mathbf{D}$ be partitioned into subsets (*index sets*) $\Gamma^k$ such that each subset has at most one nonzero element per *row*. Then define the perturbation direction vector by

$$\boldsymbol{\Delta}^k = \sum_{j \in \Gamma^k} \delta_j \mathbf{e}_j, \tag{2.4}$$

where $\delta_j$ is the perturbation size for variable $j$ and $\mathbf{e}_j$ is a unit vector in direction $j$. Using this partitioning, it can be demonstrated that the central difference estimates of the first derivatives for $i = 1,\dots,\upsilon$ and $j \in \Gamma^k$ are

$$\mathbf{D}_{ij} \approx \frac{1}{2\delta_j}[q_i(\mathbf{x}+\mathbf{\Delta}^k) - q_i(\mathbf{x}-\mathbf{\Delta}^k)]. \tag{2.5}$$

In a similar fashion, second derivative estimates for $i \in \Gamma^k$ and $j \in \Gamma^\ell$ are

$$\mathbf{E}_{ij} \approx \frac{1}{\delta_i\delta_j}[\Omega(\mathbf{x}+\mathbf{\Delta}^k+\mathbf{\Delta}^\ell) + \Omega(\mathbf{x}) - \Omega(\mathbf{x}+\mathbf{\Delta}^k) - \Omega(\mathbf{x}+\mathbf{\Delta}^\ell)] \tag{2.6}$$

and

$$\mathbf{E}_{ii} \approx \frac{1}{\delta_i^2}[\Omega(\mathbf{x}+\mathbf{\Delta}^k) + \Omega(\mathbf{x}-\mathbf{\Delta}^k) - 2\Omega(\mathbf{x})]. \tag{2.7}$$

Denote the total number of index sets $\Gamma^k$ needed to span the columns of $\mathbf{D}$ by $\gamma$. Now observe that by using the same index sets for the first and second derivatives, it is possible to compute

1. central difference first derivatives using $2\gamma$ perturbations and
2. first and second derivatives using $\gamma(\gamma+3)/2$ perturbations.

Since a function evaluation can be costly, it is clear that the number of index sets $\gamma$ determines the cost of constructing this derivative information. It can be demonstrated that the maximum number of nonzeros in any row of $\mathbf{D}$ is a lower bound on the number $\gamma$. Coleman and Moré [64] have also shown that computing the smallest number of index sets is a graph-coloring problem. For most applications, acceptable approximations to the minimum number of index sets can be achieved using the "greedy algorithm" suggested by Curtis, Powell, and Reid [65]. Regardless of how the index sets are constructed, the important point is that for the optimal control problems of interest, $\gamma \ll n$. In simple terms, the number of perturbations is much smaller than the number of variables.

There are two computational issues that should be emphasized with regard to a sparse differencing implementation. First, from direct inspection of (2.3), it might appear that storage is needed for all of the individual Hessian matrices $\nabla^2 q_i(\mathbf{x})$. This is not true! In fact, the calculations can be organized such that the summation is done first, thereby making it necessary to store only the result. Second, this procedure uses a single perturbation size to construct difference approximations for many functions. As such, choosing the "best" perturbation to balance truncation and roundoff errors as described in Section 1.17 for all of the functions is a compromise, and some inaccuracy can be expected. Nevertheless, for well-scaled functions, this is not a significant issue, especially because the central difference gradient information given by (2.5) is $\mathcal{O}(\delta^2)$.

### 2.2.2 Sparse Hessian Using Gradient Differences

Equations (2.6) and (2.7) provide a means to compute an approximation to the sparse Hessian directly from the function values $\Omega(\mathbf{x})$. On the other hand if gradients are readily available, this information can be used to construct the Hessian as proposed by Coleman, Garbow, and Moré [63]. Since the approximate Hessian $\mathbf{E}$ is a symmetric matrix of order $n$, the goal is to obtain a set of vectors $\Delta^1, \Delta^2, \dots, \Delta^\gamma$ such that $\mathbf{E}$ is uniquely determined by the products $\mathbf{E}\Delta^1, \mathbf{E}\Delta^2, \ldots, \mathbf{E}\Delta^\gamma$. Thus a forward difference estimate in the direction $\boldsymbol{\Delta}^k$ is expressed as

$$\mathbf{E}\Delta^k = \nabla^2\Omega(\mathbf{x})\Delta^k = \nabla\Omega(\mathbf{x}+\boldsymbol{\Delta}^k) - \nabla\Omega(\mathbf{x}) \tag{2.8}$$

and a central difference estimate is given by

$$\mathbf{E}\Delta^k = \nabla^2\Omega(\mathbf{x})\Delta^k = \frac{1}{2}\left[\nabla\Omega(\mathbf{x}+\boldsymbol{\Delta}^k) - \nabla\Omega(\mathbf{x}-\boldsymbol{\Delta}^k)\right]. \tag{2.9}$$

Given a sparsity pattern for the symmetric matrix $\mathbf{E}$, they determine a symmetric permutation of $\mathbf{E}$ and a partition of the columns of $\mathbf{E}$ (i.e., the index sets), consistent with determination of $\mathbf{E}$ by a lower triangular substitution method. Two techniques for constructing the index sets are implemented in their software, namely a direct method and an indirect method. Their computational experience (cf. [63]) suggests the indirect method yields a smaller number of index sets $\gamma$, whereas the direct method produces more accurate Hessian approximations.

### 2.2.3 Sparse Differences in Nonlinear Programming

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,

$$\mathbf{q} = \begin{bmatrix} \mathbf{c} \\ F \end{bmatrix} \tag{2.10}$$

and

$$\mathbf{D} = \begin{bmatrix} \mathbf{G} \\ \mathbf{g}^\mathsf{T} \end{bmatrix}. \tag{2.11}$$

It is also natural to define

$$\boldsymbol{\omega}^\mathsf{T} = (-\lambda_1, \ldots, -\lambda_m, 1), \tag{2.12}$$

where $\lambda_k$ are the Lagrange multipliers with $\upsilon = m+1$, so that (2.2)

$$\Omega(\mathbf{x}) = \sum_{i=1}^{\upsilon} \omega_i q_i(\mathbf{x}) = F - \sum_{i=1}^{m} \lambda_i c_i(\mathbf{x}) = L(\mathbf{x}, \boldsymbol{\lambda}) \tag{2.13}$$

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).

## 2.3 Sparse QP Subproblem

The efficient solution of the sparse QP subproblem can be achieved using a method proposed by Gill et al. [97, 98]. In general, the calculation of a step in a QP subproblem requires the solution of the KKT system (1.66) as described in Sections 1.8 and 1.10. For convenience, recall that the *QP subproblem* (1.102)–(1.103) is as follows:

Compute $\mathbf{p}$ to minimize a quadratic approximation to the Lagrangian

$$\mathbf{g}^\mathsf{T}\mathbf{p} + \frac{1}{2}\mathbf{p}^\mathsf{T}\mathbf{H}\mathbf{p} \tag{2.14}$$

subject to the linear approximation to the constraints

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{Gp} \\ \mathbf{p} \end{bmatrix} \leq \mathbf{b}_U \tag{2.15}$$

with bound vectors defined by

$$\mathbf{b}_L = \begin{bmatrix} \mathbf{c}_L - \mathbf{c} \\ \mathbf{x}_L - \mathbf{x} \end{bmatrix}, \qquad \mathbf{b}_U = \begin{bmatrix} \mathbf{c}_U - \mathbf{c} \\ \mathbf{x}_U - \mathbf{x} \end{bmatrix}. \tag{2.16}$$

First, the QP subproblem (2.14)–(2.15) is restated in the following *standard form*: Compute $\widetilde{\mathbf{p}}$ to minimize

$$\widetilde{\mathbf{g}}^{\mathsf{T}}\widetilde{\mathbf{p}} + \frac{1}{2}\widetilde{\mathbf{p}}^{\mathsf{T}}\widetilde{\mathbf{H}}\widetilde{\mathbf{p}} \tag{2.17}$$

subject to the linear equality constraints

$$\widetilde{\mathbf{G}}\widetilde{\mathbf{p}} = \widetilde{\mathbf{b}} \tag{2.18}$$

and simple bounds

$$\widetilde{\mathbf{p}}_L \leq \widetilde{\mathbf{p}} \leq \widetilde{\mathbf{p}}_U. \tag{2.19}$$

Notice that this formulation involves only simple bounds (2.19) and equalities (2.18). This transformation can be accomplished by introducing *slack variables*, $s_k$. For example, a general inequality constraint of the form $\mathbf{a}_k^{\mathsf{T}}\mathbf{x} \geq b_k$ is replaced by the equality constraint $\mathbf{a}_k^{\mathsf{T}}\mathbf{x} + s_k = b_k$ and the bound $s_k \leq 0$. Notice that we have introduced the tilde notation to indicate that the original variables have been augmented to include the slacks. Observe that in this formulation, when a slack variable is "fixed" on an upper or lower bound, it is equivalent to the original inequality being in the active set. Thus, for a particular QP iteration, the search direction in the "free" variables can be computed by solving the KKT system (1.66), which in this case is

$$\begin{bmatrix} \widetilde{\mathbf{H}}_f & \widetilde{\mathbf{G}}_f^{\mathsf{T}} \\ \widetilde{\mathbf{G}}_f & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\widetilde{\mathbf{p}}_f \\ \widetilde{\overline{\boldsymbol{\lambda}}} \end{bmatrix} = \begin{bmatrix} \widetilde{\mathbf{g}}_f \\ \mathbf{0} \end{bmatrix}. \tag{2.20}$$

We have used the $f$ subscript to denote quantities corresponding to the "free" variables, and assume the iteration begins at a feasible point so that $\widetilde{\mathbf{b}} = \mathbf{0}$. Let us define the large, sparse symmetric indefinite KKT matrix by

$$\mathbf{K}_0 = \begin{bmatrix} \widetilde{\mathbf{H}}_f & \widetilde{\mathbf{G}}_f^{\mathsf{T}} \\ \widetilde{\mathbf{G}}_f & \mathbf{0} \end{bmatrix}. \tag{2.21}$$

If the initial estimate of the active set is correct, the solution of the KKT system defines the solution of the QP problem. However, in general, it will be necessary to change the active set and solve a series of equality-constrained problems. In [97, 98], it is demonstrated that the solution to a problem with a new active set can be obtained by the symmetric addition of a row and column to the original $\mathbf{K}_0$, with a corresponding augmentation of the right-hand side. In fact, after $k$ iterations, the KKT system is of dimension $n_0+k$ and has the form

$$\begin{bmatrix} \mathbf{K}_0 & \mathbf{U} \\ \mathbf{U}^\mathsf{T} & \mathbf{V} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{w} \end{bmatrix}, \tag{2.22}$$

where $\mathbf{U}$ is $n_0 \times k$ and $\mathbf{V}$ is $k \times k$. The initial right-hand side of (2.20) is denoted by the $n_0$-vector $\mathbf{f}_0$. The $k$-vector $\mathbf{w}$ defines the additions to the right-hand side to reflect changes in the active set.

The fundamental feature of the method is that the system (2.22) can be solved using factorizations of $\mathbf{K}_0$ and $\mathbf{C}$, the $k \times k$ *Schur-complement* of $\mathbf{K}_0$:

$$\mathbf{C} \equiv \mathbf{V} - \mathbf{U}^\mathsf{T}\mathbf{K}_0^{-1}\mathbf{U}. \tag{2.23}$$

Using the Schur-complement, the values for $\mathbf{y}$ and $\mathbf{z}$ are computed by solving in turn

$$\mathbf{K}_0\mathbf{v}_0 = \mathbf{f}_0, \tag{2.24}$$

$$\mathbf{C}\mathbf{z} = \mathbf{w} - \mathbf{U}^\mathsf{T}\mathbf{v}_0, \tag{2.25}$$

$$\mathbf{K}_0\mathbf{y} = \mathbf{f} - \mathbf{U}\mathbf{z}. \tag{2.26}$$

Thus, each QP iteration requires one solve with the factorization of $\mathbf{K}_0$ and one solve with the factorization of $\mathbf{C}$. The solve for $\mathbf{v}_0$ needs to be done only once at the first iteration. Each change in the active set adds a new row and column to $\mathbf{C}$. It is relatively straightforward to update both $\mathbf{C}$ and its factorization to accommodate the change. It is important to keep $\mathbf{C}$ small enough to maintain a stable, dense factorization. This is achieved by refactoring the entire KKT matrix whenever $k > 100$. In general, the penalty for refactoring may be substantial. However, when the QP algorithm is used within the general NLP algorithm, it is possible to exploit previous estimates of the active set to give the QP algorithm a "warm start." In fact, as the NLP algorithm approaches a solution, it is expected that the active set will be correctly identified and the resulting number of iterations $k$ for the QP subproblem will become small.

The Schur-complement method derives its efficiency from two facts. First, the KKT matrix is factored only *once* using a very efficient *multifrontal algorithm* [4]. This software solves $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{x}$, where $\mathbf{A}$ is an $n \times n$ real *symmetric indefinite* sparse matrix. Since $\mathbf{A}$ is symmetric, it can be factored as $\mathbf{A} = \mathbf{LDL}^\mathsf{T}$, where $\mathbf{L}$ is a unit lower-triangular matrix and $\mathbf{D}$ is a block-diagonal matrix composed solely of $1 \times 1$ and $2 \times 2$ blocks. Since $\mathbf{A}$ is not necessarily positive definite, pivoting to preserve stability is required. The package uses the threshold-pivoting generalization of Bunch and Kaufman with $2 \times 2$ block pivoting for sparse symmetric indefinite matrices. The software requires storage for the nonzero elements in the lower-triangular portion of the matrix and a work array. Second, subsequent changes to the QP active set can be computed using a *solve* with the previously factored KKT matrix and a *solve* with the small, dense Schur-complement matrix. Since the factorization of the KKT matrix is significantly more expensive than the solve operation, the overall method is quite effective. The algorithm is described in [32].

## 2.4 Merit Function

When a QP algorithm is used to approximate a general nonlinearly constrained problem, it may be necessary to adjust the steplength $\alpha$ in order to achieve "sufficient reduction" in a merit function as discussed in Section 1.11. The merit function we use is a modified version of (1.110), which was proposed by Gill et al. in [95]. It is related to the function given by Rockafellar in [151]:

$$M(\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\nu},\mathbf{s},\mathbf{t}) = F - \boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{c}-\mathbf{s}) - \boldsymbol{\nu}^{\mathsf{T}}(\mathbf{x}-\mathbf{t}) + \frac{1}{2}(\mathbf{c}-\mathbf{s})^{\mathsf{T}}\boldsymbol{\Theta}(\mathbf{c}-\mathbf{s}) + \frac{1}{2}(\mathbf{x}-\mathbf{t})^{\mathsf{T}}\boldsymbol{\Xi}(\mathbf{x}-\mathbf{t}). \tag{2.27}$$

The diagonal penalty matrices are defined by $\boldsymbol{\Theta}_{ii} = \theta_i$ and $\boldsymbol{\Xi}_{ii} = \xi_i$ for $\theta_i > 0$ and $\xi_i > 0$. The merit function is written to explicitly include terms for the bounds that were not present in the original formulation (1.110) [95]. For this merit function, the slack variables **s** and **t** at the beginning of a step are defined by

$$s_i = \begin{cases} c_{Li} & \text{if } c_{Li} > c_i - \lambda_i/\theta_i, \\ c_i - \lambda_i/\theta_i & \text{if } c_{Li} \le c_i - \lambda_i/\theta_i \le c_{Ui}, \\ c_{Ui} & \text{if } c_i - \lambda_i/\theta_i > c_{Ui}; \end{cases} \tag{2.28}$$

$$t_i = \begin{cases} x_{Li} & \text{if } x_{Li} > x_i - \nu_i/\xi_i, \\ x_i - \nu_i/\xi_i & \text{if } x_{Li} \le x_i - \nu_i/\xi_i \le x_{Ui}, \\ x_{Ui} & \text{if } x_i - \nu_i/\xi_i > x_{Ui}. \end{cases} \tag{2.29}$$

These expressions for the slack variables yield a minimum value for the merit function $M$ for given values of the variables $\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}$ and penalty weights subject to the bounds on the slacks. The search direction in the real variables **x** is augmented to permit the multipliers and the slack variables to vary according to

$$\begin{bmatrix} \overline{\mathbf{x}} \\ \overline{\boldsymbol{\lambda}} \\ \overline{\boldsymbol{\nu}} \\ \overline{\mathbf{s}} \\ \overline{\mathbf{t}} \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \\ \boldsymbol{\nu} \\ \mathbf{s} \\ \mathbf{t} \end{bmatrix} + \alpha \begin{bmatrix} \mathbf{p} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{\nu} \\ \Delta\mathbf{s} \\ \Delta\mathbf{t} \end{bmatrix}. \tag{2.30}$$

The multiplier search directions, $\Delta\boldsymbol{\lambda}$ and $\Delta\boldsymbol{\nu}$, are defined using the QP multipliers $\widehat{\boldsymbol{\lambda}}$ and $\widehat{\boldsymbol{\nu}}$ according to

$$\Delta\boldsymbol{\lambda} \equiv \widehat{\boldsymbol{\lambda}} - \boldsymbol{\lambda} \tag{2.31}$$

and

$$\Delta\boldsymbol{\nu} \equiv \widehat{\boldsymbol{\nu}} - \boldsymbol{\nu}. \tag{2.32}$$

As in (1.107), the predicted slack variables are just

$$\overline{\mathbf{s}} = \mathbf{G}\mathbf{p} + \mathbf{c} = \mathbf{s} + \Delta\mathbf{s}. \tag{2.33}$$

Using this expression, the slack-vector step analogous to (1.108) is just

$$\Delta\mathbf{s} = \mathbf{G}\mathbf{p} + (\mathbf{c} - \mathbf{s}). \tag{2.34}$$

A similar technique defines the bound slack-vector search direction

$$\Delta\mathbf{t} = \mathbf{p} + (\mathbf{x} - \mathbf{t}). \tag{2.35}$$

Note that when a full step is taken ($\alpha = 1$), the updated estimates for the Lagrange multipliers $\overline{\boldsymbol{\lambda}}$ and $\overline{\boldsymbol{\nu}}$ are just the QP estimates $\widehat{\boldsymbol{\lambda}}$ and $\widehat{\boldsymbol{\nu}}$. The slack variables $\mathbf{s}$ and $\mathbf{t}$ are just the linear estimates of the constraints. The terms $(\mathbf{c} - \mathbf{s})$ and $(\mathbf{x} - \mathbf{t})$ in the merit function are measures of the *deviation* from linearity. In [95], Gill et al. prove global convergence for an SQP algorithm that uses this merit function, provided the QP subproblem has a solution, which requires bounds on the derivatives and Hessian condition number.

It is also necessary to define the penalty weights $\boldsymbol{\Theta}$ and $\boldsymbol{\Xi}$. In [95], it is shown that convergence of the method assumes the weights are chosen such that

$$M_0' \le -\frac{1}{2}\mathbf{p}^\mathsf{T}\mathbf{H}\mathbf{p}, \tag{2.36}$$

where $M_0'$ denotes the directional derivative of the merit function (2.27) with respect to the steplength $\alpha$ evaluated at $\alpha = 0$. To achieve this, let us define the vector

$$\Psi_i = \begin{cases} \theta_i - \psi_0 & \text{if} \quad 1 \le i \le m, \\ \xi_{i-m} - \psi_0 & \text{if} \quad m < i \le (m+n), \end{cases}$$

where $\psi_0 > 0$ is a strictly positive "threshold." Since (2.36) provides a single condition for the $(m+n)$ penalty parameters, we make the choice unique by minimizing the norm $\|\boldsymbol{\Psi}\|_2$. This yields

$$\boldsymbol{\Psi} = \mathbf{a}(\mathbf{a}^\mathsf{T}\mathbf{a})^{-1}\varsigma, \tag{2.37}$$

where

$$a_i = \begin{cases} (c_i - s_i)^2 & \text{if} \quad 1 \le i \le m, \\ (x_{i-m} - t_{i-m})^2 & \text{if} \quad m < i \le (m+n), \end{cases}$$

and

$$\begin{aligned} \varsigma = &-\frac{1}{2}\mathbf{p}^\mathsf{T}\mathbf{H}\mathbf{p} + \widehat{\boldsymbol{\lambda}}^\mathsf{T}\Delta\mathbf{s} + \widehat{\boldsymbol{\nu}}^\mathsf{T}\Delta\mathbf{t} - 2(\Delta\boldsymbol{\lambda})^\mathsf{T}(\mathbf{c} - \mathbf{s}) - 2(\Delta\boldsymbol{\nu})^\mathsf{T}(\mathbf{x} - \mathbf{t}) \\ &-\psi_0(\mathbf{c} - \mathbf{s})^\mathsf{T}(\mathbf{c} - \mathbf{s}) - \psi_0(\mathbf{x} - \mathbf{t})^\mathsf{T}(\mathbf{x} - \mathbf{t}). \end{aligned}$$

Typically, the threshold parameter $\psi_0$ is set to machine precision and increased only if the minimum norm solution is zero. In essence, then, the penalty weights are chosen to be as small as possible consistent with the descent condition (2.36).

## 2.5 Hessian Approximation

A positive definite Hessian matrix ensures that the solution to the QP subproblem is unique and also makes it possible to compute $\boldsymbol{\Theta}$ and $\boldsymbol{\Xi}$ to satisfy the descent condition (2.36). For NLP applications, the Hessian of the Lagrangian is

$$\mathbf{H}_L = \nabla_x^2 F - \sum_{i=1}^{m} \lambda_i \nabla_x^2 c_i. \tag{2.38}$$

An approximation to $\mathbf{H}_L$ can be constructed using the methods described in Section 2.2; however, in general, it is not positive definite. (This does not imply that a finite difference approximation is poor, since the true Hessian may also be indefinite.) In fact, it is necessary only that the reduced Hessian of the Lagrangian be positive definite at the solution with the correct active set of constraints. Similar restrictions are required at $\mathbf{x} \neq \mathbf{x}^*$ to ensure that *each* QP subproblem has a solution. Consequently, for the QP subproblem, we use the modified matrix

$$\mathbf{H} = \mathbf{H}_L + \tau(|\sigma| + 1)\mathbf{I}. \tag{2.39}$$

The parameter $\tau$ is chosen such that $0 \leq \tau \leq 1$ and is normalized using the *Gerschgorin bound* for the most negative eigenvalue of $\mathbf{H}_L$, i.e.,

$$\sigma = \min_{1 \leq i \leq n} \left\{ h_{ii} - \sum_{i \neq j}^{n} |h_{ij}| \right\}. \tag{2.40}$$

$h_{ij}$ is used to denote the nonzero elements of $\mathbf{H}_L$. An approach for modifying an approximation to the Hessian for least squares problems by the matrix $\bar{\tau}\mathbf{I}$ was originally suggested by Levenberg [131] and, because of this similarity, we refer to $\tau$ as the *Levenberg parameter*. As a practical matter, normalization, using the Gerschgorin bound, is useful even though the accuracy of the Gerschgorin estimate is not critical. It is also instructive to recall the trust-region interpretation of the parameter as defined by (1.93).

The proper choice for the Levenberg parameter $\tau$ can greatly affect the performance of the NLP algorithm. A fast rate of convergence can be obtained only when $\tau = 0$ and the correct active set has been identified. On the other hand, if $\tau = 1$, in order to guarantee a positive definite Hessian, the search direction $\mathbf{p}$ is significantly biased toward a gradient direction and convergence is degraded. A strategy similar to that used for adjusting a trust region (cf. [82]) is employed by the algorithm to maintain a current value for the Levenberg parameter $\tau$ and adjust it from iteration to iteration. The *inertia* (i.e., the number of positive, negative, and zero eigenvalues) of the related KKT matrix (2.21) is used to infer that the reduced Hessian is positive definite. Using results from Gould [104], Gill et al. [97, 98] show that the reduced Hessian will be positive definite if the inertia of $\mathbf{K}_0$ (2.21) is

$$In(\mathbf{K}_0) = (n_f, m, 0), \tag{2.41}$$

where $n_f$ is the number of rows in $\widetilde{\mathbf{H}}_f$ and $m$ is the number of rows in $\widetilde{\mathbf{G}}_f$. Basically, the philosophy is to reduce the Levenberg parameter when the predicted reduction in the merit function agrees with the actual reduction and increase the parameter when the agreement is poor. The process is accelerated by making the change in $\tau$ proportional to the observed rate of change in the gradient of the Lagrangian. To be more precise, at iteration $k$, three quantities are computed, namely

1. the actual reduction
$$\varrho_1 = M^{(k-1)} - M^{(k)}, \tag{2.42}$$

2. the predicted reduction
$$\varrho_2 = M^{(k-1)} - \tilde{M}^{(k)} = -M_0' - \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}, \tag{2.43}$$

   where $\tilde{M}^{(k)}$ is the predicted value of the merit function, and

3. the rate of change in the norm of the gradient of the Lagrangian

$$\varrho_3 = \frac{\|\boldsymbol{\vartheta}^{(k)}\|_\infty}{\|\boldsymbol{\vartheta}^{(k-1)}\|_\infty}, \tag{2.44}$$

where the error in the gradient of the Lagrangian is

$$\boldsymbol{\vartheta} = \mathbf{g} - \mathbf{G}^\mathsf{T}\boldsymbol{\lambda} - \boldsymbol{\nu}. \tag{2.45}$$

Then, if $\varrho_1 \le 0.25\varrho_2$, the actual behavior is much worse than predicted, so bias the step toward the gradient by setting $\tau^{(k+1)} = \min(2\tau^{(k)}, 1)$. On the other hand, if $\varrho_1 \ge 0.75\varrho_2$, then the actual behavior is sufficiently close to predicted, so bias the step toward a Newton direction by setting $\tau^{(k+1)} = \tau^{(k)}\min(0.5, \varrho_3)$. It is important to note that this strategy does not ensure that the reduced Hessian is positive definite. In fact, it may be necessary to supersede this adaptive adjustment and increase $\tau^{(k+1)}$ whenever the inertia of the KKT matrix is incorrect. The inertia is easily computed as a byproduct of the symmetric indefinite factorization by counting the number of positive and negative elements in the diagonal matrix (with a positive and negative contribution coming from each $2 \times 2$ block).

## 2.6 Sparse SQP Algorithm

### 2.6.1 Minimization Process

Let us now summarize the steps in the algorithm. The iteration begins at the point $\mathbf{x}$, with $k = 1$, and proceeds as follows:

1. *Gradient Evaluation.* Evaluate gradient information $\mathbf{g}$ and $\mathbf{G}$ and then
   (a) evaluate the error in the gradient of the Lagrangian from (2.45);
   (b) terminate if the KKT conditions are satisfied;
   (c) compute $\mathbf{H}_L$ from (2.38); if this is the first iteration, go to step 2; otherwise
   (d) *Levenberg modification:*
      i. compute the rate of change in the norm of the gradient of the Lagrangian from (2.44);
      ii. if $\varrho_1 \le 0.25\varrho_2$, then set $\tau^{(k)} = \min(2\tau^{(k-1)}, 1)$; otherwise
      iii. if $\varrho_1 \ge 0.75\varrho_2$, then set $\tau^{(k)} = \tau^{(k-1)}\min(0.5, \varrho_3)$.
2. *Search Direction.* Construct the optimization search direction:
   (a) compute $\mathbf{H}$ from (2.39);
   (b) compute $\mathbf{p}$ by solving the QP subproblem (2.14)–(2.15);
   (c) *Inertia control:* if inertia of $\mathbf{K}$ is incorrect and
      i. if $\tau^{(k)} < 1$, then set $\tau^{(k)} \leftarrow \min(10\tau^{(k)}, 1)$ and return to step 2(a);
      ii. if $\tau^{(k)} = 1$ and $\mathbf{H} \ne \mathbf{I}$, then set $\tau^{(k)} = 0$ and $\mathbf{H} = \mathbf{I}$ and return to step 2(a);
      iii. if $\mathbf{H} = \mathbf{I}$, then QP constraints are locally inconsistent—terminate the minimization process and attempt to locate a feasible point for the nonlinear constraints;

   (d) compute $\Delta\boldsymbol{\lambda}$ and $\Delta\boldsymbol{\nu}$ from (2.31) and (2.32);

   (e) compute $\Delta\mathbf{s}$ and $\Delta\mathbf{t}$ from (2.34) and (2.35);

   (f) compute penalty parameters to satisfy (2.36); and

   (g) initialize $\alpha = 1$.

3. *Prediction:*

   (a) compute the predicted point for the variables, the multipliers, and the slacks from (2.30);

   (b) evaluate the constraints $\overline{\mathbf{c}} = \mathbf{c}(\overline{\mathbf{x}})$ at the predicted point.

4. *Line Search.* Evaluate the merit function $M(\overline{\mathbf{x}}, \overline{\boldsymbol{\lambda}}, \overline{\boldsymbol{\nu}}, \overline{\mathbf{s}}, \overline{\mathbf{t}}) = \bar{M}$ and

   (a) if the merit function $\bar{M}$ is "sufficiently" less than $M$, then $\overline{\mathbf{x}}$ is an improved point—terminate the line search and go to step 5;

   (b) else change the steplength $\alpha$ to reduce $M$ and return to step 3.

5. *Update.* Update all quantities and set $k = k + 1$;

   (a) compute the actual reduction from (2.42);

   (b) compute the predicted reduction from (2.43), where $\tilde{M}^{(k)}$ is the predicted value of the merit function; and

   (c) return to step 1.

The steps outlined describe the fundamental elements of the optimization process; however, a number of points deserve additional clarification. First, note that the algorithm requires a line search in the direction defined by (2.30) with the steplength $\alpha$ adjusted to reduce the merit function. Adjusting the value of the steplength $\alpha$, as required in step 4(b), is accomplished using a line-search procedure that constructs a quadratic and cubic model of the merit function. The reduction is considered "sufficient" when $M(\alpha) - M(0) < \kappa_1 \alpha M'(0)$. Instead of (1.89), the Wolfe rule, $M'(\alpha) < \kappa_2 M'(0)$ for $0 < \kappa_1 < \kappa_2 < 1$, is imposed to prevent steplengths from becoming too small.

In order to evaluate the Hessian matrix (2.38), an estimate of the Lagrange multipliers is needed. The values obtained by solving the QP problem with $\mathbf{H} = \mathbf{I}$ are used for the first iteration and, thereafter, the values $\overline{\boldsymbol{\lambda}}$ from (2.30) are used. Note that, at the very first iteration, *two* QP subproblems are solved—one to compute first order multiplier estimates and the second to compute the step. Furthermore, for the very first iteration, the multiplier search directions are $\Delta\boldsymbol{\lambda} = 0$ and $\Delta\boldsymbol{\nu} = 0$, so that the multipliers will be initialized to the QP estimates $\overline{\boldsymbol{\lambda}} = \boldsymbol{\lambda} = \widehat{\boldsymbol{\lambda}}$ and $\overline{\boldsymbol{\nu}} = \boldsymbol{\nu} = \widehat{\boldsymbol{\nu}}$. The multipliers are reset in a similar fashion, after a *defective QP subproblem* is encountered, in step 2(c)iii. The subject of defective subproblems will be covered in Section 2.7. The Levenberg parameter $\tau$ in (2.39) and the penalty weights $\theta_i$ and $\xi_i$ in (2.27) are initialized to zero and, consequently, the merit function is initially just the Lagrangian.

### 2.6.2 Algorithm Strategy

The basic algorithm described above has been implemented in FORTRAN as part of the $\mathbb{SOCS}$ library and is documented in [29]. In the software, the preceding approach is referred to as strategy M since the iterates follow a path from the initial point to the solution. However, in practice it may be desirable and/or more efficient to first locate a feasible point. Consequently, the software provides four different algorithm strategies:

M Minimize. Beginning at $\mathbf{x}^0$, solve a sequence of quadratic programs until the solution $\mathbf{x}^*$ is found.

FM Find a Feasible point and then Minimize. Beginning at $\mathbf{x}^0$, solve a sequence of quadratic programs to locate a feasible point $\mathbf{x}^f$ and then, beginning from $\mathbf{x}^f$, solve a sequence of quadratic programs until the solution $\mathbf{x}^*$ is found.

FME Find a Feasible point and then Minimize subject to Equalities. Beginning at $\mathbf{x}^0$, solve a sequence of quadratic programs to locate a feasible point $\mathbf{x}^f$ and then, beginning from $\mathbf{x}^f$, solve a sequence of quadratic programs while maintaining feasible equalities until the solution $\mathbf{x}^*$ is found.

F Find a Feasible point. Beginning at $\mathbf{x}^0$, solve a sequence of quadratic programs to locate a feasible point $\mathbf{x}^f$.

The default strategy in the software is FM since computational experience suggests that it is more robust and efficient. Details on the FME strategy can be found in [32]. The fourth strategy, F, to locate a feasible point only, is also useful when debugging a new problem formulation.

## 2.7 Defective Subproblems

The fundamental step in the optimization algorithm requires the solution of a QP subproblem. In Section 1.16, we discussed a number of things that can go wrong that will prevent the solution of the subproblem. In particular, the QP subproblem can be *defective* because

1. the linear constraints (2.15) are inconsistent (i.e., have no solution);
2. the Jacobian matrix $\mathbf{G}$ is rank deficient;
3. the linear constraints (2.15) are redundant or extraneous, which can correspond to Lagrange multipliers that are zero at the solution;
4. the quadratic objective (2.14) is unbounded in the null space of the active constraints.

Unfortunately, because the QP problem is a subproblem within the overall NLP, it is not always obvious how to determine the cause of the difficulty. In particular, the QP subproblem may be defective *locally* simply because the quadratic/linear model does not approximate the nonlinear behavior. On the other hand, the QP subproblem may be defective because the original NLP problem is inherently ill-posed. Regardless of the cause of the defective QP subproblem, the overall algorithm behavior can be significantly impacted. In particular, a defective QP subproblem often produces a large increase in the solution time because

1. there is a large amount of *fill* caused by pivoting for stability in the sparse linear algebra software, which makes the sparse method act like a dense method;

2. there are many iterations in the QP subproblem, which, in turn, necessitates the factorization of a large, dense Schur-complement matrix and/or repeated KKT sparse matrix factorizations.

In order to avoid these detrimental effects, the strategy employed by the NLP algorithm has been constructed to minimize the impact of a defective QP subproblem. The first premise in designing the NLP strategy is to segregate difficulties caused by the constraints from difficulties caused by the objective function. The second basic premise is to design an NLP strategy that will eliminate a defective subproblem rather than solve an ill-conditioned system. This philosophy has been implemented in the default FM strategy. Specifically, we find a feasible point first. During this phase, difficulties that could be attributed to the objective function, Lagrange multipliers, and Hessian matrix are ignored. Instead, difficulties are attributed solely to the constraints during this phase. After a feasible point has been located with a full-rank Jacobian, it is assumed that the constraints are OK. Thus, during the optimization phase, defects related to the objective function are treated. The primary mechanism for dealing with a defective QP subproblem during the optimization process was the Levenberg Hessian modification technique. In particular, note that in step 2(c)i of the algorithm, first the Levenberg parameter is increased. If that fails, in step 2(c)ii, the Hessian is reset to the identity matrix. Only after the above two steps fail is it concluded that the defect in the QP subproblem must be caused by the constraints, and an attempt is made to locate a (nearby) feasible point.

It is worth noting that a defective subproblem is not something unique to SQP methods. In fact, all of the globalization strategies discussed in Section 1.11 can be considered remedies for a defective subproblem. However, it is curious that the remedies we will discuss are computationally attractive for large, sparse problems, but are generally not used for small, dense applications!

## 2.8 Feasible Point Strategy

### 2.8.1 QP Subproblem

Finding a point that is feasible with respect to the constraints is the first phase in either the FM or FME strategy and is often used when attempting to deal with a defective problem. The approach employed is to take a series of steps of the form given by (1.87) with the search direction computed to solve a *least distance program* (LDP). This can be accomplished if we impose the requirement that the search direction have minimum norm, i.e., $\|\mathbf{p}\|_2$. The *primary* method for computing the search direction is to minimize

$$\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{p} \tag{2.46}$$

subject to the linear constraints

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{G}\mathbf{p} \\ \mathbf{p} \end{bmatrix} \leq \mathbf{b}_U. \tag{2.47}$$

For problems with no inequality constraints, the LDP search direction is equivalent to

$$\mathbf{p} = -\mathbf{G}^{\#}\mathbf{b}, \tag{2.48}$$

where $\mathbf{G}^{\#}$ is the *pseudoinverse* of $\mathbf{G}$ and can be viewed as a generalization of the basic Newton step (1.28).

However, as previously suggested, it is possible to encounter a defective subproblem and, in this case, it is useful to construct the search direction from a problem that has a solution even if the Jacobian is singular and/or the linear constraints are inconsistent. Thus, the *relaxation* method requires finding the augmented set of variables $(\mathbf{p},\mathbf{u})$ to minimize

$$\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{p} + \frac{\rho}{2}\mathbf{u}^{\mathsf{T}}\mathbf{u} \tag{2.49}$$

subject to the linear constraints

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{G}\mathbf{p}+\mathbf{u} \\ \mathbf{p} \end{bmatrix} \leq \mathbf{b}_U, \tag{2.50}$$

where the constant $\rho \gg 0$. Typically, $\rho = 10^6$. Notice that, by adding the residual or slack variables $\mathbf{u}$, the linear equations (2.50) always have a solution. Although the size of the QP problem has been increased, both the Hessian and Jacobian for the augmented problem are sparse. Although the condition number of the KKT matrix for the relaxation QP problem is $\mathcal{O}(\rho^2)$, an accurate solution can often be obtained using a few steps of iterative refinement. It is interesting that the notion of adding variables is *not* usually considered attractive for dense problems because the associated linear algebra cost becomes excessive. However, when sparse matrix techniques are employed, this technique is, in fact, quite reasonable.

Since the solution of this subproblem is based on a linear model of the constraint functions, it may be necessary to adjust the steplength $\alpha$ in (1.87) to produce a reduction in the constraint error. Specifically, a line search is used to adjust $\alpha$ to achieve "sufficient decrease" (1.89) in the constraint violation merit function as defined by

$$M_v(\mathbf{x}) = \sum_{i=1}^{m} \chi^2(c_{Li}, c_i, c_{Ui}) + \sum_{i=1}^{n} \chi^2(x_{Li}, x_i, x_{Ui}) \tag{2.51}$$

with $\chi(l,y,u) \equiv \max[0, l-y, y-u]$.

## 2.8.2 Feasible Point Strategy

The overall strategy for locating a feasible point can now be described. Beginning at the point $\mathbf{x}$, with the "primary strategy," the procedure is as follows:

1. *Gradient Evaluation.* Evaluate the constraints and Jacobian. Terminate if the constraints are feasible.

2. *Search Direction.* Compute search direction:

   (a) if (*primary strategy*), solve the QP subproblem (2.46)–(2.47), and

      i. if QP solution is possible, then go to step 3; otherwise
      ii. change to *relaxation strategy* and go to step 2(b);

   (b) if (*relaxation strategy*), solve the QP subproblem (2.49)–(2.50).

3. *Line Search.* Choose the steplength $\alpha$ to reduce the constraint violation $M_v(\overline{\mathbf{x}})$ given by (2.51). If relaxation strategy is used, do an accurate line search.

4. *Strategy Modification:*

   (a) if the primary strategy is being used, then return to step 1;

   (b) if the relaxation strategy is being used and if $\alpha = 1$, then switch to primary strategy and return to step 1.

This overall algorithm gives priority to the primary method for computing the search direction. If the primary strategy fails, then presumably there is something defective with the QP subproblem and the relaxation strategy is used. If a switch to the relaxation strategy is made, then subsequent steps use this approach and perform an accurate line search. When full-length steps are taken with the relaxation strategy (i.e., $\alpha = 1$), the primary strategy is again invoked. This logic is motivated by the fact that the relaxation step with $\alpha = 1$ is “approximately” a Newton step and, therefore, it is worthwhile to switch back to the primary (LDP) step. Although this strategy is somewhat ad hoc, it has been quite effective in practice.

Detecting failure of the primary strategy in step 2(a) is based on a number of factors. Specifically, the primary strategy is abandoned whenever

1. the amount of “fill” in the sparse linear system exceeds expectations (implying an ill-conditioned linear system), or

2. the condition number of the KKT matrix is large, or

3. the inertia of the KKT matrix is incorrect, or

4. the number of QP iterations is excessive.

### 2.8.3 An Illustration

**Example 2.1.** The performance of the feasible point algorithm is illustrated on an example derived from an ill-conditioned two-point boundary value problem (BVP) (Burgers’ equation). The basic problem is to solve

$$\begin{aligned} \dot{y}_1 &= y_2, \\ \dot{y}_2 &= \epsilon^{-1} y_1 y_2, \\ 0 &\le y_1(t) \end{aligned}$$

subject to the boundary conditions $y_1(0) = 2\tanh(\epsilon^{-1})$ and $y_1(1) = 0$. For this illustration, the parameter $\epsilon = 10^{-3}$. The continuous problem is replaced by a discrete approximation with $M = 50$ grid points. Thus, it is required to compute the values of $\mathbf{x}^\mathsf{T} = (y_1(0), y_2(0), \ldots, y_1(1), y_2(1))$ such that the constraints

$$\mathbf{c}(\mathbf{x}) = \mathbf{y}_{j+1} - \mathbf{y}_j - \frac{h}{2}\left[\dot{\mathbf{y}}_{j+1} + \dot{\mathbf{y}}_j\right] = 0$$

for $j = 1, \ldots, (M-1)$ are satisfied in addition to the boundary conditions. For this example, the iterations began with a linear initial guess between the boundary conditions

$$\mathbf{y}_k = \mathbf{y}(0) + \frac{(k-1)}{(M-1)}\left[\mathbf{y}(1) - \mathbf{y}(0)\right]$$

for $k = 1, \ldots, M$ with $y_2(0) = -2\epsilon^{-1}[1 - \tanh(\epsilon^{-1})]$ and $y_2(1) = -2\epsilon^{-1}$. We defer details of the discretization process to subsequent chapters and simply view this as a system of nonlinear equations to be solved by proper choice of the variables $\mathbf{x}$.

The behavior of the algorithm is summarized in Table 2.1. At the first iteration, an attempt to compute the search direction using the primary least distance programming method (ldp) failed, and the relaxation (r) strategy was used. A step of length $\alpha = 1$ reduced the constraint error $\|\mathbf{c}\|$ from 97.3976 to 1.83373. Since a full Newton step was used, the second iteration began with the primary ldp strategy, which also failed, forcing the use of the relaxation method. For the second iteration, the steplength $\alpha$ was 0.317 (which is not a Newton step) and, consequently, the relaxation strategy was employed for iteration 3. At iteration 6, an attempt was made to switch back to the primary strategy, but again it was unsuccessful. Finally, at iteration 9, it was possible to return to the primary strategy, which was then used for all subsequent iterations. Notice that the condition number of the symmetric indefinite KKT system is rather moderate at the solution, even though it is very large for some of the early iterations.

**Table 2.1.** *Burgers' equation example.*

| Iter. | Method | KKT Cond. | $\alpha$ | $\|\mathbf{c}\|$ |
|---|---|---|---|---|
| 1 | ldp r | $0.48 \times 10^{+12}$ | 1.000 | 97.3976 |
| 2 | ldp r | $0.54 \times 10^{+11}$ | 0.317 | 1.83373 |
| 3 | r | $0.26 \times 10^{+13}$ | 0.149 | 1.31122 |
| 4 | r | $0.38 \times 10^{+13}$ | 0.149 | 1.16142 |
| 5 | r | $0.31 \times 10^{+13}$ | 1.000 | 1.02214 |
| 6 | ldp r | $0.22 \times 10^{+12}$ | $0.46 \times 10^{-1}$ | 0.337310 |
| 7 | r | $0.16 \times 10^{+13}$ | $0.35 \times 10^{-1}$ | 0.323168 |
| 8 | r | $0.16 \times 10^{+13}$ | 1.000 | 0.308115 |
| 9 | ldp | $0.14 \times 10^{+09}$ | $0.37 \times 10^{-2}$ | 0.182173 |
| 10 | ldp | $0.27 \times 10^{+08}$ | $0.10 \times 10^{-1}$ | 0.181395 |
| 11 | ldp | $0.50 \times 10^{+06}$ | $0.88 \times 10^{-1}$ | 0.179771 |
| 12 | ldp | $0.15 \times 10^{+06}$ | 0.457 | 0.165503 |
| 13 | ldp | $0.78 \times 10^{+05}$ | 1.000 | $0.892 \times 10^{-1}$ |
| 14 | ldp | $0.69 \times 10^{+05}$ | 1.000 | $0.251 \times 10^{-1}$ |
| 15 | ldp | $0.69 \times 10^{+05}$ | 1.000 | $0.748 \times 10^{-4}$ |
| 16 | ldp | $0.69 \times 10^{+05}$ | 1.000 | $0.212 \times 10^{-8}$ |

## 2.9 Computational Experience

### 2.9.1 Large, Sparse Test Problems

The NLP algorithm described has been tested on problems derived from optimal control and data-fitting applications. An extensive collection of test results for trajectory optimization and optimal control problems is found in [37]. The test set includes simple quadratic programs, nonlinear root solving, and poorly posed problems. Also included are examples with a wide range in the number of degrees of freedom and function nonlinearity. All problems in the test set

1. exhibit a *banded* sparsity pattern for the Jacobian and Hessian and
2. have some active constraints (i.e., no unconstrained problems).

With regard to the first attribute, all problems have some nonzero elements that are not along the diagonal (i.e., they are not separable). On the other hand, none of the problems is characterized by matrices with truly random structure. In general, the results described in [30] are very positive and, consequently, it seems worthwhile to understand the reasons for this promising behavior.

The calculation and treatment of the Hessian matrix are fundamental to the observed performance of the algorithm. As stated, the basic algorithm requires that the Hessian matrix $\mathbf{H}_L$ be computed from (2.38). Observe that the evaluation of $\mathbf{H}_L$ requires an estimate for both the variables and the Lagrange multipliers, i.e., $(\mathbf{x}, \boldsymbol{\lambda})$. Since the accuracy of the Hessian is affected by the accuracy of the multipliers, it seems desirable to use an NLP strategy that tends to produce "accurate" values for $\boldsymbol{\lambda}$. The default FM strategy, which first locates a feasible point and then stays "near" the constraints, presumably benefits from multiplier estimates $\boldsymbol{\lambda}$ that are more accurate near the constraints.

A summary of the results for the test problem set in [30] is given in Figure 2.1. All 109 problems were run using the three optimization strategies (M, FM, FME). The algorithm performance was measured in terms of the number of function evaluations (the number of times $f(\mathbf{x})$ and $\mathbf{c}(\mathbf{x})$ are evaluated) and the solution time. For each test problem, a first-, second-, and third-place strategy has been selected, where the first-place strategy required the smallest number of function evaluations. If a particular strategy failed to solve the problem, this was counted as a failure. It is clear from Figure 2.1 that strategy FM was in first place over 63% of the time. Furthermore, FM was either the best or second-best strategy nearly 89% of the time. Finally, notice that strategy FM solved all 109 problems (no failures). For all but 7 cases, the least number of function evaluations corresponds to the shortest solution time. Consequently, comparing strategies based on run time leads to the same conclusions. These results clearly indicate why strategy FM has been selected as the default. We note that for 3 problems, there are no degrees of freedom, in which case F is the only possible strategy and these cases were eliminated from the comparison.

There are at least three alternatives for computing the Hessian matrix. For some applications, it is possible to evaluate the relevant matrices analytically. Unfortunately, this is often cumbersome. For all of the test results in [30], the sparse finite difference approximations described in Section 2.2 were used. Additional information on sparse differencing is given in [35]. The third alternative, which is often effective for small, dense problems, is to use a quasi-Newton approximation to the Hessian. In general, the errors introduced by finite difference approximations are far smaller than those for quasi-Newton estimates and,

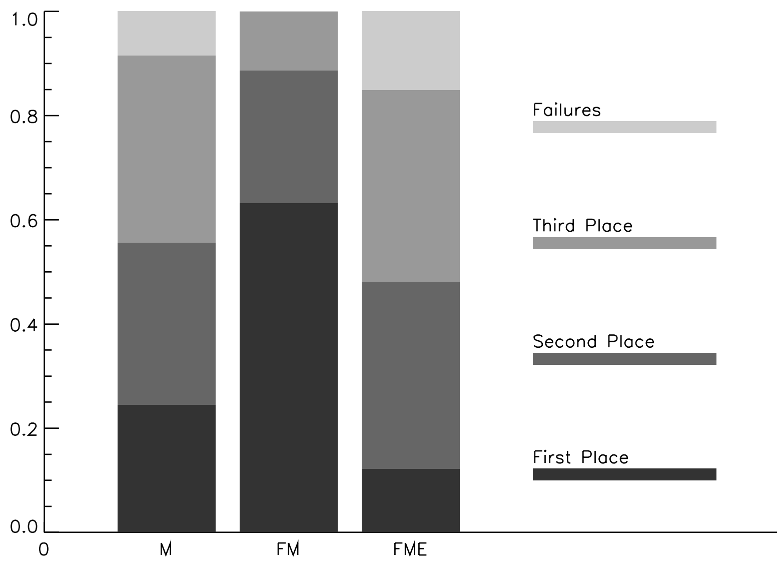


**Figure 2.1.** *Strategy comparison.*

for all practical purposes, one can consider the first two alternatives to be "exact." Probably the most significant advantage of methods with an exact Hessian is that ultimately one can expect quadratic convergence. The adaptive Levenberg strategy described in (2.42)–(2.45) is designed so that ultimately $\tau = 0$ and $\mathbf{H} = \mathbf{H}_L$ in the QP subproblem.

## 2.9.2 Small, Dense Test Problems

Although the strategy described was developed to accommodate sparse NLP applications, it is interesting to consider whether the same techniques may also be worthwhile for small, dense problems. One significant feature developed for large, sparse problems is the Levenberg modification technique, which permits using the exact Hessian without altering the matrix sparsity. A second feature is the default FM strategy, which first locates a feasible point. In contrast to the large, sparse case, for small, dense problems it is common to use a quasi-Newton approximation for the Hessian matrix. One possible approximation is the SR1 formula given by (1.49). The update formula requires the change in position given by $\Delta\mathbf{x} = \overline{\mathbf{x}} - \mathbf{x}$. For the SR1 update, it is appropriate to use $\Delta\mathbf{g} = \nabla_x L(\overline{\mathbf{x}}) - \nabla_x L(\mathbf{x})$. If the denominator is zero, making the SR1 correction undefined, we simply skip the update. Now recall that the SR1 recursive estimate is symmetric but not necessarily positive definite. Since the reduced Hessian must be positive definite at the solution, the most common approximation is the BFGS update formula (1.50). In this case, it is possible to keep the entire approximate Hessian positive definite (thereby ensuring the reduced Hessian is positive definite) provided the update also is constructed such that $\Delta\mathbf{x}^\mathsf{T}\Delta\mathbf{g} > 0$. We make an attempt to satisfy this condition by adjusting the Lagrange multiplier estimates used to construct the gradient difference $\Delta\mathbf{g}$. If this fails, the update is skipped. Thus, there are two alternative approaches for incorporating a recursive estimate into the NLP framework described. Since the update generated by the SR1 formula is symmetric, but indefinite, one might expect to generate a "more accurate" approximation to the Hessian $\mathbf{H}_L$. However, as in the case of an exact Hessian, it will be necessary to modify the approximation using the Levenberg strategy. In contrast, the BFGS update will not require any modification to maintain positive definiteness. Nevertheless, the BFGS approximation may not be an accurate estimate for an indefinite Hessian.

Table 2.2 summarizes the results of these different strategies on a set of small, dense test problems. The test set given in [108] consists of 68 test problems, nearly all of them found in the collection by Hock and Schittkowski [114]. The NLP algorithm described was used with three different methods to construct the Hessian. Both the FM and M strategies were employed. Finally, the NPSOL [96] algorithm was used as a benchmark. The *baseline* strategy referred to as SR1-FM incorporates the SR1 update in conjunction with the FM option. All results in Table 2.2 are relative—thus, the second column labeled FDH-FM compares the results for a finite difference Hessian and FM option to the baseline SR1-FM performance. The number of function evaluations (including finite difference perturbations) is the quantitative measure used to assess algorithm performance. By definition, all computed quantities (objective and constraints) are evaluated on a single function evaluation. Thus, when comparing FDH-FM and SR1-FM (reading down the second column of the table), one finds that better results were obtained on 9 problems out of 68, where "better" means that the solution was obtained with fewer function evaluations. Worse results were obtained on 48 of the 68 problems using the FDH-FM option and 3 cases were the same. The FDH-FM option also failed on 1 problem that was solved by the baseline and did not solve any more problems than the baseline. Both options failed to find a solution in 7 cases. The final row in the table presents the *average percentage change* in the number of function evaluations. Thus, on average, the FDH-FM option required 70.89% more function evaluations to obtain a solution than the SR1-FM option. It should be noted that a "failure" can occur because of either an algorithmic factor (e.g., maximum iterations) or a problem characteristic (e.g., no solution exists).

**Table 2.2.** *Dense test summary.*

| Algor. | FDH-FM | BFGS-FM | SR1-M | FDH-M | BFGS-M | NPSOL |
|---|---|---|---|---|---|---|
| Better | 9 | 19 | 19 | 7 | 25 | 25 |
| Worse | 48 | 25 | 13 | 51 | 33 | 34 |
| Same | 3 | 15 | 28 | 2 | 7 | 1 |
| Fail | 1 | 2 | 1 | 1 | 4 | 2 |
| Solve | 0 | 0 | 1 | 1 | 0 | 3 |
| Both | 7 | 7 | 6 | 6 | 7 | 4 |
| % $\Delta$ | 70.89 | 22.48 | 2.016 | 69.83 | 22.82 | 16.88 |

An analysis of the results in Table 2.2 suggests a number of trends. First, the use of a finite difference Hessian approximation for small, dense problems is much more expensive than a recursive quasi-Newton method. Second, the SR1 update seems to be somewhat better on average than the BFGS update. Presumably this is because the SR1 update yields a better approximation to the Hessian because it does not require positive definiteness. Third, although there is a slight benefit to the FM strategy in comparison to the M strategy, the advantage is not nearly as significant as it is for large, sparse applications. One can speculate that since a recursive Hessian estimate is poor during early iterations, there is no particular advantage to having "good" multiplier estimates. Finally, it is interesting to note that the results in the last two columns comparing NPSOL with the BFGS-M strategy are quite similar, which is to be expected since they employ nearly identical methods. The minor differences in performance are undoubtedly due to different line-search algorithms and other subtle implementation issues.

## 2.10 Nonlinear Least Squares

### 2.10.1 Background

In contrast to the general NLP problem as defined by (1.98)–(1.100) in Section 1.12, the nonlinear least squares (NLS) problem is characterized by an objective function

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{r}^{\mathsf{T}}(\mathbf{x})\mathbf{r}(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{\ell}\mathbf{r}_i^2, \tag{2.52}$$

where $\mathbf{r}(\mathbf{x})$ is an $\ell$-vector of *residuals*. As for the general NLP, it is necessary to find an $n$-vector $\mathbf{x}$ to minimize $F(\mathbf{x})$ while satisfying the constraints (1.99) and bounds (1.100). The $\ell \times n$ *residual Jacobian* matrix $\mathbf{R}$ is defined by

$$\mathbf{R}^{\mathsf{T}} = [\nabla \mathbf{r}_1, \ldots, \nabla \mathbf{r}_\ell] \tag{2.53}$$

and the gradient vector is

$$\mathbf{g} = \mathbf{R}^{\mathsf{T}}\mathbf{r} = \sum_{i=1}^{\ell} r_i \nabla \mathbf{r}_i. \tag{2.54}$$

And finally, the Hessian of the Lagrangian is given by

$$\mathbf{H}_L(\mathbf{x}, \boldsymbol{\lambda}) = \sum_{i=1}^{\ell} r_i \nabla^2 r_i - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i + \mathbf{R}^{\mathsf{T}}\mathbf{R} \tag{2.55}$$

$$\equiv \mathbf{V} + \mathbf{R}^{\mathsf{T}}\mathbf{R}. \tag{2.56}$$

The matrix $\mathbf{R}^{\mathsf{T}}\mathbf{R}$ is referred to as the *normal matrix* and we shall refer to the matrix $\mathbf{V}$ as the *residual Hessian*.

### 2.10.2 Sparse Least Squares

Having derived the appropriate expressions for the gradient and Hessian of the least squares objective function, one obvious approach is to simply use these quantities as required within the framework of a general NLP problem. Unfortunately, there are a number of well-known difficulties that are related to the normal matrix $\mathbf{R}^{\mathsf{T}}\mathbf{R}$. First, it is quite possible that the Hessian $\mathbf{H}_L$ may be dense even when the residual Jacobian $\mathbf{R}$ is sparse. For example, this can occur when there is one dense row in $\mathbf{R}$. Even if the Hessian is not completely dense, in general, formation of the normal matrix introduces "fill" into an otherwise sparse problem.

Second, it is well known that the normal matrix approach is subject to ill-conditioning even for small, dense problems. In particular, formation of $\mathbf{R}^\mathsf{T}\mathbf{R}$ is prone to cancellation, and the condition number of $\mathbf{R}^\mathsf{T}\mathbf{R}$ is the square of the condition number of $\mathbf{R}$. It is for this reason that the use of the normal matrix is not recommended.

For linearly constrained, linear least squares problems, the augmented matrix of particular significance is

$$\widetilde{\mathbf{D}} = \begin{bmatrix} \widetilde{\mathbf{G}} \\ \mathbf{R} \end{bmatrix}, \tag{2.57}$$

where $\widetilde{\mathbf{G}}$ is the $n_d \times n$ Jacobian of the active constraints and bounds. For small, dense problems the preferred solution technique (cf. [127]) is to introduce an orthogonal decomposition for (2.57) without forming the normal matrix. Furthermore the solution is unique if $\widetilde{\mathbf{D}}$ has rank $n$. Conversely, as example (5.6) illustrates, the linear least squares problem has no unique solution when the number of degrees of freedom $n_d = n - \hat{m}$ exceeds the number of residuals $\ell$. Unfortunately, these techniques do not readily generalize to large, sparse systems when it is necessary to repeatedly modify the active set (and, therefore, the factorization). A survey of the various alternatives is found in [110]. In order to ameliorate the difficulties associated with the normal matrix, while maintaining the benefits of the general sparse NLP Schur-complement method, a "sparse tableau" approach has been adopted.

In general, the objective function is approximated by a quadratic model of the form

$$\mathbf{g}^\mathsf{T}\mathbf{p} + \frac{1}{2}\mathbf{p}^\mathsf{T}\mathbf{H}\mathbf{p}.$$

Combining (2.14) with (2.56), let us proceed formally to "complete the square" and derive an alternative representation for the term

$$\begin{aligned}
\mathbf{p}^\mathsf{T}\mathbf{H}\mathbf{p} &= \mathbf{p}^\mathsf{T}\left[\mathbf{V} + \mathbf{R}^\mathsf{T}\mathbf{R}\right]\mathbf{p} \\
&= \mathbf{p}^\mathsf{T}\mathbf{V}\mathbf{p} + \mathbf{p}^\mathsf{T}\mathbf{R}^\mathsf{T}\mathbf{R}\mathbf{p} \\
&= \mathbf{p}^\mathsf{T}\mathbf{V}\mathbf{p} + \mathbf{p}^\mathsf{T}\mathbf{R}^\mathsf{T}\mathbf{R}\mathbf{p} + \mathbf{p}^\mathsf{T}\mathbf{R}^\mathsf{T}\mathbf{R}\mathbf{p} - \mathbf{p}^\mathsf{T}\mathbf{R}^\mathsf{T}\mathbf{R}\mathbf{p} \\
&= \mathbf{p}^\mathsf{T}\mathbf{V}\mathbf{p} + \mathbf{p}^\mathsf{T}\mathbf{R}^\mathsf{T}\mathbf{w} + \mathbf{w}^\mathsf{T}\mathbf{R}\mathbf{p} - \mathbf{w}^\mathsf{T}\mathbf{w} \\
&= [\mathbf{p}, \mathbf{w}]^\mathsf{T} \begin{bmatrix} \mathbf{V} & \mathbf{R}^\mathsf{T} \\ \mathbf{R} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{w} \end{bmatrix}.
\end{aligned} \tag{2.58}$$

Notice that in the last two lines, we have introduced $\ell$ new variables $\mathbf{w} = \mathbf{R}\mathbf{p}$.

As in the general NLP problem, it is necessary that the QP subproblem be well-posed. Consequently, to correct a defective QP subproblem, the *residual Hessian* is modified:

$$\overline{\mathbf{V}} = \mathbf{V} + \tau(|\sigma| + 1)\mathbf{I}, \tag{2.59}$$

where $\sigma$ is the Gerschgorin bound for the most negative eigenvalue of $\mathbf{V}$. Notice that it is not necessary to modify the entire Hessian matrix for the augmented problem but simply the portion that can contribute to directions of negative curvature. However, since the artificial variables $\mathbf{w}$ are introduced, the inertia test (2.41) must become

$$In(\mathbf{K}_0) = (n_f - \ell, m + \ell, 0) \tag{2.60}$$

to account for the artificial directions.

We then solve an augmented subproblem for the variables $\mathbf{q}^\mathsf{T} = [\mathbf{p},\mathbf{w}]^\mathsf{T}$, i.e., minimize

$$\frac{1}{2}[\mathbf{p},\mathbf{w}]^\mathsf{T} \begin{bmatrix} \overline{\mathbf{V}} & \mathbf{R}^\mathsf{T} \\ \mathbf{R} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{w} \end{bmatrix} + \mathbf{g}^\mathsf{T}\mathbf{p} \tag{2.61}$$

subject to

$$\mathbf{b}_L \le \begin{bmatrix} \mathbf{G} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{w} \end{bmatrix} \le \mathbf{b}_U. \tag{2.62}$$

An alternative form for the augmented problem is suggested in [14] and [16] that requires the explicit introduction of the constraints $\mathbf{w} = \mathbf{R}\mathbf{p}$. However, the technique defined by (2.61) and (2.62) is more compact than the approach in [14] and still avoids formation of the normal matrix. As for the general NLP problem, we choose the Levenberg parameter, $0 \le \tau \le 1$, such that the projected Hessian of the augmented problem is positive definite. We modify the Levenberg parameter at each iteration such that ultimately $\tau \to 0$. Observe that for linear residuals and constraints, the residual Hessian $\mathbf{V} = 0$ and, consequently, $|\sigma| = 0$. Thus, in the linear case, no modification is necessary unless $\mathbf{R}$ is rank deficient. Furthermore, for linearly constrained problems with small residuals at the solution as $\|\mathbf{r}\| \to 0$, $|\sigma| \to 0$, so the modification to the Hessian is "small" even when the Levenberg parameter $\tau \ne 0$. Finally, for large residual problems, i.e., even when $\|\mathbf{r}^*\| \ne 0$, the accelerated trust-region strategy used for the general NLP method adjusts the modification $\tau \to 0$, which ultimately leads to quadratic convergence.

### 2.10.3 Residual Hessian

Because NLS problems require the residual Hessian $\mathbf{V}$, a number of special techniques have been developed specifically for this purpose. One approach is to construct a quasi-Newton approximation for $\mathbf{V}$ itself rather than the full Hessian $\mathbf{H}$. Inserting the definition of the least squares Hessian (2.56) into the secant equation (1.47) gives

$$\overline{\mathbf{B}}\Delta\mathbf{x} = (\overline{\mathbf{V}} + \mathbf{R}^\mathsf{T}\mathbf{R})\Delta\mathbf{x} = \Delta\mathbf{g}. \tag{2.63}$$

Rearranging this expression leads to

$$\overline{\mathbf{V}}\Delta\mathbf{x} = \Delta\mathbf{g} - \mathbf{R}^\mathsf{T}\mathbf{R}\Delta\mathbf{x} \equiv \Delta\mathbf{g}^\#. \tag{2.64}$$

The idea of using an SR1 update (1.49) to construct a quasi-Newton approximation to $\mathbf{V}$ was proposed in [10] and [11]. Dennis, Gay, and Welsch [70] suggest a similar technique in which the DFP update (1.51) is used and the quantity

$$\Delta\mathbf{g}^\# \equiv (\overline{\mathbf{R}} - \mathbf{R})^\mathsf{T}\overline{\mathbf{r}}. \tag{2.65}$$

Other variations have also been suggested and these methods are generally quite effective for dense least squares problems, especially when $\|\mathbf{r}^*\| \gg 0$.

A finite difference method can also be used to construct gradient information for NLS problems. Since both the residual Jacobian and the constraint Jacobian are needed, the quantities being differentiated in (2.1) are defined as

$$\mathbf{q} = \begin{bmatrix} \mathbf{c} \\ \mathbf{r} \end{bmatrix}, \tag{2.66}$$

and then it follows that

$$\mathbf{D} = \begin{bmatrix} \mathbf{G} \\ \mathbf{R} \end{bmatrix}. \tag{2.67}$$

It is also natural to define

$$\boldsymbol{\omega}^{\mathsf{T}} = (-\lambda_1, \ldots, -\lambda_m, r_1, \ldots, r_\ell), \tag{2.68}$$

where $\lambda_k$ are the Lagrange multipliers with $\upsilon = m + \ell$, so that (2.2) becomes

$$\Omega(\mathbf{x}) = \sum_{i=1}^{\upsilon} \omega_i q_i(\mathbf{x}) = -\sum_{i=1}^{m} \lambda_i c_i(\mathbf{x}) + \sum_{i=1}^{\ell} [r_i] r_i(\mathbf{x}). \tag{2.69}$$

It should be recalled that elements of $\boldsymbol{\omega}$ are not perturbed during the finite difference operation. To emphasize this, we have written the second term above as $[r_i]r_i(\mathbf{x})$ since the quantities $[r_i]$ do *not* change during the perturbations. Then it follows that the residual Hessian $\mathbf{V} = \mathbf{E}$, where $\mathbf{E}$ is given by (2.3). This technique plays an important role in large-scale parameter estimation as discussed in Section 5.4.

## 2.11 Barrier Algorithm

### 2.11.1 External Format

For simplicity in presentation, in this section we temporarily drop the notational conventions used elsewhere in the book. The general NLP problem was defined in Section 1.12 and for convenience can be restated using slightly different notation as follows.

Determine the set of $\mathsf{n}$ variables to minimize the nonlinear function $\mathsf{F}$

$$\mathsf{F}(\mathsf{x}) \tag{2.70}$$

subject to $\mathsf{m_C}$ nonlinear constraints

$$\underline{\mathsf{a}} \le \mathsf{a}(\mathsf{x}) \le \overline{\mathsf{a}} \tag{2.71}$$

and $\mathsf{m_B}$ simple variable bounds

$$\underline{\mathsf{x}} \le \mathsf{x} \le \overline{\mathsf{x}}. \tag{2.72}$$

This is referred to as the *external format* of the problem. Equality constraints or fixed variables are specified by setting the lower and upper limits to the same value. One-sided constraints or bounds can be specified as lower or upper bounds by using an “infinite” value for the unconstrained side of the inequalities. When solving a series of related problems it also may be convenient to ignore specified constraints or bounds. With this generality in mind, denote the set of indices of subscripts corresponding to nonignored or *included* equality constraints by $\mathcal{E}$ and the subset of constraints themselves by

$$\underline{\mathsf{a}}_{\mathcal{E}} = \mathsf{a}_{\mathcal{E}}(\mathsf{x}) = \overline{\mathsf{a}}_{\mathcal{E}}.$$

Denote the set of indices for included inequality constraints by $\mathcal{I}$ and the inequality constraints by

$$\underline{\mathsf{a}}_{\mathcal{I}} \le \mathsf{a}_{\mathcal{I}}(\mathsf{x}) \le \overline{\mathsf{a}}_{\mathcal{I}}.$$

Similarly, denote the subset of the $\mathsf{n_{FX}}$ simple bounds that are not ignored and that fix variables as

$$\underline{\mathsf{x}}_{\mathcal{F}} = \mathsf{x}_{\mathcal{F}} = \overline{\mathsf{x}}_{\mathcal{F}}$$

and the subset of included simple bounds on the $\mathsf{n_F}$ free variables as

$$\underline{\mathsf{x}}_{\mathcal{A}} \le \mathsf{x}_{\mathcal{A}} \le \overline{\mathsf{x}}_{\mathcal{A}}.$$

The relevant dimensions of the problem, as they may appear later, are

| | |
|---|---|
| $\mathsf{n}$ | total number of variables |
| $\mathsf{n_{FX}}$ | number of fixed variables |
| $\mathsf{n_F}$ | number of free variables, also the number of included inequality simple bounds |
| $\mathsf{m_C}$ | total number of nonlinear constraints |
| $\mathsf{m_E}$ | number of included equality constraints |
| $\mathsf{m_I}$ | number of included inequality constraints |
| $\mathsf{m_B}$ | total number of simple bounds |

Note that the dimensions as seen by the external format are always denoted in a sans serif font. In contrast, dimensions (and variables) of the problem transformed to internal form will be given in a conventional mathematics font.

## 2.11.2 Internal Format

The external problem format is converted to an internal form that is more convenient for algorithm description. The transformation to internal form includes the following steps.

- Eliminate all constraints marked as ignored or having infinite bounds on both sides. (Further transformations take place only on constraints that are really part of the problem.)
- Eliminate all fixed variables. Only the free variables $x = \mathsf{x}_{\mathcal{A}}$ appear explicitly.
- Introduce one slack variable for each nonlinear inequality constraint to transform the inequality constraint into an equality constraint. Thus the inequality constraint

  $$\underline{\mathsf{a}}_k \le \mathsf{a}_k(\mathsf{x}) \le \overline{\mathsf{a}}_k$$

  is replaced by a nonlinear equality constraint

  $$c_k(\mathbf{x}) \equiv \mathsf{a}_k(\mathsf{x}) - s_k = 0,$$

  and the simple bound

  $$\underline{\mathsf{a}}_k \le s_k \le \overline{\mathsf{a}}_k.$$

  There are $\mathsf{m_I}$ slack variables, which we denote by $\mathbf{s}$.
- Replace two-sided bounds by one-sided bounds. In doing so, eliminate one-sided bounds when the limit is $\pm\infty$. The naming convention for the bounds that are not eliminated is as follows. Denote by $\mathcal{B}_1$ the subset of indices corresponding to the

$\mathsf{n_F}$ inequality simple bounds when a finite lower bound is given. The similar subset when a finite upper bound appears is $\mathcal{B}_2$. The corresponding subsets of the bounds on slack variables (or of the nonlinear inequality constraints) are $\mathcal{B}_3$ and $\mathcal{B}_4$. Each slack variable appears in one or two one-sided simple bounds.

- Modify the form of the problem so that all bounds are simple positivity bounds.

The NLP transformed to internal format is as follows:

Minimize the function

$$F(\mathbf{y}) \equiv \mathsf{F}(\mathsf{x}) \tag{2.73}$$

of the $n = \mathsf{n_F} + \mathsf{m_I}$ variables

$$\mathbf{y} \equiv \begin{pmatrix} \mathbf{x} \\ \mathbf{s} \end{pmatrix} \tag{2.74}$$

subject to the $m_E = \mathsf{m_E} + \mathsf{m_I}$ nonlinear equality constraints

$$\mathbf{c}(\mathbf{y}) \equiv \begin{pmatrix} \mathsf{a}_{\mathcal{E}}(\mathsf{x}) - \underline{\mathsf{a}}_{\mathcal{E}} \\ \mathsf{a}_{\mathcal{I}}(\mathsf{x}) - \mathbf{s} \end{pmatrix} = \mathbf{0} \tag{2.75}$$

and the $m_B$ linear bounds

$$\mathbf{b}(\mathbf{y}) \equiv \begin{pmatrix} (\mathbf{x} - \underline{\mathsf{x}}_{\mathcal{A}})_{\mathcal{B}_1} \\ (\overline{\mathsf{x}}_{\mathcal{A}} - \mathbf{x})_{\mathcal{B}_2} \\ (\mathbf{s} - \underline{\mathsf{a}}_{\mathcal{I}})_{\mathcal{B}_3} \\ (\overline{\mathsf{a}}_{\mathcal{I}} - \mathbf{s})_{\mathcal{B}_4} \end{pmatrix} \geq \mathbf{0}. \tag{2.76}$$

The relevant dimensions of the problem in the internal format are:

| | | |
|---|---|---|
| $n$ | $\equiv \mathsf{n_F} + \mathsf{m_I}$ | total number of internal variables |
| $m_E$ | $\equiv \mathsf{m_E} + \mathsf{m_I}$ | number of nonlinear equality constraints |
| $m_B$ | $\equiv \lvert\mathcal{B}_1\rvert + \lvert\mathcal{B}_2\rvert + \lvert\mathcal{B}_3\rvert + \lvert\mathcal{B}_4\rvert$ | number of simple positivity bounds |

There are a number of features of the transformed problem (2.73)–(2.76) that deserve comment in light of the developments to follow. Observe that the equality constraints (2.75) may be nonlinear functions of the variables $\mathbf{y}$, whereas the inequality constraints (2.76) are strictly linear functions of $\mathbf{y}$. This property makes it straightforward to construct an initial guess $\mathbf{y}^{(0)}$ that is strictly feasible with respect to the inequalities $\mathbf{b}(\mathbf{y}^{(0)}) > \mathbf{0}$. Furthermore, the method to be described constructs iterates that maintain feasibility $\mathbf{b}(\mathbf{y}^{(k)}) > \mathbf{0}$ for all $k$. However, this also implies that the original (external format) inequalities (2.71) may not be strictly feasible until a solution is found. Consequently, let us refer to the approach as an *infeasible barrier method*. With the problem transformed to the internal format (2.73)–(2.76) let us now proceed to develop the details of the interior-point algorithm.

### 2.11.3 Definitions

The *Lagrangian* for the internal format NLP is defined as

$$L(\mathbf{y},\boldsymbol{\eta},\boldsymbol{\lambda}) = F(\mathbf{y}) - \boldsymbol{\eta}^{\mathsf{T}}\mathbf{c}(\mathbf{y}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{b}(\mathbf{y}), \tag{2.77}$$

where $\boldsymbol{\eta}$ is the $m_E$-vector of Lagrange multipliers corresponding to the equality constraints, and $\boldsymbol{\lambda}$ is the $m_B$-vector of Lagrange multipliers corresponding to the inequality (bound) constraints. The gradient of the Lagrangian is given by

$$\nabla_y L(\mathbf{y},\boldsymbol{\eta},\boldsymbol{\lambda}) = \mathbf{g} - \mathbf{C}^{\mathsf{T}}\boldsymbol{\eta} - \mathbf{B}^{\mathsf{T}}\boldsymbol{\lambda}, \tag{2.78}$$

where the gradient of the objective is just

$$\mathbf{g} = \nabla_y F(\mathbf{y}) = \begin{pmatrix} \nabla_x F(\mathbf{x}) \\ \mathbf{0} \end{pmatrix}, \tag{2.79}$$

and the $m_E \times n$ Jacobian of equalities is

$$\mathbf{C} \equiv \begin{pmatrix} \mathbf{A}_{\mathcal{E}} & \mathbf{0} \\ \mathbf{A}_{\mathcal{I}} & -\mathbf{I} \end{pmatrix}, \tag{2.80}$$

with the $m_B \times n$ Jacobian of the bounds given by

$$\mathbf{B} \equiv \begin{pmatrix} \mathbf{I}_{\mathcal{B}_1} & \mathbf{0} \\ -\mathbf{I}_{\mathcal{B}_2} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{\mathcal{B}_3} \\ \mathbf{0} & -\mathbf{I}_{\mathcal{B}_4} \end{pmatrix}. \tag{2.81}$$

The matrices $\mathbf{I}_{\mathcal{B}_2}$, $\mathbf{I}_{\mathcal{B}_4}$, $\mathbf{I}_{\mathcal{B}_1}$, and $\mathbf{I}_{\mathcal{B}_3}$ are rectangular submatrices of an identity matrix, each row having a single nonzero entry of 1.

The Hessian of the Lagrangian is given by

$$\begin{aligned} \nabla^2_{yy} L(\mathbf{y},\boldsymbol{\eta},\boldsymbol{\lambda}) &= \nabla^2_{yy} F(\mathbf{y}) - \sum_{k=1}^{m_E} \eta_k \nabla^2_{yy} c_k(\mathbf{y}) - \sum_{k=1}^{m_B} \lambda_k \nabla^2_{yy} b_k(\mathbf{y}) && (2.82) \\ &= \nabla^2_{yy} F(\mathbf{y}) - \sum_{k=1}^{m_E} \eta_k \nabla^2_{yy} c_k(\mathbf{y}) && (2.83) \\ &= \begin{pmatrix} \mathbf{H}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, && (2.84) \end{aligned}$$

where

$$\mathbf{H}_L \equiv \nabla^2_{xx} F(\mathbf{x}) - \sum_{k=1}^{m_E} \eta_k \nabla^2_{xx} c_k(\mathbf{x}). \tag{2.85}$$

When the projected Hessian is not positive definite consider a modified Hessian matrix that is defined as

$$\mathbf{W} = \begin{pmatrix} \mathbf{H} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{H}_L + \tau(|\sigma|+1)\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{2.86}$$

where $0 \le \tau \le 1$ is a Levenberg parameter and $\sigma$ is the Gerschgorin bound for the most negative eigenvalue of $\mathbf{H}_L$.

Let us denote the solution to (2.73)–(2.76) by $(\mathbf{y}^*, \boldsymbol{\eta}^*, \boldsymbol{\lambda}^*)$. The solution is characterized by the following:

1. **Feasibility.** $\mathbf{c}(\mathbf{y}^*) = \mathbf{0}$ and $\mathbf{b}(\mathbf{y}^*) \ge \mathbf{0}$.

2. **Constraint Qualification.** The gradients of all constraints active at $\mathbf{y}^*$ are linearly independent.

3. **First Order KKT Condition.** The point $(\mathbf{y}^*, \boldsymbol{\eta}^*, \boldsymbol{\lambda}^*)$ is a stationary point of the Lagrangian, i.e., $\nabla_y L = \mathbf{0}$, which means

$$\mathbf{g} - \mathbf{C}^{\mathsf{T}}\boldsymbol{\eta} - \mathbf{B}^{\mathsf{T}}\boldsymbol{\lambda} = \mathbf{0} \tag{2.87}$$

and

$$\lambda_k^* \ge 0, \qquad \lambda_k b_k = 0 \tag{2.88}$$

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

4. **Strict Complementarity.** $\lambda_k^* > 0$ if $b_k = 0$ for $k = 1, \ldots, m_B$.

5. **Second Order KKT Condition.** The projected Hessian of the Lagrangian $\mathbf{Z}^{\mathsf{T}}\mathbf{W}^*\mathbf{Z}$ is positive definite, where $\mathbf{Z}$ is a basis for the null space of the Jacobian of the constraints that are equal to zero at $\mathbf{y}^*$, and $\mathbf{W}^* \equiv \mathbf{W}(\mathbf{y}^*, \boldsymbol{\eta}^*, \boldsymbol{\lambda}^*)$.

### 2.11.4 Logarithmic Barrier Function

The interior-point approach is based on the definition of the *logarithmic barrier function*

$$\beta(\mathbf{y}, \mu) = F(\mathbf{y}) - \mu \sum_{k=1}^{m_B} \ln b_k(\mathbf{y}), \tag{2.89}$$

where $\mu$ is called the *barrier parameter*. Fiacco and McCormick [79] showed that the original constrained problem (2.73)–(2.76) could be replaced by a sequence of problems for successively smaller values of the barrier parameter $\mu$. A classical *primal-barrier* approach would solve a sequence of equality constrained problems, namely minimize

$$\beta(\mathbf{y}, \mu) \tag{2.90}$$

subject to the equalities

$$\mathbf{c}(\mathbf{y}) = \mathbf{0}. \tag{2.91}$$

The necessary optimality conditions for the barrier subproblem (2.90)–(2.91) can be stated in terms of the *barrier Lagrangian*

$$L_\beta(\mathbf{y}, \boldsymbol{\eta}_\mu, \mu) = \beta(\mathbf{y}, \mu) - \boldsymbol{\eta}_\mu^{\mathsf{T}}\mathbf{c}(\mathbf{y}). \tag{2.92}$$

The gradient of the barrier Lagrangian is given by

$$\nabla_y L_\beta = \nabla_y \beta - \mathbf{C}^\mathsf{T}\boldsymbol{\eta}_\mu \tag{2.93}$$

$$= \mathbf{g} - \mu \mathbf{B}^\mathsf{T}\mathbf{D}_b^{-1}\mathbf{e} - \mathbf{C}^\mathsf{T}\boldsymbol{\eta}_\mu \tag{2.94}$$

$$= \mathbf{g} - \mathbf{B}^\mathsf{T}\boldsymbol{\pi}_b - \mathbf{C}^\mathsf{T}\boldsymbol{\eta}_\mu, \tag{2.95}$$

where $\mathbf{D}_b$ is a diagonal matrix

$$\mathbf{D}_b = \mathrm{Diag}(b_1, b_2, \ldots, b_{m_B}), \tag{2.96}$$

and $\mathbf{e}$ is a vector of ones. Observe that the components of the vector

$$\boldsymbol{\pi}_b = \mu \mathbf{D}_b^{-1}\mathbf{e} \tag{2.97}$$

are just $(\pi_b)_k = \mu/b_k$.

The classical approach is to solve a sequence of equality constrained problems (2.90)–(2.91) for successively smaller barrier parameters $\mu$. Ill-conditioning in the solution process is reduced by solving a sequence of problems rather than just beginning with a small value for $\mu$ as illustrated in the example (1.116). Let us denote the local minimizer by $\mathbf{y}_\mu$. Then it follows that the necessary conditions $\nabla_y L_\beta = \mathbf{0}$ and $\nabla_\eta L_\beta = \mathbf{0}$ are just

$$\mathbf{g} - \mathbf{B}^\mathsf{T}\boldsymbol{\pi}_b - \mathbf{C}^\mathsf{T}\boldsymbol{\eta}_\mu = \mathbf{0}, \tag{2.98}$$

$$\mathbf{c}(\mathbf{y}_\mu) = \mathbf{0}. \tag{2.99}$$

Furthermore, it can be shown that

$$\lim_{\mu \to 0} \mathbf{y}_\mu = \mathbf{y}^*, \tag{2.100}$$

$$\lim_{\mu \to 0} \boldsymbol{\eta}_\mu = \boldsymbol{\eta}^*, \tag{2.101}$$

$$\lim_{\mu \to 0} \boldsymbol{\pi}_b = \boldsymbol{\lambda}^*, \tag{2.102}$$

where $(\mathbf{y}^*, \boldsymbol{\eta}^*, \boldsymbol{\lambda}^*)$ is the solution of the original inequality-constrained problem (2.73)–(2.76). This limiting behavior can be exploited to form a modified set of necessary conditions for the unknown quantities $(\mathbf{y}, \boldsymbol{\eta}, \boldsymbol{\lambda})$. In particular if we treat $(\mathbf{y}_\mu, \boldsymbol{\eta}_\mu, \boldsymbol{\pi}_b)$ as estimates for the true values $(\mathbf{y}^*, \boldsymbol{\eta}^*, \boldsymbol{\lambda}^*)$ and explicitly include the condition (2.102), we obtain the *modified primal-dual necessary conditions*

$$\boldsymbol{\Phi}_\mu \equiv \begin{pmatrix} \mathbf{g} - \mathbf{C}^\mathsf{T}\boldsymbol{\eta} - \mathbf{B}^\mathsf{T}\boldsymbol{\lambda} \\ \mathbf{c} \\ \mathbf{D}_b(\boldsymbol{\lambda} - \boldsymbol{\pi}_b) \end{pmatrix} = \mathbf{0}. \tag{2.103}$$

The conditions in the last row of (2.103) often written as

$$b_k(\mathbf{y})\lambda_k - \mu = 0 \tag{2.104}$$

for $k = 1, \ldots, m_B$ are referred to as the "centering" or "approximate complementarity" conditions, since as $\mu \to 0$, the centering condition (2.104) approaches the complementarity condition $b_k(\mathbf{y})\lambda_k = 0$. Like other primal-dual methods (e.g., [173], [88], [87]), this condition is explicitly included here in contrast to a classical approach based solely on (2.98)–(2.99).

### 2.11.5 Computing a Search Direction

The equations (2.103) form a nonlinear system of equations $\boldsymbol{\Phi}_\mu = \mathbf{0}$ in the variables $(\mathbf{y}, \boldsymbol{\eta}, \boldsymbol{\lambda})$. A Taylor series expansion about the current point leads to the Newton equations for the modified primal-dual necessary conditions

$$\mathbf{W}\Delta\mathbf{y} - \mathbf{C}^\mathsf{T}\Delta\eta - \mathbf{B}^\mathsf{T}\Delta\lambda = -(\mathbf{g} - \mathbf{C}^\mathsf{T}\boldsymbol{\eta} - \mathbf{B}^\mathsf{T}\boldsymbol{\lambda}); \tag{2.105}$$

$$\nabla\mathbf{c}_k^\mathsf{T}(\mathbf{y})\Delta\mathbf{y} = -c_k(\mathbf{y}), \qquad k = 1, \dots, m_E; \tag{2.106}$$

$$\lambda_k \nabla\mathbf{b}_k^\mathsf{T}(\mathbf{y})\Delta\mathbf{y} + b_k(\mathbf{y})\Delta\lambda_k = -(b_k(\mathbf{y})\lambda_k - \mu), \qquad k = 1, \dots, m_B, \tag{2.107}$$

where $\mathbf{W}$ is the Hessian matrix defined by (2.86). Rewriting these equations yields the following *unsymmetric primal-dual KKT system*:

$$\begin{pmatrix} \mathbf{W} & \mathbf{C}^\mathsf{T} & \mathbf{B}^\mathsf{T} \\ \mathbf{C} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}_\lambda\mathbf{B} & \mathbf{0} & -\mathbf{D}_b \end{pmatrix} \begin{pmatrix} \Delta\mathbf{y} \\ -\Delta\eta \\ -\Delta\lambda \end{pmatrix} = - \begin{pmatrix} \mathbf{g} - \mathbf{C}^\mathsf{T}\eta - \mathbf{B}^\mathsf{T}\boldsymbol{\lambda} \\ \mathbf{c} \\ \mathbf{D}_b(\boldsymbol{\lambda} - \boldsymbol{\pi}_b) \end{pmatrix}, \tag{2.108}$$

where $\mathbf{D}_\lambda = \mathrm{Diag}(\lambda_1, \lambda_2, \dots, \lambda_{m_B})$.

In principle the unsymmetric system (2.108) can be solved for the Newton step; however, for large, sparse linear systems it may be preferable to solve a KKT system that is scaled so that it is symmetric. Here consider using both row and column scaling. Let us apply scaling only to the third block row and the third block column of (2.108). Scale the rows of the coefficient matrix by the diagonal matrix $\mathbf{D}_r = \mathrm{Diag}(r_1, r_2, \dots, r_{m_B})$. Scale the columns (or the variables) by the diagonal matrix $\mathbf{D}_v = \mathrm{Diag}(v_1, v_2, \dots, v_{m_B})$. Thus application of the scaling matrices to (2.108) yields

$$\begin{aligned} &\begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_r \end{pmatrix} \begin{pmatrix} \mathbf{W} & \mathbf{C}^\mathsf{T} & \mathbf{B}^\mathsf{T} \\ \mathbf{C} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}_\lambda\mathbf{B} & \mathbf{0} & -\mathbf{D}_b \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_v \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_v^{-1} \end{pmatrix} \begin{pmatrix} \Delta\mathbf{y} \\ -\Delta\eta \\ -\Delta\lambda \end{pmatrix} \\ &\qquad = - \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_r \end{pmatrix} \begin{pmatrix} \mathbf{g} - \mathbf{C}^\mathsf{T}\eta - \mathbf{B}^\mathsf{T}\boldsymbol{\lambda} \\ \mathbf{c} \\ \mathbf{D}_b(\boldsymbol{\lambda} - \boldsymbol{\pi}_b) \end{pmatrix}. \end{aligned} \tag{2.109}$$

Multiplying (2.109) through by the scaling matrices gives the *symmetric primal-dual KKT system*

$$\begin{pmatrix} \mathbf{W} & \mathbf{C}^\mathsf{T} & \mathbf{B}^\mathsf{T}\mathbf{D}_v \\ \mathbf{C} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}_r\mathbf{D}_\lambda\mathbf{B} & \mathbf{0} & -\mathbf{D}_r\mathbf{D}_b\mathbf{D}_v \end{pmatrix} \begin{pmatrix} \Delta\mathbf{y} \\ -\Delta\eta \\ -\mathbf{D}_v^{-1}\Delta\lambda \end{pmatrix} = - \begin{pmatrix} \mathbf{g} - \mathbf{C}^\mathsf{T}\eta - \mathbf{B}^\mathsf{T}\boldsymbol{\lambda} \\ \mathbf{c} \\ \mathbf{D}_r\mathbf{D}_b(\boldsymbol{\lambda} - \boldsymbol{\pi}_b) \end{pmatrix}. \tag{2.110}$$

There are a number of possible ways to choose the matrices $\mathbf{D}_r$ and $\mathbf{D}_v$ such that the resulting KKT system is symmetric. In order to achieve symmetry of the KKT system we must have $\mathbf{D}_v = \mathbf{D}_r\mathbf{D}_\lambda$, implying that

$$r_k\lambda_k = v_k \tag{2.111}$$

for $k = 1, \dots, m_B$. Here consider three choices of diagonal entries for $D_r$ and $D_v$ that satisfy (2.111). One alternative is to choose

$$\begin{cases} r_k = \frac{1}{\lambda_k}, \\ v_k = 1, \end{cases} \tag{2.112}$$

which leads to

$$\mathbf{B}_k^{\mathsf{T}}\Delta\mathbf{y} - \frac{b_k(\mathbf{y})}{\lambda_k}\,(-\Delta\lambda_k) = -\frac{b_k(\mathbf{y})}{\lambda_k}(\boldsymbol{\lambda} - \boldsymbol{\pi}_b)_k\,. \tag{2.113}$$

Note that the lower-right diagonal block entry of the KKT matrix is $\frac{b_k(\mathbf{y})}{\lambda_k}$. Unfortunately, for inactive inequality constraints $\lambda_k \to 0$, leading to an entry that becomes infinite. A second alternative is

$$\begin{cases} r_k = 1, \\ v_k = \lambda_k, \end{cases} \tag{2.114}$$

which leads to

$$\lambda_k\mathbf{B}_k^{\mathsf{T}}\Delta\mathbf{y} - b_k(\mathbf{y})\lambda_k\left(-\frac{\Delta\lambda_k}{\lambda_k}\right) = -b_k(\mathbf{y})(\boldsymbol{\lambda} - \boldsymbol{\pi}_b)_k\,. \tag{2.115}$$

In this case, the lower-right diagonal block entry is $b_k(\mathbf{y})\lambda_k$ and the lower left block entry is $\lambda_k\mathbf{B}_k^{\mathsf{T}}$. This situation may also cause difficulties for inactive inequalities, since both of the above entries approach zero as $\mu \to 0$. Finally, consider

$$\begin{cases} r_k = \frac{1}{\sqrt{\lambda_k}}, \\ v_k = \sqrt{\lambda_k}, \end{cases} \tag{2.116}$$

which leads to

$$\sqrt{\lambda_k}\mathbf{B}_k^{\mathsf{T}}\Delta\mathbf{y} - b_k(\mathbf{y})\left(-\frac{\Delta\lambda_k}{\sqrt{\lambda_k}}\right) = -\frac{b_k(\mathbf{y})}{\sqrt{\lambda_k}}(\boldsymbol{\lambda} - \boldsymbol{\pi}_b)_k\,. \tag{2.117}$$

In this case, the lower-right diagonal block entry is $b_k(\mathbf{y})$, which should be well behaved. It is for this reason that (2.116) is the preferred scaling technique.

Further simplification is possible by choosing the row and column scaling matrices $\mathbf{D}_v = \mathbf{D}_r = \mathbf{I}$. By eliminating $\Delta\lambda$ from (2.108) one obtains the *condensed primal-dual KKT system*

$$\begin{pmatrix} \mathbf{W} + \mathbf{B}^{\mathsf{T}}\mathbf{D}_b{}^{-1}\mathbf{D}_\lambda\mathbf{B} & \mathbf{C}^{\mathsf{T}} \\ \mathbf{C} & \mathbf{0} \end{pmatrix}\begin{pmatrix} \Delta\mathbf{y} \\ -\Delta\eta \end{pmatrix} = -\begin{pmatrix} \mathbf{g} - \mathbf{C}^{\mathsf{T}}\boldsymbol{\eta} - \mathbf{B}^{\mathsf{T}}\boldsymbol{\pi}_b \\ \mathbf{c} \end{pmatrix}, \tag{2.118}$$

where

$$\Delta\lambda = -\mathbf{D}_b{}^{-1}\mathbf{D}_\lambda\mathbf{B}\Delta\mathbf{y} - \boldsymbol{\lambda} + \boldsymbol{\pi}_b. \tag{2.119}$$

A Newton iteration requires solving either the linear system (2.110) or (2.118) for the search direction $(\Delta\mathbf{y}, \Delta\eta, \Delta\lambda)$. These sparse symmetric indefinite linear systems can be solved efficiently using the same multifrontal algorithm [4] described in Section 2.3. This technique also provides the *inertia* of the system (2.110) given as a triple defining the number of positive, negative, and zero eigenvalues, respectively. It can be shown that if

$$In(\mathbf{K}) = (n,m,0), \tag{2.120}$$

where $\mathbf{K}$ is the matrix on the left-hand side of (2.110), then the projected Hessian matrix $\mathbf{Z}^{\mathsf{T}}\mathbf{W}\mathbf{Z}$ is positive definite as required by the necessary conditions. The Levenberg parameter $\tau$ in (2.86) can be adjusted to obtain the correct inertia using the procedure described in Section 2.5 and [17]. This creates a well-defined subproblem for the NLP. It is shown in the next section that the above inertia requirements on the barrier method KKT matrix are consistent with the optimality requirements for the original problem.

The search direction computed is a linear prediction for the nonlinear behavior, and as such may not satisfy the inequality constraints $\mathbf{b}(\mathbf{y}) \geq \mathbf{0}$ and $\boldsymbol{\lambda} \geq \mathbf{0}$. However, both of these conditions are linear functions of the variables, and consequently we can modify the *length* of the step such that these conditions are met. Specifically, using a linear estimate for the boundary, i.e., $\mathbf{b}(\overline{\mathbf{y}}) = \mathbf{0}$, the steplength is given by

$$\sigma_y = \min_k \left[ \frac{-b_k(\mathbf{y})}{\nabla \mathbf{b}_k^{\mathsf{T}}(\mathbf{y})\Delta \mathbf{y}} \right], \tag{2.121}$$

where the minimization is over all constraints in the downhill direction, that is, for all $k$ such that $\mathbf{b}_k^{\mathsf{T}}(\mathbf{y})\Delta\mathbf{y} < 0$. In a similar fashion the length of the step in the multipliers must be changed so that $\boldsymbol{\lambda} \geq \mathbf{0}$. Again, making a linear estimate for the boundary at $\overline{\boldsymbol{\lambda}} = \mathbf{0}$ one finds

$$\sigma_\lambda = \min_k \left[ \frac{-\lambda_k}{\Delta\lambda_k} \right] \tag{2.122}$$

for all $k$ such that $\Delta\lambda_k < 0$. Now, a full length step as defined by the scalars $\sigma_y$ and $\sigma_\lambda$ corresponds to a move to the boundary of the feasible region. In order to maintain strict feasibility it is necessary to take some fraction of this step. Following the approach in Gay, Overton, and Wright [88] let us define a fraction of the full step according to

$$\varphi = 1 - \min(.01, 100\mu^2). \tag{2.123}$$

Thus, a "pad" has been introduced that ensures all subproblem iterates remain strictly feasible. Note that because the format of the problem has only *linear* inequalities this computation is particularly simple. In contrast, a formulation that permits nonlinear inequality constraints to appear directly in the barrier function would require a more complex line-search procedure to guarantee the nonlinear constraints remain feasible.

In keeping with most primal-dual methods [173], we attempt to use different steplengths in the primal variables $\mathbf{y}$ and the dual variables $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$. Specifically, consider a correction of the form

$$\begin{pmatrix} \overline{\mathbf{y}} \\ \overline{\boldsymbol{\eta}} \\ \overline{\boldsymbol{\lambda}} \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ \boldsymbol{\eta} \\ \boldsymbol{\lambda} \end{pmatrix} + \alpha \begin{pmatrix} \Delta\mathbf{y} \\ \gamma\Delta\eta \\ \gamma\Delta\lambda \end{pmatrix}. \tag{2.124}$$

Different primal and dual step scaling is defined by setting $\alpha = \hat{\alpha}$, where

$$\hat{\alpha} = \min(1, \varphi\sigma_y), \tag{2.125}$$

$$\gamma = \frac{1}{\hat{\alpha}} \min(1, \varphi\sigma_\lambda). \tag{2.126}$$

The primal and dual steps have the same scaling when

$$\alpha = \tilde{\alpha} = \min(1, \varphi\sigma_y, \varphi\sigma_\lambda), \tag{2.127}$$

with $\gamma = 1$, and in this case the step is just a multiple of the Newton step. Clearly it is desirable to take a Newton step when converging to a solution. Furthermore a scalar multiple of the Newton step must provide local improvement on the KKT conditions. On the other hand computational experience suggests using different primal-dual scaling may improve efficiency. Consequently we first try to use the scaled primal-dual step. If an acceptable point is obtained with $\alpha^{(0)} = \hat{\alpha}$ and $\gamma \neq 1$, it is accepted. If the point is not accepted, we force the same scaling by setting $\alpha^{(0)} = \tilde{\alpha}$ with $\gamma = 1$ and then choose subsequent steps so that $\alpha^{(k)} \leq \hat{\alpha}$ based on some globalization strategy.

## 2.11.6 Inertia Requirements for the Barrier KKT System

Using terminology from an active set method, one can show that the barrier algorithm asymptotically requires the Hessian positive definiteness properties indicated by optimality theory. The analysis also provides some insight into the workings of the interior-point method.

First, consider an active set method. When an active set SQP method solves a QP subproblem, inequalities go in and out of the active set (and the KKT matrix) until the correct active set, say

$$\widehat{\mathbf{A}} \equiv \begin{pmatrix} \mathbf{A}_{\mathcal{E}} \\ \widehat{\mathbf{A}}_{\mathcal{I}} \end{pmatrix}, \tag{2.128}$$

is identified. We know that optimality conditions require only that the projected Hessian $\mathbf{Z}^{\mathsf{T}}\mathbf{H}\mathbf{Z}$ be positive definite, where $\mathbf{Z}$ is a basis for the null space of $\hat{A}$, and $\mathbf{H}$ is the modified Hessian of the Lagrangian with respect to the original problem variables. Note that the Hessian may need to be modified as in [32] even when $\mathbf{Z}^{\mathsf{T}}\mathbf{H}\mathbf{Z}$ is positive definite. This is because the initial active set may result in a projected Hessian that extends outside $\mathbf{Z}^{\mathsf{T}}\mathbf{H}\mathbf{Z}$.

Now, consider an interior-point algorithm. The questions under consideration are the following:

- How does the KKT system change from one iteration to the next?
- Asymptotically, what ensures that the interior-point method does not require more than $\mathbf{Z}^{\mathsf{T}}\mathbf{H}\mathbf{Z}$ to be positive definite?

Note that the only part of the interior-point KKT matrix in (2.118) that changes during the iterations is the $\mathbf{B}^{\mathsf{T}}\mathbf{D}_b{}^{-1}\mathbf{D}_\lambda\mathbf{B}$ portion of the Hessian. The hope is that it changes in a way such that only $\mathbf{Z}^{\mathsf{T}}\mathbf{H}\mathbf{Z}$ need be positive definite near a solution. In (2.118), the matrix $\mathbf{M} \equiv \mathbf{W}+\mathbf{B}^{\mathsf{T}}\mathbf{D}_b{}^{-1}\mathbf{D}_\lambda\mathbf{B}$ becomes the Hessian of the barrier Lagrangian $\nabla^2_{yy}\beta = \mathbf{W}+\mu\mathbf{B}^{\mathsf{T}}\mathbf{D}_b{}^{-2}\mathbf{B}$ when $b_k(y)\lambda_k = \mu$, $k = 1,\ldots,m_B$. Also, the upper block of the right-hand-side vector in (2.118) becomes the gradient of the barrier Lagrangian (2.95) when $b_k(y)\lambda_k = \mu$, $k = 1,\ldots,m_B$.

Consider the positive definiteness requirements for $\mathbf{M}$ that are imposed by the interior-point method. The inertia requirements for the interior-point KKT system imply that $\mathbf{N}^{\mathsf{T}}\mathbf{M}\mathbf{N}$ is positive definite, where $\mathbf{N}$ is the null space of the $\mathbf{C}$ matrix in (2.118). For any

$$\mathbf{y} \equiv \begin{pmatrix} \mathbf{x} \\ \mathbf{s} \end{pmatrix},$$

we have

$$\mathbf{y}^{\mathsf{T}}(\mathbf{W}+\mathbf{B}^{\mathsf{T}}\mathbf{D}_b{}^{-1}\mathbf{D}_\lambda\mathbf{B})\mathbf{y} = \mathbf{x}^{\mathsf{T}}\mathbf{H}\mathbf{x}+\mathbf{s}^{\mathsf{T}}(\mathbf{B}^{\mathsf{T}}\mathbf{D}_b{}^{-1}\mathbf{D}_\lambda\mathbf{B})\mathbf{s}\,. \tag{2.129}$$

Suppose $\mathbf{y}$ is a vector in the null space $\mathbf{N}$. Then, from the definition of $\mathbf{C}$ we have

$$\mathbf{s} = \mathbf{A}_{\mathcal{I}}\mathbf{x}\,. \tag{2.130}$$

Now, positive definiteness requirements for the interior-point method can be examined by considering the two types of vectors $\mathbf{y}$ that can be in the null space $\mathbf{N}$. First, assume that $\mathbf{x}$ belongs to $\mathbf{Z}$, the null space of $\hat{A}$. In this case, the optimality conditions ensure that $\mathbf{x}^{\mathsf{T}}\mathbf{H}\mathbf{x} > 0$. Since the entries in $\mathbf{D}_b$ and $\mathbf{D}_\lambda$ are positive, the second term in (2.129) is nonnegative. Thus, the right-hand side of (2.129) is positive and these null space vectors follow the optimality theory.

Next, suppose that $\mathbf{y} \in \mathbf{N}$ but $\mathbf{x}$ does not belong to $\mathbf{Z}$. Since $\mathbf{y} \in \mathbf{N}$, we must have $\mathbf{A}_{\mathcal{E}}\mathbf{x} = \mathbf{0}$ and $\widehat{\mathbf{s}} = \widehat{\mathbf{A}}_{\mathcal{I}}\mathbf{x} \neq \mathbf{0}$. The definition of $\mathbf{B}$, combined with $\widehat{\boldsymbol{\lambda}} > \mathbf{0}$, $\widehat{\mathbf{b}}(\mathbf{y}) \to \mathbf{0}$, and $\widehat{\mathbf{s}} \neq \mathbf{0}$, ensures that the second term in (2.129) will become more positive than any negative value for $\mathbf{x}^{\mathsf{T}}\mathbf{H}\mathbf{x}$. Thus, the Hessian positive definiteness requirements for the interior-point method asymptotically match those for the QP theory.

## 2.11.7 Filter Globalization

The role of a globalization strategy within the context of a line-search method is essentially to decide whether to accept or reject a particular step. Presumably, the point $(\overline{\mathbf{y}}, \overline{\boldsymbol{\eta}}, \overline{\boldsymbol{\lambda}})$ defined by (2.124) should be accepted if something "good" happens, and otherwise rejected. Many approaches have been proposed for the step acceptance criteria. One obvious technique is to monitor the error in the KKT conditions, i.e., the right-hand side of (2.108)

$$\|\boldsymbol{\Phi}_\mu\| \equiv \left\| \begin{matrix} \mathbf{g} - \mathbf{C}^{\mathsf{T}}\boldsymbol{\eta} - \mathbf{B}^{\mathsf{T}}\boldsymbol{\lambda} \\ \mathbf{c} \\ \mathbf{D}_b(\boldsymbol{\lambda} - \boldsymbol{\pi}_b) \end{matrix} \right\|_\infty . \tag{2.131}$$

Unfortunately, it is well known that reducing $\|\boldsymbol{\Phi}_\mu\|$ does not necessarily correspond to finding a minimizer of $\beta(\mathbf{y}, \mu)$. Gay, Overton, and Wright [88] define an augmented Lagrangian merit function corresponding to the equality constrained barrier subproblem (2.90)–(2.91), and use a "watchdog" technique to force descent while monitoring $\|\boldsymbol{\Phi}_\mu\|$. Forsgren and Gill [87] propose an alternate merit function. However, both of these techniques require the computation of penalty weights, a task that can prove to be problematic. Instead let us consider use of a filter (cf. Section 1.11.4) as an alternative to constructing a merit function. We apply the nonlinear filter as a globalization strategy for the equality constrained barrier subproblem. In particular, for *fixed* $\mu$ we would like to do the following:

Minimize

$$\beta(\mathbf{y}, \mu) = F(\mathbf{y}) - \mu \sum_{k=1}^{m_B} \ln b_k(\mathbf{y}) \tag{2.132}$$

and minimize

$$\left\| \begin{matrix} \mathbf{c}(\mathbf{y}) \\ \mathbf{D}_b(\boldsymbol{\lambda} - \boldsymbol{\pi}_b) \end{matrix} \right\|_\infty . \tag{2.133}$$

Denote the values of the objective and constraint violation at the point $(\mathbf{y}^{(k)}, \boldsymbol{\lambda}^{(k)})$ by

$$\left\{\beta^{(k)}, v^{(k)}\right\} \equiv \left\{\beta(\mathbf{y}^{(k)}, \mu), \left\| \begin{matrix} \mathbf{c}(\mathbf{y}^{(k)}) \\ \mathbf{D}_b(\boldsymbol{\lambda}^{(k)} - \boldsymbol{\pi}_b) \end{matrix} \right\|_\infty \right\}. \tag{2.134}$$

When comparing the information at two different points $(\mathbf{y}^{(k)}, \boldsymbol{\lambda}^{(k)})$ and $(\mathbf{y}^{(j)}, \boldsymbol{\lambda}^{(j)})$, a pair $\{\beta^{(k)}, v^{(k)}\}$ is said to *dominate* another pair $\{\beta^{(j)}, v^{(j)}\}$ if and only if both $\beta^{(k)} \leq \beta^{(j)}$ and $v^{(k)} \leq v^{(j)}$. Using this definition we can then define a *filter* as a list of pairs

$$\mathcal{F} = \begin{bmatrix} \beta^{(1)}, v^{(1)} \\ \beta^{(2)}, v^{(2)} \\ \vdots \\ \beta^{(K)}, v^{(K)} \end{bmatrix} \tag{2.135}$$

such that no pair dominates any other. A new point $\{\beta^{(\ell)}, v^{(\ell)}\}$ is said to be acceptable for inclusion in the filter if it is not dominated by any point in the filter. Conversely, if a new point $\{\beta^{(\ell)}, v^{(\ell)}\}$ is dominated by any point in the filter, it is not acceptable. Thus, if a trial point produces an improvement in *either* the objective *or* the constraint violation over previous iterates, it is accepted. In keeping with [84], two special entries are included in the filter corresponding to the "northwest" and the "southeast" corners. For the northwest corner, the filter includes $\{\beta_{NW}, 0\}$, where $\beta_{NW}$ is a liberal estimate for the upper bound on the objective function. At the southeast corner, an entry $\{-\infty, c_{max}\}$ is included, where $c_{max}$ is an upper bound on the absolute constraint violation. An estimate for $c_{max}$ must be specified by the user, and may be reduced if necessary to ensure convergence of the algorithm. It is important to note that the filter globalization strategy is used within a *single* barrier subproblem. When the barrier parameter is changed, the filter is restarted. Consequently, we are comparing only iterates with the same value for the barrier parameter.

We would like to choose the steplength $\alpha^{(k)}$ such that the new point given by (2.124) is "acceptable" to the filter. If the point is accepted, the information is used to augment the filter for subsequent iterations. If the point is not accepted, we utilize a special line-search strategy that forces step contraction, i.e., $\alpha^{(k)} < \alpha^{(k-1)}$, in order to find a point that is acceptable for inclusion in the filter. This step contraction is utilized only when the primal-dual step scaling is equal, i.e., $\gamma = 1$ in (2.124).

In contrast to a traditional line search, a filter has two different quantities that determine acceptance. Consequently, we compute two different estimates for the steplength—one chosen to reduce the objective, and one chosen to reduce the constraint error. Since $\nabla_y \beta = \mathbf{g} - \mathbf{B}^\mathsf{T} \boldsymbol{\pi}_b$, the slope of the barrier function evaluated at $\alpha = 0$ is

$$\beta_0' = (\Delta \mathbf{y})^\mathsf{T} \nabla_y \beta = \mathbf{g}^\mathsf{T} \Delta \mathbf{y} - (\mathbf{B} \Delta \mathbf{y})^\mathsf{T} \boldsymbol{\pi}_b. \tag{2.136}$$

Let us assume that the current iterate $\bar{\alpha} \equiv \alpha^{(k)}$ has been rejected by the filter. Three pieces of information are available: the function and slope at $\alpha = 0$ and the function value at $\bar{\alpha}$. For a standard merit function, it is common to estimate the new step as the minimizer of a polynomial model that interpolates the known information. However, Murray and Wright [136] suggest a model of the following form:

$$q(\alpha) = a + b\alpha + c\alpha^2 - \mu \ln(d - \alpha), \tag{2.137}$$

where the constant $d$ defines the location of the known singularity in the objective. Clearly, one can choose $d = \sigma_y$ from (2.121) and then compute the remaining coefficients $a, b, c$ so the quadratic log barrier model (2.137) interpolates the known data. Thus, by requiring

$q(0) = \beta(0) = \beta_0$, $q(\bar{\alpha}) = \beta(\bar{\alpha}) = \bar{\beta}$, and $q'(0) = \beta_0'$, one obtains

$$a = \beta_0 + \mu \ln(d); \tag{2.138}$$
$$b = \beta_0' - \mu/d; \tag{2.139}$$
$$c = \frac{1}{\bar{\alpha}^2}\left[\bar{\beta} - a - b\bar{\alpha} + \mu \ln(d - \bar{\alpha})\right]. \tag{2.140}$$

From this information, the minimizer of the quadratic log barrier model (2.137) is

$$\alpha_q^* = \begin{cases} \frac{1}{4c}\left[2cd - b - \sqrt{(b+2cd)^2 + 8\mu c}\right] & \text{if } c \neq 0 \text{ and } \beta_0' < 0, \\ d + \frac{\mu}{b} & \text{if } c = 0 \text{ and } \beta_0' < 0, \\ 0 & \text{if } \beta_0' > 0. \end{cases} \tag{2.141}$$

As a second alternative, we would like to choose a steplength to reduce the constraint violation (2.133). For simplicity in constructing the steplength, let us model the violation by

$$\phi(\alpha) = \frac{1}{2}\left[\mathbf{c}^{\mathsf{T}}\mathbf{c} + (\mathbf{D}_b(\boldsymbol{\lambda} - \boldsymbol{\pi}_b))^{\mathsf{T}}(\mathbf{D}_b(\boldsymbol{\lambda} - \boldsymbol{\pi}_b))\right]. \tag{2.142}$$

The behavior by a quadratic model can be approximated by

$$q(\alpha) = a + b\alpha + c\alpha^2 \tag{2.143}$$

and the model coefficients can be computed to interpolate the function and slope at $\alpha = 0$ as well as the function value at $\bar{\alpha}$. After simplification one finds that the step that minimizes the constraint violation is

$$\alpha_v^* = \frac{\bar{\alpha}^2\phi_0}{\left[\bar{\phi} - \phi_0 + 2\bar{\alpha}\phi_0\right]}. \tag{2.144}$$

As with any line search it is necessary to introduce some heuristics that safeguard the procedure. The basic goal is to choose the largest step when both models predict a step, use one of the models if possible, and resort to bisection otherwise. This philosophy can be implemented as follows:

$$\alpha^* = \begin{cases} \max\left[\alpha_v^*, \alpha_q^*, \kappa_2\bar{\alpha}\right] & \text{if } 0 < \alpha_v^* < \kappa_1\bar{\alpha} \text{ and } 0 < \alpha_q^* < \kappa_1\bar{\alpha}, \\ \max\left[\alpha_v^*, \kappa_2\bar{\alpha}\right] & \text{if } 0 < \alpha_v^* < \kappa_1\bar{\alpha} \text{ and } 0 = \alpha_q^* \text{ or } \alpha_q^* > \kappa_1\bar{\alpha}\ , \\ \max\left[\alpha_q^*, \kappa_2\bar{\alpha}\right] & \text{if } 0 < \alpha_q^* < \kappa_1\bar{\alpha} \text{ and } 0 = \alpha_v^* \text{ or } \alpha_v^* > \kappa_1\bar{\alpha}\ , \\ \max\left[\bar{\alpha}/2, \kappa_2\bar{\alpha}\right] & \text{otherwise.} \end{cases} \tag{2.145}$$

Typically, we choose the constants $\kappa_1 = .99$ and $\kappa_2 = \frac{1}{10}$. The computed value $\alpha^*$ from (2.145) provides an estimate for the steplength $\alpha^{(k)}$ in (2.124) which will construct a strictly contracting sequence until finding a point that is acceptable to the filter. Computational experience suggests that the line search seldom requires more than one or two steps.

It should be noted that the approximate complementarity equations (2.104) appear in the filter. The solution of (2.108) is guaranteed to have the properties of a solution to the barrier KKT system only if the approximate complementarity equations are satisfied. Thus, filter convergence theory [83] can be relied on only when (2.104) is satisfied.

### 2.11.8 Barrier Parameter Update Strategy

An important practical matter when implementing a barrier method is the choice for the barrier parameter $\mu$. Clearly, $\mu$ must converge to zero, in order that $\mathbf{y}_\mu \to \mathbf{y}^*$. Typical early implementations of interior-point methods, as described in Fiacco and McCormick [79], calculate the solution to the barrier subproblem very accurately. Recent computational experience suggests that not only is this unnecessary, but it is computationally expensive. Instead, the preferred technique is to simply get "close" to the central path and then reduce the barrier parameter. Obviously, some quantitative definition of "close" is required, and we have adopted the approach in Gay, Overton, and Wright [88]. Specifically, we will consider a point "close" to the central path if

$$\|\mathbf{\Phi}_\mu\| < \min[\kappa\mu, \epsilon_c], \tag{2.146}$$

where (2.131) defines the error in the KKT conditions, and $\epsilon_c$ is a user-specified central path tolerance. Typically, $\epsilon_c = \kappa = 10$. We will use this relation to decide when the barrier parameter should be reduced.

Let us now describe the procedure for updating the barrier parameter estimate. The following test is applied after completing every step and is essentially a modified version of the procedure described in [88] for computing the new barrier parameter $\widehat{\mu}$:

**if** $(\alpha \geq .1)$ **and** $\|\mathbf{\Phi}_\mu\| < \min[\kappa\mu, \epsilon_c]$ **then**

$$\widehat{\mu} = \begin{cases} 10\mu^2 & \text{if } \mu < 10^{-4}, \\ \mu/10 & \text{if } \mu \geq 10^{-4}, \end{cases} \tag{2.147}$$

**elseif** $\mu$ unchanged for $N_u$ iterations

$$\widehat{\mu} = .9\mu. \tag{2.148}$$

The philosophy of the procedure is to aggressively reduce the barrier parameter when it appears promising to do so, based on the observed behavior of the KKT error $\|\mathbf{\Phi}_\mu\|$. Conversely, if it appears that progress is slow and the barrier parameter is unchanged after $N_u$ iterations, a modest reduction is made. Typically, we use $N_u = 10$ and $\epsilon_c = \kappa = 10$.

### 2.11.9 Initialization

A fundamental property of the barrier algorithm is that the sequence of iterates remain strictly feasible with respect to the bounds (2.76) and inequalities $\boldsymbol{\lambda} \geq \mathbf{0}$, as required by (2.88). In this section, we describe how to initialize the interior-point method. The primal variables $\mathbf{y}$ are simply reset to lie strictly within their bounds as a part of the transformation to internal format (2.74). However, in general, good values for the dual variables $(\boldsymbol{\eta}, \boldsymbol{\lambda})$ and the related barrier parameter $\mu$ are not available. There are a number of possibilities and we suggest three options.

*Option* 1. Suppose values for $\mathbf{y}$ and $\mu$ are known and the corresponding gradient information is also available. In this case, we can choose to use the central path estimate

$$\boldsymbol{\lambda} = \mu\mathbf{D}_b^{-1}\mathbf{e} \tag{2.149}$$

and then compute estimates $\widetilde{\boldsymbol{\eta}}$ for the multipliers $\boldsymbol{\eta}$ to minimize the error in the KKT conditions (2.87), i.e., minimize

$$\|\mathbf{g} - \mathbf{C}^{\mathsf{T}}\widetilde{\boldsymbol{\eta}} - \mathbf{B}^{\mathsf{T}}\boldsymbol{\lambda}\| = \|\mathbf{C}^{\mathsf{T}}\widetilde{\boldsymbol{\eta}} - (\mathbf{g} - \mu\mathbf{B}^{\mathsf{T}}\mathbf{D}_b^{-1}\mathbf{e})\|. \tag{2.150}$$

This is an overdetermined linear least squares problem. Since $\mathbf{b} > \mathbf{0}$, it is clear from (2.149) that $\boldsymbol{\lambda} > \mathbf{0}$ (provided $\mu > 0$). On the other hand, we cannot expect that $\boldsymbol{\lambda}$ is a good approximation to the optimal multipliers $\boldsymbol{\lambda}^*$ unless $\mathbf{y}$ is "near" the central path.

*Option* 2. A second possible approach is to use the central path estimate (2.149) for $\boldsymbol{\lambda}$ and the gradient information at $\mathbf{y}$ and then compute estimates $\tilde{\mu}$ and $\widetilde{\boldsymbol{\eta}}$ to minimize

$$\|\mathbf{g} - \mathbf{C}^{\mathsf{T}}\widetilde{\boldsymbol{\eta}} - \mathbf{B}^{\mathsf{T}}\boldsymbol{\lambda}\| = \left\| \begin{pmatrix} \mathbf{C}^{\mathsf{T}} & \mathbf{B}^{\mathsf{T}}\mathbf{D}_b^{-1}\mathbf{e} \end{pmatrix} \begin{pmatrix} \widetilde{\boldsymbol{\eta}} \\ \tilde{\mu} \end{pmatrix} - \mathbf{g} \right\|. \tag{2.151}$$

Unfortunately, the value of $\tilde{\mu}$ that solves this linear least squares problem may not be positive. So, if the computed estimate $\tilde{\mu} < \mu_L$, where $\mu_L = \max(\mu_\ell, \|c\|_\infty/\bar{\kappa})$ for a user-specified value $\mu_\ell$, we set $\mu = \mu_L$ and recompute using Option 1. The constant $\bar{\kappa}$ is chosen slightly larger than $\kappa$ (say $\bar{\kappa} = 11$).

*Option* 3. The third possibility is to compute estimates $\tilde{\mu}$, $\widetilde{\boldsymbol{\eta}}$, and $\widetilde{\boldsymbol{\lambda}}$ to minimize

$$\left\| \begin{matrix} \mathbf{g} - \mathbf{C}^{\mathsf{T}}\widetilde{\boldsymbol{\eta}} - \mathbf{B}^{\mathsf{T}}\widetilde{\boldsymbol{\lambda}} \\ \mathbf{D}_b\widetilde{\boldsymbol{\lambda}} - \tilde{\mu}\mathbf{e} \end{matrix} \right\| = \left\| \begin{pmatrix} \mathbf{C}^{\mathsf{T}} & \mathbf{B}^{\mathsf{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_b & -\mathbf{e} \end{pmatrix} \begin{pmatrix} \widetilde{\boldsymbol{\eta}} \\ \widetilde{\boldsymbol{\lambda}} \\ \tilde{\mu} \end{pmatrix} - \begin{pmatrix} \mathbf{g} \\ \mathbf{0} \end{pmatrix} \right\|. \tag{2.152}$$

As with Option 2, if the computed solution $\tilde{\mu} < \mu_L$, where $\mu_L = \max(\mu_\ell, \|c\|_\infty/\bar{\kappa})$, then we set $\mu = \mu_L$ and recompute using Option 1. Otherwise, we use $\tilde{\mu}$ and simply truncate the multipliers $\lambda_k = \max(\epsilon_m, \widetilde{\lambda}_k)$, where $\epsilon_m$ is machine precision, to ensure they are strictly positive.

All three initialization options require the solution of a sparse linear least squares problem. The multifrontal algorithm [4] used for solving the KKT system can also be used to construct the minimum norm solution of the linear least squares problem

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|. \tag{2.153}$$

Since the multifrontal method performs a symmetric factorization, if we assume $\mathbf{A}$ is full rank, then we can solve

$$\begin{pmatrix} \mathbf{I} & \mathbf{A}^{\mathsf{T}} \\ \mathbf{A} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ -\mathbf{r} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{b} \end{pmatrix} \tag{2.154}$$

in the *underdetermined* case and

$$\begin{pmatrix} \mathbf{I} & \mathbf{A} \\ \mathbf{A}^{\mathsf{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{r} \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{0} \end{pmatrix} \tag{2.155}$$

when the system is *overdetermined.*

### 2.11.10 Outline of the Primary Algorithm

The preceding sections described the main features of the interior-point algorithm. Let us now outline the main algorithmic operations. The primary algorithm proceeds as follows:

1. Initialization: Compute external to internal transformation; (2.73)–(2.76), define sparsity of $\mathbf{B}$, $\mathbf{C}$, $\mathbf{W}$, permutations, etc. Evaluate function and gradient information at the initial point.
2. Multiplier and Barrier Parameter Initialization: Compute initial values for $\boldsymbol{\eta}$, $\boldsymbol{\lambda}$, and $\mu$ using (2.150), (2.151), or (2.152).
3. Gradient Evaluation and Convergence Tests: Evaluate gradients and then check error in KKT conditions (2.131). Terminate if $\|\boldsymbol{\Phi}_\mu\| \leq \epsilon$ and $\mu \leq \epsilon$ for a tolerance $\epsilon$ (go to step 9).
4. Step Calculations:
   (a) Compute log-barrier function. Initialize filter if necessary.
   (b) Iteration print. Hessian evaluation.
   (c) Levenberg modification.
   (d) Barrier parameter update using (2.147) or (2.148). Restart filter when $\mu$ changes.
   (e) Multiplier reset. If $\|\boldsymbol{\eta}\|_\infty$ is too large, then recompute multipliers using (2.151), and, if this fails, using (2.150). If they are still too large, terminate (go to step 9).
5. Search Direction: Solve the KKT system (2.110) or (2.118) and (2.119) and modify the Hessian (2.86) by adjusting the Levenberg parameter if necessary.
6. Step Calculations: Compute the step scaling (2.121), (2.122), and (2.123).
7. Line Search:
   (a) Compute predicted point (2.124) and then evaluate functions and log-barrier function.
   (b) Check point against filter—if acceptable, then update filter (2.135) and go on; otherwise reduce $\alpha$ using (2.145) and repeat.
8. Update Information: Set $\mathbf{y} \leftarrow \overline{\mathbf{y}}$, $\boldsymbol{\eta} \leftarrow \overline{\boldsymbol{\eta}}$, and $\boldsymbol{\lambda} \leftarrow \overline{\boldsymbol{\lambda}}$, etc. Return to step 3.
9. Barrier Algorithm Termination.

### 2.11.11 Computational Experience

First, let us revisit the Powell example (1.116) introduced previously. Figure 2.2 illustrates the example for $m = 20$ with $\epsilon_c = 10^{-2}$. Notice that the central path is a vertical line in the $x_1x_2$-space. For the sake of illustration a number of points on the central path have been computed accurately, that is, by accurately solving the equality constrained subproblem, and the results are summarized in Table 2.3. Observe that as $\mu \to 0$ the computed values approach a solution at $(0, -1)$ more and more accurately.

The iteration history followed by the interior-point algorithm is presented in Table 2.4 and is shown in Figure 2.2. Note that 12 iterations were required to compute the solution. If the number of constraints in this simple example are increased, it can be used to illustrate another important aspect of interior-point methods. Table 2.5 presents a comparison of the

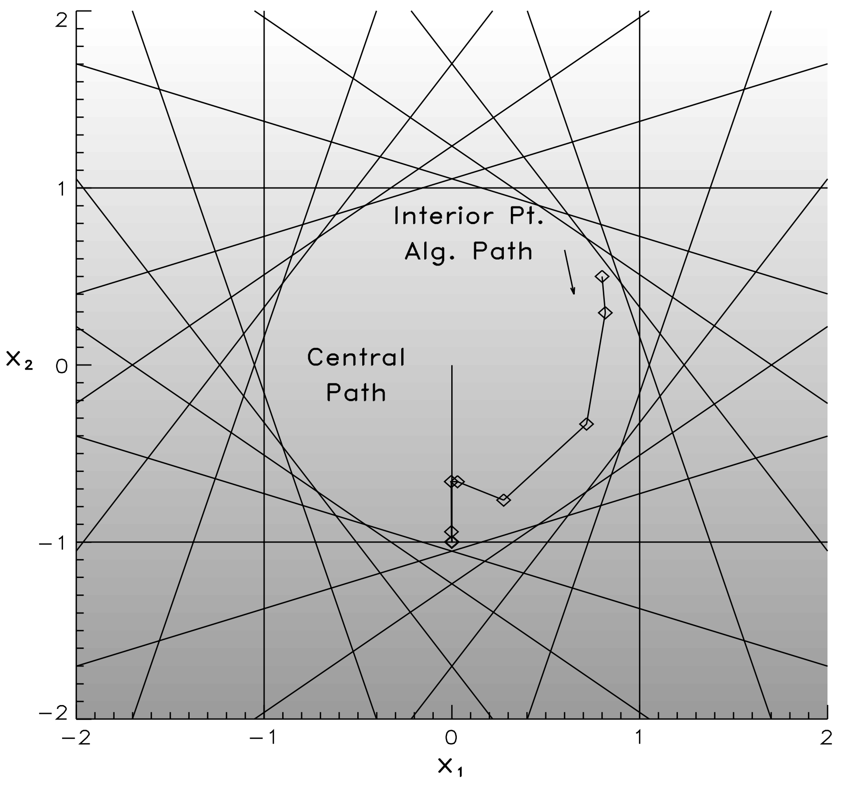


**Figure 2.2.** *Interior-point algorithm behavior.*

**Table 2.3.** *Central path.*

| $\mu$ | $x_1$ | $x_2$ |
|---|---|---|
| 1 | $.443143 \times 10^{-16}$ | $-.992604 \times 10^{-1}$ |
| $10^{-1}$ | $-.326292 \times 10^{-10}$ | $-.658967$ |
| $10^{-2}$ | $-.687390 \times 10^{-9}$ | $-.983966$ |
| $10^{-3}$ | $-.159620 \times 10^{-11}$ | $-.998951$ |
| $10^{-4}$ | $.865020 \times 10^{-6}$ | $-.999900$ |
| $10^{-5}$ | $.353902 \times 10^{-5}$ | $-.999990$ |
| $10^{-6}$ | $.220346 \times 10^{-3}$ | $-.999999$ |

interior-point method with the sparse SQP method presented in Section 2.6, which employs an active set strategy. In particular, we present the number of iterations needed to obtain a solution for three different sized problems. Observe that the number of iterations grows substantially for the active set method, whereas the barrier method computes the solution in nearly the same number of iterations.

**Table 2.4.** *Interior-point iteration history.*

| $x_1$ | $x_2$ | $\|\Phi_\mu\|$ | $\mu$ |
|---|---|---|---|
| .800000 | .500000 | .441090 | $10^{-1}$ |
| .817167 | .294690 | .200266 | $10^{-1}$ |
| .717009 | $-.333391$ | .113042 | $10^{-1}$ |
| .275835 | $-.763470$ | $9.703972 \times 10^{-2}$ | $10^{-1}$ |
| $.299099 \times 10^{-1}$ | $-.659337$ | $5.747828 \times 10^{-2}$ | $10^{-1}$ |
| $-.350958 \times 10^{-2}$ | $-.658608$ | $9.259092 \times 10^{-2}$ | $10^{-2}$ |
| $-.254854 \times 10^{-2}$ | $-.942010$ | $2.195237 \times 10^{-2}$ | $10^{-2}$ |
| $-.496161 \times 10^{-3}$ | $-.999420$ | $1.244132 \times 10^{-2}$ | $10^{-3}$ |
| $-.468141 \times 10^{-3}$ | $-.998461$ | $1.379102 \times 10^{-3}$ | $10^{-4}$ |
| $-.406328 \times 10^{-3}$ | $-.999949$ | $1.096671 \times 10^{-4}$ | $10^{-5}$ |
| $-.370598 \times 10^{-3}$ | $-.999990$ | $1.011066 \times 10^{-5}$ | $10^{-9}$ |
| $-.370515 \times 10^{-3}$ | $-1.00000$ | $1.925492 \times 10^{-9}$ | $10^{-9}$ |

**Table 2.5.** *Iteration comparison.*

| Constraints | Barrier Iterations | QP Iterations |
|---|---|---|
| 20 | 12 | 11 |
| 200 | 12 | 65 |
| 2000 | 13 | 593 |

One final observation with regard to barrier methods is illustrated by this simple example. At the solution the limiting constraint (1.116) takes the form

$$x_1 \cos\left(\frac{\pi}{2}\right) + x_2 \sin\left(\frac{\pi}{2}\right) = x_2 \geq -1. \tag{2.156}$$

Observe from Figure 2.2 that when $x_2 = -1$, there are many points on the limiting constraint with the same value for the objective function. In other words, for this example, the optimal solution point is not unique, even though the optimal objective value is. An active set method such as the simplex algorithm would compute a solution with two active constraints, whereas the optimal solution computed by the barrier algorithm at $(-.370515 \times 10^{-3}, -1)$ has only one limiting constraint. Thus, the barrier method which does not utilize an active set has computed a solution on the optimal *facet*, whereas an active-set method would compute a solution that is at a *vertex*.

Chapter 3

# Optimal Control Preliminaries

## 3.1 The Transcription Method

The preceding chapters focus on methods for solving NLP problems. In the remainder of the book, we turn our attention to the *optimal control problem*. An NLP problem is characterized by a *finite* set of variables $\mathbf{x}$ and constraints $\mathbf{c}$. In contrast, optimal control problems can involve continuous *functions* such as $\mathbf{y}(t)$ and $\mathbf{u}(t)$. It will be convenient to view the optimal control problem as an infinite-dimensional extension of an NLP problem. However, practical methods for solving optimal control problems require Newton-based iterations with a *finite* set of variables and constraints. This goal can be achieved by *transcribing* or converting the infinite-dimensional problem into a finite-dimensional approximation.

Thus, the *transcription method* has three fundamental steps:

1. convert the dynamic system into a problem with a *finite* set of variables; then

2. solve the finite-dimensional problem using a parameter optimization method (i.e., the NLP subproblem); and then

3. assess the accuracy of the finite-dimensional approximation and if necessary repeat the transcription and optimization steps.

We will begin the discussion by focusing on the first step in the process, namely identifying the NLP variables, constraints, and objective function for common applications. In simple terms, we will focus on how to convert an optimal control problem into an NLP problem.

## 3.2 Dynamic Systems

A *dynamic system* is usually characterized mathematically by a set of *ordinary differential equations* (ODEs). Specifically, the dynamics are described for $t_I \le t \le t_F$ by a system of $n_y$ ODEs

$$\dot{\mathbf{y}} = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \vdots \\ \dot{y}_{n_y} \end{bmatrix} = \begin{bmatrix} f_1[y_1(t),\dots,y_{n_y}(t),t] \\ f_2[y_1(t),\dots,y_{n_y}(t),t] \\ \vdots \\ f_{n_y}[y_1(t),\dots,y_{n_y}(t),t] \end{bmatrix} = \mathbf{f}[\mathbf{y}(t),t]. \tag{3.1}$$

For many applications, the variable $t$ is time and it is common to associate the independent variable with a time scale. Of course, mathematically there is no need to make such an assumption and it is perfectly reasonable to define the independent variable in any meaningful way. The system (3.1) is referred to as an *explicit* first order ODE system.

**Example 3.1** ODE EXAMPLE. Consider the following simple system of differential equations:

$$\dot{y}_1 = y_2, \tag{3.2}$$
$$\dot{y}_2 = -y_1 \tag{3.3}$$

defined on the region $0 \le t \le t_F$. It is easy to verify that this system of two first order differential equations is equivalent to the single second order system $\ddot{p}(t) + p(t) = 0$ by identifying $p(t) = y_1(t)$. Now the solution to this system is

$$y_1(t) = \alpha \sin(t+\beta), \tag{3.4}$$
$$y_2(t) = \alpha \cos(t+\beta), \tag{3.5}$$

where $\alpha$ and $\beta$ are arbitrary constants. Since there are two arbitrary constants one can impose two side conditions, but how we do this determines the nature of the ODE.

First, suppose the side conditions are imposed at the initial time

$$y_1(0) = c_1, \qquad y_2(0) = c_2, \tag{3.6}$$

where $c_1$ and $c_2$ are specified constants. Then from (3.4)–(3.5)

$$c_1 = \alpha \sin\beta, \tag{3.7}$$
$$c_2 = \alpha \cos\beta, \tag{3.8}$$

which defines the values for $\beta = \tan^{-1}\frac{c_1}{c_2}$ and $\alpha = c_1/\sin\beta$. The solution is unique for all values of $c_1$ and $c_2$, and this is referred to as an *initial value problem* (IVP) because the side conditions are imposed at the initial time. In general for an IVP, one is given a set of initial values for the dependent variables $\mathbf{y}(t_I)$, called the *initial conditions*, and one must determine the values at some other point $t_F$.

In contrast, for the *boundary value problem*, one must determine the dependent variables such that they have specified values at two or more points, say $t_I$ and $t_F$. The conditions that define the dependent variables are called *boundary conditions*. So instead of (3.6), suppose the side conditions are imposed at both the initial and final times

$$y_1(0) = c_1, \qquad y_1(t_F) = c_2. \tag{3.9}$$

If we choose $t_F = \pi$ and $c_1 = 0$, then from (3.4) we must have

$$y_1(0) = \alpha \sin(\beta) = 0, \tag{3.10}$$
$$y_1(\pi) = \alpha \sin(\pi+\beta) = c_2. \tag{3.11}$$

But $\sin(\beta) = -\sin(\pi + \beta)$, which leads to the conditions

$$\alpha \sin\beta = 0, \tag{3.12}$$

$$-\alpha \sin\beta = c_2. \tag{3.13}$$

Clearly if $c_2 \neq 0$, there is no solution! On the other hand if $c_2 = 0$ and $\beta = 0$, the value of $\alpha$ is arbitrary, and there are an infinite number of solutions! Finally, if we choose $t_F \neq \pi$, then a unique solution can be found. To recapitulate, for the same system of differential equations (3.2)–(3.3), different boundary conditions produce dramatically different results. In contrast to the IVP, for a BVP anything is possible! Although most optimal control problems are BVPs, a number of concepts are usually introduced in the initial value setting.

When the independent variable $t$ does not appear explicitly in the right-hand-side functions, that is, $\mathbf{f} = \mathbf{f}[\mathbf{y}(t)]$, the equations are called an *autonomous system.* Since a nonautonomous system of ODEs can be transformed into an autonomous form by introducing a new variable, say $\mathbf{Y}^\mathsf{T} = (\mathbf{y}, t)^\mathsf{T}$, and a new right-hand side, say $\mathbf{F}^\mathsf{T} = (\mathbf{f}, 1)^\mathsf{T}$, it is sometimes convenient to simply write $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ without loss of generality.

BVPs are often classified by the manner in which the boundary conditions are stated. In general, for a *two-point BVP* the initial and final conditions are given by

$$\boldsymbol{\psi}\left[\mathbf{y}(t_I), \mathbf{y}(t_F)\right] = \mathbf{0}. \tag{3.14}$$

The boundary conditions are said to be *linear* when they can be written as

$$\mathbf{B}_I \mathbf{y}(t_I) + \mathbf{B}_F \mathbf{y}(t_F) = \mathbf{b}, \tag{3.15}$$

where the matrices $\mathbf{B}_I$ and $\mathbf{B}_F$ and vector $\mathbf{b}$ are constant. A boundary condition is said to be *separated* if it involves either $\mathbf{y}(t_I)$ or $\mathbf{y}(t_F)$, but *not* both. Thus for a problem with linear boundary conditions, if row $k$ of $\mathbf{B}_I$ has nonzero values and row $k$ of $\mathbf{B}_F$ is entirely zero, the boundary conditions are both linear and separable. For example, the linear, separable boundary conditions (3.9) can be written using (3.15) as

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_1(t_F) \\ y_2(t_F) \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}. \tag{3.16}$$

For an IVP there is a well-defined sense of direction; that is, it makes perfect sense to propagate *forward* in time. This is not the case for a BVP, and indeed solution methods for the latter are global, rather than local, in nature.

## 3.3 Shooting Method

**Example 3.2** SHOOTING BOUNDARY VALUE PROBLEM. To illustrate the basic concepts, let us consider the simple dynamics:

$$\frac{dy}{dt} = y.$$

Clearly, the analytic solution to the IVP is given by

$$y = y(t_I)e^{t-t_I}.$$

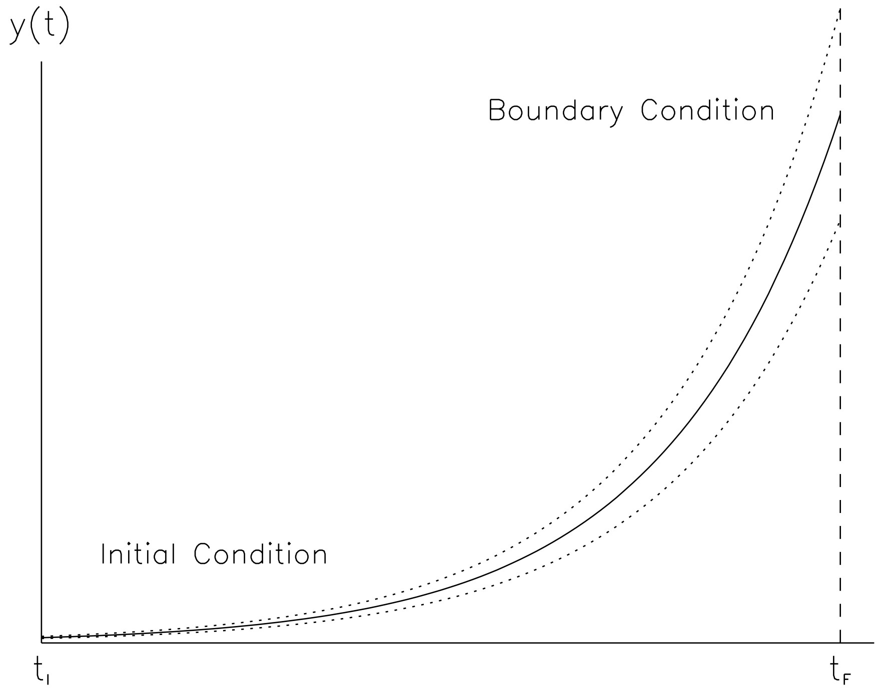


**Figure 3.1.** *Shooting method.*

However, suppose we want to find $y(t_I) \equiv y_I$ such that $y(t_F) = b$, where $b$ is a specified value. This is called a two-point BVP. In particular, using the transcription formulation, it is clear that we can formulate the problem in terms of the single (NLP) variable $x \equiv y_I$. Then it is necessary to solve the single constraint

$$\begin{aligned} c(x) &= y(t_F) - b \\ &= y_I e^{t_F - t_I} - b \\ &= x e^{t_F - t_I} - b \\ &= 0 \end{aligned}$$

by adjusting the variable $x$. Figure 3.1 illustrates the problem.

The approach described is referred to as the *shooting method* and is one of the simplest techniques for solving a BVP. An early practical application of the method required that a cannon be aimed such that the cannonball hit its target, hence explaining the colorful name. Of course, the method is not limited to just a single variable and constraint. In general, the shooting method can be summarized as follows:

1. guess initial conditions $\mathbf{x} = \mathbf{y}(t_I)$;

2. propagate the differential equations from $t_I$ to $t_F$, i.e., "shoot";

3. evaluate the error in the boundary conditions $\mathbf{c}(\mathbf{x}) = \mathbf{y}(t_F) - \mathbf{b}$;

4. use an NLP to adjust the variables $\mathbf{x}$ to satisfy the constraints $\mathbf{c}(\mathbf{x}) = \mathbf{0}$, i.e., repeat steps 1–3.

From a practical standpoint, the shooting method is widely used primarily because the transcribed problem has a small number of variables. Clearly, the number of iteration variables is equal to the number of differential equations. Consequently, any stable implementation of Newton's method is a viable candidate as the iterative technique. Unfortunately, the shooting method also suffers from one major disadvantage. In particular, a small change in the initial condition can produce a very large change in the final conditions. This "tail wagging the dog" effect can result in constraints $\mathbf{c}(\mathbf{x})$ that are *very* nonlinear and, hence, very difficult to solve. The nonlinearity also makes it difficult to construct an accurate estimate of the Jacobian matrix that is needed for a Newton iteration. It should be emphasized that this nonlinear behavior in the boundary conditions can be caused by differential equations that are either nonlinear or *stiff* (or both). Although a discussion of stiffness is deferred temporarily, it should be clear that Example 3.2 would be much harder to solve if we replaced the *linear* differential equation $\dot{y} = y$ by the *linear* differential equation $\dot{y} = 20y$. A more realistic illustration of the shooting method is presented in Example 6.6.

## 3.4 Multiple Shooting Method

**Example 3.3** MULTIPLE SHOOTING EXAMPLE. In order to reduce the sensitivity present in a shooting method, one approach is to simply break the problem into shorter steps, i.e., don't shoot as far. Thus, we might consider a "first step" for $t_I \le t \le t_2$ and a "second step" for $t_2 \le t \le t_F$. For simplicity, let us assume that $t_2 = \frac{1}{2}(t_F + t_I)$. Since the problem has now been broken into two shorter steps, we must guess a value for $y$ at the midpoint in order to start the second step. Furthermore, we must add a constraint to force continuity between the two steps, i.e.,

$$\hat{y}_1 = y_2,$$

where $y(t_2^-) \equiv y_2^- = \hat{y}_1$ denotes the value at the end of the first step and $y_2 \equiv y(t_2^+) = y_2^+$ denotes the value at the beginning of the second step. As a result of this interval splitting, the transcribed problem now has two variables, $\mathbf{x}^\mathsf{T} \equiv [y_I, y_2]$. Furthermore, we must now solve the two constraints

$$\begin{bmatrix} c_1(\mathbf{x}) \\ c_2(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} y_2 - \hat{y}_1 \\ y(t_F) - b \end{bmatrix} = \begin{bmatrix} x_2 - x_1 e^{(t_2 - t_I)} \\ x_2 e^{(t_F - t_2)} - b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Figure 3.2 illustrates the problem.

The approach described is called *multiple shooting* [55, 123] and the constraints enforcing continuity are called *defect* constraints or, simply, defects. Let us generalize the method as follows: compute the unknown initial values $\mathbf{y}(t_I) = \mathbf{y}_I$ such that the boundary condition

$$\mathbf{0} = \boldsymbol{\psi}[\mathbf{y}(t_F), t_F] \tag{3.17}$$

holds for some value of $t_F > t_I$ that satisfies the ODE system (3.1). The fundamental idea of multiple shooting is to break the trajectory into shorter pieces or segments. Thus, we

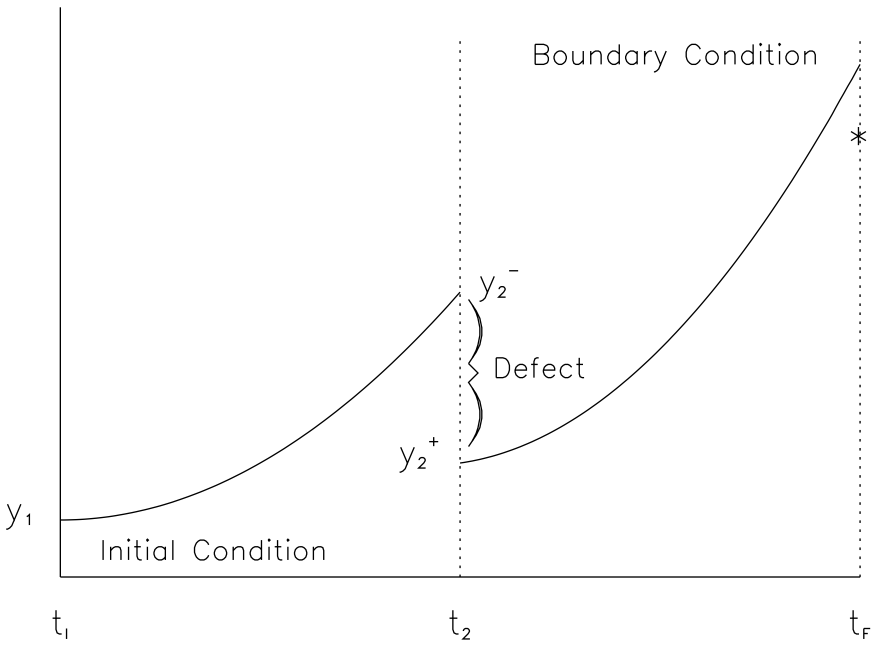


**Figure 3.2.** *Multiple shooting method.*

break the time domain into smaller intervals of the form

$$t_I = t_1 < t_2 < \cdots < t_M = t_F. \tag{3.18}$$

Let us denote $\mathbf{y}_k$ for $k = 1,\ldots,(M-1)$ as the initial value for the dynamic variable at the beginning of segment $k$. For segment $k$ we can propagate (integrate) the differential equations (3.1) from $t_k$ to the end of the segment at $t_{k+1}$. Denote the result of this integration by $\widehat{\mathbf{y}}_k$. Collecting all segments, let us define a set of NLP variables

$$\mathbf{x}^\mathsf{T} = \left(\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_{M-1}\right). \tag{3.19}$$

Now we also must ensure that the segments join at the boundaries; consequently, we impose the constraints

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} \mathbf{y}_2 - \widehat{\mathbf{y}}_1 \\ \mathbf{y}_3 - \widehat{\mathbf{y}}_2 \\ \vdots \\ \boldsymbol{\psi}[\widehat{\mathbf{y}}_M, t_F] \end{bmatrix} = \mathbf{0}. \tag{3.20}$$

One obvious result of the multiple shooting approach is an increase in the size of the problem that the Newton iteration must solve. Additional variables and constraints are introduced for each shooting segment. In particular, the number of NLP variables and constraints for a multiple shooting application is $n = n_y(M-1)$, where $n_y$ is the number of dynamic variables $\mathbf{y}$ and $(M-1)$ is the number of segments. Fortunately, the Jacobian matrix, which is needed to compute the Newton search direction, is *sparse*. In particular, only

$(M-1)n_y^2$ elements in $\mathbf{G}$ are nonzero out of a possible $[(M-1)n_y]^2$. Thus, the percentage of nonzeros is proportional to $1/(M-1)$, indicating that the matrix gets sparser as the number of intervals grows. This sparsity is a direct consequence of the multiple shooting formulation, since variables early in the trajectory do not change constraints later in the trajectory. In fact, Jacobian sparsity is the mathematical consequence of *un*coupling between the multiple shooting segments. For the simple case described, the Jacobian matrix is banded with $n_y \times n_y$ blocks along the diagonal, and the very efficient methods described in Section 2.3 can be used. It is important to note that the multiple shooting segments are introduced strictly for numerical reasons. A more complete discussion of sparsity is presented in Section 4.6. Example 6.7 describes a practical application of multiple shooting to an orbit transfer problem.

An interesting benefit of the multiple shooting algorithm is the ability to exploit a parallel processor. The method is sometimes called *parallel shooting* because the simulation of each segment can be implemented on an individual processor. This technique was explored for a trajectory optimization application [34] and remains an intriguing prospect for multiple shooting methods in general.

## 3.5 Initial Value Problems

In the preceding sections, both the shooting and multiple shooting methods require "propagation" of a set of differential equations. For the simple motivational examples, it was possible to analytically propagate the solution from $t_I$ to $t_F$. However, in general, analytic propagation is not feasible and numerical methods must be employed. The numerical solution of the IVP for ODEs is fundamental to most optimal control methods. The problem can be stated as follows: compute the value of $\mathbf{y}(t_F)$ for some value of $t_I < t_F$ that satisfies (3.1) with the known initial value $\mathbf{y}(t_I) = \mathbf{y}_I$. Numerical methods for solving the ODE IVP are relatively mature in comparison to the other fields in optimal control.

Most schemes can be classified as *one-step methods* or *multistep methods*. Let us begin the discussion with one-step methods. Our goal is to construct an expression over a single step from $t_i$ to $t_{i+1}$, where $h_i$ is referred to as the *integration stepsize* for step $i$. We denote the value of the vector $\mathbf{y}$ at $t_i$ by $\mathbf{y}(t_i) \equiv \mathbf{y}_i$. Proceeding formally to integrate (3.1) yields

$$\begin{aligned} \mathbf{y}_{i+1} &= \mathbf{y}_i + \int_{t_i}^{t_{i+1}} \dot{\mathbf{y}} dt \\ &= \mathbf{y}_i + \int_{t_i}^{t_{i+1}} \mathbf{f}(\mathbf{y}, t) dt. \end{aligned} \tag{3.21}$$

To evaluate the integral, first let us subdivide the integration step into $K$ subintervals

$$\tau_j = t_i + h_i \rho_j \tag{3.22}$$

with

$$0 \le \rho_1 \le \rho_2 \le \cdots \le \rho_K \le 1$$

for $1 \le j \le K$. With this subdivided interval, we now apply a quadrature formula within a quadrature formula. Specifically, we have

$$\int_{t_i}^{t_{i+1}} \mathbf{f}(\mathbf{y},t)dt \approx h_i \sum_{j=1}^{K} \beta_j \widehat{\mathbf{f}}_j, \tag{3.23}$$

where $\widehat{\mathbf{f}}_j \equiv \mathbf{f}(\tau_j, \widehat{\mathbf{y}}_j)$. Notice that this approximation requires values for the variables $\mathbf{y}$ at the intermediate points $\tau_j$, i.e., $\widehat{\mathbf{y}}(\tau_j) \equiv \widehat{\mathbf{y}}_j$. Consequently, we construct these intermediate values using the second expression

$$\int_{t_i}^{\tau_j} \mathbf{f}(\mathbf{y},t)dt \approx h_i \sum_{\ell=1}^{K} \alpha_{j\ell} \mathbf{f}_\ell \tag{3.24}$$

for $1 \le j \le K$.

Collecting results, we obtain a popular family of one-step methods called the *$K$-stage Runge–Kutta* scheme:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \sum_{j=1}^{K} \beta_j \mathbf{f}_{ij}, \tag{3.25}$$

where

$$\mathbf{f}_{ij} = \mathbf{f}\left[\left(\mathbf{y}_i + h_i \sum_{\ell=1}^{K} \alpha_{j\ell} \mathbf{f}_{i\ell}\right), \left(t_i + h_i \rho_j\right)\right] \tag{3.26}$$

for $1 \le j \le K$. $K$ is referred to as the "stage." In these expressions, $\{\rho_j, \beta_j, \alpha_{j\ell}\}$ are known constants with $0 \le \rho_1 \le \rho_2 \le \cdots \le \rho_K \le 1$. A convenient way to define the coefficients is to use the so-called Butcher array

$$\begin{array}{c|ccc} \rho_1 & \alpha_{11} & \dots & \alpha_{1K} \\ \vdots & \vdots & & \vdots \\ \rho_K & \alpha_{K1} & \dots & \alpha_{KK} \\ \hline & \beta_1 & \dots & \beta_K \end{array}.$$

The schemes are called *explicit* if $\alpha_{j\ell} = 0$ for $l \ge j$ and *implicit* otherwise. Four common examples of $K$-stage Runge–Kutta schemes are summarized below:

**Euler Method** (explicit, $K = 1$)

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}.$$

*Common Representation:*

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \mathbf{f}_i. \tag{3.27}$$

**Classical Runge–Kutta Method** (explicit, $K = 4$)

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}.$$

*Common Representation:*

$$\mathbf{k}_1 = h_i \mathbf{f}_i, \tag{3.28}$$

$$\mathbf{k}_2 = h_i \mathbf{f}\left(\mathbf{y}_i + \frac{1}{2}\mathbf{k}_1, t_i + \frac{h_i}{2}\right), \tag{3.29}$$

$$\mathbf{k}_3 = h_i \mathbf{f}\left(\mathbf{y}_i + \frac{1}{2}\mathbf{k}_2, t_i + \frac{h_i}{2}\right), \tag{3.30}$$

$$\mathbf{k}_4 = h_i \mathbf{f}(\mathbf{y}_i + \mathbf{k}_3, t_{i+1}), \tag{3.31}$$

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4). \tag{3.32}$$

**Trapezoidal Method** (implicit, $K = 2$)

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}.$$

*Common Representation:*

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{h_i}{2}(\mathbf{f}_i + \mathbf{f}_{i+1}). \tag{3.33}$$

**Hermite–Simpson Method** (implicit, $K = 3$)

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 5/24 & 1/3 & -1/24 \\ 1 & 1/6 & 2/3 & 1/6 \\ \hline & 1/6 & 2/3 & 1/6 \end{array}.$$

*Common Representation:*

$$\bar{\mathbf{y}} = \frac{1}{2}(\mathbf{y}_i + \mathbf{y}_{i+1}) + \frac{h_i}{8}(\mathbf{f}_i - \mathbf{f}_{i+1}), \tag{3.34}$$

$$\bar{\mathbf{f}} = \mathbf{f}_i\left(\bar{\mathbf{y}}, t_i + \frac{h_i}{2}\right), \tag{3.35}$$

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{h_i}{6}\left(\mathbf{f}_i + 4\bar{\mathbf{f}} + \mathbf{f}_{i+1}\right). \tag{3.36}$$

The Runge–Kutta scheme (3.25)–(3.26) is often motivated in another way. Suppose we consider approximating the solution of the ODE (3.1) by a function $\widetilde{\mathbf{y}}(t)$. As an approximation, let us use a polynomial of degree $K$ (order $K + 1$) over each step $t_i \le t \le t_{i+1}$:

$$\widetilde{\mathbf{y}}(t) = a_0 + a_1(t - t_i) + \cdots + a_K(t - t_i)^K \tag{3.37}$$

with the coefficients $(a_0, a_1, \ldots, a_K)$ chosen such that the approximation matches at the beginning of the step $t_i$, that is,

$$\widetilde{\mathbf{y}}(t_i) = \mathbf{y}_i, \tag{3.38}$$

and has derivatives that match at the points (3.22):

$$\frac{d\tilde{\mathbf{y}}(\tau_j)}{dt} = \mathbf{f}[\mathbf{y}(\tau_j), \tau_j]. \tag{3.39}$$

The conditions (3.39) are called *collocation* conditions and the resulting method is referred to as a *collocation method.* Thus, the Runge–Kutta scheme (3.25)–(3.26) is a collocation method [2], and the solution produced by the method is a piecewise polynomial. While the polynomial representation (3.37) (called a monomial representation) has been introduced for simplicity in this discussion, in practice we will use an equivalent but computationally preferable form called a B-spline representation.

The collocation schemes of particular interest for the remainder of the book, namely (3.33) and (3.36), are both *Lobatto methods.* More precisely, the trapezoidal method is a Lobatto IIIA method of order 2, and the Hermite–Simpson method is a Lobatto IIIA method of order 4 [107, p. 75]. For a Lobatto method, the endpoints of the interval are also collocation points, and consequently $\rho_1 = 0$ and $\rho_K = 1$. The trapezoidal method is based on a quadratic interpolation polynomial. The three coefficients $(a_0, a_1, a_2)$ are constructed such that the function matches at the beginning of the interval (3.38) and the slope matches at the beginning and end of the interval (3.39). For the Hermite–Simpson scheme, the four coefficients $(a_0, a_1, a_2, a_3)$ defining a cubic interpolant are defined by matching the function at the beginning of the interval and the slope at the beginning, midpoint, and end of the interval. A unique property of Lobatto methods is that mesh points are also collocation points. In contrast, *Gauss* schemes impose the collocation conditions strictly interior to the interval, that is, $\rho_1 > 0$ and $\rho_K < 1$. The simplest Gauss collocation scheme is the *midpoint rule*:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \mathbf{f}\left[\frac{1}{2}(\mathbf{y}_{i+1} + \mathbf{y}_i), t_i + \frac{h_i}{2}\right]. \tag{3.40}$$

A *Radau* method imposes the collocation condition at only one end of the interval, specifically $\rho_1 > 0$ and $\rho_K = 1$. The simplest Radau collocation scheme is the *backward Euler method*:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \mathbf{f}_{i+1}. \tag{3.41}$$

An obvious appeal of an explicit scheme is that the computation of each integration step can be performed without iteration; that is, given the value $\mathbf{y}_i$ at the time $t_i$, the value $\mathbf{y}_{i+1}$ at the new time $t_{i+1}$ follows directly from available values of the right-hand-side functions $\mathbf{f}$. In contrast, for an implicit method, the unknown value $\mathbf{y}_{i+1}$ appears nonlinearly; e.g., the trapezoidal method requires

$$0 = \mathbf{y}_{i+1} - \mathbf{y}_i - \frac{h_i}{2}\left[\mathbf{f}(\mathbf{y}_{i+1}, t_{i+1}) + \mathbf{f}(\mathbf{y}_i, t_i)\right] \equiv \boldsymbol{\zeta}_i. \tag{3.42}$$

Consequently, to compute $\mathbf{y}_{i+1}$, given the values $t_{i+1}$, $\mathbf{y}_i$, $t_i$, and $\mathbf{f}[\mathbf{y}_i, t_i]$, requires solving the nonlinear expression (3.42) to drive the *defect* $\boldsymbol{\zeta}_i$ to zero. The iterations required to solve this equation are called *corrector* iterations. An initial guess to begin the iteration is usually provided by the so-called predictor step. There is considerable latitude in the choice of predictor and corrector schemes. For some well-behaved differential equations, a single predictor and corrector step is adequate. In contrast, it may be necessary to perform multiple corrector iterations, e.g., using Newton's method, especially when the differential equations are *stiff*. To illustrate this, suppose that the dynamic behavior is described by two types of variables, namely $\mathbf{y}(t)$ and $\mathbf{u}(t)$. Instead of (3.1), the system dynamics are described by

$$\begin{aligned}\dot{\mathbf{y}} &= \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t], \\ \epsilon\dot{\mathbf{u}} &= \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t],\end{aligned} \tag{3.43}$$

where $\epsilon$ is a "small" parameter. Within a very small region $0 \le t \le t_\epsilon$, the solution displays a rapidly changing behavior and, thereafter, the second equation can effectively be replaced by its limiting form[3]

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{3.44}$$

The resulting system given by

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t], \tag{3.45}$$

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t] \tag{3.46}$$

is called a *semiexplicit differential-algebraic equation* (DAE). In modern control theory, the differential variables $\mathbf{y}(t)$ are called *state variables* and the algebraic variables $\mathbf{u}(t)$ are called *control variables*. The system is semiexplicit because the differential variables appear explicitly (on the left-hand side), whereas the algebraic variables appear implicitly in $\mathbf{f}$ and $\mathbf{g}$. The original stiff system of ODEs (3.43) is referred to as a *singular perturbation* problem and in the limit approaches a DAE system.

**Example 3.4** VAN DER POL'S EQUATION. An example of a singular perturbation problem that arises in electrical circuits is given in second order form by

$$\epsilon\ddot{z} + (z^2 - 1)\dot{z} + z = 0. \tag{3.47}$$

For a modest value of the parameter $\epsilon = 1$, the second order system can be rewritten as

$$\dot{y}_1 = y_2, \tag{3.48}$$

$$\dot{y}_2 = (1 - y_1^2)y_2 - y_1 \tag{3.49}$$

by identifying $y_1 = z$ and $y_2 = \dot{z}$. Now observe that the first two terms in (3.47)

$$\epsilon\ddot{z} + (z^2 - 1)\dot{z} = \frac{d}{dt}\left[\epsilon\dot{z} + \left(\frac{z^3}{3} - z\right)\right],$$

so if we identify the quantity in brackets as

$$y = \epsilon\dot{z} + \left(\frac{z^3}{3} - z\right)$$

and set $u = z$, we can also write

$$\dot{y} = -u, \tag{3.50}$$

$$\epsilon\dot{u} = y - \left(\frac{u^3}{3} - u\right), \tag{3.51}$$

[3]Strictly speaking $\mathbf{g}_u$ (3.59) must be uniformly negative definite.

which is the singular perturbation format (3.43). Clearly, when the parameter $\epsilon = 0$ one obtains the DAE system

$$\dot{y} = -u, \tag{3.52}$$

$$0 = y - \left(\frac{u^3}{3} - u\right). \tag{3.53}$$

The second class of integration schemes are termed *multistep methods* and have the general form

$$\mathbf{y}_{i+k} = \sum_{j=0}^{k-1} \alpha_j \mathbf{y}_{i+j} + h \sum_{j=0}^{k} \beta_j \mathbf{f}_{i+j}, \tag{3.54}$$

where $\alpha_j$ and $\beta_j$ are known constants. If $\beta_k = 0$, then the method is explicit; otherwise, it is implicit. The *Adams schemes* are members of the multistep class that are based on approximating the functions $\mathbf{f}(t)$ by interpolating polynomials. The Adams–Bashforth method is an explicit multistep method [5], whereas the Adams–Moulton method is implicit [135]. Multistep methods must address three issues that we have not discussed for single-step methods. First, as written, the method requires information at $(k-1)$ previous points. Clearly, this implies some method must be used to start the process, and one common technique is to take one or more steps with a one-step method (e.g., Euler). Second, as written, the multistep formula assumes the stepsize $h$ is a fixed value. When the stepsize is allowed to vary, careful implementation is necessary to ensure that the calculation of the coefficients is both efficient and well-conditioned. Finally, similar remarks apply when the number of steps $k$ (i.e., the order) of the method is changed.

Regardless of whether a one-step or multistep method is used, a successful implementation must address the accuracy of the solution. How well does the discrete solution $\mathbf{y}_i$ for $i = 1, 2, \ldots, M$, produced by the integration scheme, agree with the "real" answer $\mathbf{y}(t)$? All well-implemented schemes have some mechanism for adjusting the integration stepsize and/or order to control the integration error. The reader is urged to consult the works of Dahlquist and Björk [66], Stoer and Bulirsch [163], Hindmarsh [113], Shampine and Gordon [158], Hairer, Norsett, and Wanner [106], and Gear [89] for additional information. It is also worth noting that a great deal of discussion has been given to the distinction between explicit and implicit methods. Indeed, it is often tempting to use an explicit method simply because it is more easily implemented (and understood?). However, the optimal control problem is a BVP, *not* an IVP, and, to quote Ascher, Mattheij, and Russell [2, p. 69],

> ... for a boundary value problem ... any scheme becomes effectively, implicit. Thus, the distinction between explicit and implicit initial value schemes becomes less important in the BVP context.

Methods for solving IVPs when dealing with a system of DAEs have appeared more recently. For a semiexplicit DAE system such as (3.45)–(3.46), it is tempting to try to "eliminate" the algebraic (control) variables in order to use a more standard method for solving ODEs. Proceeding formally to solve (3.46) one can write

$$\mathbf{u}(t) = \mathcal{G}^{-1}[\mathbf{y}, t], \tag{3.55}$$

where $\mathcal{G}^{-1}$ is used to denote the inverse of the function $\mathbf{g}$. When this value is substituted into (3.45), one obtains the nonlinear differential equation

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathcal{G}^{-1}[\mathbf{y}, t], t], \tag{3.56}$$

which is amenable to solution using any of the ODE techniques described above.

When analytic elimination is impossible, one alternative is to introduce a nonlinear iterative technique (e.g., Newton's method) that must be executed at every integration step. Consider solving the equations

$$\mathbf{0} = \mathbf{g}[\mathbf{y}_k, \mathbf{u}_k, t_k] \tag{3.57}$$

for the variables $\mathbf{u}_k$ given the variables $\mathbf{y}_k, t_k$. Applying (1.28) and (1.29), we obtain the iteration

$$\overline{\mathbf{u}}_k = \mathbf{u}_k - \mathbf{g}_u^{-1}\mathbf{g}, \tag{3.58}$$

where

$$\mathbf{g}_u = \begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \frac{\partial g_1}{\partial u_2} & \cdots & \frac{\partial g_1}{\partial u_m} \\ \frac{\partial g_2}{\partial u_1} & \frac{\partial g_2}{\partial u_2} & \cdots & \frac{\partial g_2}{\partial u_m} \\ \vdots & & \ddots & \\ \frac{\partial g_m}{\partial u_1} & \frac{\partial g_m}{\partial u_2} & \cdots & \frac{\partial g_m}{\partial u_m} \end{bmatrix}. \tag{3.59}$$

At least in principle, we could repeat the iteration (3.58) until the equations (3.57) are satisfied. This approach is not only very time-consuming, but it can conflict with logic used to control integration error in the dynamic variables $\mathbf{y}$. If an implicit method is used for solving the ODEs, this "elimination" iteration must be performed within each corrector iteration; in other words, it becomes an iteration within an iteration. In simple terms, this is usually *not* the way to solve the problem! However, this discussion does provide one useful piece of information, namely, that successful application of Newton's method requires that $\mathbf{g}_u^{-1}$ can be computed, i.e., that $\mathbf{g}_u$ has full rank.

A word of caution must be interjected at this point. In order to eliminate the variables in the manner presented here, the inverse operation must exist and uniquely define the algebraic variables. However, it is not hard to construct a setting that precludes this operation. For example, suppose the matrix $\mathbf{g}_u$ is rank deficient. In this case, the algebraic constraint does not uniquely specify all of the degrees of freedom, and we could expect some subset (or subspace) of the variables to be undefined. Furthermore, it is not hard to imagine a situation where the rank deficiency in this inverse operation changes with time—full rank in some regions and rank deficient in others.

Instead of using (3.46) to eliminate the control, let us take a slightly different approach. Now if $\mathbf{g}(t) = \mathbf{0}$ must be satisfied over some range of time, then we also expect that the first derivative $\dot{\mathbf{g}} = \mathbf{0}$. Therefore, let us differentiate (3.46) with respect to $t$ yielding

$$0 = \mathbf{g}_y\dot{\mathbf{y}} + \mathbf{g}_u\dot{\mathbf{u}} + \mathbf{g}_t \tag{3.60}$$

$$= \mathbf{g}_y\mathbf{f}[\mathbf{y}, \mathbf{u}] + \mathbf{g}_u\dot{\mathbf{u}} + \mathbf{g}_t, \tag{3.61}$$

where (3.61) follows by substituting the expression for $\dot{\mathbf{y}}$ from (3.45). Now if $\mathbf{g}_u$ is nonsingular, we can solve (3.61) for $\dot{\mathbf{u}}$ and replace the original DAE system (3.45)–(3.46) with

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t], \tag{3.62}$$

$$\dot{\mathbf{u}} = -\mathbf{g}_u^{-1}\left[\mathbf{g}_y \mathbf{f}[\mathbf{y}, \mathbf{u}] + \mathbf{g}_t\right]. \tag{3.63}$$

This is now just a system of differential equations in the new dynamic variables $\mathbf{z}^\mathsf{T} = (\mathbf{y}, \mathbf{u})$. If $\mathbf{g}_u$ is nonsingular, we say the system (3.45)–(3.46) is an *index-one DAE*. On the other hand, if $\mathbf{g}_u$ is rank deficient, we can differentiate (3.60) a second time and repeat the process. If an ODE results, we say the DAE has *index two*. It should also be clear that each differentiation may determine some (but not all) of the algebraic variables. In general, the *DAE index* is the minimum number of times that all or part of the original system must be differentiated with respect to $t$ in order to explicitly determine the algebraic variable. A more complete definition of the DAE index can be found in Brenan, Campbell, and Petzold [48]. For obvious reasons, the process just described is called *index reduction*. It does provide one (not necessarily good) technique for solving DAEs. The example described in Section 4.12 illustrates the use of index reduction.

The first general technique for solving DAEs was proposed by Gear [90] and uses a backward differentiation formula (BDF) in a linear multistep method. The algebraic variables $\mathbf{u}(t)$ are treated the same as the differential variables $\mathbf{y}(t)$. The method was originally proposed for the semiexplicit index-one system described by (3.45)–(3.46) and soon extended to the fully implicit form

$$\mathbf{F}[\dot{\mathbf{z}}, \mathbf{z}, t] = \mathbf{0}, \tag{3.64}$$

where $\mathbf{z} = (\mathbf{y}, \mathbf{u})$. The basic idea of the BDF approach is to replace the derivative $\dot{\mathbf{z}}$ by the derivative of the polynomial that interpolates the solution computed over the preceding $k$ steps. The simplest example is the implicit Euler method, which replaces (3.64) with

$$\mathbf{F}\left[\frac{\mathbf{z}_i - \mathbf{z}_{i-1}}{h_i}, \mathbf{z}_i, t_i\right] = \mathbf{0}. \tag{3.65}$$

The resulting nonlinear system in the unknowns $\mathbf{z}_i$ is usually solved by some form of Newton's method at each time step $t_i$. The widely used production code DASSL, developed by Petzold [140, 141], essentially uses a variable-stepsize, variable-order implementation of the BDF formulas. The method is appropriate for index-one DAEs with consistent initial conditions. Current research into the solution of DAEs with higher index ($\geq 2$) has renewed interest in one-step methods, specifically, the implicit Runge–Kutta (IRK) schemes described for ODEs. RADAU5 [107] implements an IRK method for problems of this type. A discussion of methods for solving DAEs can be found in the books by Brenan, Campbell, and Petzold [48] and Hairer and Wanner [107].

Throughout the book, we have adopted the semiexplicit definition of a DAE as given by (3.45)–(3.46). It is worth noting that a fully implicit form such as (3.64) can be converted to the semiexplicit form by simply writing

$$\dot{\mathbf{y}} = \mathbf{u}(t), \tag{3.66}$$

$$\mathbf{0} = \mathbf{F}\left[\mathbf{y}, \mathbf{u}, t\right]. \tag{3.67}$$

However, as a rule of thumb [48], if the original implicit system (3.64) has index $\nu$, then the semiexplicit form has index $\nu + 1$.

**Example 3.5** DAEs ARE NOT ODEs. As a final note consider the very simple DAE

$$\dot{y} = u, \tag{3.68}$$
$$0 = y - t. \tag{3.69}$$

Clearly the solution is $y(t) = t$ and $u(t) = 1$. Furthermore it is not necessary to specify any initial or final conditions! The problem is uniquely specified as it stands. In fact if one tries to specify $y(0) = y_0$, there is no solution if $y_0 \neq 0$.

## 3.6 Boundary Value Example

**Example 3.6** PUTTING EXAMPLE. To motivate the BVP, Alessandrini [1] describes a problem as follows:

> Suppose that Arnold Palmer is on the 18th green at Pebble Beach. He needs to sink this putt to beat Jack Nicklaus and walk away with the \$1,000,000 grand prize. What should he do? Solve a BVP! By modeling the surface of the green, Arnie sets up the equations of motion of his golf ball.

In [1] the acceleration acting on the golf ball is given by

$$\ddot{\mathbf{z}} = -g\mathbf{k} + gn_3\mathbf{n} - \mu_k g n_3 \frac{\dot{\mathbf{z}}}{\|\dot{\mathbf{z}}\|},$$

where the geometry of the green is modeled using a paraboloid with a minimum at $(10,5)$ given by

$$z_3 = S(z_1,z_2) = \frac{(z_1-10)^2}{125} + \frac{(z_2-5)^2}{125} - 1, \tag{3.70}$$

where the normal force is in the direction defined by

$$\mathbf{n} = \frac{\mathbf{N}}{\|\mathbf{N}\|}, \qquad \mathbf{N} = \left(-\tfrac{\partial S}{\partial z_1}, -\tfrac{\partial S}{\partial z_2}, 1\right).$$

For this example, distance is given in feet, the constant $g = 32.174$ ft/sec$^2$ is the acceleration of gravity, and the constant $\mu_k = 0.2$ is called the kinetic coefficient of friction.

Now since the dynamics are described by a second order differential equation, we can construct an equivalent first order system by introducing new state variables $\mathbf{p} = (\mathbf{z}, \dot{\mathbf{z}})$. The dynamics are then described by the first order system

$$\dot{p}_1 = p_4, \tag{3.71}$$
$$\dot{p}_2 = p_5, \tag{3.72}$$
$$\dot{p}_3 = p_6, \tag{3.73}$$
$$\dot{p}_4 = gn_1n_3 - \mu_k g n_3 \frac{p_4}{s}, \tag{3.74}$$
$$\dot{p}_5 = gn_2n_3 - \mu_k g n_3 \frac{p_5}{s}, \tag{3.75}$$
$$\dot{p}_6 = gn_3n_3 - \mu_k g n_3 \frac{p_6}{s} - g, \tag{3.76}$$

where $s = \sqrt{p_4^2 + p_5^2 + p_6^2}$ is the speed of the ball. The geometry has been defined such that the ball is located at $\mathbf{z} = (p_1, p_2, p_3) = (0,0,0)$ and the hole is located at $\mathbf{z}_H = (20,0,0)$. In [1], the problem is formulated as a two-point BVP with four variables $\mathbf{x} = (\dot{\mathbf{z}}(0), t_F)$ representing the initial velocity imparted when the golf club strikes the ball and the duration of the putt. Obviously, to win \$1,000,000, Arnold Palmer hopes the final position of the ball is in the hole, thus producing three boundary conditions $\mathbf{z}(t_F) = \mathbf{z}_H$. Since there are four variables and only three conditions, Alessandrini suggests that the final boundary condition should be either $\|\dot{\mathbf{z}}(t_F)\| = 0$ or $\|\dot{\mathbf{z}}(t_F)\| \leq s_F$, where $s_F$ is the maximum final speed.

Unfortunately, neither of the proposed boundary conditions is entirely satisfactory. If we use the first suggestion $\|\dot{\mathbf{z}}(t_F)\| = s = 0$, then, clearly, (3.74)–(3.76) have a singularity at the solution! Obviously, this will cause difficulties when the numerical integration procedure attempts to evaluate the ODE at the final time $t_F$. On the other hand, suppose the second alternative is used and the integration is terminated when $0 < s \leq s_F$. This will avoid the singularity at the solution. However, the solution is not unique since there are many possible putts that will yield $s \leq s_F$ and it is not clear how to choose the value $s_F$.

There is a second, less obvious, difficulty with the original formulation that was pointed out by R. Vanderbei. Since the surface of the green is given by (3.70), it follows that the vertical position of the center of mass for the golf ball while on the green is just

$$p_3 = \frac{(p_1 - 10)^2}{125} + \frac{(p_2 - 5)^2}{125} - 1 + r_b, \tag{3.77}$$

where $r_b$ is the radius of the ball. Differentiating, we find that

$$\begin{aligned} \dot{p}_3 &= \frac{2}{125}(p_1 - 10)\dot{p}_1 + \frac{2}{125}(p_2 - 5)\dot{p}_2 \\ &= \frac{2}{125}(p_1 - 10)p_4 + \frac{2}{125}(p_2 - 5)p_5 = p_6. \end{aligned} \tag{3.78}$$

When this equation is differentiated a second time, the resulting expression for $\dot{p}_6$ is not consistent with (3.76) unless the surface of the green given by (3.70) is a plane. Essentially, the formulation in [1] is "too simple" to produce a well-posed numerical problem!

A better approach is to model the motion of the golf ball in two different regions, namely on the green and in the hole. While the ball is on the green, the motion can be defined by four state variables $\mathbf{y} = (p_1, p_2, p_4, p_5)$ and the differential equations

$$\dot{y}_1 = y_3, \tag{3.79}$$

$$\dot{y}_2 = y_4, \tag{3.80}$$

$$\dot{y}_3 = g n_1 n_3 - \mu_k g n_3 \frac{y_3}{s}, \tag{3.81}$$

$$\dot{y}_4 = g n_2 n_3 - \mu_k g n_3 \frac{y_4}{s}. \tag{3.82}$$

By using (3.78) to compute $p_6$, the speed of the ball $s = \sqrt{y_1^2 + y_2^2 + p_6^2(\mathbf{y})}$. One also finds that

$$\mathbf{N} = \left[ -\frac{2}{125}(y_1 - 10), -\frac{2}{125}(y_2 - 5), 1 \right].$$

Thus, all of the quantities in the system (3.79)–(3.82) are completely specified by the four elements in $\mathbf{y}$. If we assume that the hole has a diameter of 4.25 in and the ball has a diameter of 1.68 in, then, when the distance from the center of the ball to the center of the hole is greater than 1.29 in, the ball is on the green. The location of the hole is just $\mathbf{y}_H = (20, 0)$. Thus, mathematically, when $\|\mathbf{y}(t) - \mathbf{y}_H\| \geq R_H$, where $R_H = (4.25/2 - 1.68/2)/12$, the ball is rolling on the green and the dynamics are given by (3.79)–(3.82).

On the other hand, when the golf ball is inside the hole, there is no frictional force term, and the surface geometry constraint is not needed. Consequently, inside the hole, the complete dynamics are described by six (not four) state variables and

$$\dot{y}_1 = y_4, \tag{3.83}$$

$$\dot{y}_2 = y_5, \tag{3.84}$$

$$\dot{y}_3 = y_6, \tag{3.85}$$

$$\dot{y}_4 = 0, \tag{3.86}$$

$$\dot{y}_5 = 0, \tag{3.87}$$

$$\dot{y}_6 = -g. \tag{3.88}$$

Since the time when the ball leaves the green is unknown, we must introduce an additional variable $t_2$, where $t_2 < t_F$, and an additional constraint $\|\mathbf{y}(t_2) - \mathbf{y}_H\| = R_H$. Furthermore, as long as $\sqrt{(y_1(t_F) - 20)^2 + y_2^2(t_F)} \leq R_H$, the ball is in the hole at the end of the trajectory. However, the final state is still not uniquely defined. Although there are other possibilities, it seems reasonable to minimize the horizontal velocity at the final time, i.e., minimize $\dot{y}_1^2 + \dot{y}_2^2$. To summarize, the four NLP variables $\mathbf{x}^\mathsf{T} = (\dot{\mathbf{y}}(0), t_2, t_F)$ must be chosen to satisfy the two NLP constraints

$$\|\mathbf{y}(t_2) - \mathbf{y}_H\| = R_H, \tag{3.89}$$

$$\sqrt{(y_1(t_F) - 20)^2 + y_2^2(t_F)} \leq R_H \tag{3.90}$$

and minimize the objective

$$F(\mathbf{x}) = \dot{y}_1^2 + \dot{y}_2^2. \tag{3.91}$$

Figure 3.3 illustrates two (local) solutions to the problem. It is interesting that the long-time solution takes $t_F = 4.46564$ sec with an optimal objective value $F^* = 0.1865527926$, whereas the short-time solution, which takes $t_F = 2.93612$ sec, yields an almost identical objective value of $F^* = 0.1865528358$.

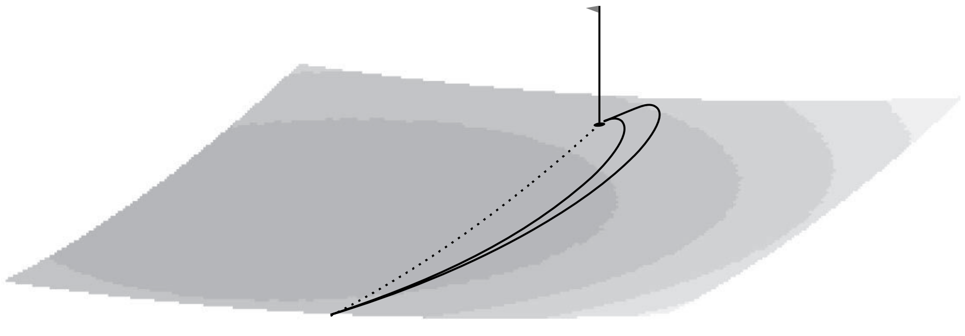


**Figure 3.3.** *Putting example.*

## 3.7 Dynamic Modeling Hierarchy

The golf-putting example described in the previous section suggests that it may be necessary to break a problem into one or more pieces in order to correctly model the physical situation. In the putting example, it made sense to do this because the differential equations were different on and off the green. In fact, in general, it is convenient to view a dynamic trajectory as made up of a collection of *phases*. Specifically, let us define a *phase* as a portion of a trajectory in which the system of DAEs remains *unchanged*. Conversely, different sets of DAEs *must* be in different phases.

The complete description of a problem may require one or more phases. When necessary, phases may be *linked* together by linkage conditions. For the golf-putting example, the first phase (on the green) was linked to the second phase (in the hole) by forcing continuity in the position and velocity across the phase boundary. A phase terminates at an *event*. Thus, for the golf-putting example, phase 1 was terminated at the boundary time $t_2$ by the event *condition* or criterion $\|\mathbf{y}(t_2) - \mathbf{y}_H\| = R_H$. Although it is common for phases to occur sequentially in time, this is not always the case. Rather, a phase should be viewed as a fundamental building block that defines a "chunk" of the dynamics that is needed to construct a complete problem description. In fact, multiphase trajectories with nontrivial linkage conditions permit very complex problem definitions.

Now to reiterate, we have stated that

1. the DAEs must be unchanged within a phase and
2. different sets of DAEs must be in different phases.

However, this does not imply that the DAEs *must* change across a phase boundary. In other words, a phase may be introduced strictly for numerical reasons, as opposed to modeling different physical phenomena. In particular, multiple shooting segments may (or may not) be treated as phases. Furthermore, within a phase, we have a spectrum of possibilities:

1. the phase may have a single multiple shooting segment and a multiple number of integration steps;
2. the phase may be subdivided into a limited number of multiple shooting segments (e.g., 5), with multiple integration steps per segment; or
3. the number of multiple shooting segments may be *equal* to the number of integration steps (i.e., one step per segment).

Traditionally, the first approach is what we have described as the *shooting method* and the second what we have described as *multiple shooting*. Although the remainder of the book will concentrate on the third approach, it is often useful to recall this modeling hierarchy.

## 3.8 Function Generator

### 3.8.1 Description

The whole focus of this chapter has been to present different methods for transcribing a continuous problem described by DAEs into a finite-dimensional problem that can be solved using an NLP algorithm. In fact, we have described different ways to construct a *function generator* for the NLP algorithm. The input to the function generator is the set of NLP variables $\mathbf{x}$. The purpose of the function generator is to compute the constraints $\mathbf{c}(\mathbf{x})$ and objective function $F(\mathbf{x})$ by using one of the transcription methods we have outlined. If it is not possible to compute this information, the function generator can communicate this status by setting a *function error* flag. Thus, the output of the function generator is either the requested constraint and objective function values or a flag indicating that this information cannot be computed. In fact, it is often convenient to view the function generator as a giant subroutine or "black box." Figure 3.4 illustrates this concept.

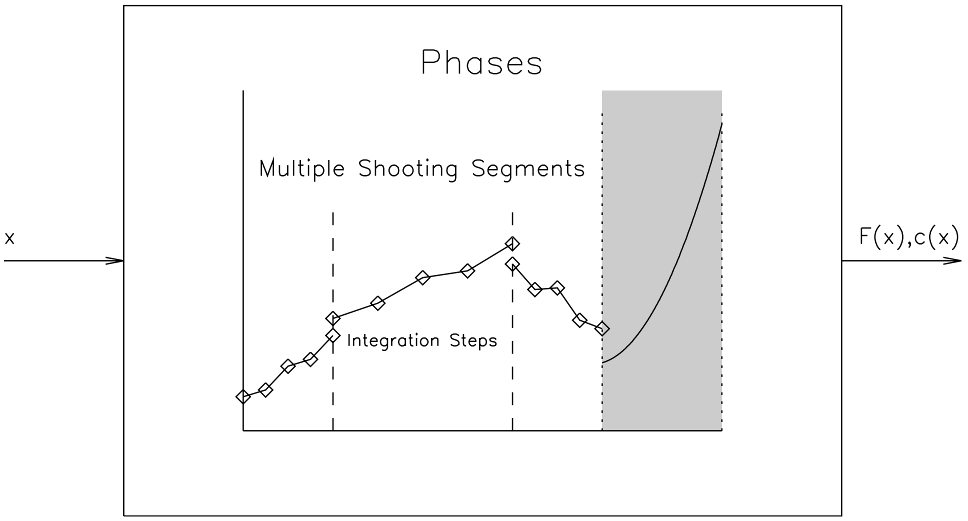


**Figure 3.4.** *Function generator.*

### 3.8.2 NLP Considerations

Section 1.16 described a number of considerations that should be addressed in order to construct a problem that is well-posed and amenable to solution by an NLP algorithm. How do those goals relate to the formulation of an optimal control problem? Clearly, a major goal is to construct a function generator, i.e., select a transcription method that is *noise free*. However, before proceeding any further, it is important to clarify the distinction between *consistency* and *accuracy*. To be more precise

- a *consistent function generator* executes the same sequence of arithmetic operations for all values of $\mathbf{x}$;
- an *accurate* function generator computes accurate approximations to the dynamics $\dot{\mathbf{z}} = \mathbf{f}(\mathbf{z}, t)$.

Thus, an adaptive quadrature method, which implements a sophisticated variable-order and/or stepsize technique, will appear as "noise" to the NLP algorithm. This approach is accurate, but it is not consistent! In contrast, a fixed number of steps with an explicit integration method is a consistent process, but it is not necessarily accurate.

To fully appreciate the importance of this matter, it is worth reviewing our approach. Recall that we intend to choose the NLP variables $\mathbf{x}$ to optimize an NLP objective function $F(\mathbf{x})$ and satisfy a set of NLP constraints defined by the functions $\mathbf{c}(\mathbf{x})$. Newton's method requires first and second derivative information, i.e., the Jacobian and Hessian matrices. When the NLP functions are computed by a function generator, it is clear that we must differentiate the function generator in order to supply the requisite Jacobian and Hessian. The most common approach to computing gradients is via finite difference approximations as described in Section 2.2. A *forward difference* approximation to column $j$ of the Jacobian matrix $\mathbf{G}$ is

$$\mathbf{G}_{\cdot j} = \frac{1}{\delta_j}\left[\mathbf{c}(\mathbf{x}+\mathbf{\Delta}_j) - \mathbf{c}(\mathbf{x})\right], \tag{3.92}$$

where the vector $\mathbf{\Delta}_j = \delta_j \mathbf{e}_j$ and $\mathbf{e}_j$ is a unit vector in direction $j$. From (2.5), a *central difference* approximation is

$$\mathbf{G}_{\cdot j} = \frac{1}{2\delta_j}\left[\mathbf{c}(\mathbf{x}+\mathbf{\Delta}_j) - \mathbf{c}(\mathbf{x}-\mathbf{\Delta}_j)\right]. \tag{3.93}$$

In order to calculate gradient information this way, it is necessary to integrate the differential equations for each perturbation. Consequently, at least $n$ trajectories are required to evaluate a finite difference gradient, and this information may be required for each NLP iteration. A less common alternative to finite difference gradients is to integrate the so-called variational equations. For this technique, described in Section 3.9.4, an additional differential equation is introduced for each derivative and the augmented system of differential equations must be solved along with the state equations. Unfortunately, the variational equations must be derived for each application and, consequently, are used far less in general-purpose software.

While the cost of computing derivatives by finite differences is a matter of practical concern, a more important issue is the accuracy of the gradient information. Forward difference estimates are of order $\delta$, whereas central difference estimates are $\mathcal{O}(\delta^2)$. Of course, the more accurate central difference estimates are twice as expensive as forward difference gradients. Typically, numerical implementations use forward difference estimates until nearly converged and then switch to the more accurate derivatives for convergence. While techniques for selecting the finite difference perturbation size might seem to be critical to accurate gradient evaluation, a number of effective methods are available to deal with this matter [99]. A more crucial matter is the interaction between the gradient computations and the underlying numerical interpolation and integration algorithms. A complete discussion of the matter is given in Section 3.9. We have already discussed how linear interpolation of tabular data can introduce gradient errors. However, it should be emphasized that sophisticated predictor-corrector variable-step, variable-order numerical integration algorithms also introduce "noise" into the gradients. Why? Because the evaluation of a perturbed trajectory is *not* consistent with the nominal trajectory. Thus, while these sophisticated techniques enhance the efficiency of the integration, they degrade the efficiency of the optimization. In fact, a simple fixed-step, fixed-order integrator may yield better overall efficiency in the optimization because the gradient information is more accurate. Two integration methods that are suitable for use inside the function generator are described in Brenan [47], Gear and Vu [91], and Vu [170]. Another approach, referred to as *internal differentiation*, will be described in Section 3.9.3.

Another issue arises in the context of an optimal control application when the final time $t_F$ is defined implicitly by a boundary or event condition instead of explicitly. In this case, we are not asking to integrate from $t_I$ to $t_F$ but rather from $t_I$ *until* $\psi[\mathbf{y}(t_F), t_F] = 0$. Most numerical integration schemes *interpolate* the solution to locate the final point. On the other hand, if the final point is found by *iteration* (e.g., using a root-finding method), the net effect is to introduce noise into the external Jacobian evaluations. A better alternative is to simply add an extra variable and constraint to the overall NLP problem and avoid the use of an "internal" iteration.

In order to avoid the aforementioned difficulties, the approach we will describe in the next chapter does two things. First, we select a transcription method with one integration step per multiple shooting segment. This ensures the function generator is *consistent*. Second, we will treat the solution accuracy *outside* the NLP problem.

## 3.9 Dynamic System Differentiation

Let us consider a slightly more general definition of the dynamics than given in (3.1), namely

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{p}, t], \tag{3.94}$$

where $\mathbf{y}$ is the $n$ dimension state vector, and $\mathbf{p}$ is an $m$ dimension vector of parameters. The initial conditions at the fixed initial time $t_I$ are

$$\boldsymbol{\psi}[\mathbf{y}_I, \mathbf{p}, t_I] = \mathbf{0}, \tag{3.95}$$

where the initial state $\mathbf{y}(t_I)$ is denoted as $\mathbf{y}_I$. Now the solution to (3.94) is a function of the $n+m$ variables

$$\mathbf{x} = \left[\mathbf{y}_I, \mathbf{p}\right] \tag{3.96}$$

as well as time $t$. Let us denote this solution by $\mathbf{y}(\mathbf{x}, t)$. In order to solve a BVP it is necessary to compute derivatives of this ODE solution with respect to the variables $\mathbf{x}$, and clearly the numerical integration and differentiation processes interact. The goal of this section is how to deal effectively with this interaction.

### 3.9.1 Simple Example

To illustrate the alternatives first consider the simple ODE

$$\dot{y} = y^2 \tag{3.97}$$

with initial condition $y(t_I) = y_I$ at the initial time $t = t_I = 0$. The analytic solution to the IVP is given by

$$y^*(t) = \frac{-y_I}{y_I t - 1}. \tag{3.98}$$

Suppose we want to choose $y_I$ to satisfy the boundary condition

$$c(y_I) = y_F^* - a = \left[\frac{-y_I}{y_I t_F - 1}\right] - a = 0. \tag{3.99}$$

Clearly the analytic solution to this simple BVP is just

$$y_I^* = \frac{a}{1+at_F}. \tag{3.100}$$

The derivative of the boundary condition (3.99) with respect to the variable $y_I$ is just

$$c' = (y_I t_F - 1)^{-2}. \tag{3.101}$$

Let us now consider an approximate solution to the dynamic system and subdivide the domain into $M$ steps, where $h = \frac{t_F}{M}$. A simple Euler integration leads to the approximation

$$y_{i+1} = y_i + hf(y_i) \tag{3.102}$$

for $i = 1,\dots,M-1$. Now if the formula is applied recursively with $y_1 = y_I$, one finds

$$\begin{aligned} y_2 &= y_1 + hy_1^2, \\ y_3 &= y_2 + hy_2^2, \\ &\vdots \\ y_M &= y_{M-1} + hy_{M-1}^2. \end{aligned} \tag{3.103}$$

Differentiating (3.103) yields the derivative of the boundary condition when the dynamics are computed using the Euler approximation

$$\begin{aligned} y_2' &= 1 + 2hy_1, \\ y_3' &= y_2' + 2hy_2 y_2', \\ &\vdots \\ c_E' &= y_M'. \end{aligned} \tag{3.104}$$

Clearly the (exact) derivative of the approximate dynamics given by (3.104) is not the same as the (exact) derivative of the true solution given by (3.101).

Now, let us consider solving this very simple BVP in four different ways. For all cases treating the initial condition $y_I$ as the iteration variable $x$ let us use a simple Newton iteration scheme (1.11)

$$x^{(k+1)} = x^{(k)} - \frac{c[x^{(k)}]}{c'[x^{(k)}]}, \tag{3.105}$$

and for simplicity let us choose $t_F = a_F = 1$. With these values the true solution from (3.100) is $x^* = \frac{1}{2}$, and so let us begin the iteration with $x^{(1)} = .6$.

**Example 3.7** Analytic ODE Solution, Analytic Derivative. First let us consider solving the single constraint (3.99) using the Newton iteration (3.105). For the derivative we use the exact expression given by (3.101). Table 3.1 presents the error between the true solution $|\bar{x} - y_I^*|$ and the current iterate, as well as the error in the constraint $|c(x)|$. As expected the iteration converges quadratically to the correct solution.

**Table 3.1.** *BVP iteration: analytic ODE solution, analytic derivative.*

| $k$ | $\lvert x^{(k)}-y_I^*\rvert$ | $\lvert c(x)\rvert$ |
|---|---|---|
| 1 | 0.100 | 0.500 |
| 2 | $0.200\times10^{-1}$ | $0.833\times10^{-1}$ |
| 3 | $0.800\times10^{-3}$ | $0.321\times10^{-2}$ |
| 4 | $0.128\times10^{-5}$ | $0.512\times10^{-5}$ |
| 5 | $0.328\times10^{-11}$ | $0.131\times10^{-10}$ |
| 6 | 0.00 | 0.00 |

**Example 3.8** Discretized ODE Solution, Analytic Derivative of Discretization. For realistic problems it is seldom possible to express the ODE solution in closed form as in (3.98). So let us repeat the experiment using the discretized approximation to the solution given by (3.103). A derivative is needed for the Newton iteration (3.105), and this is given by (3.104). Table 3.2 summarizes the iteration history for discretizations with 10, 100, and 1000 grid points, respectively. Since the Newton iteration uses the exact derivative of the discretization, quadratic convergence of the iterates is still observed. However, now the accuracy of the solution when compared to $y_I^*$ is dictated by the number of grid points.

**Table 3.2.** *BVP iteration: discretized ODE solution with analytic derivative of discretization.*

| $M$ | 10 | | 100 | | 1000 | |
|---|---|---|---|---|---|---|
| $k$ | $\lvert x^{(k)}-y_I^*\rvert$ | $\lvert c(x)\rvert$ | $\lvert x^{(k)}-y_I^*\rvert$ | $\lvert c(x)\rvert$ | $\lvert x^{(k)}-y_I^*\rvert$ | $\lvert c(x)\rvert$ |
| 1 | 0.100 | 0.200 | 0.100 | 0.459 | 0.100 | 0.496 |
| 2 | $0.474\times10^{-1}$ | $0.138\times10^{-1}$ | $0.217\times10^{-1}$ | $0.708\times10^{-1}$ | $0.202\times10^{-1}$ | $0.820\times10^{-1}$ |
| 3 | $0.432\times10^{-1}$ | $0.761\times10^{-4}$ | $0.483\times10^{-2}$ | $0.230\times10^{-2}$ | $0.120\times10^{-2}$ | $0.310\times10^{-2}$ |
| 4 | $0.432\times10^{-1}$ | $0.236\times10^{-8}$ | $0.424\times10^{-2}$ | $0.257\times10^{-5}$ | $0.425\times10^{-3}$ | $0.478\times10^{-5}$ |
| 5 | $0.432\times10^{-1}$ | $0.222\times10^{-15}$ | $0.424\times10^{-2}$ | $0.325\times10^{-11}$ | $0.423\times10^{-3}$ | $0.114\times10^{-10}$ |
| 6 | $0.432\times10^{-1}$ | 0.00 | $0.424\times10^{-2}$ | $-0.666\times10^{-15}$ | $0.423\times10^{-3}$ | $-0.167\times10^{-14}$ |
| 7 | | | $0.424\times10^{-2}$ | $0.444\times10^{-15}$ | $0.423\times10^{-3}$ | $0.888\times10^{-15}$ |
| 8 | | | $0.424\times10^{-2}$ | 0.00 | $0.423\times10^{-3}$ | 0.00 |

**Example 3.9** Discretized ODE Solution, Finite Difference Derivative of Discretization. The results illustrated in Example 3.8 demonstrate rapid convergence of the BVP iteration provided exact (analytic) derivatives of the ODE discretization are used. What if finite difference derivatives are used instead? So let us repeat the experiment using the discretized approximation to the solution given by (3.103). However, let us use the forward difference derivative

$$c' \approx \frac{y_M[x^{(k)}+\delta]-y_M[x^{(k)}]}{\delta} \tag{3.106}$$

with $\delta = 10^{-3}$ for the Newton iteration (3.105) instead of the exact expression given by (3.104). Table 3.3 summarizes the iteration history for discretizations with 10, 100, and 1000 grid points, respectively. Since the Newton iteration uses the forward difference approximation to the derivative of the discretization, convergence of the iterates is degraded somewhat when compared to Example 3.8. Nevertheless, the overall behavior compares favorably in all other respects.

**Table 3.3.** *BVP iteration: discretized ODE solution with finite difference derivative of discretization.*

| $M$ | 10 | | 100 | | 1000 | |
|---|---|---|---|---|---|---|
| $k$ | $\|x^{(k)} - y_I^*\|$ | $\|c(x)\|$ | $\|x^{(k)} - y_I^*\|$ | $\|c(x)\|$ | $\|x^{(k)} - y_I^*\|$ | $\|c(x)\|$ |
| 1 | 0.100 | 0.200 | 0.100 | 0.459 | 0.100 | 0.496 |
| 2 | $0.474\times10^{-1}$ | $0.140\times10^{-1}$ | $0.219\times10^{-1}$ | $0.716\times10^{-1}$ | $0.204\times10^{-1}$ | $0.829\times10^{-1}$ |
| 3 | $0.432\times10^{-1}$ | $0.972\times10^{-4}$ | $0.487\times10^{-2}$ | $0.248\times10^{-2}$ | $0.125\times10^{-2}$ | $0.332\times10^{-2}$ |
| 4 | $0.432\times10^{-1}$ | $0.133\times10^{-6}$ | $0.424\times10^{-2}$ | $0.774\times10^{-5}$ | $0.426\times10^{-3}$ | $0.121\times10^{-4}$ |
| 5 | $0.432\times10^{-1}$ | $0.176\times10^{-9}$ | $0.424\times10^{-2}$ | $0.149\times10^{-7}$ | $0.423\times10^{-3}$ | $0.242\times10^{-7}$ |
| 6 | $0.432\times10^{-1}$ | $0.233\times10^{-12}$ | $0.424\times10^{-2}$ | $0.285\times10^{-10}$ | $0.423\times10^{-3}$ | $0.481\times10^{-10}$ |
| 7 | $0.432\times10^{-1}$ | $0.222\times10^{-15}$ | $0.424\times10^{-2}$ | $0.540\times10^{-13}$ | $0.423\times10^{-3}$ | $0.888\times10^{-13}$ |
| 8 | $0.432\times10^{-1}$ | 0.00 | $0.424\times10^{-2}$ | 0.00 | $0.423\times10^{-3}$ | $0.933\times10^{-14}$ |
| 9 | | | | | $0.423\times10^{-3}$ | $-0.233\times10^{-14}$ |
| 10 | | | | | $0.423\times10^{-3}$ | $0.222\times10^{-15}$ |
| 11 | | | | | $0.423\times10^{-3}$ | 0.00 |

**Example 3.10** DISCRETIZED ODE SOLUTION, ANALYTIC DERIVATIVE OF EXACT SOLUTION. As a final experiment let us consider solving the discretized ODE problem using the analytic (exact) derivative of the true solution (3.101). Table 3.4 summarizes the iteration history for discretizations with 10, 100, and 1000 grid points, respectively. It might be expected that the analytic derivative should approximate the derivative of the discretization, especially as $M \to \infty$. In fact the BVP iteration converges rapidly for $M = 1000$. However, the analytic derivative (3.101) is *not* the derivative of the discrete approximation (3.103), and for small values of $M$ the BVP iteration does not converge at all!

**Table 3.4.** *BVP iteration: discretized ODE solution, analytic derivative of exact solution.*

| $M$ | 10 | | 100 | | 1000 | |
|---|---|---|---|---|---|---|
| $k$ | $\|x^{(k)} - y_I^*\|$ | $\|c(x)\|$ | $\|x^{(k)} - y_I^*\|$ | $\|c(x)\|$ | $\|x^{(k)} - y_I^*\|$ | $\|c(x)\|$ |
| 1 | 0.100 | 0.200 | 0.100 | 0.459 | 0.100 | 0.496 |
| 2 | $0.680\times10^{-1}$ | $0.836\times10^{-1}$ | $0.266\times10^{-1}$ | $0.914\times10^{-1}$ | $0.207\times10^{-1}$ | $0.843\times10^{-1}$ |
| 3 | $0.524\times10^{-1}$ | $0.304\times10^{-1}$ | $0.611\times10^{-2}$ | $0.734\times10^{-2}$ | $0.132\times10^{-2}$ | $0.360\times10^{-2}$ |
| 4 | $0.463\times10^{-1}$ | $0.102\times10^{-1}$ | $0.432\times10^{-2}$ | $0.306\times10^{-3}$ | $0.428\times10^{-3}$ | $0.205\times10^{-4}$ |
| 5 | $0.442\times10^{-1}$ | $0.333\times10^{-2}$ | $0.424\times10^{-2}$ | $0.117\times10^{-4}$ | $0.423\times10^{-3}$ | $0.795\times10^{-7}$ |
| 6 | $0.435\times10^{-1}$ | $0.107\times10^{-2}$ | $0.424\times10^{-2}$ | $0.446\times10^{-6}$ | $0.423\times10^{-3}$ | $0.308\times10^{-9}$ |
| 7 | $0.433\times10^{-1}$ | $0.344\times10^{-3}$ | $0.424\times10^{-2}$ | $0.170\times10^{-7}$ | $0.423\times10^{-3}$ | $0.120\times10^{-11}$ |
| 8 | $0.432\times10^{-1}$ | $0.110\times10^{-3}$ | $0.424\times10^{-2}$ | $0.647\times10^{-9}$ | $0.423\times10^{-3}$ | $0.488\times10^{-14}$ |
| 9 | $0.432\times10^{-1}$ | $0.353\times10^{-4}$ | $0.424\times10^{-2}$ | $0.246\times10^{-10}$ | $0.423\times10^{-3}$ | $-0.167\times10^{-14}$ |
| 10 | $0.432\times10^{-1}$ | $0.113\times10^{-4}$ | $0.424\times10^{-2}$ | $0.939\times10^{-12}$ | $0.423\times10^{-3}$ | $0.888\times10^{-15}$ |
| 11 | $0.432\times10^{-1}$ | $0.362\times10^{-5}$ | $0.424\times10^{-2}$ | $0.344\times10^{-13}$ | $0.423\times10^{-3}$ | 0.00 |
| 12 | $0.432\times10^{-1}$ | $0.116\times10^{-5}$ | $0.424\times10^{-2}$ | 0.00 | | |
| $\vdots$ | | | | | | |
| 100 | $0.432\times10^{-1}$ | $0.222\times10^{-15}$ | | | | |

### 3.9.2 Discretization versus Differentiation

When solving a BVP it is necessary to address two distinct numerical tasks. Specifically a discretization and/or quadrature technique is needed to solve the differential equations (3.94). Furthermore to solve the BVP it is also necessary to compute derivatives of the boundary conditions. But which comes first? The previous examples suggest that it is important to

- first discretize the ODE and then
- compute derivatives of the discretization.

Conversely, reversing the order of these operations as illustrated by Example 3.10 is both contraindicated and counterproductive! It is worth noting that similar observations can be found in Gill, Murray, and Wright [99, pp. 266–267]. This illustrates a principle we refer to as *discretize then optimize* which will be revisited in Section 4.12.

What is less obvious is an explanation of this behavior. The key notion illustrated here is the concept of a *consistent function generator*. In all four examples the evaluation of the ODE solution is performed consistently—that is, with the same number of additions, subtractions, multiplications, and divisions—regardless of what value the iteration variable $x$ takes on. For Examples 3.7, 3.8, and 3.9 the Newton iteration used the derivative of the function generator. For Example 3.10—which behaves poorly—the Newton iteration did not use the derivative of the function generator. It is important that

- the function generator be consistent and
- the iterative process use derivatives of the function generator.

Does this preclude the use of adaptive quadrature procedures for numerical integration error control? Not necessarily, as we shall see in the next section.

### 3.9.3 External and Internal Differentiation

The examples in the preceding section illustrate different techniques for computing derivatives of the ODE solution $\mathbf{y}(\mathbf{x}, t_F)$ with respect to the variables $\mathbf{x}$. More precisely we are interested in constructing the matrix

$$\mathbf{G}(t_F) = \frac{\partial \mathbf{y}(t_F)}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1(t_F)}{\partial x_1} & \frac{\partial y_1(t_F)}{\partial x_2} & \cdots & \frac{\partial y_1(t_F)}{\partial x_{n+m}} \\ \frac{\partial y_2(t_F)}{\partial x_1} & \frac{\partial y_2(t_F)}{\partial x_2} & \cdots & \frac{\partial y_2(t_F)}{\partial x_{n+m}} \\ \vdots & & \ddots & \\ \frac{\partial y_n(t_F)}{\partial x_1} & \frac{\partial y_n(t_F)}{\partial x_2} & & \frac{\partial y_n(t_F)}{\partial x_{n+m}} \end{bmatrix}. \tag{3.107}$$

If we define perturbation vectors

$$\boldsymbol{\delta}_k^{\mathsf{T}} = [0, \ldots, 0, \delta_k, 0, \ldots, 0]^{\mathsf{T}} \tag{3.108}$$

for $k = 1, 2, \ldots, (n+m)$, then a forward difference approximation to column $k$ is of the form

$$\mathbf{G}_{*,k}(t_F) \approx \frac{1}{\delta_k}\left[\mathbf{y}(\mathbf{x}+\boldsymbol{\delta}_k, t_F) - \mathbf{y}(\mathbf{x}, t_F)\right]. \tag{3.109}$$

The computational implementation of (3.109) is deceptively simple and is referred to as *external differentiation*. The basic algorithm is as follows:

*External Differentiation*

**Nominal Point** Compute $\mathbf{y}(\mathbf{x}, t_F)$
Integrate the ODEs (3.94) from $t_I$ to $t_F$ with initial conditions $\mathbf{x}$

**Perturbations** For $k = 1, \ldots, (n+m)$
· Set $\overline{\mathbf{x}} = \mathbf{x} + \boldsymbol{\delta}_k$
· Compute $\mathbf{y}(\overline{\mathbf{x}}, t_F)$ ... Integrate the ODEs (3.94) from $t_I$ to $t_F$ with initial conditions $\overline{\mathbf{x}}$
· Construct $\mathbf{G}_{*,k}(t_F)$ from (3.109)

In effect this approach wraps a "loop" around the numerical integration of the ODE system. Since there are many effective software packages for solving the IVP this approach is very appealing because the dynamic simulation can be treated as a "black box." It is a serial process, in effect simulating each perturbation one at a time. Unfortunately the approach is not consistent. To control integration error sophisticated software may alter the number of integration steps and/or the order of the method. In particular, there is no reason to expect that the integration history will be consistent for all of the perturbations.

Instead of integrating the perturbed trajectories one at a time, let us consider doing all of them at the same time. To do so, define an augmented system of ODEs by making $(1+n+m)$ copies of (3.94)

$$\dot{\mathbf{z}}_k = \mathbf{f}[\mathbf{z}_k, \mathbf{q}_k, t] \qquad \text{for} \quad k = 0, 1, \ldots, (n+m). \tag{3.110}$$

By construction the augmented system (3.110) involves $n(1+n+m)$ differential variables; i.e., $\mathbf{z}$ is the $n(1+n+m)$-dimensional vector $\mathbf{z}^{\mathsf{T}} = (\mathbf{z}_0^{\mathsf{T}}, \ldots, \mathbf{z}_{n+m}^{\mathsf{T}}) = (\mathbf{y}^{\mathsf{T}}, \ldots, \mathbf{y}^{\mathsf{T}})$. However, now define the initial conditions for this augmented system as

$$\begin{bmatrix} \mathbf{z}_0(t_I) \\ \mathbf{z}_1(t_I) \\ \vdots \\ \mathbf{z}_n(t_I) \\ \mathbf{z}_{n+1}(t_I) \\ \vdots \\ \mathbf{z}_{n+m}(t_I) \end{bmatrix} = \begin{bmatrix} \mathbf{y}_I \\ \mathbf{y}_I + \delta_1 \widehat{\mathbf{e}}_1 \\ \vdots \\ \mathbf{y}_I + \delta_n \widehat{\mathbf{e}}_n \\ \mathbf{y}_I \\ \vdots \\ \mathbf{y}_I \end{bmatrix}, \tag{3.111}$$

where the $n$-vector $\widehat{\mathbf{e}}_k$ has a one in row $k$ and zero elsewhere. In a similar fashion we define the parameter vector

$$\begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_n \\ \mathbf{q}_{n+1} \\ \vdots \\ \mathbf{q}_{n+m} \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{p} \\ \vdots \\ \mathbf{p} \\ \mathbf{p} + \delta_{n+1}\widetilde{\mathbf{e}}_{n+1} \\ \vdots \\ \mathbf{p} + \delta_{n+m}\widetilde{\mathbf{e}}_{n+m} \end{bmatrix}, \tag{3.112}$$

where the $m$-vector $\widetilde{\mathbf{e}}_k$ has a one in row $k-n$ and zero elsewhere. In particular if the augmented system is propagated from $t_I$ to the final time $t_F$, we find that

$$\begin{bmatrix} \mathbf{z}_0(t_F) \\ \mathbf{z}_1(t_F) \\ \vdots \\ \mathbf{z}_{n+m}(t_F) \end{bmatrix} = \begin{bmatrix} \mathbf{y}(\mathbf{x}, t_F) \\ \mathbf{y}(\mathbf{x} + \boldsymbol{\delta}_1, t_F) \\ \vdots \\ \mathbf{y}(\mathbf{x} + \boldsymbol{\delta}_{n+m}, t_F) \end{bmatrix}. \tag{3.113}$$

Observe that all of the vectors needed to compute the matrix $\mathbf{G}(t_F)$ as given by (3.109) are available at the final point in (3.113). Hairer, Norsett, and Wanner [106] refer to this as a "stepsize freeze" technique for *internal differentiation*. The original idea for internal differentiation was proposed by Bock [43] using a slightly different implementation.

*Internal Differentiation*

**Propagation** Compute $\mathbf{y}(\mathbf{x}, t_F), \ldots, \mathbf{y}(\mathbf{x} + \boldsymbol{\delta}_{n+m}, t_F)$
Integrate the augmented ODE system (3.110) from $t_I$ to $t_F$ with
· initial conditions (3.111)
· and parameter values (3.112)

**Derivative Evaluation** For $k = 1, \ldots, (n+m)$
· Construct $\mathbf{G}_{*,k}(t_F)$ from (3.109) and (3.113)

There are three additional points that deserve comment. First, although internal differentiation was described using forward differences, clearly the approach is applicable to any of the higher-order methods outlined in Section 1.17. Second, the same techniques for choosing an optimal perturbation size, e.g., (1.166), can be applied in this setting. Finally, although the initial and final times $t_I$ and $t_F$ were assumed fixed in the discussion, variable time applications can also be treated. Specifically, a problem with variable initial and/or final time can be converted to one on the fixed domain $0 \le \tau \le 1$ by introducing the transformation

$$t = t_I + (t_F - t_I)\tau. \tag{3.114}$$

**Example 3.11** BRUSSELATOR PROBLEM. To demonstrate the difference between external and internal differentiation let us consider the *Brusselator* problem with dynamics given by the ODEs

$$\dot{y}_1 = 1 + y_1^2 y_2 - 4y_1, \qquad y_1(0) = 1.5, \tag{3.115}$$

$$\dot{y}_2 = 3y_1 - y_1^2 y_2, \qquad y_2(0) = y_{2I} \tag{3.116}$$

for $0 \le t \le 20$. For the sake of illustration let us compute forward difference approximations to

$$\left.\frac{\partial y_2(t)}{\partial y_{2I}}\right|_{t=20} = \frac{1}{\delta}\left\{ y_2(20)[y_{2I}+\delta] - y_2(20)[y_{2I}] \right\} \tag{3.117}$$

for different values of the initial condition $y_{2I} = 2.90, 2.91, \ldots, 3.09, 3.1$. The finite difference perturbation size is chosen to be $\delta = |y_{2I}| \times 10^{-3}$. The ODEs were solved using a variable-step, eighth order Dormand–Prince Runge–Kutta integrator DOP853 [107] for three different values of the absolute and relative error tolerances, $\delta_A = \delta_R = 1 \times 10^{-4}, 1 \times 10^{-6}, 1 \times 10^{-8}$. The results were computed using external and internal differentiation, and are illustrated in Figure 3.5. The left-hand column in the figure illustrates the results for external differentiation and the right-hand column presents the corresponding derivatives computed by internal differentiation. Each row in the figure corresponds to a different integration tolerance. As the nominal initial value for $y_{2I}$ changes smoothly from a small to large value, one intuitively expects the derivative to change accordingly. Indeed, this consistent behavior is observed for the internal derivatives. In contrast, the external derivatives shown in the left-hand column change erratically. Furthermore when using internal differentiation, the derivative values appear consistent regardless of whether a large or small numerical integration tolerance is used. In contrast, the integration tolerance must be very tight to compute a reasonable derivative estimate using external derivatives.

An explanation of this behavior is afforded by examining the details of one particular case at $y_{2I} = 2.91$. Specifically the details of external and internal differentiation are given in Tables 3.5 and 3.6, respectively. In Table 3.5 the nominal trajectory required 31 integration steps, with 484 evaluations of the right-hand-side functions $\mathbf{f}(\mathbf{y},t)$. However, the positive perturbation required 30 integration steps, with 461 right-hand-side evaluations. Even though both the nominal and perturbed trajectories satisfy the relative integration error tolerance, the trajectories are clearly inconsistent. The same sequence of arithmetic operations was not performed on the nominal and perturbed trajectories. When the finite difference derivative is calculated as shown in (3.119) this inconsistent behavior is accentuated leading to the value $-1.4344680$. In contrast for internal differentiation the nominal and perturbed trajectories have the same number of steps and right-hand-side evaluations as shown in Table 3.6. The finite difference derivative computations given in (3.120) lead to the value $-0.13319669$. A consistent function generator yields a consistent derivative!

## 3.9.4 Variational Derivatives

Generalizing the techniques in the previous section, let us consider computing the derivatives of the state at time $t$ with respect to the variables $\mathbf{x}$. Specifically define the $n \times (n+m)$ matrix of *variational derivatives*

$$\mathbf{G}(t) = \frac{\partial \mathbf{y}(t)}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1(t)}{\partial x_1} & \frac{\partial y_1(t)}{\partial x_2} & \cdots & \frac{\partial y_1(t)}{\partial x_{n+m}} \\ \frac{\partial y_2(t)}{\partial x_1} & \frac{\partial y_2(t)}{\partial x_2} & \cdots & \frac{\partial y_2(t)}{\partial x_{n+m}} \\ \vdots & & \ddots & \\ \frac{\partial y_n(t)}{\partial x_1} & \frac{\partial y_n(t)}{\partial x_2} & & \frac{\partial y_n(t)}{\partial x_{n+m}} \end{bmatrix}. \tag{3.118}$$

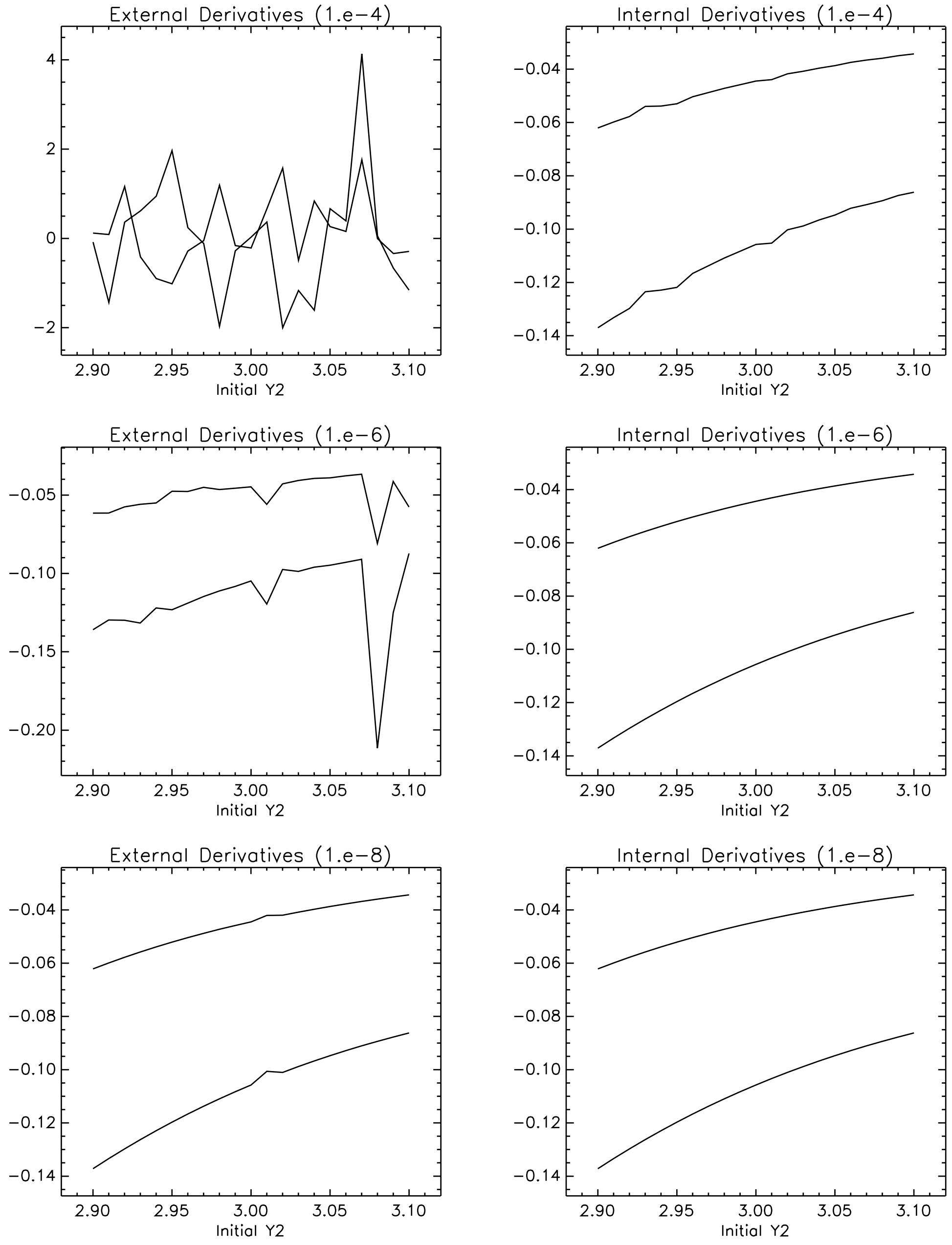


**Figure 3.5.** *Differentiation: external (left) versus internal (right).*

**Table 3.5.** *External differentiation behavior at $y_{2I} = 2.91$.*

| $y_2(0)$ | NSTEPS | NRHSFE | $y_2(20)$ |
|---|---|---|---|
| 2.91000000000000 | 31 | 484 | 4.605741850415169 |
| 2.91003162277660 | 30 | 461 | 4.605696488553900 |

$$\left.\frac{\partial y_2(t)}{\partial y_{2I}}\right|_{t=20} = \frac{4.605696488553900 - 4.605741850415169}{2.91003162277660 - 2.91000000000000} = -1.4344680 \tag{3.119}$$

**Table 3.6.** *Internal differentiation behavior at $y_{2I} = 2.91$.*

| $y_2(0)$ | NSTEPS | NRHSFE | $y_2(20)$ |
|---|---|---|---|
| 2.91000000000000 | 31 | $1485/3 = 495$ | 4.607480645589384 |
| 2.91003162277660 | * | * | 4.607476433540143 |

$$\left.\frac{\partial y_2(t)}{\partial y_{2I}}\right|_{t=20} = \frac{4.607476433540143 - 4.607480645589384}{2.91003162277660 - 2.91000000000000} = -0.13319669 \tag{3.120}$$

Differentiating (3.94) leads to the *variational differential equations*

$$\dot{\mathbf{G}} = \mathbf{f}_y[\mathbf{y}(t), \mathbf{p}, t]\mathbf{G} + \mathbf{f}_p[\mathbf{y}(t), \mathbf{p}, t], \tag{3.121}$$

where

$$\mathbf{f}_y[\mathbf{y}(t), \mathbf{p}, t] = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \cdots & \frac{\partial f_2}{\partial y_n} \\ \vdots & & \ddots & \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & & \frac{\partial f_n}{\partial y_n} \end{bmatrix} \tag{3.122}$$

and

$$\mathbf{f}_p[\mathbf{y}(t), \mathbf{p}, t] = \begin{bmatrix} \frac{\partial f_1}{\partial p_1} & \frac{\partial f_1}{\partial p_2} & \cdots & \frac{\partial f_1}{\partial p_m} \\ \frac{\partial f_2}{\partial p_1} & \frac{\partial f_2}{\partial p_2} & \cdots & \frac{\partial f_2}{\partial p_m} \\ \vdots & & \ddots & \\ \frac{\partial f_n}{\partial p_1} & \frac{\partial f_n}{\partial p_2} & & \frac{\partial f_n}{\partial p_m} \end{bmatrix}. \tag{3.123}$$

In general the initial conditions for (3.121) are

$$\mathbf{G}(t_I) = \frac{\partial \boldsymbol{\psi}(t_I)}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \psi_1(t_I)}{\partial x_1} & \frac{\partial \psi_1(t_I)}{\partial x_2} & \cdots & \frac{\partial \psi_1(t_I)}{\partial x_{n+m}} \\ \frac{\partial \psi_2(t_I)}{\partial x_1} & \frac{\partial \psi_2(t_I)}{\partial x_2} & \cdots & \frac{\partial \psi_2(t_I)}{\partial x_{n+m}} \\ \vdots & & \ddots & \\ \frac{\partial \psi_n(t_I)}{\partial x_1} & \frac{\partial \psi_n(t_I)}{\partial x_2} & & \frac{\partial \psi_n(t_I)}{\partial x_{n+m}} \end{bmatrix}. \tag{3.124}$$

However, when the initial conditions (3.95) take the simple form

$$\mathbf{y}(t_I) = \mathbf{y}_I, \tag{3.125}$$

then the initial conditions for the variational equations are just

$$\mathbf{G}(t_I) = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{3.126}$$

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

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

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

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

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

Unlike an equality constraint, which must be satisfied for all $t_I \le t \le t_F$, 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

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

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

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

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

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

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

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

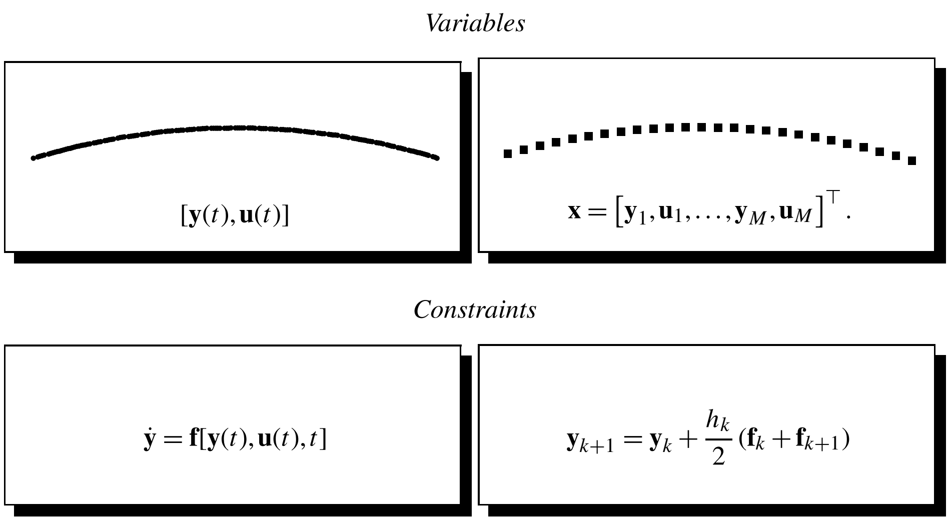

Variables
$[\mathbf{y}(t), \mathbf{u}(t)]$
$\mathbf{x} = [\mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_M, \mathbf{u}_M]^\top.$
Constraints
$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t]$
$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h_k}{2}(\mathbf{f}_k + \mathbf{f}_{k+1})$

**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

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

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

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

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 $\overline{\mathbf{u}}_k \equiv \mathbf{u}(\bar{t})$ with $\bar{t} = \frac{1}{2}(t_k + t_{k-1})$. 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:*

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

*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

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

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

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

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

**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:*

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

*Defects:*

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

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

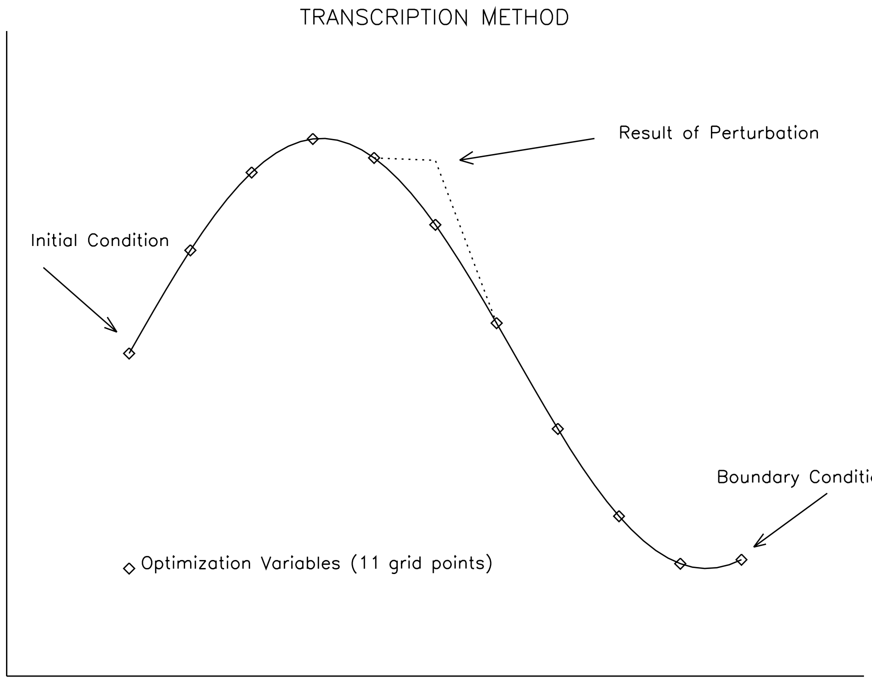


**Figure 4.2.** *Transcription sensitivity.*

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

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

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 $\lambda_k$ 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[\begin{array}{*{51}{c}}
\times&\times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times&\times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times&\times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times&\times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times \\
\times &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& & \times&\times&\times&\times&\times&\times&\times&\times&\times&\times
\end{array}\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

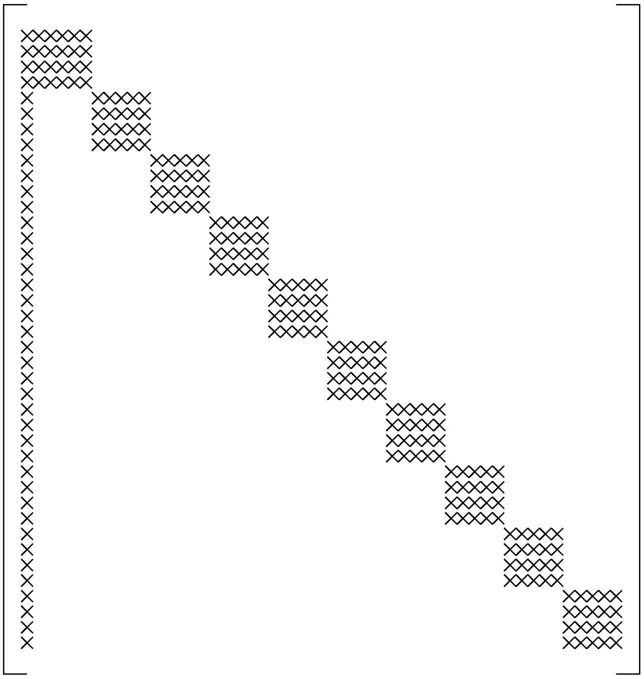
$$\mathbf{D} \equiv \frac{\partial \mathbf{q}}{\partial \mathbf{x}} \tag{4.79}$$

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 **D**:

struct(**D**) =



. (4.81)

Notice that the matrix **D** has approximately half as many nonzero elements per row as the matrix **G**. Consequently, the number of index sets needed to construct **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 **A** and **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

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

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 **D** in (4.87) involves partial derivatives of the right-hand-side functions **f** with respect to the state and control, all evaluated at the same grid point. In particular, let us define the nonzero pattern

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

as the *sparsity template*. The nonzero pattern defined by the sparsity template appears repeatedly in the matrix **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

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

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 **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 **D**. For our linear tangent steering example,

$$\mathrm{struct}(\mathbf{D}) = \left[\begin{array}{ccccccccccccccccccccccccccccccc}
\times&\times \\
\times&&\times \\
\times&&&\times \\
\times&&&\times \\
\times&&&&\times \\
\times&&&&&\times \\
\times&&&&&&\times \\
\times&&&&&&\times \\
\times&&&&&&&\times \\
\times&&&&&&&&\times \\
\times&&&&&&&&&\times \\
\times&&&&&&&&&\times \\
\times&&&&&&&&&&\times \\
\times&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times \\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times
\end{array}\right]. \tag{4.91}$$

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, $\gamma$ | 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

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

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

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

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

For the linear tangent steering example, we find that

$$\operatorname{struct}(\mathbf{D})=\left[\begin{array}{*{54}{c}}
\times&&&\times&&\times&\times&&&\times&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&\times&\times&\times&&&&\times&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&\times&&\times&\times&&&\times&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&\times&\times&\times&&&&\times&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&\times&&\times&\times&&&\times&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&\times&\times&\times&&&&\times&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&\times&&\times&\times&&&\times&&\times&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&\times&\times&\times&&&&\times&\times&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&&\times&\times&&&\times&&\times&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&\times&&&&\times&\times&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&&\times&\times&&&\times&&\times&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&\times&&&&\times&\times&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&&\times&\times&&&\times&&\times&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&\times&&&&\times&\times&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times&&&&&&\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&&\times&\times&&&\times&&\times\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&\times&&&&\times&\times\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times\\
\times&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\times&\times&&&&&\times
\end{array}\right]. \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

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

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

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

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

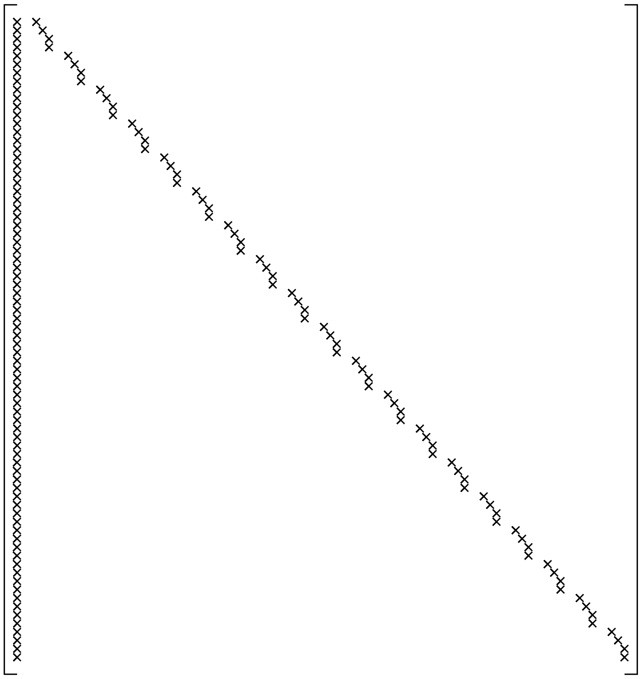
As before, the NLP constraints are

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

where **A** and **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

struct($\mathbf{D}$) =  . (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

$$\begin{aligned}
\mathbf{y}_{kj} &= \mathbf{y}_k + h_k \sum_{\ell=1}^{K} \alpha_{j\ell} \mathbf{f}_{k\ell}, && (4.110)\\
\mathbf{f}_{kj} &= \mathbf{f}\left[\mathbf{y}_{kj}, \mathbf{u}_{kj}, t_{kj}\right], && (4.111)\\
\mathbf{u}_{kj} &= \mathbf{u}\left(t_{kj}\right), && (4.112)\\
t_{kj} &= t_k + h_k \rho_j && (4.113)
\end{aligned}$$

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 $(\mathbf{BD})^{\mathsf{T}}(\mathbf{BD})$, 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 $\boldsymbol{\sigma}^{\mathsf{T}}\mathbf{x}$, it is then straightforward to compute the actual Hessian matrix using the sparse differencing formulas (2.6) and (2.7) with $\boldsymbol{\omega}^{\mathsf{T}} = [-\boldsymbol{\lambda}^{\mathsf{T}}, 1]\mathbf{B}$.

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., $\gamma_s \le \gamma_c$. 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*:

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

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 $\gamma$, 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.

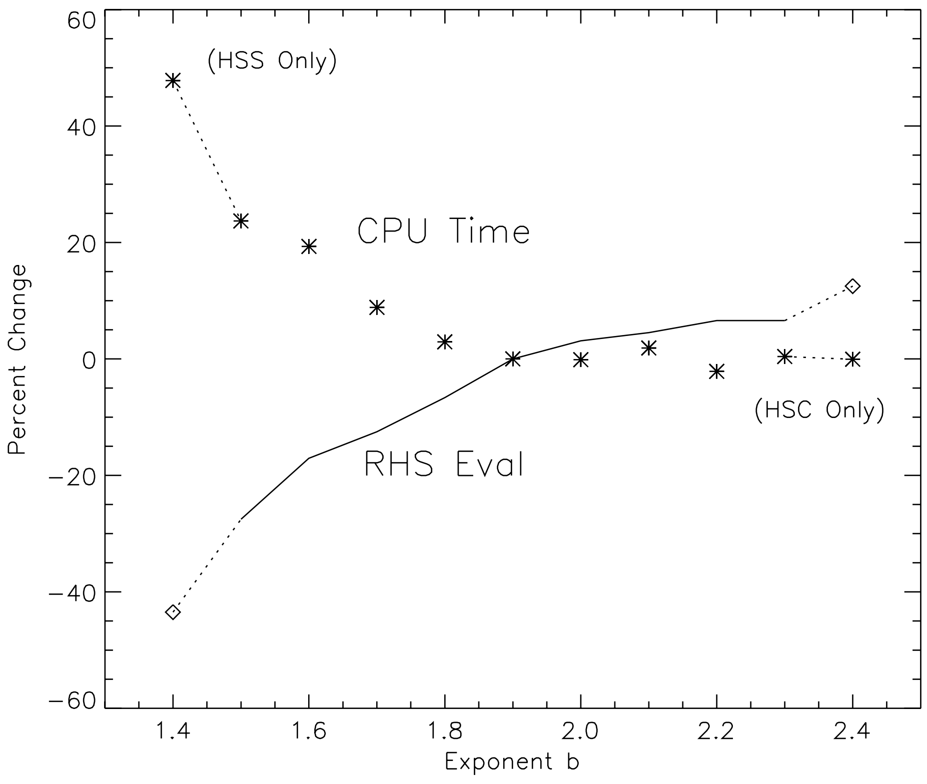


**Figure 4.3.** *Discretization tradeoffs.*

### 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

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

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

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

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

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

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

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

and parameter values

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

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 $i = 0, 1, \dots, N, N+1$, 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 $\mathbf{y}^{\mathsf{T}} = (y_1, \dots, y_N)$ 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 $i = 2, \dots, N-1$ 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

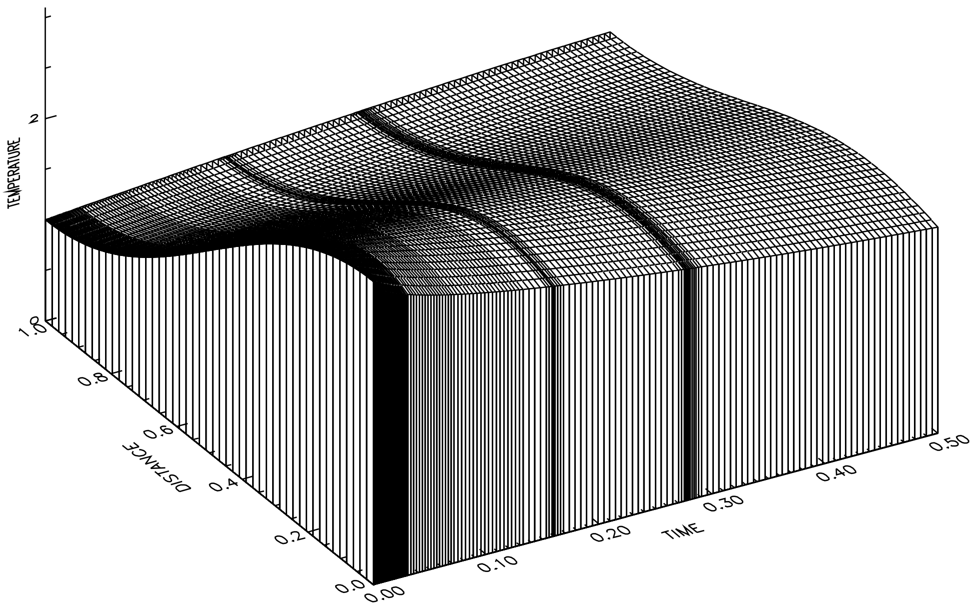


**Figure 4.4.** *Optimal temperature distribution.*

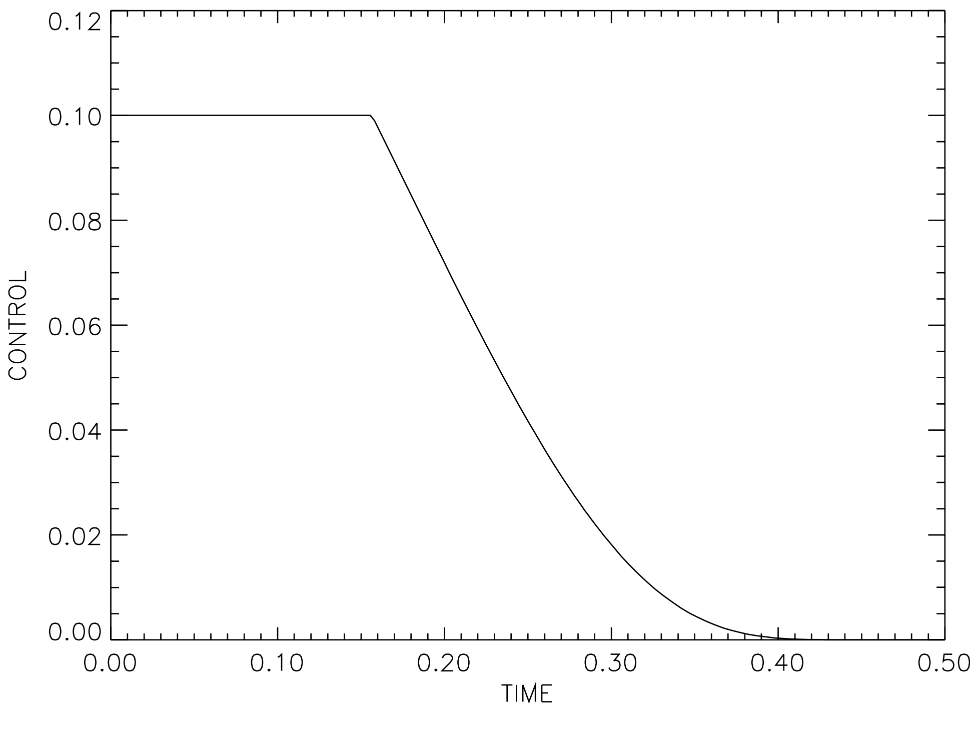


**Figure 4.5.** *Optimal control distribution.*

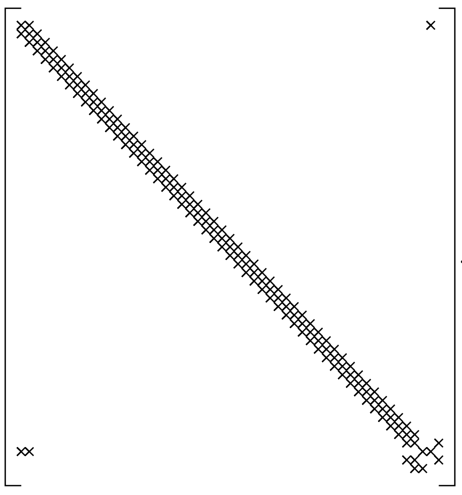
obtained after 5 mesh-refinement iterations (using the algorithm described in Section 4.7), with 164 points in the final time-dependent grid. The final NLP problem involved 9181 variables, 9025 active constraints, and 156 degrees of freedom.

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

| | Dense | Sparse | Reduction (%) |
|---|---|---|---|
| $\gamma_s$ | 53 | 4 | −92.5% |
| $\gamma_c$ | 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 $\gamma$ for both HSS and HSC discretization forms. This is due primarily to the right-hand-side sparsity template

$$\mathcal{T} = \text{struct}\left[\begin{array}{c|c} \frac{\partial \mathbf{f}}{\partial \mathbf{y}} & \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{g}}{\partial \mathbf{y}} & \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{w}}{\partial \mathbf{y}} & \frac{\partial \mathbf{w}}{\partial \mathbf{u}} \end{array}\right] = \qquad (4.137)$$



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 $k = 1,\dots,M$. 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 $k = 1,\dots,M$.

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 $k = 1,\dots,M$. 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 $k$th 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

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

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

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

where the scale weight

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

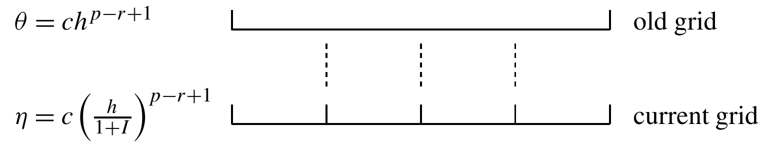
defines the maximum value for the $i$th state variable or its derivative over the $M$ grid points in the phase. Notice that $\epsilon_k$ is the maximum relative error over all components $i$ in the state equations $\dot{\mathbf{y}} - \mathbf{f}$ evaluated in the interval $k$.

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 $\eta_k$. 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 $\theta$. 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

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

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., $\theta_k$, 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 $\eta$ and $\theta$. 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 $\tau$. 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

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

so that

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

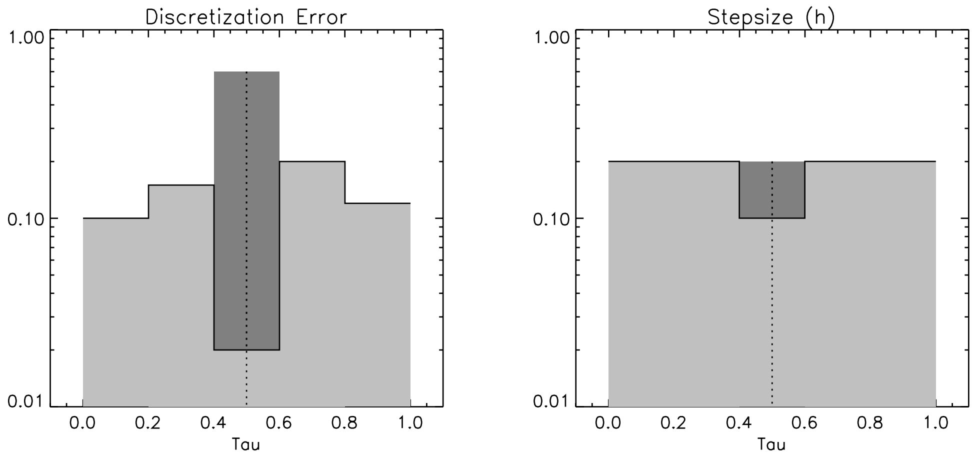


**Figure 4.6.** *Subdividing the grid.*

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 $k = 1,\dots,n_s$. 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 $\alpha$, i.e.,

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

for $k = 1,\dots,n_s$ 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 $\alpha$. 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 $\delta$, 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., $\epsilon_k$ for $k = 1,\dots,n_s$, 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 $\epsilon_k$ 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 $\epsilon_\alpha \le 2\overline{\epsilon}$, 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 $\alpha$ 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 $\alpha$, i.e., $I_\alpha \leftarrow I_\alpha + 1$.
   (d) Update the predicted error for interval $\alpha$ 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 $\epsilon_\alpha$ 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 ($\mathbb{SOCS}$) [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:

$$\begin{aligned}
\dot{y}_1 &= -10y_1 + u_1 + u_2,\\
\dot{y}_2 &= -2y_2 + u_1 + 2u_2,\\
\dot{y}_3 &= -3y_3 + 5y_4 + u_1 - u_2,\\
\dot{y}_4 &= 5y_3 - 3y_4 + u_1 + 3u_2,
\end{aligned}$$

$$y_1^2 + y_2^2 + y_3^2 + y_4^2 \geq 3p(t,3,12) + 3p(t,6,10) + 3p(t,10,6) + 8p(t,15,4) + 0.01,$$

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).

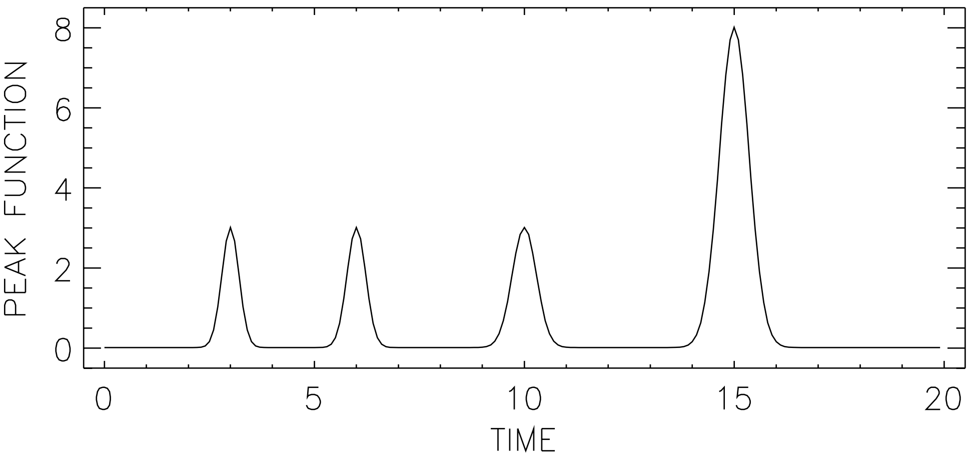


**Figure 4.7.** *Alp rider peaks.*

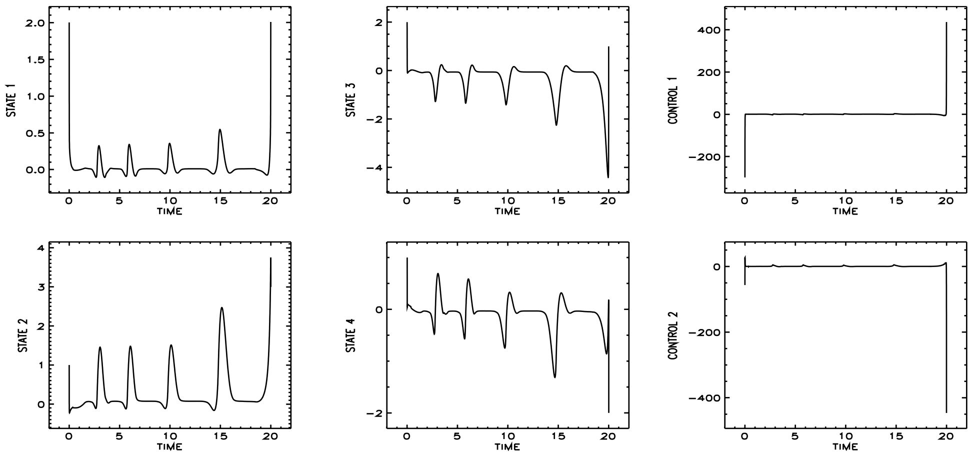


**Figure 4.8.** *Alp rider solution.*

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 $0.32 \times 10^0$, which was obtained in $0.22 \times 10^2$ 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 | $0.32 \times 10^{0}$ | $0.22 \times 10^{2}$ |
| 2 | 41 | 93 | 91 | 10969 | 888489 | $0.29 \times 10^{-1}$ | $0.66 \times 10^{2}$ |
| 3 | 76 | 37 | 35 | 4244 | 640844 | $0.74 \times 10^{-2}$ | $0.63 \times 10^{2}$ |
| 4 | 84 | 19 | 16 | 1993 | 332831 | $0.89 \times 10^{-3}$ | $0.35 \times 10^{2}$ |
| 5 | 119 | 26 | 23 | 2833 | 671421 | $0.18 \times 10^{-3}$ | $0.65 \times 10^{2}$ |
| 6 | 194 | 18 | 16 | 1964 | 760068 | $0.31 \times 10^{-4}$ | $0.95 \times 10^{2}$ |
| 7 | 253 | 17 | 15 | 1844 | 931220 | $0.11 \times 10^{-4}$ | $0.14 \times 10^{3}$ |
| 8 | 304 | 10 | 8 | 1004 | 609428 | $0.37 \times 10^{-5}$ | $0.16 \times 10^{3}$ |
| 9 | 607 | 6 | 4 | 524 | 635612 | $0.35 \times 10^{-6}$ | $0.54 \times 10^{3}$ |
| 10 | 785 | 5 | 3 | 404 | 633876 | $0.16 \times 10^{-6}$ | $0.92 \times 10^{3}$ |
| 11 | 992 | 5 | 3 | 404 | 801132 | $0.29 \times 10^{-7}$ | $0.14 \times 10^{4}$ |
|  | 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.

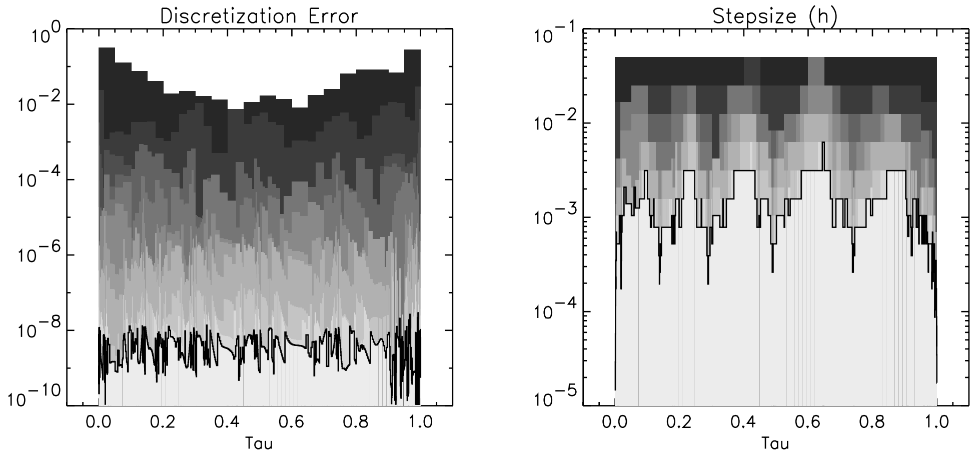


**Figure 4.9.** *Alp rider mesh-refinement history.*

## 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 $\pi$, 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

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

which relates the scaled state $\tilde{\mathbf{y}}$ and control $\tilde{\mathbf{u}}$ 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

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

will relate the scaled constraints $\tilde{\mathbf{c}}$ 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

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

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 $\zeta_k$ 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 $\sigma_L$ and $\sigma_U$ 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 $\mathbb{SOCS}$ 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

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

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

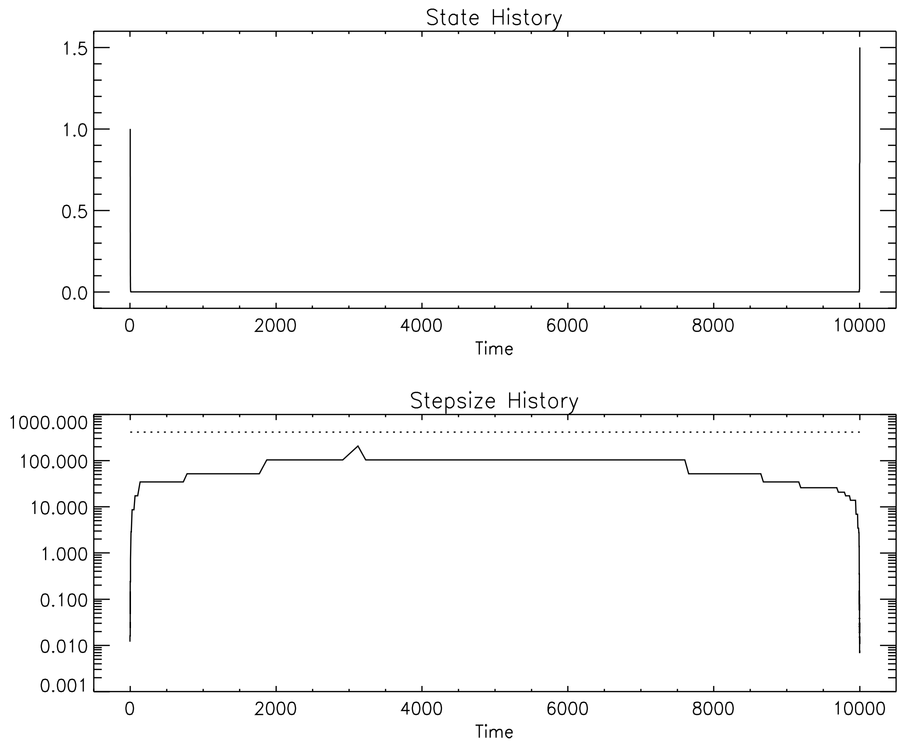


**Figure 4.10.** *Hypersensitive problem.*

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 | $0.15 \times 10^5$ | $0.62 \times 10^0$ |
| 2 | 25 | 21 | 19 | 124 | 6076 | $0.37 \times 10^3$ | $0.15 \times 10^1$ |
| 3 | 49 | 11 | 9 | 63 | 6111 | $0.28 \times 10^2$ | $0.18 \times 10^1$ |
| 4 | 97 | 11 | 8 | 61 | 11773 | $0.49 \times 10^1$ | $0.29 \times 10^1$ |
| 5 | 109 | 29 | 27 | 585 | 126945 | $0.11 \times 10^1$ | $0.58 \times 10^1$ |
| 6 | 115 | 10 | 8 | 185 | 42365 | $0.13 \times 10^0$ | $0.25 \times 10^1$ |
| 7 | 121 | 8 | 6 | 144 | 34704 | $0.12 \times 10^{-1}$ | $0.22 \times 10^1$ |
| 8 | 128 | 8 | 1 | 93 | 23715 | $0.96 \times 10^{-3}$ | $0.18 \times 10^1$ |
| 9 | 137 | 5 | 1 | 60 | 16380 | $0.19 \times 10^{-4}$ | $0.15 \times 10^1$ |
| 10 | 181 | 4 | 1 | 49 | 17689 | $0.55 \times 10^{-6}$ | $0.16 \times 10^1$ |
| 11 | 229 | 4 | 1 | 49 | 22393 | $0.31 \times 10^{-7}$ | $0.19 \times 10^1$ |
|  | 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 $k = 1,\dots,(M-1)$, 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 $m \le n$ 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 $k = 1, \dots, M$. 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

$$\begin{aligned}
\mu_1 &= -\left(\frac{2}{h_1}\right)\alpha_1, && (4.235)\\
\mu_k &= -\left(\frac{2}{h_{k-1}+h_k}\right)\alpha_k, \qquad k = 2, \dots, M-1, && (4.236)\\
\mu_M &= -\left(\frac{2}{h_{M-1}}\right)\alpha_M. && (4.237)
\end{aligned}$$

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 $k = 1, \dots, M-1$. 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

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

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

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

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

### 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 $\lambda$. 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

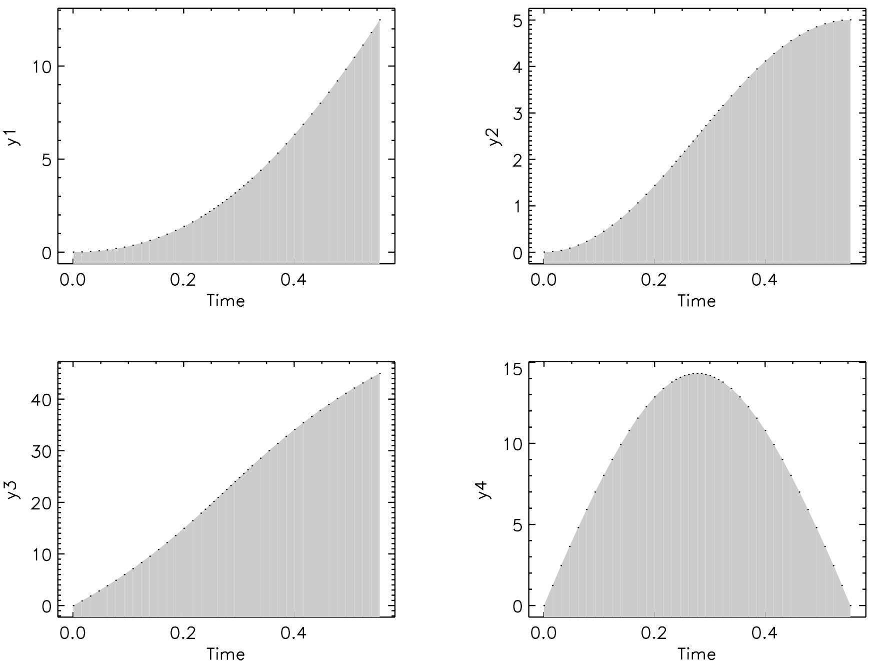


**Figure 4.11.** *Linear tangent steering states* $(y_1, y_2, y_3, y_4)$.

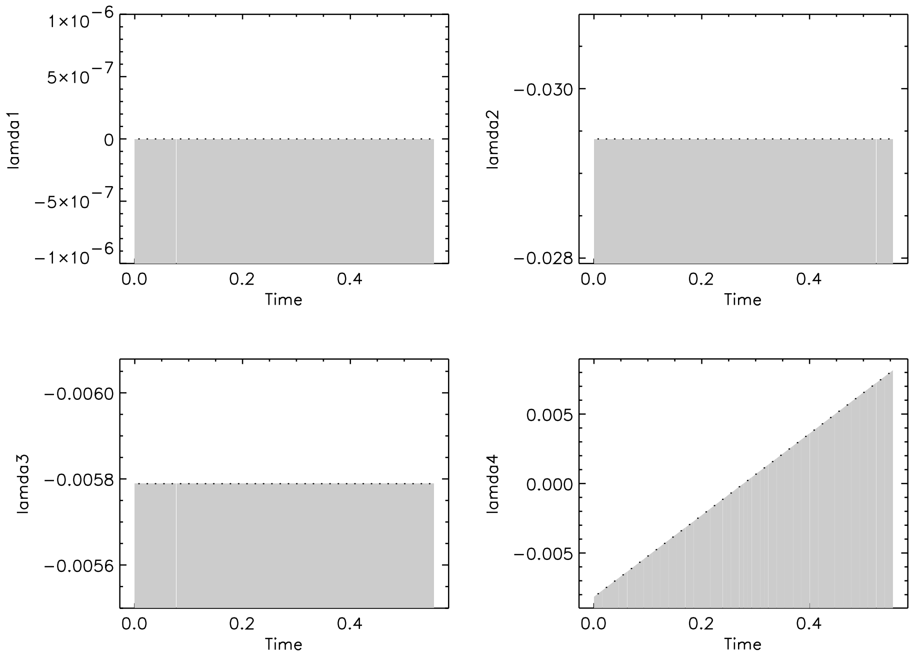


**Figure 4.12.** *Linear tangent steering adjoints* $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$.

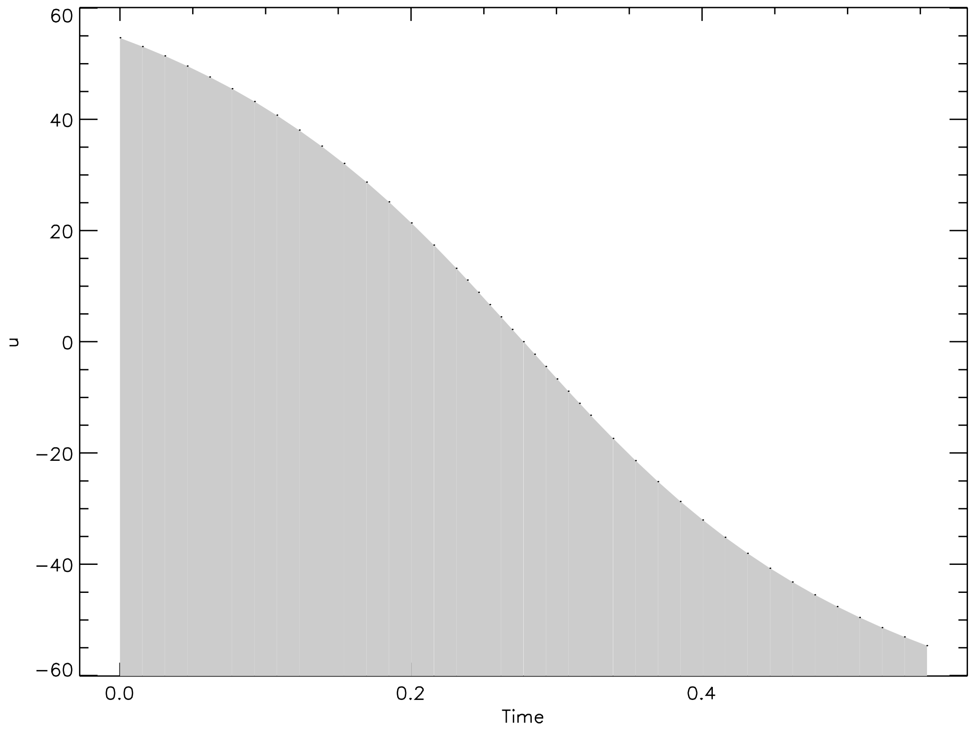


**Figure 4.13.** *Linear tangent steering control.*

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 $\varepsilon$ is quite small for all adjoint estimates.

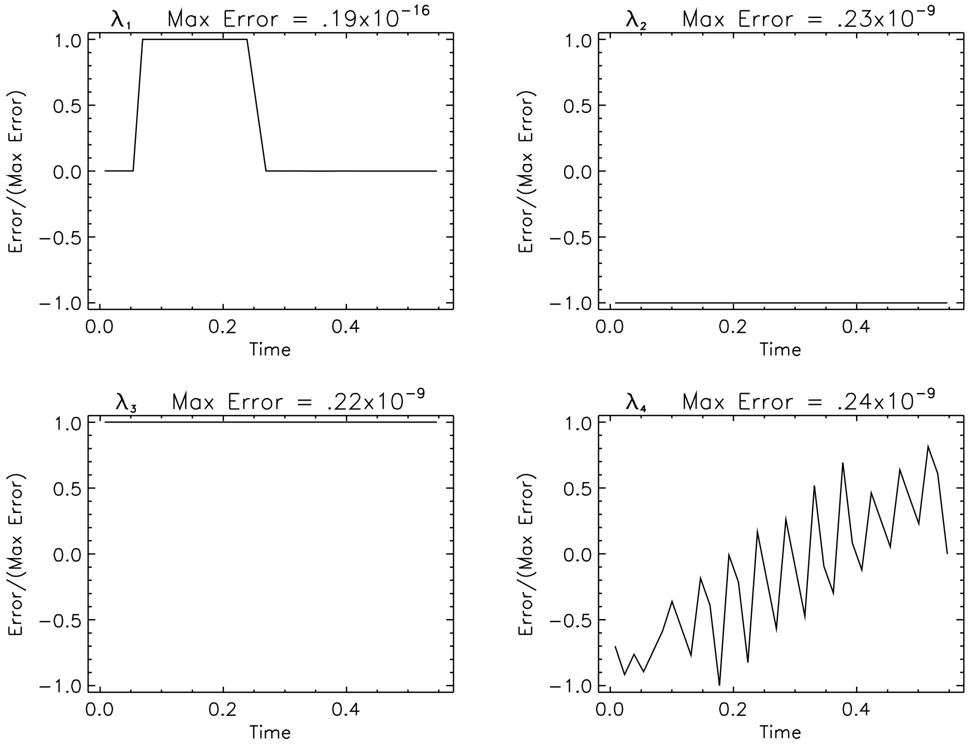


**Figure 4.14.** *Linear tangent steering, error in discrete adjoint estimates.*

**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

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

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 $\lambda$ 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 $\varepsilon$.

**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

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

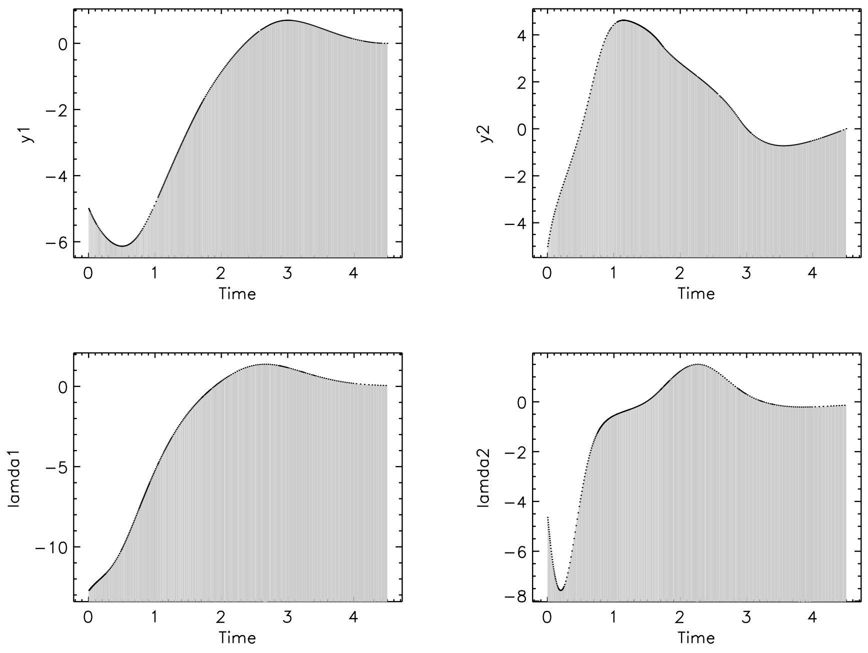


**Figure 4.15.** *Rayleigh problem, states* $(y_1, y_2)$ *and adjoints* $(\lambda_1, \lambda_2)$.

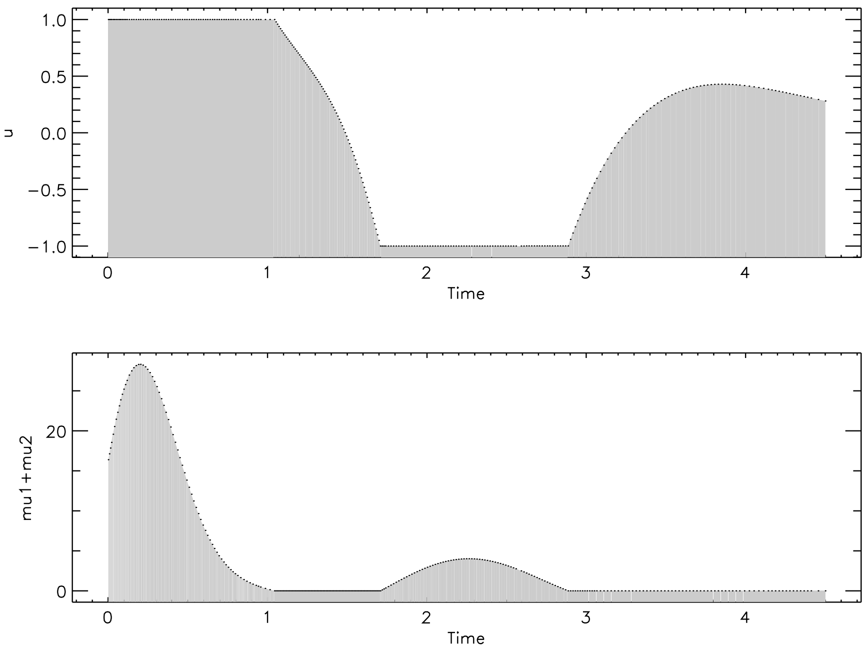


**Figure 4.16.** *Rayleigh problem, control and* $\mu_1 + \mu_2$.

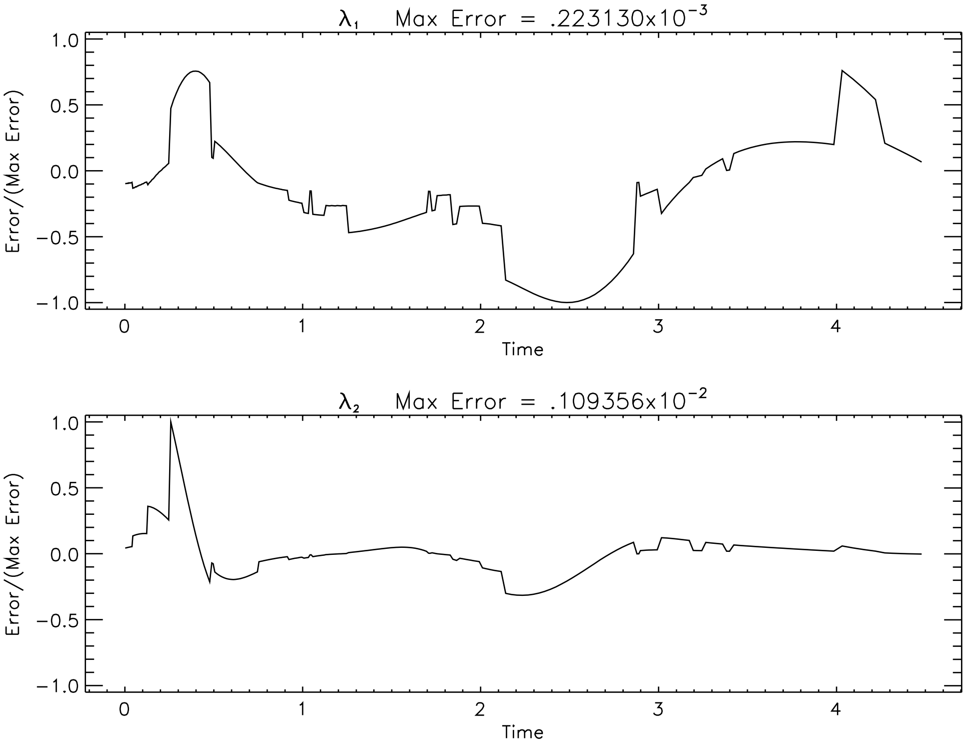


**Figure 4.17.** *Rayleigh problem, error in discrete adjoint estimates.*

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.

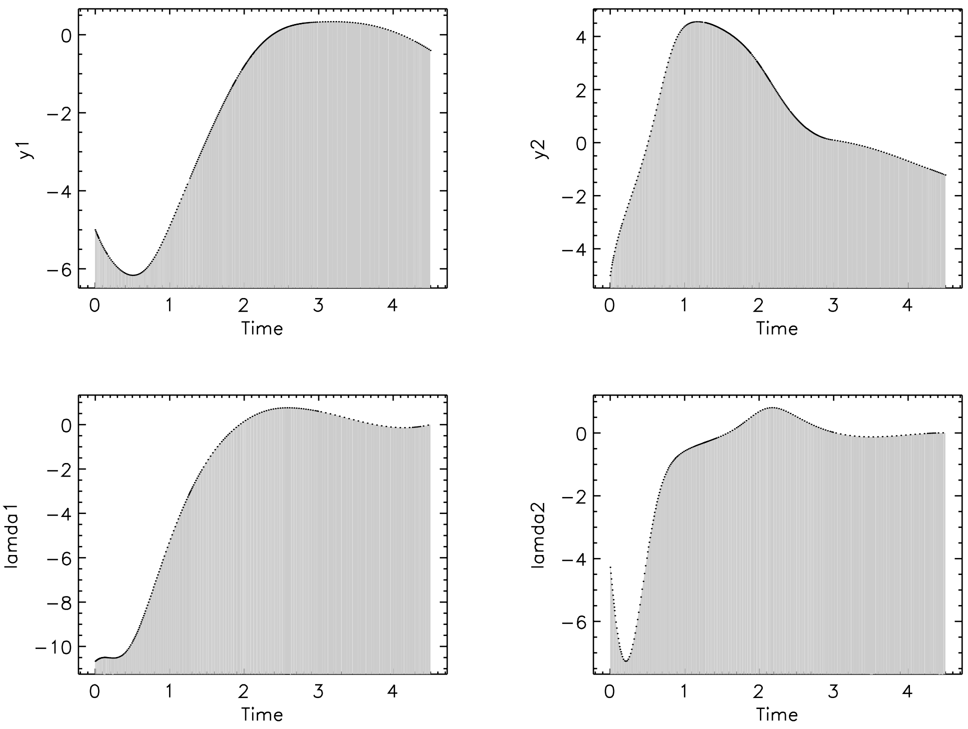


**Figure 4.18.** *Rayleigh mixed constraint problem, states $(y_1, y_2)$ and adjoints $(\lambda_1, \lambda_2)$.*

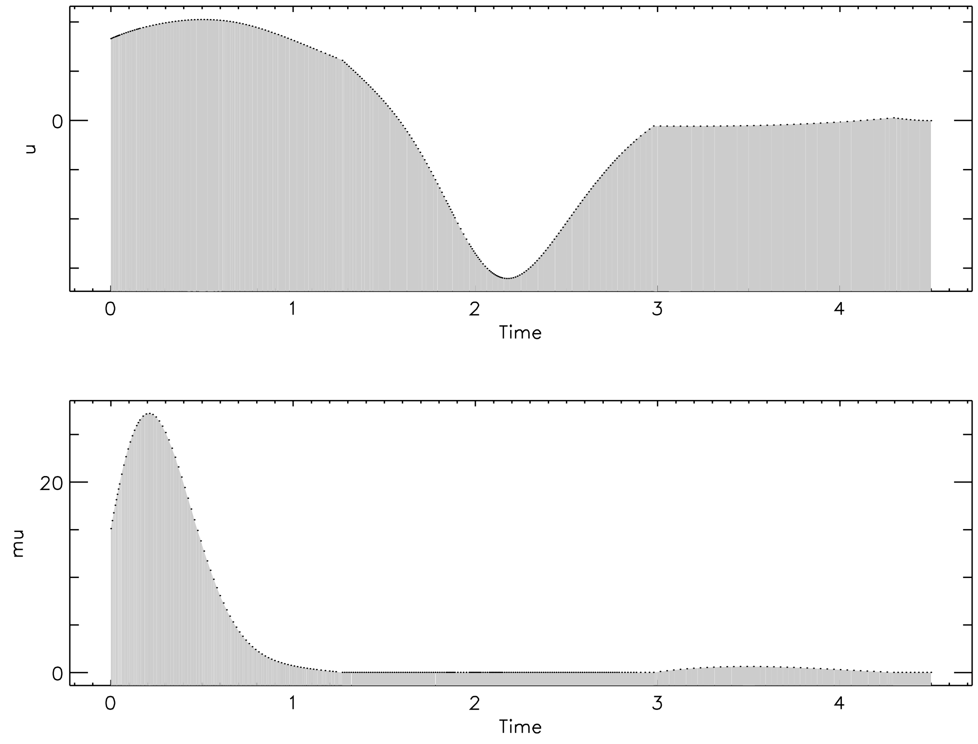


**Figure 4.19.** *Rayleigh mixed constraint problem, control and $\mu$.*

**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

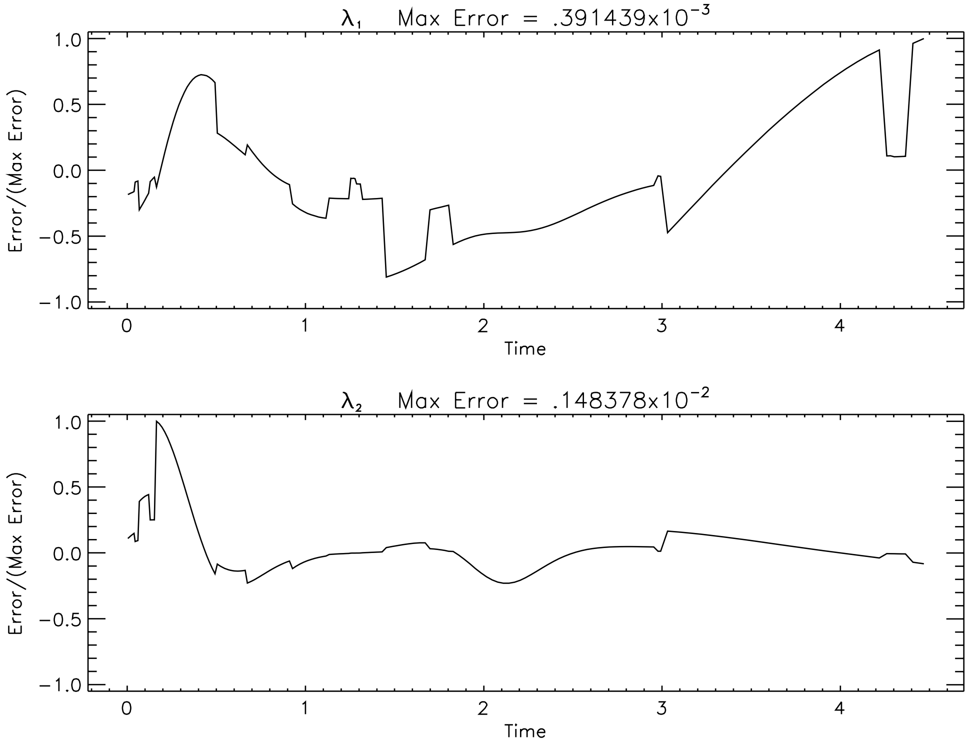


**Figure 4.20.** *Rayleigh mixed constraint problem, error in discrete adjoint estimates.*

$H$. 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 $\lambda_2$ 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

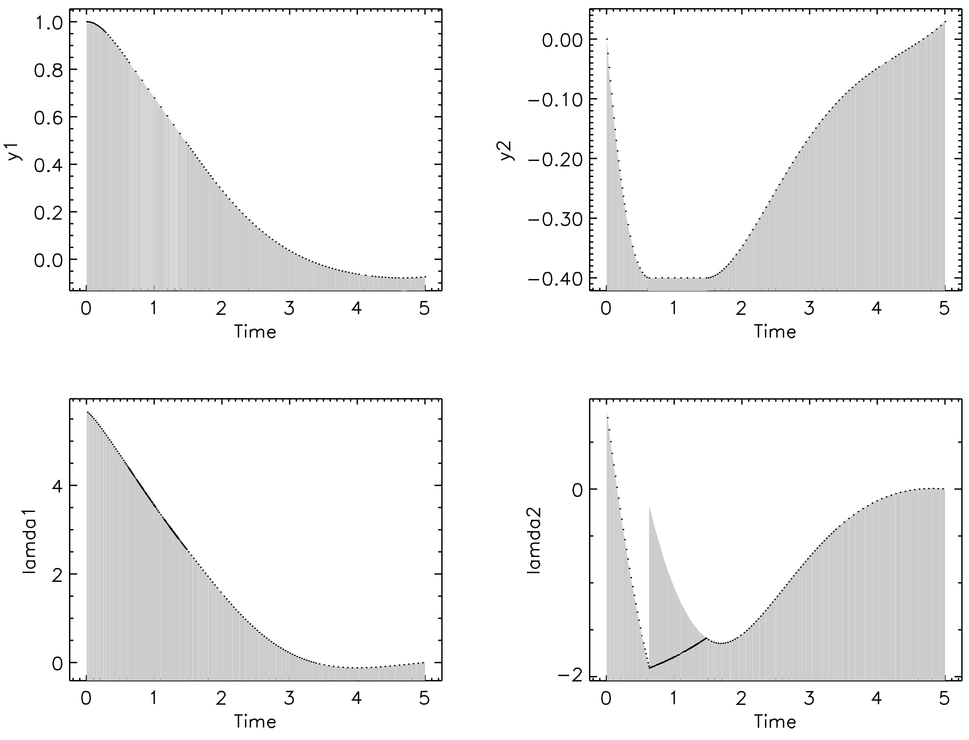


**Figure 4.21.** *Van der Pol problem, states* $(y_1, y_2)$ *and adjoints* $(\lambda_1, \lambda_2)$.

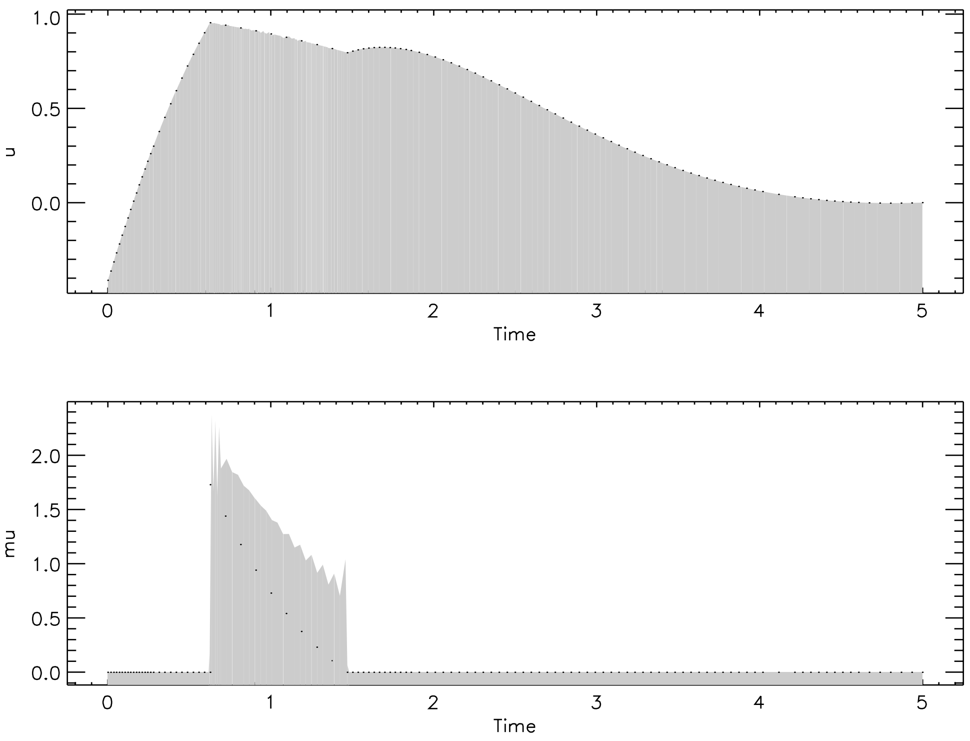


**Figure 4.22.** *Van der Pol problem, control and $\mu$.*

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 $\lambda$ agree when the state constraint is not active. However, along the boundary arc it is clear that the true value of $\lambda_2$ differs significantly from the discrete adjoint estimate. Furthermore, the true value of $\mu$ 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 $\lambda_2$ has no influence on either the optimal control $u$ or the states. In this region $\lambda_2$ 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 $\mu$ 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

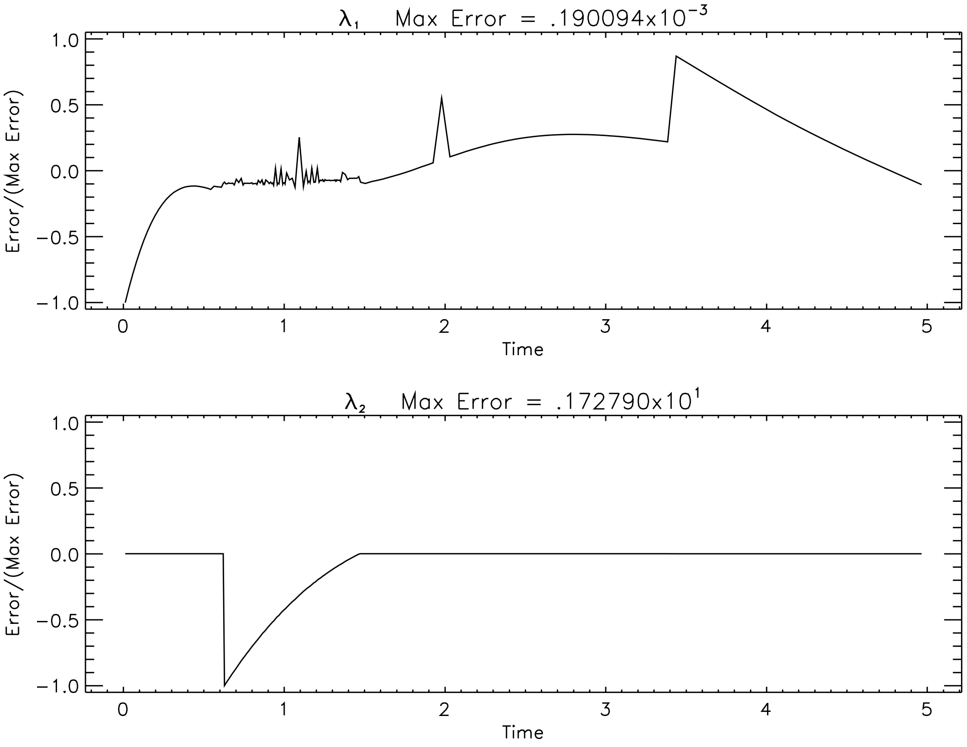


**Figure 4.23.** *Van der Pol problem, error in discrete adjoint estimates.*

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 $q$th 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 $\lambda_2$ 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

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

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

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

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 $\tau$, 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 $2\times 2$ 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 $k$th derivative of $s(\mathbf{y},\tau)$ by $d^k s/d\tau^k = s^{(k)}(\tau)$. Then

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

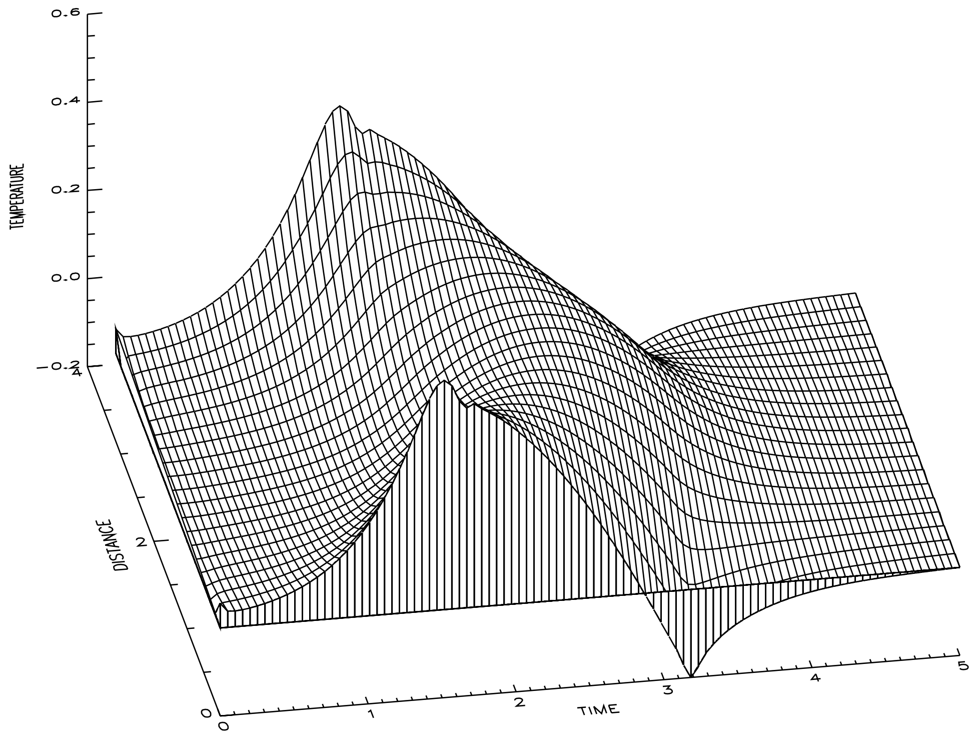


Figure 4.24. $y(x,t)$.

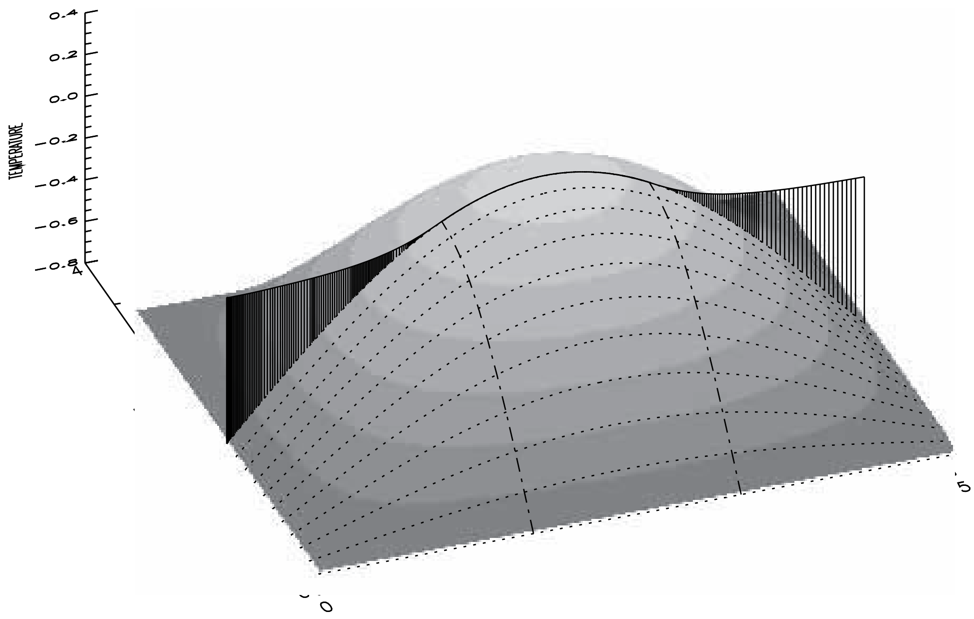


Figure 4.25. *Inequality* $y(x_{10},t) \geq g(x_{10},t)$.

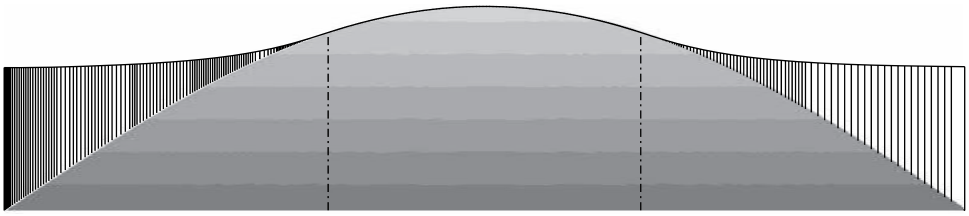


**Figure 4.26.** *Solution regions.*

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 $k$th derivative of the constraint function is given by

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

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 $\varepsilon_m$ 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

$\mathbb{SOCS}$, 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 $m$th 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.

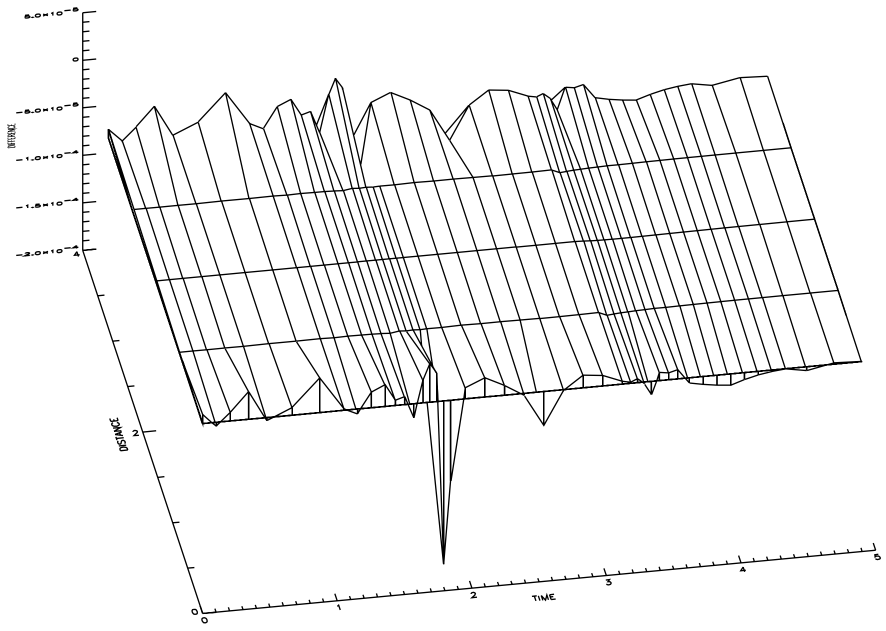


**Figure 4.27.** *Difference between direct and indirect solutions $n = 4$.*

## 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 $\varepsilon_m$ is the machine precision. Since $\epsilon$ is the NLP constraint tolerance, we

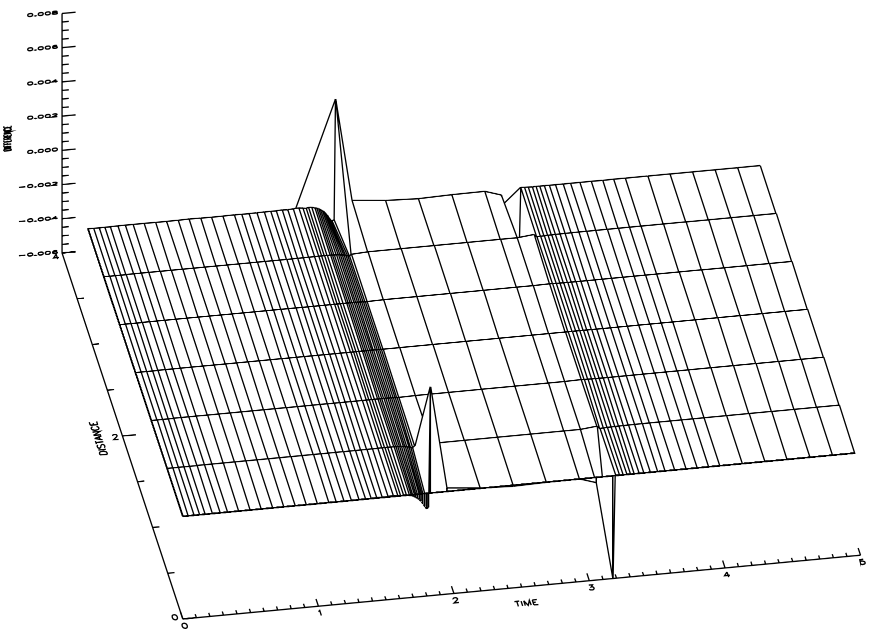


**Figure 4.28.** *Difference between direct and indirect solutions n* = 6.

can view $s(\mathbf{y},\tau)/\epsilon$ as a *normalized path constraint error*. More precisely, the constraint $|s(\mathbf{y},\tau)| \le \epsilon$ when $-1 \le s(\mathbf{y},\tau)/\epsilon \le 1$. 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

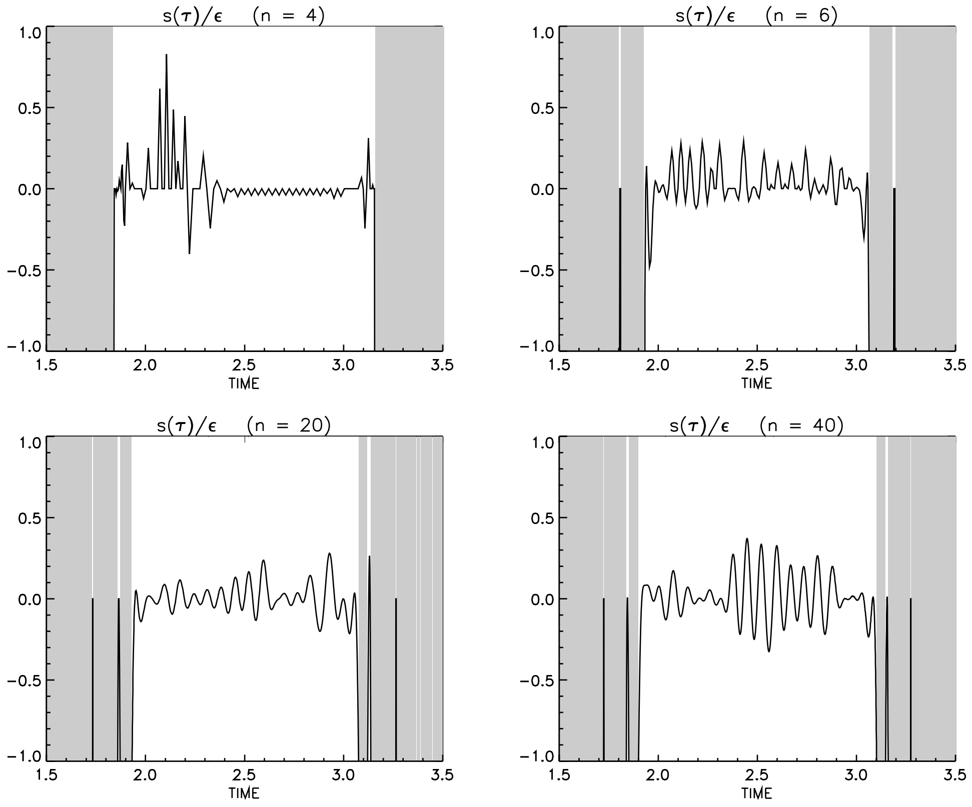


**Figure 4.29.** *Inequality constraint error.*

[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

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

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 $\bar{\mathbf{u}}_j \equiv \mathbf{u}[\bar{\tau}_j]$. 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

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

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 $\epsilon_\alpha$ 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) | $\epsilon_\alpha$ |
|---|---|---|---|---|---|---|---|---|
| 50 | 351 | 253 | 98 | 405 | 16 | 7 | 0.120000 | $1.19\times10^{-2}$ |
| 100 | 701 | 503 | 198 | 494 | 18 | 9 | 0.260000 | $2.25\times10^{-3}$ |
| 200 | 1401 | 1003 | 398 | 494 | 18 | 9 | 0.570000 | $3.64\times10^{-4}$ |
| 400 | 2801 | 2003 | 798 | 405 | 16 | 7 | 1.06000 | $5.26\times10^{-5}$ |
| 800 | 5601 | 4003 | 1598 | 440 | 16 | 8 | 2.75000 | $7.08\times10^{-6}$ |
| 1600 | 11201 | 8003 | 3198 | 367 | 15 | 6 | 6.83000 | $9.18\times10^{-7}$ |
| 3200 | 22401 | 16003 | 6398 | 413 | 16 | 7 | 21.7700 | $1.17\times10^{-7}$ |
| 6400 | 44801 | 32003 | 12798 | 502 | 18 | 9 | 92.7300 | $1.47\times10^{-8}$ |
| 12800 | 89601 | 64003 | 25598 | 457 | 17 | 8 | 356.660 | $1.84\times10^{-9}$ |
| 25600 | 179201 | 128003 | 51198 | 503 | 18 | 9 | 4588.33 | $6.64\times10^{-10}$ |

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 $M$.

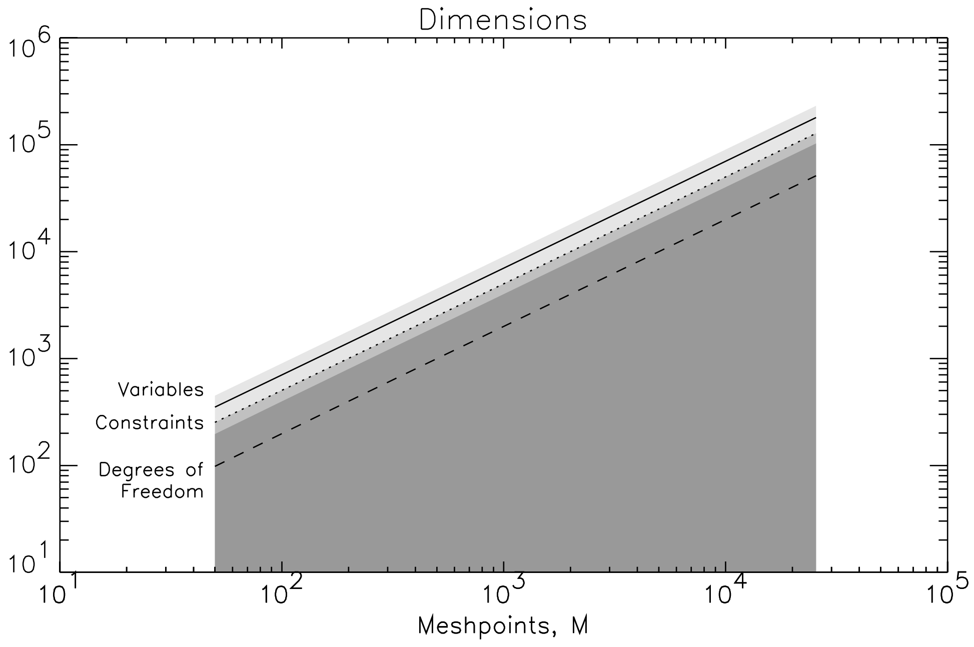


**Figure 4.30.** *NLP dimensions increase with mesh.*

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.

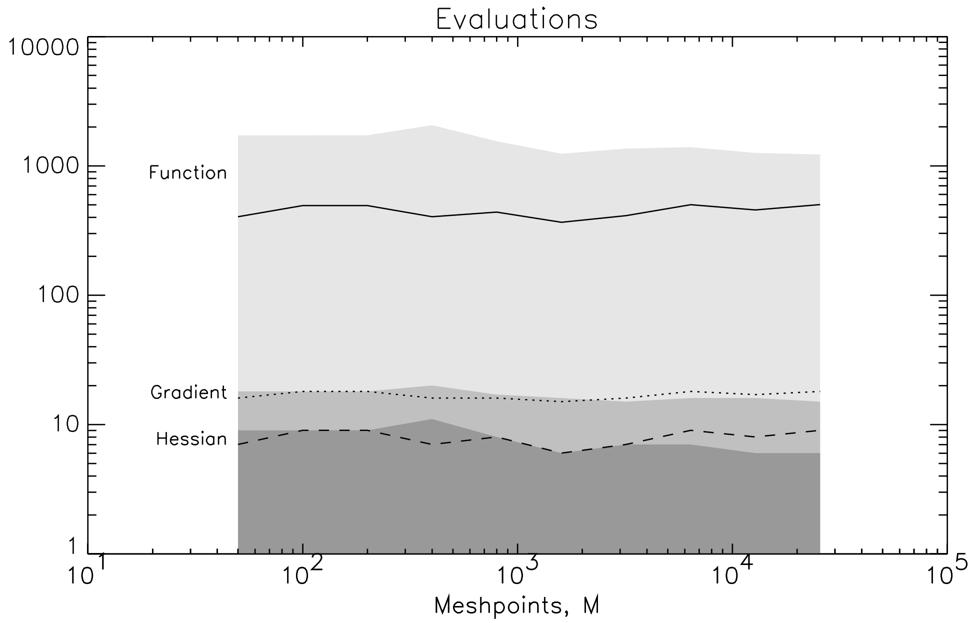


**Figure 4.31.** *NLP evaluations do not increase with mesh.*

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

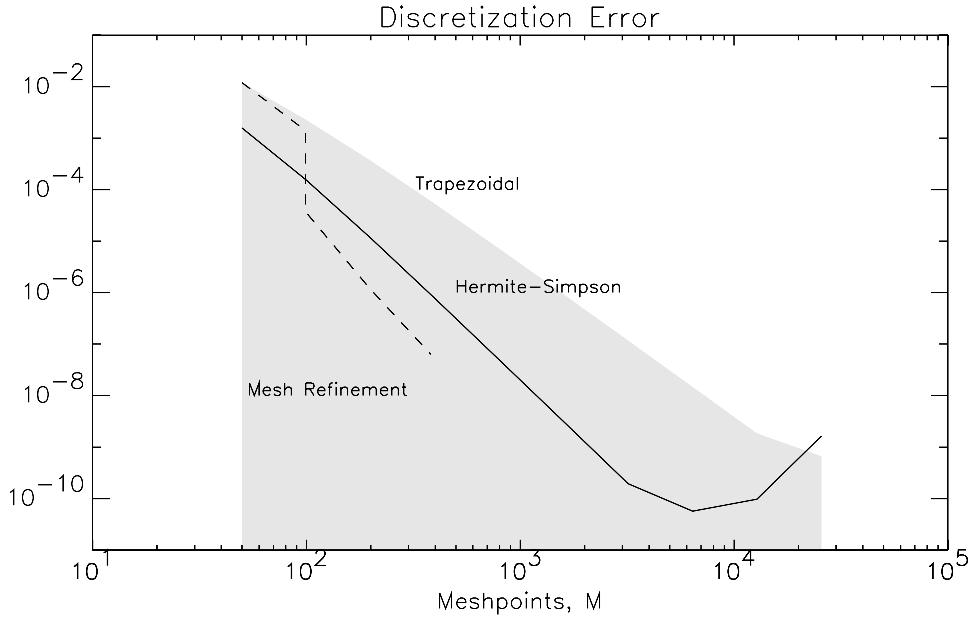


**Figure 4.32.** *Discretization error decreases with mesh.*

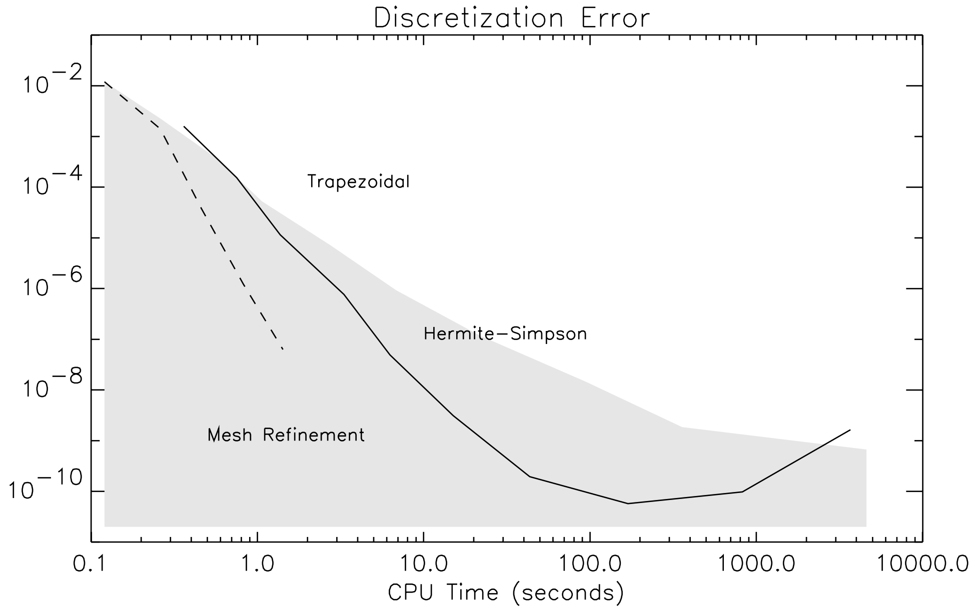


**Figure 4.33.** *Discretization error versus CPU time.*

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 $\kappa$ 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 $\mathbb{SOCS}$ as well as the LOQO algorithm of Vanderbei and Shanno [166].

$\boxed{\text{Answer: Newton.}}$

### 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 $\epsilon_\alpha \le \delta = 10^{-7}$ 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 $\epsilon_m$ 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 $\mu$ 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.

$\boxed{\text{Answer: SQP.}}$

## 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]:

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

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

$$\begin{aligned}
0 &= T(t) - \sigma v^2 \exp[-h/h_0] - mg \\
&\quad - \frac{mg}{1 + 4(c/v) + 2(c^2/v^2)}\left[\frac{c^2}{h_0 g}\left(1 + \frac{v}{c}\right) - 1 - 2\frac{c}{v}\right],
\end{aligned} \tag{4.367}$$

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 | $0.67 \times 10^{-4}$ | $0.15 \times 10^{1}$ |
| 2 | 79 | 26 | $0.55 \times 10^{-5}$ | $0.12 \times 10^{1}$ |
| 3 | 91 | 14 | $0.70 \times 10^{-7}$ | $0.96 \times 10^{0}$ |
|  |  | 70 |  | 3.64 |

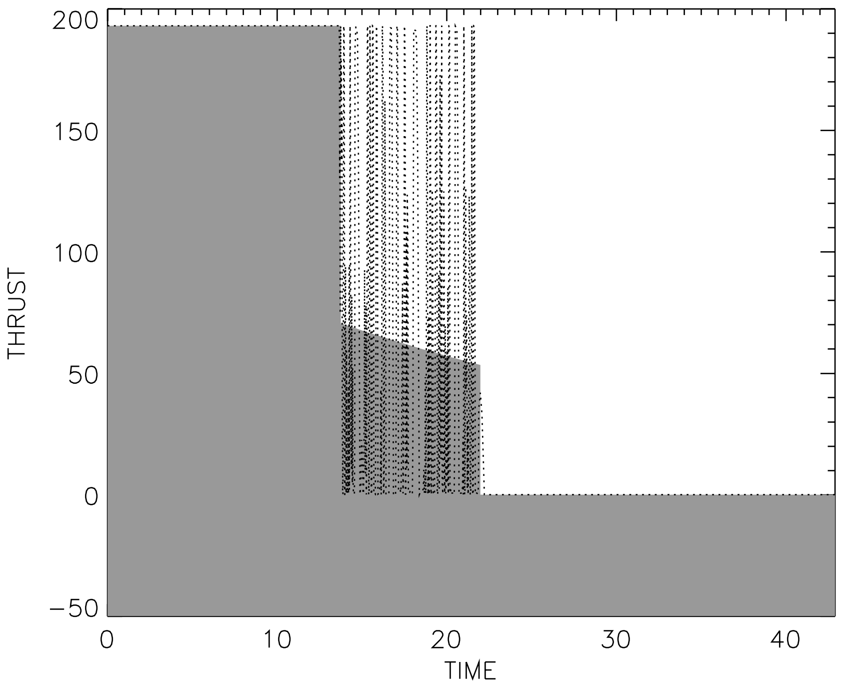


**Figure 4.34.** *Singular arc solution.*

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 | $0.52\times10^{-3}$ | $0.46\times10^{1}$ |
| 2 | 96 | 416 | $0.72\times10^{-3}$ | $0.17\times10^{2}$ |
| 3 | 103 | 2078 | $0.13\times10^{-3}$ | $0.58\times10^{2}$ |
| 4 | 188 | 3398 | $0.64\times10^{-4}$ | $0.34\times10^{3}$ |
| 5 | 375 | 230 | $0.26\times10^{-6}$ | $0.19\times10^{3}$ |
| 6 | 450 | 230 | $0.84\times10^{-7}$ | $0.17\times10^{3}$ |
|  |  | 6541 |  | 775.30 |

### 4.14.2 State Constraints

A standard approach to a problem with path constraint(s)

$$\mathbf{0}\le\mathbf{g}[\mathbf{y}(t),\mathbf{u}(t),t]$$

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

$$\frac{d^{q-k}}{dt^{q-k}}(\mathbf{g})\mid_{t=t_a}=\mathbf{0}$$

for $k=1,\dots,q$ 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

$$\begin{aligned}\dot{x}&=v\cos u,\\ \dot{y}&=v\sin u,\\ \dot{v}&=g\sin u,\\ 0&\ge y-x/2-h\end{aligned}$$

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 | $0.16 \times 10^{-2}$ | $0.13 \times 10^{1}$ |
| 2 | 19 | 35 | $0.19 \times 10^{-3}$ | $0.54 \times 10^{0}$ |
| 3 | 21 | 140 | $0.13 \times 10^{-4}$ | $0.10 \times 10^{1}$ |
| 4 | 41 | 119 | $0.55 \times 10^{-6}$ | $0.21 \times 10^{1}$ |
| 5 | 69 | 119 | $0.53 \times 10^{-7}$ | $0.72 \times 10^{1}$ |
|  |  | 562 |  | 12.16 |

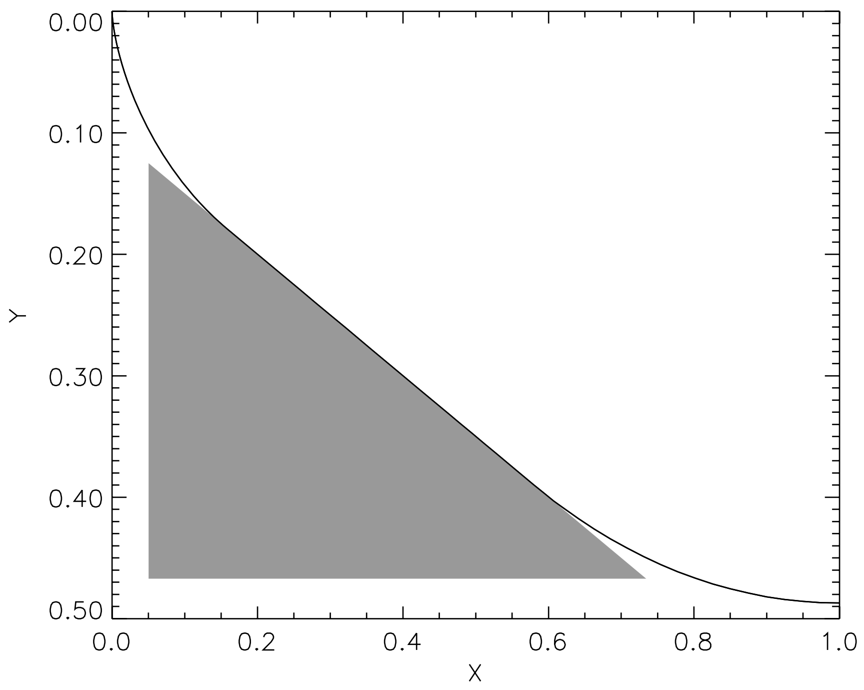


**Figure 4.35.** *State-constrained solution.*

## 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 $\mathbb{SOCS}$, it is worth reviewing the behavior on a problem

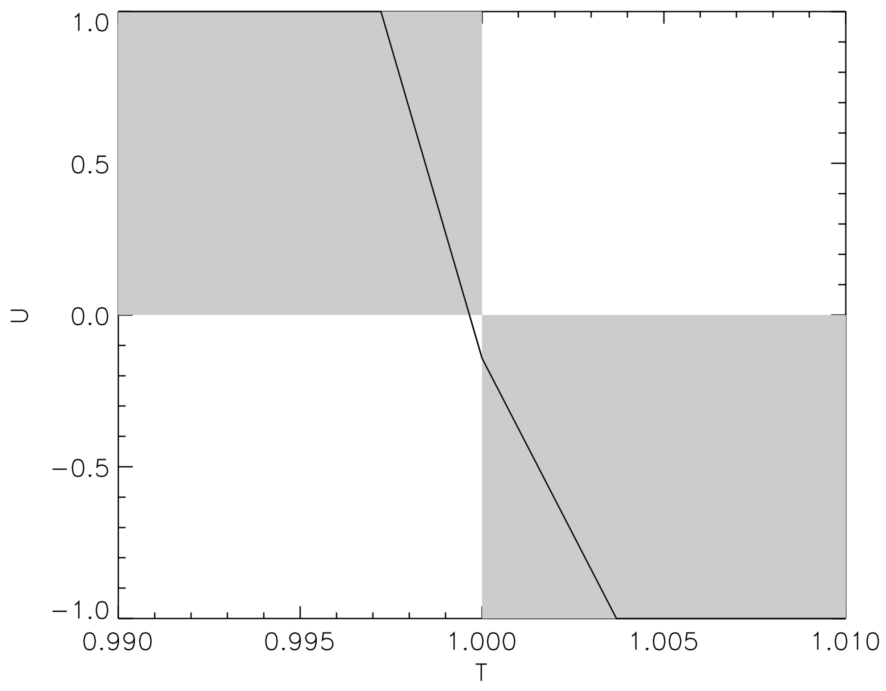


**Figure 4.36.** *Bang-bang control approximation.*

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

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= u, \\ -1 \leq u &\leq 1 \end{aligned}$$

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.

Chapter 5

# Parameter Estimation

## 5.1 Introduction

The behavior of many physical processes can be described mathematically by ordinary differential or differential-algebraic equations. Commonly a finite number of parameters appear in the description of the system dynamics. A parameter estimation problem arises when it is necessary to compute values for these parameters based on observations of the system dynamics. Methods for solving these so-called *inverse problems* have been used for many years [155]. In fact, most techniques in use today are based on ideas proposed by Gauss nearly 200 years ago that he used to solve orbit determination problems.

Traditional methods for solving inverse problems usually combine a standard initial-value numerical integration technique with a Gauss–Newton method for optimization. An approach that uses neither of the traditional processes can be constructed using the same techniques introduced in Chapter 4 for solving optimal control problems. Initial value integration is replaced by a discretization of the relevant differential-algebraic equations. We then exploit sparse finite difference approximations to the Hessian matrix, which permits us to construct a quadratically convergent algorithm for solving parameter estimation problems.

## 5.2 The Parameter Estimation Problem

Typically the dynamics of the system are defined for $t_I \le t \le t_F$ by a set of ODEs written in explicit form given by (4.32) and restated here as

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t], \tag{5.1}$$

where $\mathbf{y}$ is the $n_y$ dimension state vector, and $\mathbf{u}$ is an $n_u$ dimension vector of algebraic variables. In addition the solution must satisfy *algebraic path constraints* (4.33) of the form

$$\mathbf{g}_L \le \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \le \mathbf{g}_U, \tag{5.2}$$

where $\mathbf{g}$ is a vector of size $n_g$, with elements of the form

$$g[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] = \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{v} + \boldsymbol{\beta}^{\mathsf{T}}\mathbf{a}[\mathbf{v}, t], \tag{5.3}$$

where

$$\mathbf{v} = \begin{bmatrix} \mathbf{y}(t) \\ \mathbf{u}(t) \\ \mathbf{p} \end{bmatrix} \tag{5.4}$$

and

$$\mathbf{a}[\mathbf{v},t] = \begin{bmatrix} a_0(\mathbf{y},\mathbf{u},\mathbf{p},t) \\ a_1(\mathbf{y},\mathbf{u},\mathbf{p},t) \\ \vdots \\ a_{n_a}(\mathbf{y},\mathbf{u},\mathbf{p},t) \end{bmatrix}. \tag{5.5}$$

The constraint definition can include *analytic* terms involving $\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{v}$, where the $(n_y + n_u + n_p)$ vector $\boldsymbol{\alpha}$ is constant, as well as linear combinations of the $n_a$ *auxiliary functions* $a_k(\mathbf{v},t)$ for $k = 0,\dots,n_a$, where the coefficients $\beta_k$ are nonzero constants. By convention, a path constraint with a single nonlinear term $a_0(\mathbf{y},\mathbf{u},\mathbf{p},t)$ has no auxiliary functions ($n_a = 0$). Observe that each individual path constraint may have a different number of auxiliary functions and analytic terms. In addition to the general constraints (5.2) it is computationally useful to include simple linear bounds on the state variables as in (4.34)

$$\mathbf{y}_L \le \mathbf{y}(t) \le \mathbf{y}_U, \tag{5.6}$$

the algebraic variables as in (4.35)

$$\mathbf{u}_L \le \mathbf{u}(t) \le \mathbf{u}_U, \tag{5.7}$$

and the $n_p$ parameters

$$\mathbf{p}_L \le \mathbf{p} \le \mathbf{p}_U. \tag{5.8}$$

Note that an equality constraint can be imposed if the upper and lower bounds are equal; e.g., $(g_L)_k = (g_U)_k$ for some $k$. Boundary conditions at the initial time $t_I$ and/or final time $t_F$ are defined by

$$\boldsymbol{\psi}_L \le \boldsymbol{\psi}[\mathbf{y}(t_I),\mathbf{u}(t_I),t_I,\mathbf{y}(t_F),\mathbf{u}(t_F),t_F,\mathbf{p}] \le \boldsymbol{\psi}_U. \tag{5.9}$$

Unlike an optimal control problem with objective given by (4.39), for the *parameter estimation problem* the goal is to determine the $n_p$-dimensional vector $\mathbf{p}$ to minimize the performance index

$$F = \frac{1}{2}\mathbf{r}^{\mathsf{T}}\mathbf{r} = \frac{1}{2}\sum_{k=1}^{\ell} r_k^2, \tag{5.10}$$

where $\mathbf{r}$ is the $\ell$-dimensional *residual* vector. Components of the residual vector can be of two forms. State residuals are of the form

$$r_k = w_k\left[y_{i(k)}(\theta_k) - \hat{y}_{i(k)}\right], \tag{5.11}$$

where $y_{i(k)}(\theta_k)$ is the value of state variable $i(k)$ computed at time $\theta_k$ and $\hat{y}_{i(k)}$ is the observed value at the same point. Algebraic residuals are of the form

$$r_k = w_k\left[u_{i(k)}(\theta_k) - \hat{u}_{i(k)}\right], \tag{5.12}$$

where $u_{i(k)}(\theta_k)$ is the value of algebraic variable $i(k)$ computed at time $\theta_k$ and $\hat{u}_{i(k)}$ is the observed value at the same point. The residual weights are typically positive, i.e., $w_k > 0$. It is required that data evaluation points satisfy

$$t_I \leq \theta_k \leq t_F. \tag{5.13}$$

Often the evaluation points are arranged monotonically, that is, $\theta_k \leq \theta_{k+1}$. It is also common to have many residuals evaluated at the same time, e.g., $\theta_k = \theta_{k+1}$. Although neither of these assumptions is necessary for our approach, we do require that the initial and final times $t_I$ and $t_F$ be fixed. In contrast to an optimal control problem whose objective function (4.39) involves quantities evaluated continuously over the entire domain, the parameter estimation objective (5.10) is evaluated at a finite, albeit possibly large, number of discrete points.

It is worth noting that more complicated problem descriptions can be accommodated by the formulation given. For example, suppose it is required to minimize the expression

$$F = \frac{1}{2}\sum_{k=1}^{N}[\mathbf{h}(\mathbf{y}(\theta_k),\mathbf{u}(\theta_k),\mathbf{p},\theta_k) - \widehat{\mathbf{h}}_k]^{\mathsf{T}}\boldsymbol{\Lambda}[\mathbf{h}(\mathbf{y}(\theta_k),\mathbf{u}(\theta_k),\mathbf{p},\theta_k) - \widehat{\mathbf{h}}_k], \tag{5.14}$$

where $\widehat{\mathbf{h}}_k$ are the observed values of the function $\mathbf{h}$ at the times $\theta_k$ and $\boldsymbol{\Lambda}$ is the inverse covariance matrix of these quantities. Since the positive definite matrix can be factored as $\boldsymbol{\Lambda} = \mathbf{Q}^{\mathsf{T}}\mathbf{Q}$ we can define a new set of algebraic variables

$$\mathbf{z}(t) = \mathbf{Q}\mathbf{h}(\mathbf{y}(t),\mathbf{u}(t),\mathbf{p},t) \tag{5.15}$$

and transform the observed data

$$\widehat{\mathbf{z}}_k = \mathbf{Q}\widehat{\mathbf{h}}_k. \tag{5.16}$$

The *maximum likelihood* objective function (5.14) then becomes

$$F = \frac{1}{2}\sum_{k=1}^{N}[\mathbf{z}_k - \widehat{\mathbf{z}}_k]^{\mathsf{T}}[\mathbf{z}_k - \widehat{\mathbf{z}}_k], \tag{5.17}$$

where the residuals have the form given by (5.12) and the transformation (5.15) can be treated as an equality path constraint as in (5.2).

This example suggests that in general the discrete data may involve complicated expressions of the "real" state and algebraic variables $\mathbf{y}(t), \mathbf{u}(t)$ and the parameters $\mathbf{p}$. When this occurs the problem can be restated in terms of an augmented system. In the most common situation the *observation*

$$\mathbf{z}(t) = \mathbf{h}[\mathbf{y}(t),\mathbf{u}(t),t] \tag{5.18}$$

is treated as an (additional) algebraic constraint and it is natural to augment the "real" algebraic variable $\mathbf{u}(t)$ to include the additional algebraic variable $\mathbf{z}(t)$. On the other hand, if the observation is given by

$$\mathbf{z}(t) = \mathbf{h}[\mathbf{y}(t),t], \tag{5.19}$$

then it is possible to augment the "real" *state* variable $\mathbf{y}(t)$ to include the additional state $\mathbf{z}(t)$. In this case the state equations (5.1) must be augmented to include

$$\dot{\mathbf{z}}(t) = \mathbf{h}_y\dot{\mathbf{y}} + \dot{\mathbf{h}} = \mathbf{h}_y\mathbf{f} + \dot{\mathbf{h}}, \tag{5.20}$$

where the vector $\mathbf{h}_y \doteq (\partial h/\partial y_1, \dots, \partial h/\partial y_n)$ is considered a row vector.

For the sake of simplicity we have not introduced problems with multiple "phases." Nevertheless, our software implementation $\mathbb{SOPE}$ [38] does not have these restrictions.

## 5.3 Computing the Residuals

In order to evaluate the residuals (5.11) it is necessary to compute the value of the state variable at the data evaluation time $\theta_k$ as illustrated in Figure 5.1. This quantity can be constructed from the Hermite interpolating polynomial. Thus for any particular residual $k$ there is an interval $t_j \le \theta_k \le t_{j+1}$ and a particular state $\nu = i(k)$. Then the value of the state needed in the residual calculation is

$$\begin{aligned} y_\nu(\theta_k) = (1 - 3\delta^2 + 2\delta^3)y_{\nu j} + (3\delta^2 - 2\delta^3)y_{\nu,j+1} \\ + (h_j\delta - 2h_j\delta^2 + h_j\delta^3)f_{\nu j} + (-h_j\delta^2 + h_j\delta^3)f_{\nu,j+1}, \end{aligned} \tag{5.21}$$

where $h_j = t_{j+1} - t_j$ is the length of the discretization interval and $\delta = (\theta_k - t_j)/h_j$ defines the location of the evaluation time relative to the beginning of the interval. In this expression, $y_{\nu j}$ is the value of state variable $\nu$ at grid point $j$ and $f_{\nu j}$ is the corresponding value of the right-hand side (5.1) at the same grid point. The value of the algebraic variable required in (5.12) can be constructed from a quadratic interpolant when the Hermite–Simpson discretization is used, i.e., according to

$$u_\nu(\theta_k) = (1-\delta)(1-2\delta)u_{\nu j} + 4\delta(1-\delta)\bar{u}_{\nu,j+1} - \delta(1-2\delta)u_{\nu,j+1}, \tag{5.22}$$

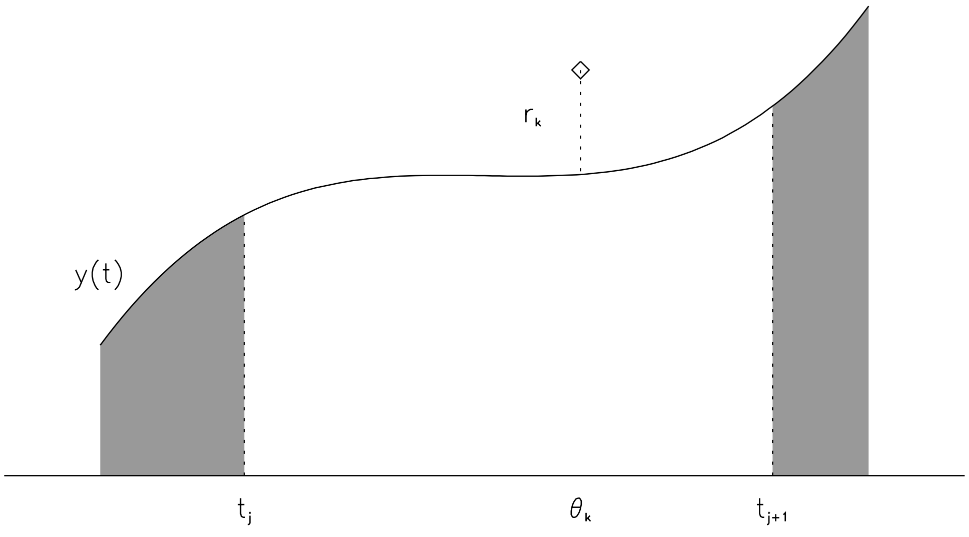


**Figure 5.1.** *Residual evaluation.*

where $\delta = (\theta_k - t_j)/h_j$. Similarly, when a trapezoidal discretization is used, linear interpolation between the grid points yields

$$u_\nu(\theta_k) = (1-\delta)u_{\nu j} + \delta u_{\nu,j+1}. \tag{5.23}$$

It is important to observe that the residuals are computed by interpolation and do not have any direct effect on the location of the discretization grid points. This is often referred to as *dense output* in methods for numerical integration (cf. [106]). It is worth emphasizing another property of the interpolation scheme. In each of the expressions (5.21), (5.22), and (5.23) the interpolated value is written as a linear combination of the NLP variables and the right-hand-side functions **f** at the grid points. In particular the quantities $h_j$ are fixed by the mesh-refinement procedure, and the quantities $\delta$ are fixed by the location of the discrete data points within a mesh. Thus, within a particular mesh-refinement step, the coefficients defining the interpolants are *constant* during the NLP optimization iterations. For example, the term $(-h_j\delta^2 + h_j\delta^3)$ in (5.21) remains unchanged by the NLP variables. This will be exploited when constructing derivatives. Furthermore to improve numerical conditioning we first normalize the time domain using (3.114) and then evaluate (5.21), (5.22), or (5.23) as functions of $\tau$ rather than $t$.

## 5.4 Computing Derivatives

First and second derivatives are constructed by exploiting the sparse finite differencing techniques described in Sections 2.10.3 and 4.6.8. The key notion is to write the complete set of transcribed NLP functions as

$$\begin{bmatrix} \mathbf{c}(\mathbf{x}) \\ \mathbf{r}(\mathbf{x}) \end{bmatrix} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}) + \boldsymbol{\zeta}, \tag{5.24}$$

where **A** and **B** are matrices (constant during the NLP) and **q** involves the nonlinear functions at grid points. Elements of the vector $\boldsymbol{\zeta}$ that correspond to defect constraints are zero. Similar information for the nonlinear boundary functions $\boldsymbol{\psi}$ can also be incorporated. We then construct finite difference estimates for the first derivatives of the set of $\upsilon$ functions $q_i(\mathbf{x})$ with respect to the $n$ variables $\mathbf{x}$ in the $\upsilon \times n$ matrix

$$\mathbf{D} \equiv \begin{bmatrix} (\nabla q_1)^\top \\ (\nabla q_2)^\top \\ \vdots \\ (\nabla q_\upsilon)^\top \end{bmatrix} = \frac{\partial \mathbf{q}}{\partial \mathbf{x}}. \tag{5.25}$$

The efficiency of the differencing technique depends on the sparsity of the matrix **D** defined in Section 2.2.1. The columns of **D** can be partitioned into subsets called *index sets* such that each subset has at most one nonzero element per *row*. Derivatives are constructed by perturbing all variables in an index set at the same time, and consequently the number of perturbations needed to construct **D** can be much smaller than the number of variables $n$. In our software, we construct this problem-dependent *sparsity template* information by random sampling of the user functions. From the sparsity template information, it is possible to construct the sparsity for the matrix **D** and compute the finite difference index sets. The first derivative information needed to solve the NLP can then be computed from

$$\begin{bmatrix} \mathbf{G} \\ \mathbf{R} \end{bmatrix} = \mathbf{A} + \mathbf{BD}, \tag{5.26}$$

where **G** is the Jacobian of the constraints and **R** is the residual Jacobian. This function separability can be further exploited to construct the residual Hessian needed in (2.56) using (2.66)–(2.69).

### 5.4.1 Residuals and Sparsity

There are a number of aspects of the approach that deserve emphasis. First, because the grid points (4.43) do not necessarily coincide with the data evaluation points (5.13), the sparsity pattern of the matrices **R** and **V** do not have a simple block form. Second, the grid points are placed to efficiently control the discretization error by the mesh-refinement procedure. However, the data points at $\theta_k$ do not have any direct relation to the grid points at $t_j$. In essence the numerical integration of the differential equations is not controlled by the observation data. This also has an impact on the sparsity of the residual Jacobian and Hessian as illustrated in Figure 5.2. In this illustration, when the mesh includes the points at $t_j$ and $t_{j+1}$, the partial derivative of the residual $r_k = w_k \left[ y_\nu(\theta_k) - \hat{y}_{\nu j} \right]$ with respect to the state at the left grid point is nonzero, i.e.,

$$\frac{\partial r_k}{\partial y(t_j)} \neq 0.$$

However, when the mesh is refined by adding a new grid point at $t_r = \frac{1}{2}(t_{j+1} + t_j)$, we find that

$$\frac{\partial r_k}{\partial y(t_r)} \neq 0 \quad \text{but} \quad \frac{\partial r_k}{\partial y(t_j)} = 0.$$

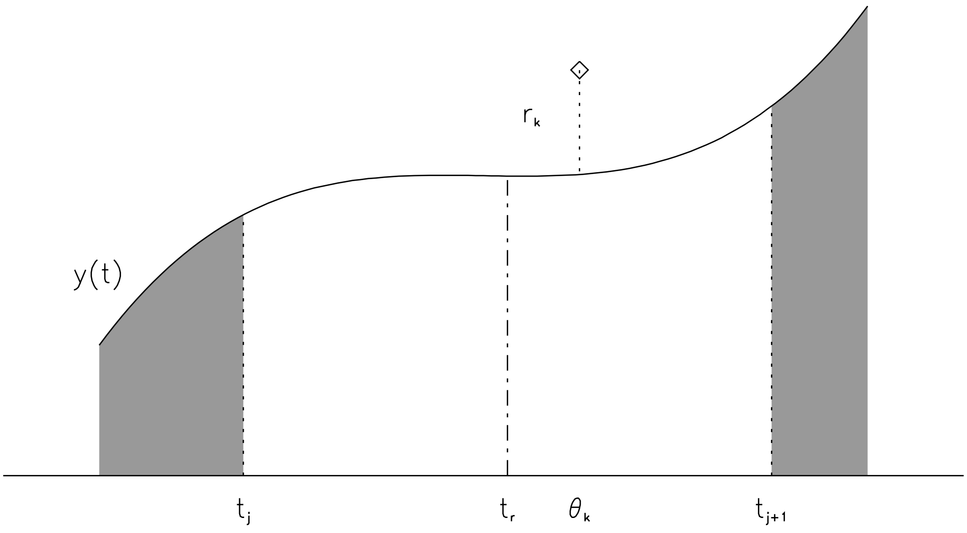


**Figure 5.2.** *Mesh refinement alters sparsity.*

Thus, mesh refinement alters the sparsity pattern of the residual Jacobian and Hessian matrices. Although it is more complicated to implement the construction of the sparsity pattern, there is no apparent impact on the solution times.

### 5.4.2 Residual Decomposition

Let us now present the details of the decomposition (5.24) for the various terms. To express a state residual given by (5.11) and (5.21) in the decomposed form we write

$$\begin{aligned}
\mathbf{r}_k(\mathbf{x}) &= \mathbf{A}_{k+m}\mathbf{x} + \mathbf{B}_{k+m}\mathbf{q}(\mathbf{x}) + \boldsymbol{\zeta}_{k+m} \\
&= \begin{bmatrix} \mathbf{0} & w_k(1-3\delta^2+2\delta^3) & \mathbf{0} & w_k(3\delta^2-2\delta^3) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \cdot \\ y_{\nu j} \\ \cdot \\ y_{\nu,j+1} \\ \cdot \end{bmatrix} \\
&\quad + \begin{bmatrix} \mathbf{0} & w_k(h_j\delta - 2h_j\delta^2 + h_j\delta^3) & \mathbf{0} & w_k(-h_j\delta^2+h_j\delta^3) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \cdot \\ f_{\nu j} \\ \cdot \\ f_{\nu,j+1} \\ \cdot \end{bmatrix} \\
&\quad - w_k\hat{y}_{\nu j}.
\end{aligned} \tag{5.27}$$

Observe that there are only two nonzero values in row $(k+m)$ of the matrices $\mathbf{A}$ and $\mathbf{B}$ denoted by $\mathbf{A}_{k+m}$ and $\mathbf{B}_{k+m}$, respectively. The nonlinear portions of the residual have been isolated in the vector $\mathbf{q}$. Furthermore, the problem-dependent sparsity of the nonlinear quantities can be exploited because of separability; i.e., there is no interdependence between grid points. Finally, it should be clear that the algebraic residuals (5.12) can also be written in the separable form required by (5.24) using either the quadratic (5.22) or the linear (5.23) interpolant.

### 5.4.3 Auxiliary Function Decomposition

The benefits of sparsity can be exploited by utilizing the separable form for the algebraic equations. Specifically an algebraic constraint function $g[\mathbf{y}(t),\mathbf{u}(t),\mathbf{p},t]$ as given in (5.3) can be expressed as

$$\begin{aligned}
\mathbf{c}_k(\mathbf{x}) &= \mathbf{A}_k\mathbf{x} + \mathbf{B}_k\mathbf{q}(\mathbf{x}) + \boldsymbol{\zeta}_k \\
&= \begin{bmatrix} \mathbf{0} & \boldsymbol{\alpha}^{\mathsf{T}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \cdot \\ \mathbf{y}_j \\ \mathbf{u}_j \\ \mathbf{p} \\ \cdot \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \beta_0 & ,\dots, & \beta_{n_a} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \cdot \\ a_0(t_j) \\ \dots \\ a_{n_a}(t_j) \\ \cdot \end{bmatrix}.
\end{aligned} \tag{5.28}$$

Here, $\mathbf{A}_k = [\mathbf{0}, \boldsymbol{\alpha}^{\mathsf{T}}, \mathbf{0}]$, $\mathbf{B}_k = [\mathbf{0}, \beta_0, \dots, \beta_{n_a}, \mathbf{0}]$, and $\boldsymbol{\zeta}_k = 0$. In contrast to the decomposition of the residual specified by (5.27), which can be performed algorithmically, this formulation must be given by the user. Nevertheless, the efficiency improvements are similar.

Again, the key notion is to define the vector $\mathbf{q}$ which is differentiated so that the nonlinearities are isolated and involve quantities at a single grid point.

Ultimately the software implementation must compute derivatives of user-supplied quantities via sparse finite differences. However, the user can reduce the cost of finite differencing by exploiting separability in the functions. To illustrate this point consider three different, yet mathematically equivalent, formulations of the same problem. Suppose the dynamic system has one state variable $y$, one control variable $u$, and one parameter $p$ that satisfy the DAE system

$$\begin{aligned} \dot{y} &= f(u), \\ 0 &= g(y,u,p), \end{aligned} \tag{5.29}$$

where $g(y,u,p) \equiv b_0(y) + u + b_1(p)$. There is some flexibility in how to group the terms in the path constraint when constructing the expression $g(y,u,p) = \boldsymbol{\alpha}^\mathsf{T}\mathbf{v} + \boldsymbol{\beta}^\mathsf{T}\mathbf{a}[\mathbf{v},t]$. One approach is to ignore separability and simply compute the terms enclosed between "{" and "}" together, i.e., compute $g(y,u,p) = \{b_0(y) + u + b_1(p)\}$. With this approach we define the quantities in the path constraint function (5.3) as follows:

$$\begin{aligned} \boldsymbol{\alpha}^\mathsf{T} &= (0,0,0), \\ n_a &= 0, \\ a_0(y,u,p) &= b_0(y) + u + b_1(p), \\ \boldsymbol{\beta}^\mathsf{T} &= (1). \end{aligned} \tag{5.30}$$

For this formulation the user must compute the functions $f$ and $a_0$, and the matrix $\mathbf{D}$ will contain repeated blocks with the sparsity template

$$struct \begin{pmatrix} \frac{\partial f}{\partial y} & \frac{\partial f}{\partial u} & \frac{\partial f}{\partial p} \\ \frac{\partial a_0}{\partial y} & \frac{\partial a_0}{\partial u} & \frac{\partial a_0}{\partial p} \end{pmatrix} = \begin{bmatrix} 0 & \mathsf{x} & 0 \\ \mathsf{x} & \mathsf{x} & \mathsf{x} \end{bmatrix}.$$

Since the rows of the matrix $\mathbf{D}$ corresponding to the path constraint will have three nonzero elements, this formulation will require three index sets and hence three perturbations to compute a finite difference approximation for $\mathbf{D}$.

A second alternative is to compute the first two terms together and explicitly identify an auxiliary function, i.e., $g(y,u,p) = \{b_0(y) + u\} + \{b_1(p)\}$. Here we define

$$\begin{aligned} \boldsymbol{\alpha}^\mathsf{T} &= (0,0,0), \\ n_a &= 1, \\ a_0(y,u,p) &= b_0(y) + u, \\ a_1(y,u,p) &= b_1(p), \\ \boldsymbol{\beta}^\mathsf{T} &= (1,1). \end{aligned} \tag{5.31}$$

Since the user must compute the functions $f$, $a_0$, and $a_1$ individually the corresponding sparsity template will have the form

$$struct \begin{pmatrix} \frac{\partial f}{\partial y} & \frac{\partial f}{\partial u} & \frac{\partial f}{\partial p} \\ \frac{\partial a_0}{\partial y} & \frac{\partial a_0}{\partial u} & \frac{\partial a_0}{\partial p} \\ \frac{\partial a_1}{\partial y} & \frac{\partial a_1}{\partial u} & \frac{\partial a_1}{\partial p} \end{pmatrix} = \begin{bmatrix} 0 & \mathsf{x} & 0 \\ \mathsf{x} & \mathsf{x} & 0 \\ 0 & 0 & \mathsf{x} \end{bmatrix}.$$

Using this formulation the finite difference derivatives can be computed using two perturbations.

A third alternative is to define

$$\begin{aligned}\boldsymbol{\alpha}^{\mathsf{T}} &= (0,1,0),\\ n_a &= 1,\\ a_0(y,u,p) &= b_0(y),\\ a_1(y,u,p) &= b_1(p),\\ \boldsymbol{\beta}^{\mathsf{T}} &= (1,1).\end{aligned} \tag{5.32}$$

Here we explicitly identify both the analytic term and the auxiliary function. Since the sparsity template is

$$struct\begin{pmatrix} \frac{\partial f}{\partial y} & \frac{\partial f}{\partial u} & \frac{\partial f}{\partial p} \\ \frac{\partial a_0}{\partial y} & \frac{\partial a_0}{\partial u} & \frac{\partial a_0}{\partial p} \\ \frac{\partial a_1}{\partial y} & \frac{\partial a_1}{\partial u} & \frac{\partial a_1}{\partial p} \end{pmatrix} = \begin{bmatrix} 0 & \mathsf{x} & 0 \\ \mathsf{x} & 0 & 0 \\ 0 & 0 & \mathsf{x} \end{bmatrix},$$

this formulation requires only one perturbation to compute the finite difference approximation for $\mathbf{D}$.

This example illustrates the need for a more general software interface. Typically when solving a semiexplicit DAE such as (5.29), the user must provide a subroutine to compute the right-hand-side functions $f(y,u,p,t)$ and $g(y,u,p,t)$ for given values of the arguments $(y,u,p,t)$. However, to fully exploit sparsity our software implementation permits the user to compute the augmented set of right-hand-side functions $f(y,u,p,t)$ and $a_k(y,u,p,t)$ for $k = 0,\dots,n_a$. Nevertheless, a twofold computational benefit is observed by exploiting separability. First, the Hessian matrix is usually more sparse since it is determined by the structure of $(\mathbf{BD})^{\mathsf{T}}(\mathbf{BD})$. This leads to computational savings when solving the linear systems required by the NLP algorithm. Second, since gradient information can be computed with fewer perturbations, it is not necessary to call the user function routines as many times, leading to additional computational savings. It is important to note that exploiting separability is computationally beneficial for both optimal control and parameter estimation problems. In fact the same separability benefits lead to the HSS discretization developed in Section 4.6.6. The issue becomes more important when solving nonlinear parameter estimation problems that require treatment with algebraic path constraints as in (5.3).

### 5.4.4 Algebraic Variable Parameterization

For many applications it is natural to consider representing one or more of the algebraic variables in terms of a finite number of parameters. In particular, let us consider

$$u(t) = \sum_j b_j B_j(t), \tag{5.33}$$

where $t_I \le t \le t_F$. If the domain is subdivided into $N$ equally spaced segments and the functions $B_j(t)$ are cubic B-splines with $C^1$ continuity, the algebraic variable $u(t)$ is uniquely determined by the $(2N+2)$ coefficients $b_j$. One alternative is to eliminate $u(t)$ by substituting this expression wherever the algebraic variable $u(t)$ appears in the problem. Another alternative is to include the coefficients $b_j$ as parameters $\mathbf{p}$ and then introduce an additional constraint

$$\begin{aligned} 0 &= g[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \\ &= u(t) - \sum_j b_j B_j(t). \end{aligned} \tag{5.34}$$

When the continuous problem is transcribed into an NLP, this constraint is enforced at the discretization grid points leading to the set of NLP constraints

$$0 = u(\tau_k) - \sum_j b_j B_j(\tau_k) \tag{5.35}$$

for $k = 1, \dots, M$, with $t_k = \tau_k \Delta t + t_I$ and $\Delta t = (t_F - t_I)$ as in (3.114). This expression can be simplified further by exploiting a property of the B-spline basis functions $B_j(\tau_k)$ referred to as *local support*. In our case, the B-spline basis is constructed over $N$ equally spaced segments, so in segment $\ell$

$$B_j(\tau_k) \geq 0, \qquad \frac{(\ell-1)}{N} \leq \tau_k \leq \frac{\ell}{N}, \tag{5.36a}$$

$$B_j(\tau_k) = 0 \qquad \text{otherwise.} \tag{5.36b}$$

Moreover for a $C^1$ cubic representation, there are at most four nonzero basis functions $B_j(\tau_k)$ at any grid point $\tau_k$. Since the discretization grid points and B-spline break points, called *knot locations*, are fixed relative to each other, the nonzero B-spline basis values $B_j(\tau_k)$ can be precomputed. Consequently each B-spline constraint (5.35) is of the form (5.3) with $\boldsymbol{\beta} = \mathbf{0}$ and strictly analytic terms $\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{v}$, where

$$\boldsymbol{\alpha}^{\mathsf{T}} = \left[\dots, 1, \dots, -B_\alpha(\tau_k), -B_{\alpha+1}(\tau_k), -B_{\alpha+2}(\tau_k), -B_{\alpha+3}(\tau_k), \dots\right], \tag{5.37}$$

$$\mathbf{v}^{\mathsf{T}} = \left[\dots, u_k, \dots, b_\alpha, b_{\alpha+1}, b_{\alpha+2}, b_{\alpha+3}, \dots\right]. \tag{5.38}$$

Observe that by treating the B-spline decomposition analytically, these constraints do not introduce any additional sparse finite difference index sets. Furthermore, the NLP constraints are strictly linear in the NLP variables and thus are easily satisfied as part of the NLP iterative process. Clearly this algebraic variable parameterization can be used for either optimal control or optimal estimation problems.

## 5.5 Computational Experience

The computational performance and behavior of the algorithm are illustrated on a number of examples.

**Example 5.1** NOTORIOUS PROBLEM. An example described by Bock [44] and Schittkowski [155, p. 141] as a "notorious test problem" was originally introduced by Bulirsch [55]. The differential equations are

$$\dot{y}_1 = y_2, \tag{5.39}$$

$$\dot{y}_2 = \mu^2 y_1 - (\mu^2 + p^2)\sin(pt) \tag{5.40}$$

with $y_1(0) = 0$, $y_2(0) = \pi$, $\mu = 60$, and $0 \le t \le 1$. It is easily verified that if the parameter $p = \pi$, then the corresponding analytic solution to (5.40) is given by

$$y_1 = \sin(\pi t), \tag{5.41}$$

$$y_2 = \pi \cos(\pi t). \tag{5.42}$$

Data for this problem can be constructed by evaluating the true solution at the data points $\theta_k$ and then adding normally distributed random variables with mean zero and standard deviation $\sigma = .05$. It is easy to demonstrate that the optimal value of the objective function $F^* \approx \ell\sigma^2$, where $\ell$ is the total number of residuals. This deceptively simple example is extremely difficult to solve using any type of shooting method, because the differential equations are unstable. In contrast, the parameter estimation process using direct transcription is very well behaved. Furthermore, we can use the example to demonstrate two major features of the new algorithm; namely,

- the grid distribution is not determined by the data location, and
- the algorithm converges quadratically for nonzero, nonlinear residuals.

Consider three different cases:

1. for $k = 1,10$ select $\theta_k = .1k$;
2. for $k = 1,2000$ select $\theta_k$ as a uniformly distributed random variable in the region $0 \le \theta_k \le 1$; and
3. for $k = 1,10$ select $\theta_k = .1k$; for $k = 11,2000$ select $\theta_k$ as a normally distributed random variable with mean $= .4$ and standard deviation $= .1$.

The first case has a relatively small number of residuals and as such a small, albeit, nonzero objective at the solution. The second case has a large amount of data spread over the entire domain, whereas the third case has lots of data clustered in only one portion of the time domain near $t = .4$. Table 5.1 summarizes the performance of the algorithm. Notice that the number of mesh-refinement iterations, grid points, and NLP iterations is essentially the same for all three cases. This occurs even though the objective function is significantly nonzero at the solution. Furthermore, the optimal parameter estimate is quite good, especially since the discretization error tolerance was also $10^{-7}$.

**Table 5.1.** *Algorithm performance summary.*

| Case | 1 | 2 | 3 |
|---|---|---|---|
| No. mesh it. | 5 | 4 | 5 |
| No. grid pt. | 92 | 73 | 91 |
| No. NLP it. | 23 | 20 | 23 |
| $F^*$ | .030372674 | 2.6563759 | 2.4887179 |
| $\lvert p^* - \pi \rvert$ | $3.6\times10^{-8}$ | $4.7\times10^{-8}$ | $3.6\times10^{-8}$ |

Figure 5.3 illustrates the final mesh distribution for case 1 and case 3. The solid line plots the stepsize history as a function of time for case 3. The shading illustrates the distribution of data points over the domain—the darkest representing the highest concentration.

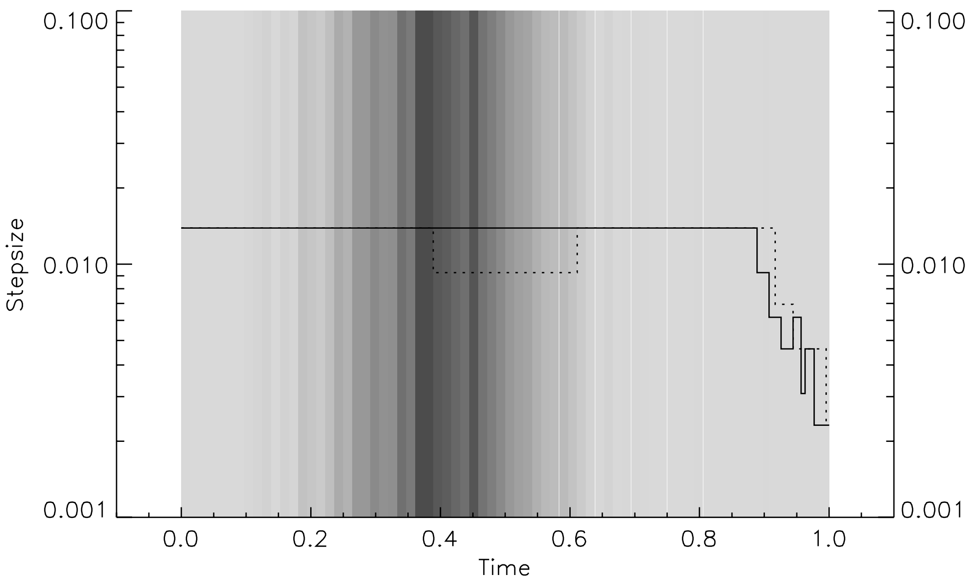


**Figure 5.3.** *Stepsize history (case* 1: *dashed; case* 3: *solid).*

The stepsize for case 1 is shown with dotted lines. Even though case 1 has only 11 points evenly spread over the time domain, and case 3 has 2000 data points clustered near $t = .4$, the final mesh distribution is nearly identical. Obviously for this example the location of the discrete data does not directly influence the location of the mesh points because the stepsize is constructed to control error in the differential equation.

## 5.5.1 Reentry Trajectory Reconstruction

A problem of some practical interest occurs when attempting to reconstruct the trajectory of an object as it reenters the earth's atmosphere using information from radar observations. Let us consider a nonlifting body of unknown size, shape, and mass, reentering the atmosphere over an oblate rotating earth. The translational motion is described by the differential equations

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{5.43}$$

$$\dot{\mathbf{v}} = -D\frac{\mathbf{v}_r}{\|\mathbf{v}_r\|} + \mathbf{g}(\mathbf{r}), \tag{5.44}$$

where $\mathbf{r}^\mathsf{T} = (x, y, z)$ is the earth centered inertial (ECI) position vector, $\mathbf{v}^\mathsf{T} = (\dot{x}, \dot{y}, \dot{z})$ is the ECI velocity vector, and $\mathbf{g}(\mathbf{r})$ is the gravitational acceleration. An oblate earth model including the first four zonal harmonics is used. The earth relative velocity vector is defined by

$$\mathbf{v}_r = \mathbf{v} - \omega \times \mathbf{r}, \tag{5.45}$$

where $\boldsymbol{\omega}^{\mathsf{T}} = (0,0,\omega)$ is the earth rotation rate vector. The drag on the object is given by

$$D = \frac{g_0 \rho \|\mathbf{v}_r\|^2}{2\beta}, \tag{5.46}$$

where $\rho(h)$ is the atmospheric density as a function of the altitude above the oblate spheroid, $g_0 = 32.174$, and $\beta$ is the *ballistic coefficient.* For this application the atmospheric density is computed using a cubic spline approximation to the 1962 Standard Atmosphere.

The goal is to reconstruct the position and velocity time history from radar information. Thus we would like to minimize

$$F = \frac{1}{2}\sum_{k=1}^{N} \mathbf{q}_k^{\mathsf{T}} \mathbf{q}_k \tag{5.47}$$

with

$$\mathbf{q}_k = \begin{bmatrix} (\psi_k - \hat{\psi}_k)/\sigma_1 \\ (\eta_k - \hat{\eta}_k)/\sigma_2 \\ (s_k - \hat{s}_k)/\sigma_3 \\ (\dot{s}_k - \hat{\dot{s}}_k)/\sigma_4 \end{bmatrix}, \tag{5.48}$$

where $\psi_k = \psi(\mathbf{r}(\theta_k), \mathbf{v}(\theta_k), \theta_k)$ is the azimuth angle from the radar site to the object evaluated at time $\theta_k$, and $\hat{\psi}_k$ is the corresponding radar measurement data, with standard deviation $\sigma_1$. Similarly, $\eta_k$ is the elevation, $s_k$ is the slant range, and $\dot{s}_k$ is the (slant) range rate. In order to restate the problem involving residuals of the form (5.12) we introduce the algebraic variables $(u_1, u_2, u_3, u_4)$ and the corresponding algebraic path equations

$$0 = \psi(\mathbf{r}, \mathbf{v}, t) - u_1(t), \tag{5.49}$$
$$0 = \eta(\mathbf{r}, \mathbf{v}, t) - u_2(t), \tag{5.50}$$
$$0 = s(\mathbf{r}, \mathbf{v}, t) - u_3(t), \tag{5.51}$$
$$0 = \dot{s}(\mathbf{r}, \mathbf{v}, t) - u_4(t). \tag{5.52}$$

After introducing the new algebraic variables it is clear that (5.47) can be rewritten in the form (5.12).

To complete the definition of the problem it is sufficient to describe how the radar quantities in (5.49)–(5.52) are computed. The position of the radar site at time $t$ is given by

$$\mathbf{w}(t) = r_e \begin{bmatrix} \cos\theta_s \cos\varphi \\ \cos\theta_s \sin\varphi \\ \sin\theta_s \end{bmatrix}, \tag{5.53}$$

where $\theta_s$ is the geocentric latitude of the radar site, $\varphi = \phi_s + \phi_0 + \omega t$ is the inertial longitude of the radar site, $\phi_s$ is the longitude of the radar site, and $r_e$ is the radius to the site. The inertial velocity of the radar site is

$$\dot{\mathbf{w}}(t) = \begin{bmatrix} -\omega[\sin(\omega t) r_1 + \cos(\omega t) r_2] \\ \omega[\cos(\omega t) r_1 - \sin(\omega t) r_2] \\ 0 \end{bmatrix}. \tag{5.54}$$

The line-of-sight vector from the radar site to the vehicle is given by

$$\mathbf{s} = \mathbf{r} - \mathbf{w}, \tag{5.55}$$

which yields the slant range

$$s(\mathbf{r}, \mathbf{v}, t) = \|\mathbf{s}\|. \tag{5.56}$$

The range rate is then given by

$$\dot{s}(\mathbf{r}, \mathbf{v}, t) = \frac{\mathbf{s}^{\mathsf{T}}(\mathbf{v} - \dot{\mathbf{w}})}{\|\mathbf{s}\|}. \tag{5.57}$$

The azimuth angle is given by

$$\psi(\mathbf{r}, \mathbf{v}, t) = \tan^{-1}\left[\frac{w_1 s_2 - w_2 s_1}{\left[(w_1^2 + w_2^2)s_3 - w_3(w_1 s_1 + w_2 s_2)\right] r_e^{-1}}\right]. \tag{5.58}$$

Now the local geodetic vertical direction at the radar site is

$$\mathbf{d} = \begin{bmatrix} \cos(\varphi)\cos(\theta_d) \\ \sin(\varphi)\cos(\theta_d) \\ \sin(\theta_d) \end{bmatrix}, \tag{5.59}$$

where $\theta_d$ is the geodetic latitude of the radar site, and the geodetic elevation is given by

$$\eta(\mathbf{r}, \mathbf{v}, t) = \frac{\pi}{2} - \cos^{-1}\left(\frac{\mathbf{d}^{\mathsf{T}}\mathbf{s}}{\|\mathbf{s}\|}\right). \tag{5.60}$$

**Example 5.2** Compton Gamma Ray Observatory Reentry. On June 4, 2000, the NASA Compton Gamma Ray Observatory satellite reentered the atmosphere, and a portion of the trajectory was observed by the Kaena Point tracking station in Hawaii. The 17 ton spacecraft, one of the largest ever launched by NASA, was deliberately de-orbited after one of the observatory's three attitude-control gyros failed in December, 1999. The radar site provided azimuth, elevation, range, and range rate data for a portion of the trajectory above 70nm altitude during a time span of approximately 4 min. Assistance provided by Dr. Wayne Hallman of The Aerospace Corporation concerning this example is gratefully acknowledged.

The parameter estimation method was used to reconstruct the reentry trajectory, and the results of the algorithm are summarized in Table 5.2. The algorithm began with 25 equally spaced grid points and after six refinement iterations increased the number of points to 410 (cf. column 2). This refinement reduced the discretization error $\epsilon$ from $1.24 \times 10^{-2}$ to $9.91 \times 10^{-8}$ as shown in column 7. The number of gradient, Hessian, function, and right-hand-side evaluations are given by the columns labeled NGC, NHC, NFE, and NRHS, respectively. Figure 5.4 displays the normalized error residuals, i.e., the components of $\mathbf{q}_k$ given by (5.48) for each set of data. The total normalized residual $\|\mathbf{q}_k^{\mathsf{T}}\mathbf{q}_k\|$ is plotted in Figure 5.5.

**Table 5.2.** *Mesh-refinement summary.*

| $k$ | $M$ | NGC | NHC | NFE | NRHS | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|
| 1 | 25 | 8 | 4 | 184 | 9016 | $1.24\times10^{-2}$ | 2.40 |
| 2 | 49 | 5 | 2 | 110 | 10670 | $8.27\times10^{-4}$ | 2.41 |
| 3 | 97 | 5 | 2 | 110 | 21230 | $4.02\times10^{-5}$ | 4.71 |
| 4 | 193 | 3 | 1 | 59 | 22715 | $2.40\times10^{-6}$ | 5.11 |
| 5 | 385 | 3 | 1 | 59 | 45371 | $1.53\times10^{-7}$ | 9.09 |
| 6 | 410 | 4 | 2 | 96 | 78624 | $9.91\times10^{-8}$ | 14.1 |
| Total | 410 | 28 | 12 | 618 | 187626 | | 37.84 |

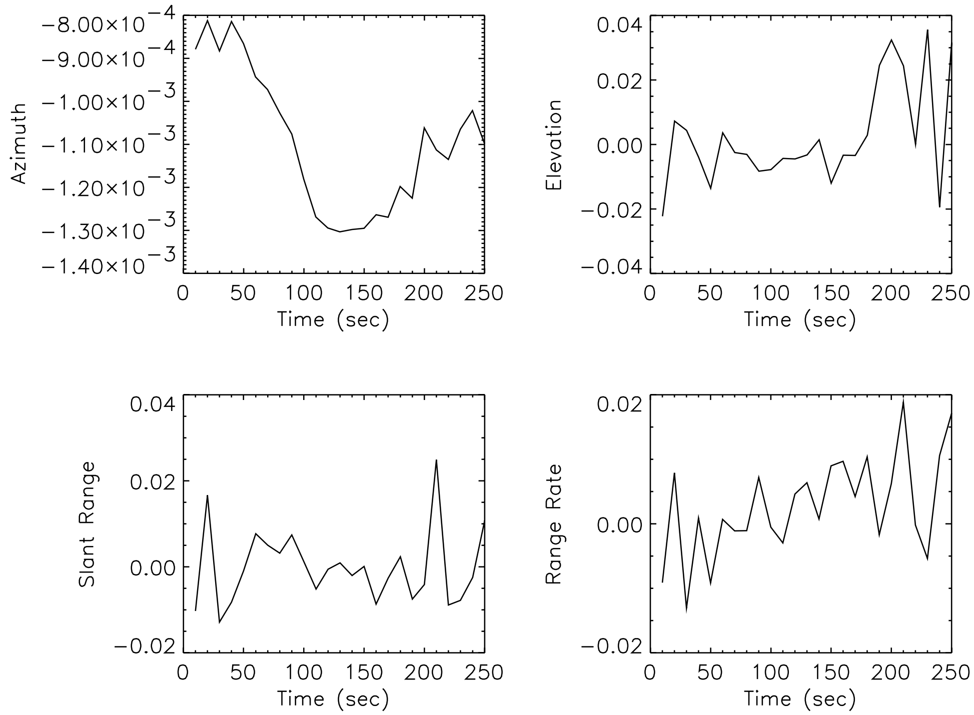


**Figure 5.4.** *Normalized residual errors.*

## 5.5.2 Commercial Aircraft Rotational Dynamics Analysis

When constructing a dynamic simulation of a commercial aircraft, flight test data are used to refine analytic models of the aerodynamic characteristics. A representative example of a parameter estimation problem occurs when attempting to estimate rotational accelerations from measured information about the aircraft orientation. We consider a particular maneuver called a "windup turn" for a 767-ER aircraft and gratefully acknowledge the contributions of Dr. Jia Luo of The Boeing Company for information related to this example.

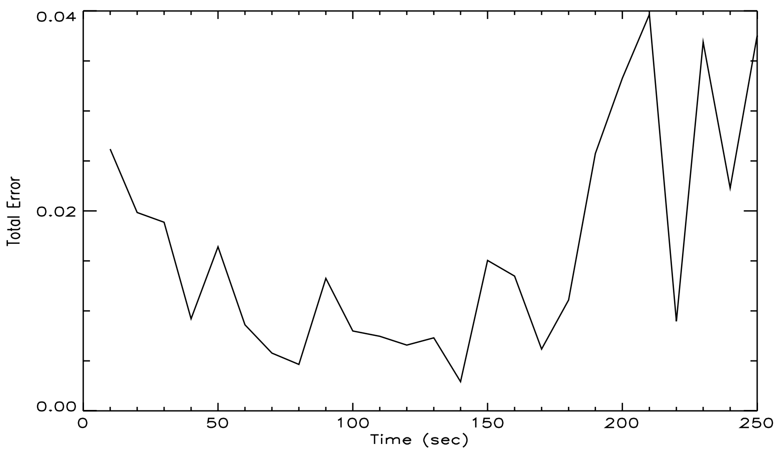


**Figure 5.5.** *Total normalized residual error.*

The rotational dynamics are described by

$$\dot{\phi} = p + q\frac{\sin\phi\sin\theta}{\cos\theta} + r\frac{\cos\phi\sin\theta}{\cos\theta}, \tag{5.61}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi, \tag{5.62}$$

$$\dot{\psi} = q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta}, \tag{5.63}$$

$$\dot{p} = a_p(\mathbf{b}), \tag{5.64}$$

$$\dot{q} = a_q(\mathbf{b}), \tag{5.65}$$

$$\dot{r} = a_r(\mathbf{b}), \tag{5.66}$$

where $\phi$ is the bank angle (rad), $\theta$ is the pitch angle (rad), $\psi$ is the heading angle (rad), $p$ is the roll rate (rad/sec), $q$ is the pitch rate (rad/sec), and $r$ is the yaw rate (rad/sec). During flight testing measurements of the bank, pitch, and heading angle are made; i.e., we have measured values $\hat{\phi}_k$, $\hat{\theta}_k$, and $\hat{\psi}_k$ at a sequence of time points—in this case 1841 values corresponding to measurements every .05 sec for 92 sec. We would like to compute the unknown accelerations $a_p$, $a_q$, and $a_r$ such that the objective

$$F = \frac{1}{2}\sum_{k=1}^{N}\left[\frac{\phi_k - \hat{\phi}_k}{\sigma_1}\right]^2 + \left[\frac{\theta_k - \hat{\theta}_k}{\sigma_2}\right]^2 + \left[\frac{\psi_k - \hat{\psi}_k}{\sigma_3}\right]^2 \tag{5.67}$$

is minimized, where the standard deviations on the data are given by $\sigma_j$. It should be clear that the residuals are of the form given by (5.11) with weights $w_k = 1/\sigma_\nu$. Note that for this example the symbol $\theta$ is used to denote a state variable, and *not* an evaluation time as in (5.11).

**Example 5.3** MULTIPHASE APPROXIMATION. There are many ways to parameterize the accelerations $a_p$, $a_q$, and $a_r$. Since the accelerations are smooth functions of time, a particularly effective approach is to utilize piecewise polynomial approximations. Let us

**Table 5.3.** *Mesh-refinement summary.*

| $k$ | $M$ | NGC | NHC | NFE | NRHS | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|
| 1 | 200 | 10 | 1 | 105 | 39900 | $3.51\times10^{-4}$ | 13.0 |
| 2 | 380 | 3 | 1 | 38 | 28120 | $2.32\times10^{-5}$ | 6.19 |
| 3 | 740 | 3 | 1 | 38 | 55480 | $1.48\times10^{-6}$ | 10.9 |
| 4 | 1429 | 3 | 1 | 38 | 107844 | $9.39\times10^{-8}$ | 20.9 |
| Total | 1429 | 19 | 4 | 219 | 231344 | | 51.07 |

introduce $N_p$ *phases*, where the independent variable $t$ for phase $k$ is defined in the region $t_I^{(k)} \le t \le t_F^{(k)}$ and the phases are sequential, that is, $t_I^{(k+1)} = t_F^{(k)}$. In addition let us construct the beginning of the first phase to coincide with the beginning of the problem $t_I^{(1)} = 0$, and the end of the last phase to coincide with the end of the problem $t_F^{(N_p)} = 92$. If we treat the values of the acceleration and their slopes at the phase boundaries as parameters, the accelerations within a phase are of the form

$$a(\mathbf{b}) = \mathcal{H}\left[a(t_I^{(k)}), \dot{a}(t_I^{(k)}), a(t_F^{(k)}), \dot{a}(t_F^{(k)})\right] \tag{5.68}$$

for $k = 1, \dots, N_p$. In this expression the Hermite interpolation $\mathcal{H}$ is given by (5.21), with the appropriate definition of symbols. Finally, we require continuity and differentiability in the state variables and accelerations across the phase boundaries. It is worth noting that in general there are three distinct levels of discretization. Within a phase, there may be many grid points selected to satisfy the differential equation accuracy requirements. Furthermore, the data observation points may or may not coincide with the phase times and/or the differential equation grid. For this 20 phase example, $N_p = 20$ and the total number of parameters $\mathbf{p}$ is $n_p = 12N_p = 240$. The particular data set used for this illustration had $N = 1841$ data points or 5523 residuals in (5.67). A summary of the mesh-refinement procedure is presented in Table 5.3. The process was initiated with 10 grid points per phase or a total of $M = 200$. The first NLP problem was solved after 10 gradient evaluations (NGC), and one Hessian evaluation (NHC), which required 39900 evaluations of the right-hand sides of the differential equations (NRHS). This problem was solved in 13 sec of CPU time with a discretization error of $\epsilon = 3.51 \times 10^{-4}$. The mesh was refined three more times as tabulated in rows 2–4. The overall solution was obtained in 51.07 sec and required 1429 mesh points. Notice that only one Hessian evaluation was required for each NLP problem, even though the objective function is quite nonlinear and the optimal value $F^* = 1.715105 \times 10^{-2} \neq 0$.

From this information it is also possible to infer how the new approach compares with a more traditional shooting method. Suppose we assume that a fourth order Runge–Kutta scheme is used to integrate the trajectory (which requires four right-hand-side evaluations per step), and there are 1429 steps (corresponding to the final grid size $M = 1429$). Then, the number of right-hand-side evaluations (231344) required by the new approach is equivalent to $231344/(1429 \times 4) \approx 41$ integrated trajectories. In comparison at least 240 trajectories would be required just to compute a single finite difference gradient in a traditional shooting method! Furthermore, if a quasi-Newton method is used to optimize this function with 132 degrees of freedom, one would expect that at least 132 iterations (and gradient evaluations) would be required to converge. An estimate of the total number of trajectories for a traditional shooting method is $(132 \times 240 = 31680)$. Thus, comparing the new versus the old algorithm suggests a ratio of $41 : 31680 \approx 1 : 773$. In short, a traditional shooting method would be extremely impractical for this application! The cost of computing first derivatives could be reduced somewhat for this problem by using a multiple shooting method; however, this approach still lacks quadratic convergence because it does not provide Hessian information.

Figure 5.6 presents the optimal time history for all of the angles as well as the data. Figure 5.7 illustrates the angular rates for the optimal solution and Figure 5.8 plots the corresponding accelerations. The phase boundaries are illustrated in all figures.

**Example 5.4** B-SPLINE APPROXIMATION. The key notion illustrated by Example 5.3 is to represent the rates $p$, $q$, and $r$ in (5.64)–(5.66) using the piecewise polynomial form given by (5.68). Twenty distinct phases were introduced in order to exploit sparsity. Similar benefits can be realized by exploiting the local support property of a B-spline representation, without explicitly introducing additional phases.

Two modifications to the original formulation are required. First, the rates are treated as new algebraic variables $a_p(t)$, $a_q(t)$, and $a_r(t)$. Second, additional path constraints are introduced to enforce the B-spline relationship. Specifically the dynamic equations (5.61)–(5.66) are replaced by the DAE system

$$\dot{\phi} = p + q\frac{\sin\phi\sin\theta}{\cos\theta} + r\frac{\cos\phi\sin\theta}{\cos\theta}, \tag{5.69}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi, \tag{5.70}$$

$$\dot{\psi} = q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta}, \tag{5.71}$$

$$\dot{p} = a_p(t), \tag{5.72}$$

$$\dot{q} = a_q(t), \tag{5.73}$$

$$\dot{r} = a_r(t), \tag{5.74}$$

$$0 = a_p(t) - \sum_k b_{pk}B_k(t), \tag{5.75}$$

$$0 = a_q(t) - \sum_k b_{qk}B_k(t), \tag{5.76}$$

$$0 = a_r(t) - \sum_k b_{rk}B_k(t). \tag{5.77}$$

The B-spline basis functions are denoted by $B_k(t)$ with corresponding coefficients $b_{pk}$, $b_{qk}$, and $b_{rk}$ for the respective rates. If 20 internal knots are equally spaced between $t_I$ and $t_F$, and a $C^1$ cubic spline is used, then the number of coefficients needed to represent each rate in (5.75)–(5.77) is $k = 42$. In comparison the piecewise polynomial representation introduces 80 parameters, and explicitly imposes $2 \times 19 = 38$ constraints to enforce continuity in the function and slope. Thus both representations have 42 degrees of freedom.

The differential, algebraic, and parametric variables, respectively, are

$$\mathbf{y}^{\mathsf{T}} = [\phi,\theta,\psi,p,q,r], \tag{5.78}$$

$$\mathbf{u}^{\mathsf{T}} = [a_p,a_q,a_r], \tag{5.79}$$

$$\mathbf{p}^{\mathsf{T}} = [b_{p1},\dots,b_{p42},b_{q1},\dots,b_{q42},b_{r1},\dots,b_{r42}]. \tag{5.80}$$

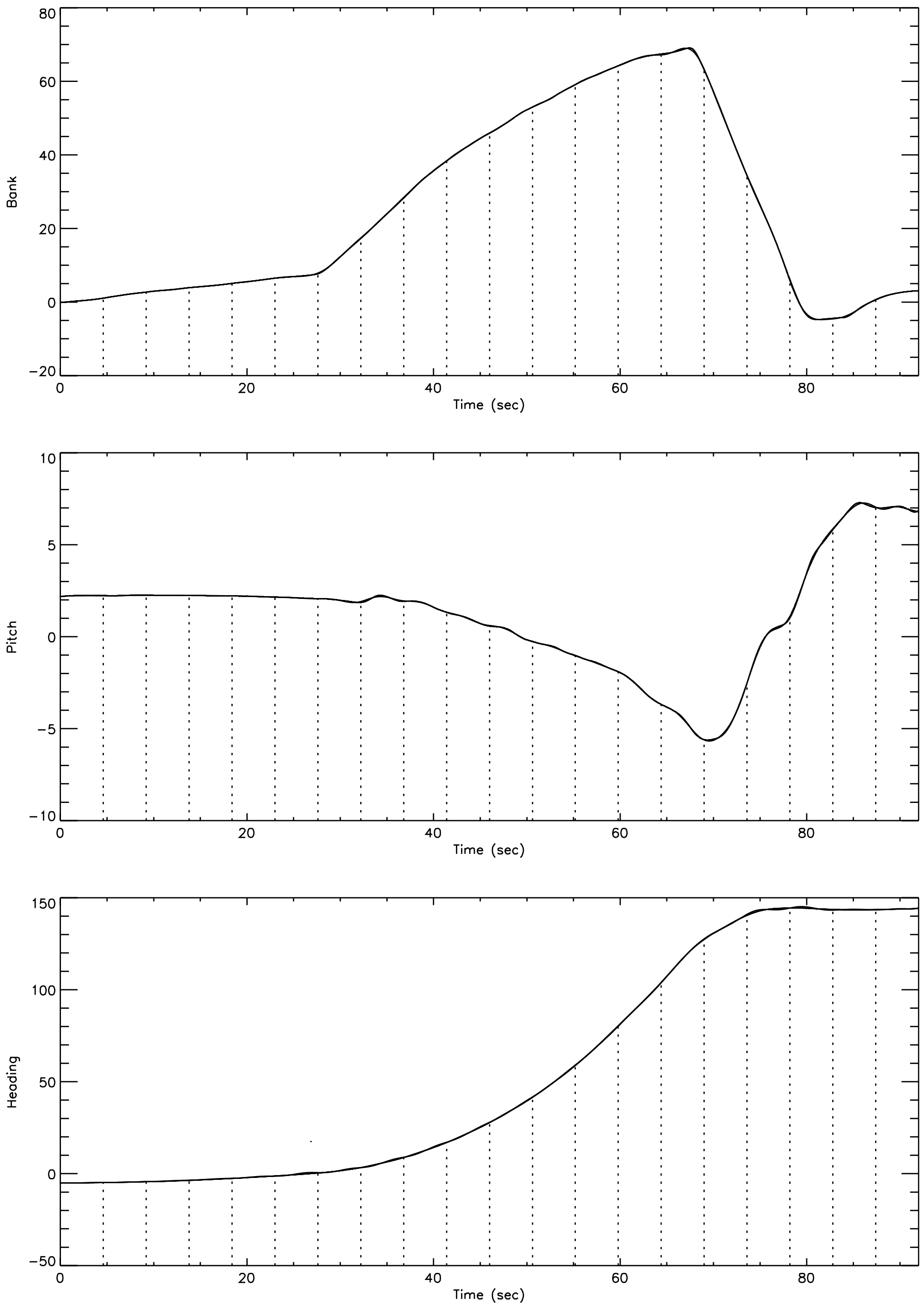


**Figure 5.6.** *Angular variable history.*

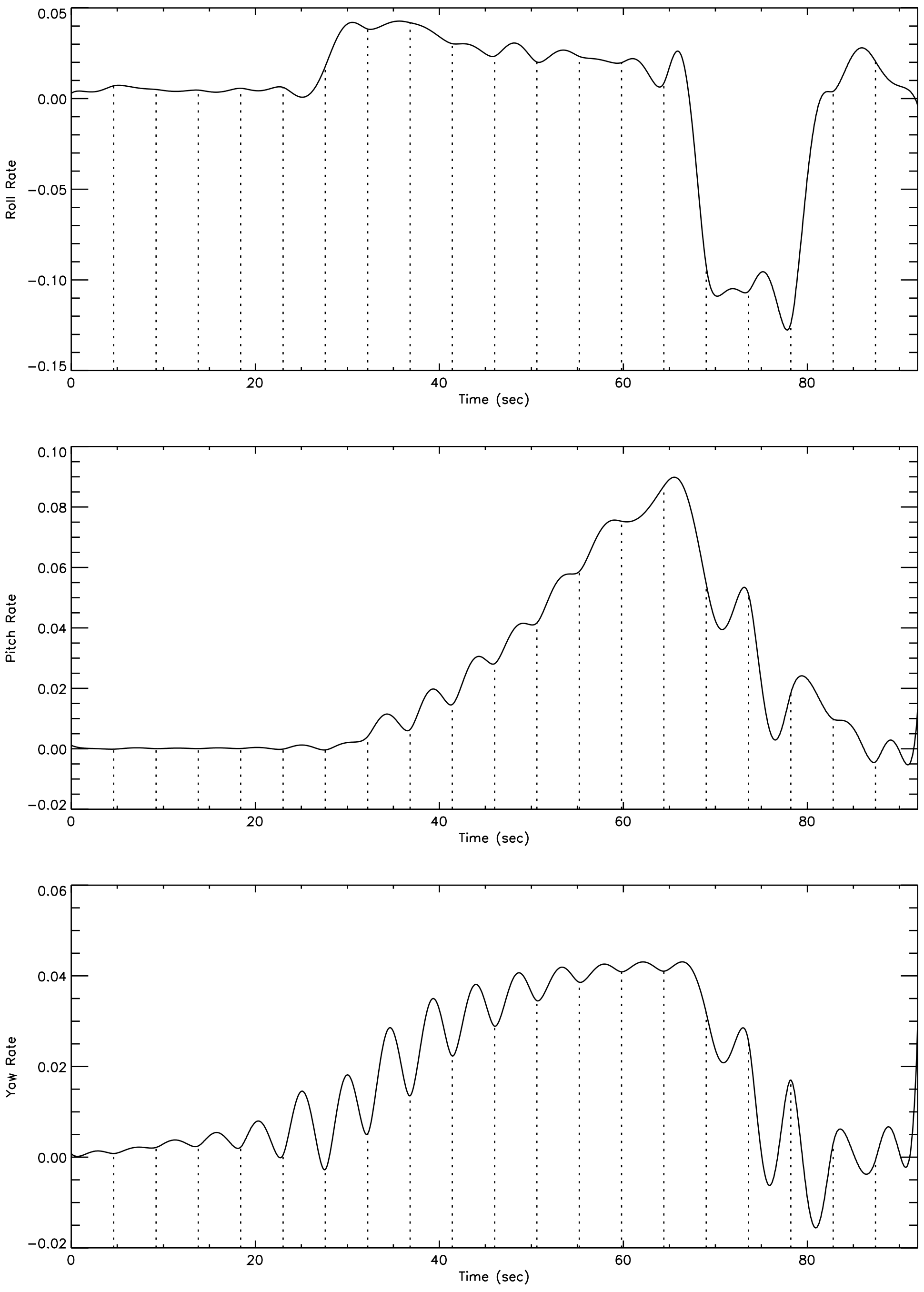


**Figure 5.7.** *Angular rate history.*

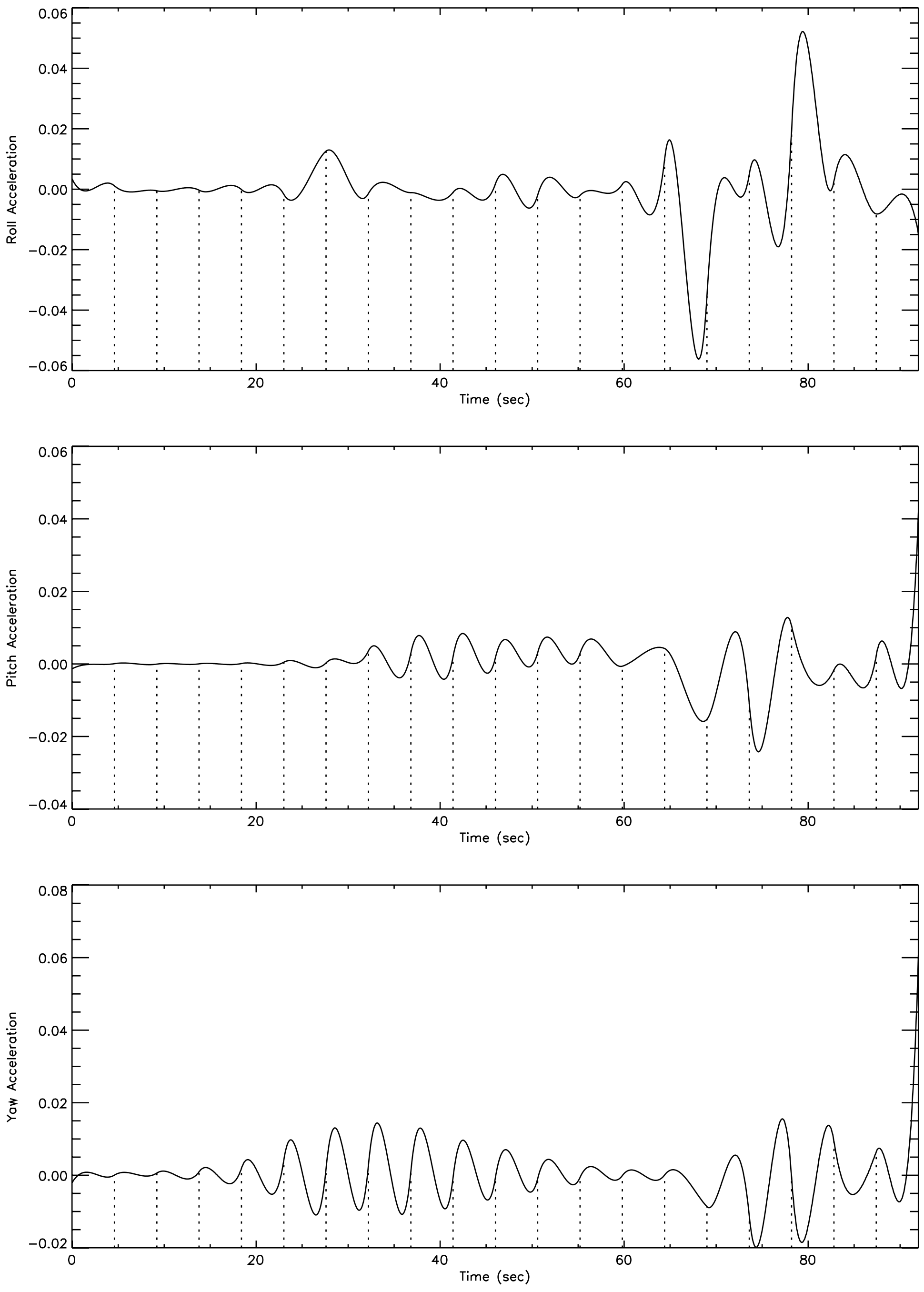


**Figure 5.8.** *Angular acceleration history.*

If the auxiliary functions in (5.5) are defined as

$$\mathbf{a}[\mathbf{v},t] = \begin{bmatrix} b_{p1}B_1(t) \\ b_{p2}B_2(t) \\ \vdots \\ b_{p42}B_{42}(t) \end{bmatrix}, \tag{5.81}$$

then path constraint (5.75) can be written as

$$g[\mathbf{y}(t),\mathbf{u}(t),\mathbf{p},t] = \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{v} + \boldsymbol{\beta}^{\mathsf{T}}\mathbf{a}[\mathbf{v},t], \tag{5.82}$$

with $\beta_j = -1$ for $j = 1,\ldots,42$, and $\alpha_7 = 1$ as the only nonzero element of $\boldsymbol{\alpha}$. Clearly (5.76) and (5.77) can be expressed in a similar fashion. When using a separated Simpson (HSS) discretization because separability is exploited both the B-spline and piecewise polynomial representation in Example 5.3 can compute finite difference derivative information using the same number of index sets ($\gamma = 5$).

**Example 5.5** ALGEBRAIC FUNCTION APPROXIMATION. The previous examples illustrate two different ways to represent the quantities $a_p(t)$, $a_q(t)$, and $a_r(t)$. Another alternative is to treat these quantities as differential rather than algebraic variables and introduce additional algebraic functions $b_p(t)$, $b_q(t)$, and $b_r(t)$. This approach leads to the system

$$\dot{\phi} = p + q\frac{\sin\phi\sin\theta}{\cos\theta} + r\frac{\cos\phi\sin\theta}{\cos\theta}, \tag{5.83}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi, \tag{5.84}$$

$$\dot{\psi} = q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta}, \tag{5.85}$$

$$\dot{p} = a_p(t), \tag{5.86}$$

$$\dot{q} = a_q(t), \tag{5.87}$$

$$\dot{r} = a_r(t), \tag{5.88}$$

$$\dot{a}_p = b_p(t), \tag{5.89}$$

$$\dot{a}_q = b_q(t), \tag{5.90}$$

$$\dot{a}_r = b_r(t). \tag{5.91}$$

Since the original variables correspond to "rates," the new variables can be viewed as "curvatures." As such it is reasonable to try to keep $b_p \sim b_q \sim b_r \sim 0$ and this can be achieved by including them in the objective, that is, minimize

$$F = \frac{1}{2}\sum_{k=1}^{N}\left[\frac{\phi_k - \hat{\phi}_k}{\sigma_1}\right]^2 + \left[\frac{\theta_k - \hat{\theta}_k}{\sigma_2}\right]^2 + \left[\frac{\psi_k - \hat{\psi}_k}{\sigma_3}\right]^2 + b_{pk}^2 + b_{qk}^2 + b_{rk}^2. \tag{5.92}$$

The formulations presented in Examples 5.3 and 5.4 share one key attribute. Specifically the number of parameters in $\mathbf{p}$ is finite. As the NLP problem size increases during the mesh-refinement procedure, each subproblem is well-posed. The number of degrees of freedom does not change as the mesh is refined.

In contrast, when additional algebraic functions $b_p(t)$, $b_q(t)$, and $b_r(t)$ are introduced, there are no parameters $\mathbf{p}$. When the algebraic functions $b(t)$ are discretized, the number of degrees of freedom can become arbitrarily large as the mesh is refined. In fact if the mesh size becomes sufficiently large, it may be possible to *interpolate* every single data point. This formulation is clearly ill-posed or at least inconsistent with the standard mesh-refinement philosophy. Thus when using this formulation, mesh refinement cannot be permitted!

## 5.6 Optimal Control or Optimal Estimation?

**Example 5.6** LINEAR TANGENT STEERING ESTIMATION. Chapter 4 was devoted to the optimal control problem, and this chapter addresses the optimal estimation problem. However, at times the distinction is not so clear cut, and so to highlight the issues, let us reconsider the *linear tangent steering* problem first presented in Example 4.1 and again in Example 4.5. The goal of this optimal control problem is to choose the steering angle $\beta(t)$ to minimize the final time subject to dynamics specified by

$$\dot{y}_1 = y_3, \tag{5.93}$$
$$\dot{y}_2 = y_4, \tag{5.94}$$
$$\dot{y}_3 = a\cos\beta, \tag{5.95}$$
$$\dot{y}_4 = a\sin\beta. \tag{5.96}$$

This optimal control problem has an analytic solution. Specifically the minimum time solution with initial conditions $y_1(0) = y_2(0) = y_3(0) = y_4(0) = 0$, final conditions $y_2(t^*) = 5$, $y_3(t^*) = 45 = ab_1/b_0$, $y_4(t^*) = 0$, and thrust $a = 100$ is given by [54, p. 82]

$$y_1^*(t) = \frac{a}{b_0^2}(b_2 - b_1\tan\beta), \tag{5.97}$$
$$y_2^*(t) = \frac{a}{2b_0^2}(b_3\sec\beta_0 - b_2\tan\beta - b_1), \tag{5.98}$$
$$y_3^*(t) = \frac{ab_1}{b_0}, \tag{5.99}$$
$$y_4^*(t) = \frac{ab_2}{b_0}, \tag{5.100}$$
$$\lambda_1^*(t) = 0, \tag{5.101}$$
$$\lambda_2^*(t) = -\frac{2\sin\beta_0}{at^*}, \tag{5.102}$$
$$\lambda_3^*(t) = -\frac{\cos\beta_0}{a}, \tag{5.103}$$
$$\lambda_4^*(t) = -\frac{\sin\beta_0}{a}\left(1 - \frac{2t}{t^*}\right), \tag{5.104}$$

where

$$\beta_0 = \left(\frac{\pi}{180}\right) 54.6263551908, \tag{5.105}$$

$$t^* = .554570878337, \tag{5.106}$$

$$b_0 = \frac{2}{t^*}\tan\beta_0, \tag{5.107}$$

$$\beta = \tan^{-1}(\tan\beta_0 - b_0 t), \tag{5.108}$$

$$b_1 = \log\left(\frac{\tan\beta_0 + \sec\beta_0}{\tan\beta + \sec\beta}\right), \tag{5.109}$$

$$b_2 = \sec\beta_0 - \sec\beta, \tag{5.110}$$

$$b_3 = \tan\beta_0 - \tan\beta. \tag{5.111}$$

Clearly the optimal control given by (5.108) is of the *linear tangent* form

$$\tan\beta^*(t) = \left[p_1 - p_2 t\right]. \tag{5.112}$$

But what if the problem is reversed? Suppose we fix the final time at $t^*$, omit the boundary conditions at $t = 0$ and $t = t^*$, and then try to find the control that best approximates the optimal state history. In this case the objective function is

$$F = \frac{1}{2}\sum_{k=1}^{N}[y_1(t_k) - \hat{y}_1(t_k)]^2 + [y_2(t_k) - \hat{y}_2(t_k)]^2 + [y_3(t_k) - \hat{y}_3(t_k)]^2 + [y_4(t_k) - \hat{y}_4(t_k)]^2 \tag{5.113}$$

with measurement times $0 \le t_k \le t^*$. Furthermore, suppose the measurements are given by

$$\hat{y}_j(t_k) = y_j^*(t_k) + \eta_j y_j^*(t_k) \tag{5.114}$$

for $j = 1,2,3,4$, with $y_j^*(t_k)$ given by (5.97)–(5.100). But what are the optimization variables for this example? One possibility is to treat the steering angle as a *function* of the two parameters $\mathbf{p} = (p_1, p_2)$, i.e.,

$$\boxed{\textit{Formulation LTP}} \qquad \beta(\mathbf{p},t) = \tan^{-1}\left[p_1 - p_2 t\right]. \tag{5.115}$$

A second alternative is to treat the steering angle as an algebraic variable, i.e.,

$$\boxed{\textit{Formulation OCP}} \qquad \beta(t) = u(t). \tag{5.116}$$

So what is the difference? Clearly, when $\eta_j = 0$ in (5.114) the two formulations must be the same. In fact this formulation is ideally suited for solution using a simple *shooting method* since the steering time history can be represented using only two unknowns. Of course a collocation method will produce equivalent results and does not require a priori knowledge of the linear tangent functional form for the control function $u(t)$.

**Table 5.4.** *Mesh-refinement summaries.*

Formulation LTP

| $k$ | $M$ | $\epsilon$ | $n_d$ | Levenberg |
|---|---|---|---|---|
| 1 | 10 | $7.07\times10^{-4}$ | 6 | 0 |
| 2 | 10 | $4.73\times10^{-5}$ | 6 | 0 |
| 3 | 19 | $3.03\times10^{-6}$ | 6 | 0 |
| 4 | 37 | $1.93\times10^{-7}$ | 6 | 0 |
| 5 | 73 | $1.20\times10^{-8}$ | 6 | 0 |

Formulation OCP

| $k$ | $M$ | $\epsilon$ | $n_d$ | Levenberg |
|---|---|---|---|---|
| 1 | 10 | $2.6443\times10^{-3}$ | 14 | $6.0\times10^{-6}$ |
| 2 | 19 | $2.2177\times10^{-3}$ | 23 | $2.6\times10^{-8}$ |
| 3 | 19 | $1.3419\times10^{-3}$ | 41 | $3.7\times10^{-8}$ |
| 4 | 37 | $7.0506\times10^{-4}$ | 75 | 0 |
| 5 | 51 | $4.2101\times10^{-4}$ | 92 | 0 |
| 6 | 60 | $8.1262\times10^{-5}$ | 96 | $7.4\times10^{-7}$ |
| 7 | 77 | $4.1914\times10^{-5}$ | 117 | $7.4\times10^{-7}$ |
| 8 | 103 | $8.0736\times10^{-6}$ | 163 | $1.5\times10^{-6}$ |
| 9 | 142 | $2.3558\times10^{-6}$ | 230 | $1.5\times10^{-6}$ |
| 10 | 175 | $6.1396\times10^{-7}$ | 292 | $1.5\times10^{-6}$ |
| 11 | 203 | $1.1786\times10^{-7}$ | 347 | $1.5\times10^{-6}$ |
| 12 | 247 | $3.8995\times10^{-8}$ | 431 | $1.5\times10^{-6}$ |

But what if $\eta_j \neq 0$? To illustrate let us assume $\eta_j$ are normally distributed random variables with mean zero and standard deviations $10^{-1}$, and let us evaluate the residuals at 10 equally spaced points between 0 and $t^*$. Table 5.4 summarizes how the $\mathbb{SOPE}$ algorithm performed for both formulations. When the linear tangent parameterization (LTP) (5.115) is used, five mesh-refinement iterations were needed to achieve the prescribed tolerance $\epsilon \leq 10^{-7}$. In contrast, when the optimal control formulation (OCP) (5.116) was used, 12 iterations were needed to achieve the same accuracy. Moreover, the LTP formulation required only 73 grid points, whereas the OCP approach needs 247 points. Furthermore a comparison of the final objective function values

$$F^*_{LTP} = 40.83222809,$$
$$F^*_{OCP} = 19.94180278$$

suggests that the OCP formulation is "better," since $F^*_{OCP} < F^*_{LTP}$. However, examination of the solutions as illustrated in Figure 5.9 suggests something entirely different. The solid line illustrates the state history obtained using the LTP parameterization and the dashed line shows the OCP solution. The exact solution $y^*_j(t)$ in (5.114) is shaded and the perturbed data points $\hat{y}_j(t_k)$ are shown with a "+." Clearly, the OCP solution is "closer" to the actual data points, which explains why the least squares objective is better. However, this behavior is achieved by constructing a control history that "follows the noise." Figure 5.10 illustrates the control history for both formulations, with the lower figure restricted to the range $.175 < t < .186$.

This behavior can be explained by the manner in which the steering angle is represented. When the linear tangent form (5.115) is used there are a fixed number of free

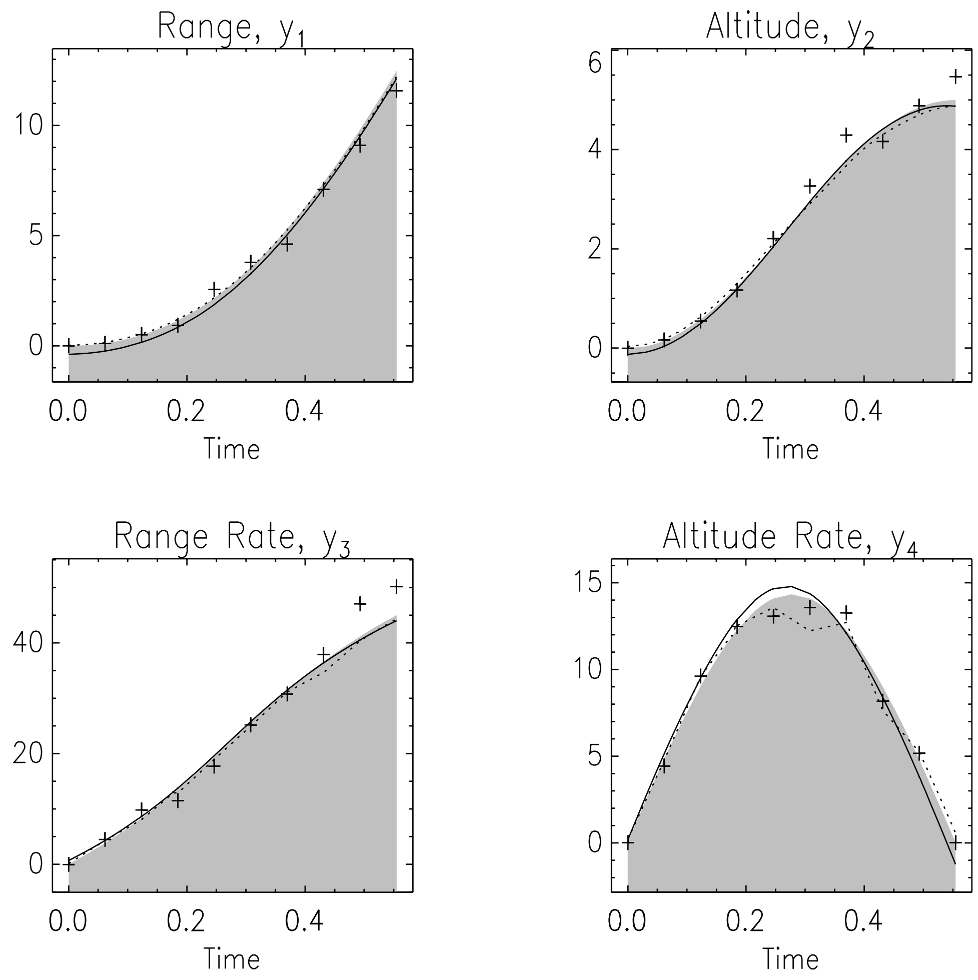


**Figure 5.9.** *Estimated state history.*

parameters (four initial conditions plus two parameters). For this formulation the number of degrees of freedom, $n_d$, remains fixed as indicated in Table 5.4. Since the underlying least squares NLP has a unique solution no Levenberg modification is required. In contrast the number of degrees of freedom grows from 14 to 431 as the mesh is refined in the OCP formulation and exceeds the number of residuals, $\ell = 40$. In this case the underlying NLP problem has no unique solution and a nonzero Levenberg parameter is introduced to compensate for the rank deficiency. In effect the linear tangent formulation is constructed from a finite-dimensional space, whereas the optimal control approximation is coming from an infinite-dimensional space.

This example suggests a more general property for a well-posed parameter estimation problem. Specifically, we expect that the number of degrees of freedom must reach a fixed, finite value as the discretization mesh is refined. In fact a linear analysis (cf. (2.57)) suggests the problem has no unique solution when the number of degrees of freedom $n_d$

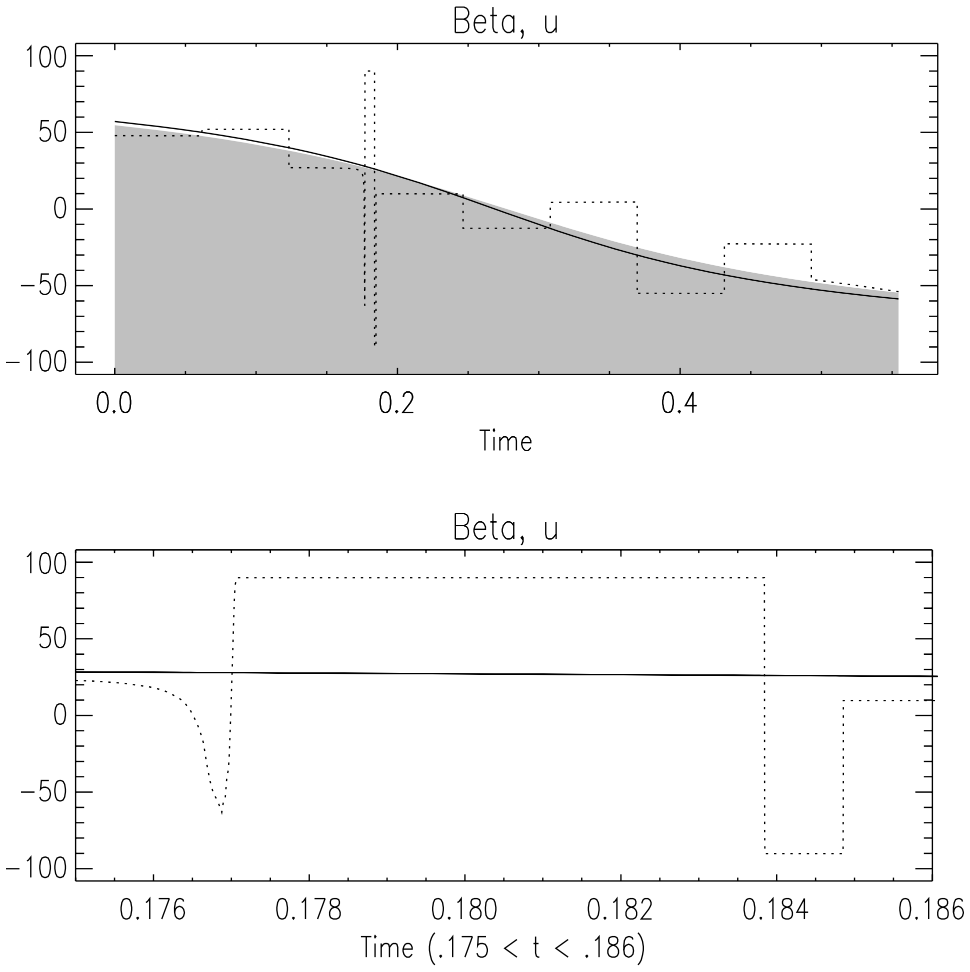


**Figure 5.10.** *Estimated control history.*

exceeds the number of residuals $\ell$. Thus if the parameterization is allowed to grow as the mesh size grows, we can expect to "chase the noise." Note that an *algebraic state* which is uniquely defined by a corresponding algebraic equation does not require a finite parameterization.

# Chapter 6

# Optimal Control Examples

## 6.1 Space Shuttle Reentry Trajectory

Construction of the reentry trajectory for the space shuttle is a classic example of an optimal control problem. The problem is of considerable practical interest and is nearly intractable using a simple shooting method because of its nonlinear behavior. Early results were presented by Bulirsch [55] on one version of the problem, as well by Dickmanns [73]. Ascher, Mattheij, and Russell present a similar problem [2, p. 23] and Brenan, Campbell, and Petzold discuss a closely related path control problem [48, p. 157]. Let us consider a particular variant of the problem originally described in [175].

The motion of the vehicle is defined by the following set of DAEs:

$$\dot{h} = v \sin\gamma, \tag{6.1}$$

$$\dot{\phi} = \frac{v}{r} \cos\gamma \sin\psi / \cos\theta, \tag{6.2}$$

$$\dot{\theta} = \frac{v}{r} \cos\gamma \cos\psi, \tag{6.3}$$

$$\dot{v} = -\frac{D}{m} - g \sin\gamma, \tag{6.4}$$

$$\dot{\gamma} = \frac{L}{mv} \cos(\beta) + \cos\gamma \left( \frac{v}{r} - \frac{g}{v} \right), \tag{6.5}$$

$$\dot{\psi} = \frac{1}{mv\cos\gamma} L \sin(\beta) + \frac{v}{r\cos\theta} \cos\gamma \sin\psi \sin\theta, \tag{6.6}$$

$$q \le q_U, \tag{6.7}$$

where the aerodynamic heating on the vehicle wing leading edge is $q = q_a q_r$ and the dynamic variables are

| | | | |
|---|---|---|---|
| $h$ | altitude (ft), | $\gamma$ | flight path angle (rad), |
| $\phi$ | longitude (rad), | $\psi$ | azimuth (rad), |
| $\theta$ | latitude (rad), | $\alpha$ | angle of attack (rad), |
| $v$ | velocity (ft/sec), | $\beta$ | bank angle (rad). |

For the sake of reference, the aerodynamic and atmospheric forces on the vehicle are specified by the following quantities (English units):

$$
\begin{aligned}
D &= \frac{1}{2} c_D S \rho v^2, & a_0 &= -0.20704,\\
L &= \frac{1}{2} c_L S \rho v^2, & a_1 &= 0.029244,\\
g &= \mu / r^2, & \mu &= 0.14076539 \times 10^{17},\\
r &= R_e + h, & b_0 &= 0.07854,\\
\rho &= \rho_0 \exp[-h/h_r], & b_1 &= -0.61592 \times 10^{-2},\\
\rho_0 &= 0.002378, & b_2 &= 0.621408 \times 10^{-3},\\
h_r &= 23800, & q_r &= 17700\sqrt{\rho}(0.0001v)^{3.07},\\
c_L &= a_0 + a_1 \hat{\alpha}, & q_a &= c_0 + c_1\hat{\alpha} + c_2\hat{\alpha}^2 + c_3\hat{\alpha}^3,\\
c_D &= b_0 + b_1\hat{\alpha} + b_2\hat{\alpha}^2, & c_0 &= 1.0672181,\\
\hat{\alpha} &= 180\alpha/\pi, & c_1 &= -0.19213774 \times 10^{-1},\\
R_e &= 20902900, & c_2 &= 0.21286289 \times 10^{-3},\\
S &= 2690, & c_3 &= -0.10117249 \times 10^{-5}.
\end{aligned}
$$

The reentry trajectory begins at an altitude where the aerodynamic forces are quite small with a weight of $w = 203000$ (lb) and mass $m = w/g_0$ (slug), where $g_0 = 32.174$ (ft/sec$^2$). The initial conditions are as follows:

$$
\begin{aligned}
h &= 260000 \text{ ft}, & v &= 25600 \text{ ft/sec},\\
\phi &= 0 \text{ deg}, & \gamma &= -1 \text{ deg},\\
\theta &= 0 \text{ deg}, & \psi &= 90 \text{ deg}.
\end{aligned}
$$

The final point on the reentry trajectory occurs at the unknown (free) time $t_F$, at the so-called terminal area energy management (TAEM) interface, which is defined by the conditions

$$
h = 80000 \text{ ft}, \qquad v = 2500 \text{ ft/sec}, \qquad \gamma = -5 \text{ deg}.
$$

To obtain realistic solutions, we also restrict the trajectory by defining the following simple bounds:

$$
\begin{aligned}
0 &\le h, & -89 \text{ deg} &\le \theta \le 89 \text{ deg},\\
1 &\le v, & -89 \text{ deg} &\le \gamma \le 89 \text{ deg},\\
-90 \text{ deg} \le \alpha &\le 90 \text{ deg}, & -89 \text{ deg} &\le \beta \le 1 \text{ deg}.
\end{aligned}
$$

**Example 6.1** MAXIMUM CROSSRANGE. The goal is to choose the control variables $\alpha(t)$ and $\beta(t)$ such that the final crossrange is maximized. There are many ways to define the crossrange, but for this case it is equivalent to maximizing the final latitude

$$
J = \theta(t_F). \tag{6.8}
$$

For comparison, the solution is computed with no limit on the aerodynamic heating, i.e.,

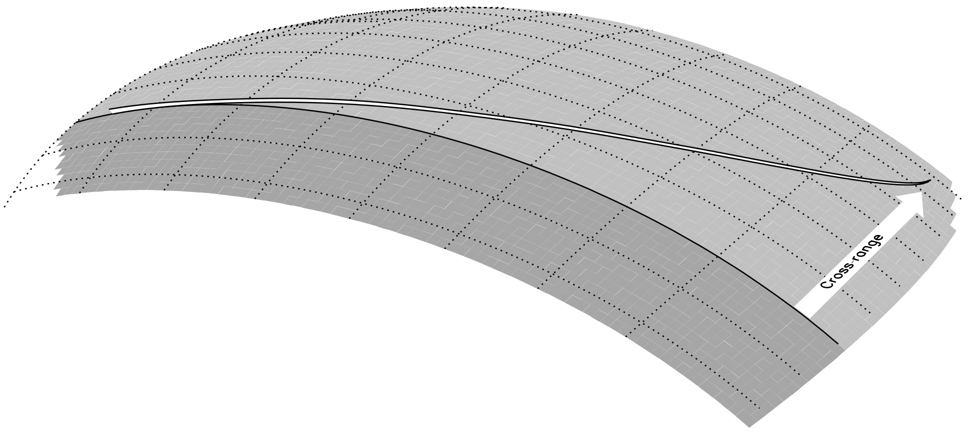


**Figure 6.1.** *Max crossrange shuttle reentry.*

$q_U = \infty$, and also with an upper bound on the aerodynamic heating of $q_U = 70$ BTU/ft$^2$/sec. Figure 6.1 illustrates the optimal trajectory when no heating limit is imposed.

The time histories for the state are shown in Figure 6.2, with the unconstrained solution shown as a solid line and the constrained solution as a dotted line. All of the angular quantities are given in degrees, the altitude in multiples of $10^5$ ft, velocity in multiples of $10^4$ ft/sec, and time in seconds. In Figure 6.3, the control time histories, as well as the aerodynamic heating, are illustrated. Note that for clarity the two solutions for angle of attack are plotted with different scales—the scale corresponding to the unconstrained solution is given on the right side of the figure. The optimal values for the final time and latitude are summarized in Table 6.1. For the heat-constrained example, Figure 6.4 illustrates the behavior of the $\mathbb{SOCS}$ mesh-refinement algorithm as the discretization error is reduced below the requested tolerance of $10^{-7}$. The first refinement iteration has the darkest shading and the last refinement has the lightest shading. For this case, using a linear initial guess between the boundary conditions, the solution was computed after 10 mesh-refinement iterations.

It is also interesting to ask whether mesh refinement is necessary. In order to assess the importance of mesh refinement, let us consider the following experiment. Suppose the mesh-refinement procedure is terminated after a specific number of iterations. Call the (prematurely obtained) solution $\widehat{\mathbf{u}}(t)$. This approximate "solution" can be used to propagate the trajectory, i.e., let us integrate the differential equations (6.1)–(6.6)

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \widehat{\mathbf{u}}, t) \tag{6.9}$$

from $t_I = 0$ to $t_F$. Let us assume the initial conditions are satisfied and assess the relative error in the terminal conditions, i.e.,

$$\epsilon = 100 \max_{k=1}^{n} \left[ \frac{|\hat{y}_k(t_F) - y_k(t_F)|}{\max(|\hat{y}_k(t_F)|, |y_k(t_F)|)} \right]. \tag{6.10}$$

In order to obtain an accurate solution of the IVP, we can use any sophisticated numerical integration software to compute the value of $\widehat{\mathbf{y}}(t_F)$. We have chosen to use a variable-order, variable-stepsize Gear [90] integrator with a relative error tolerance of $10^{-14}$. In essence,

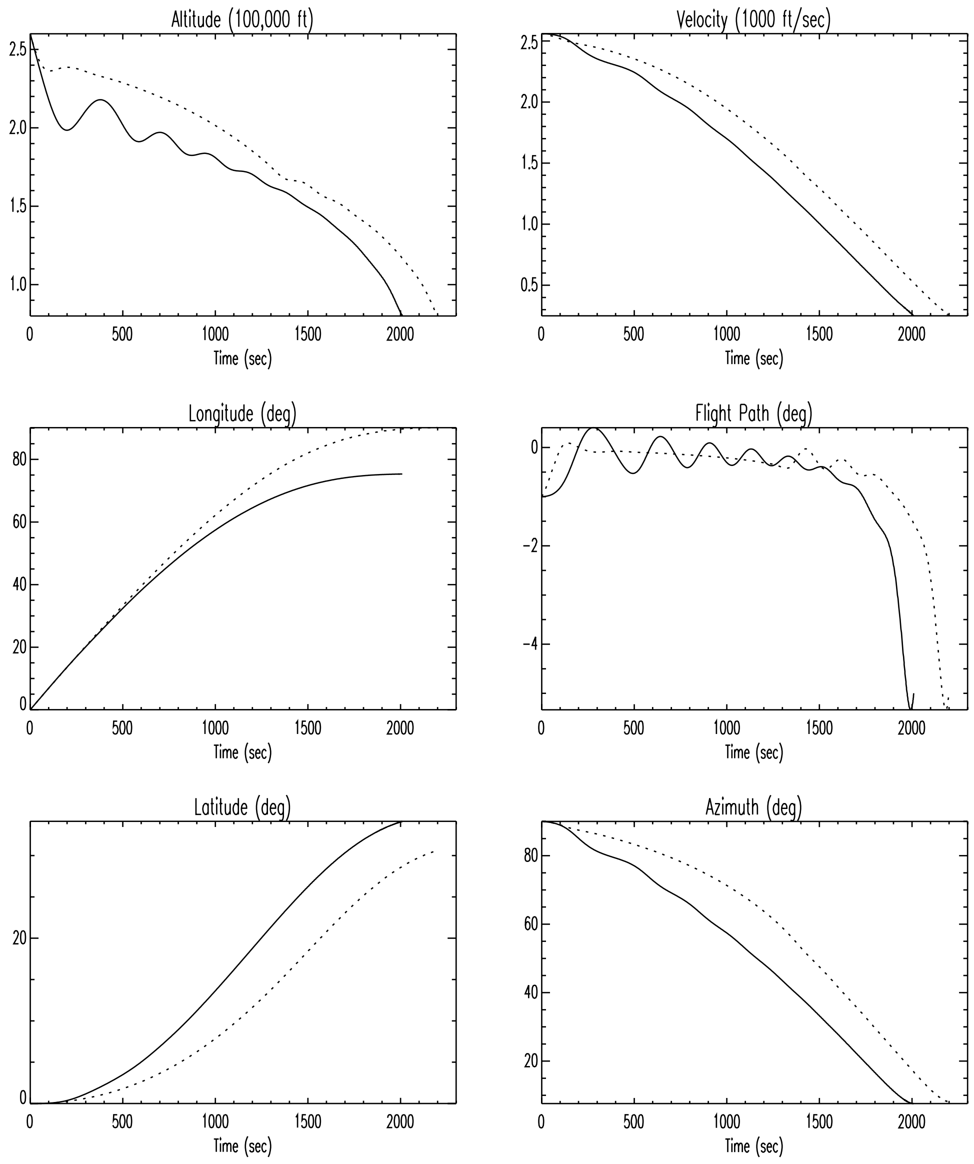


**Figure 6.2.** *Shuttle reentry—state variables.*

we are trying to assess how well the approximate solution $\widehat{\mathbf{u}}(t)$ to the BVP satisfies the corresponding IVP. Figure 6.5 summarizes the results of this experiment. Observe that when the approximate solution $\widehat{\mathbf{u}}(t)$ corresponds to the result after the first mesh-refinement iteration, the relative error in the integrated state is 46.5%. Thus, while the error in the objective function is only 0.4% (which might be reasonable for engineering purposes), the computed trajectory is totally unrealistic. In fact the final integrated trajectory with the approximate control $\widehat{\mathbf{u}}(t)$ has a position error of 12948.73 ft and the velocity error is 390.6111 ft/sec. Mesh refinement is crucial to obtain a realistic solution.

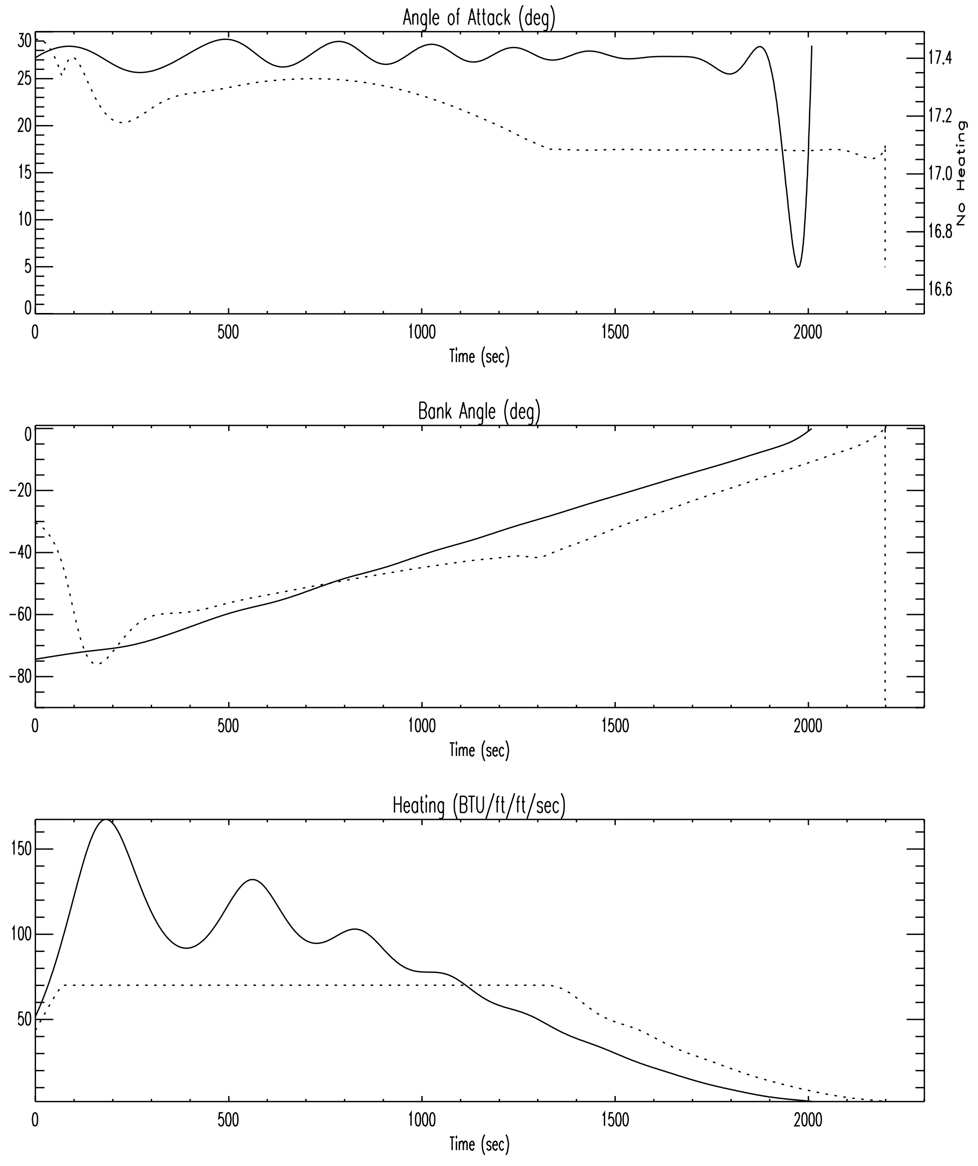


**Figure 6.3.** *Shuttle reentry—control variables.*

**Table 6.1.** *Shuttle reentry example.*

| $q_U$ | $\infty$ | 70 |
|---|---|---|
| $t_F$ (sec) | 2008.59 | 2198.67 |
| $\theta(t_F)$ (deg) | 34.1412 | 30.6255 |

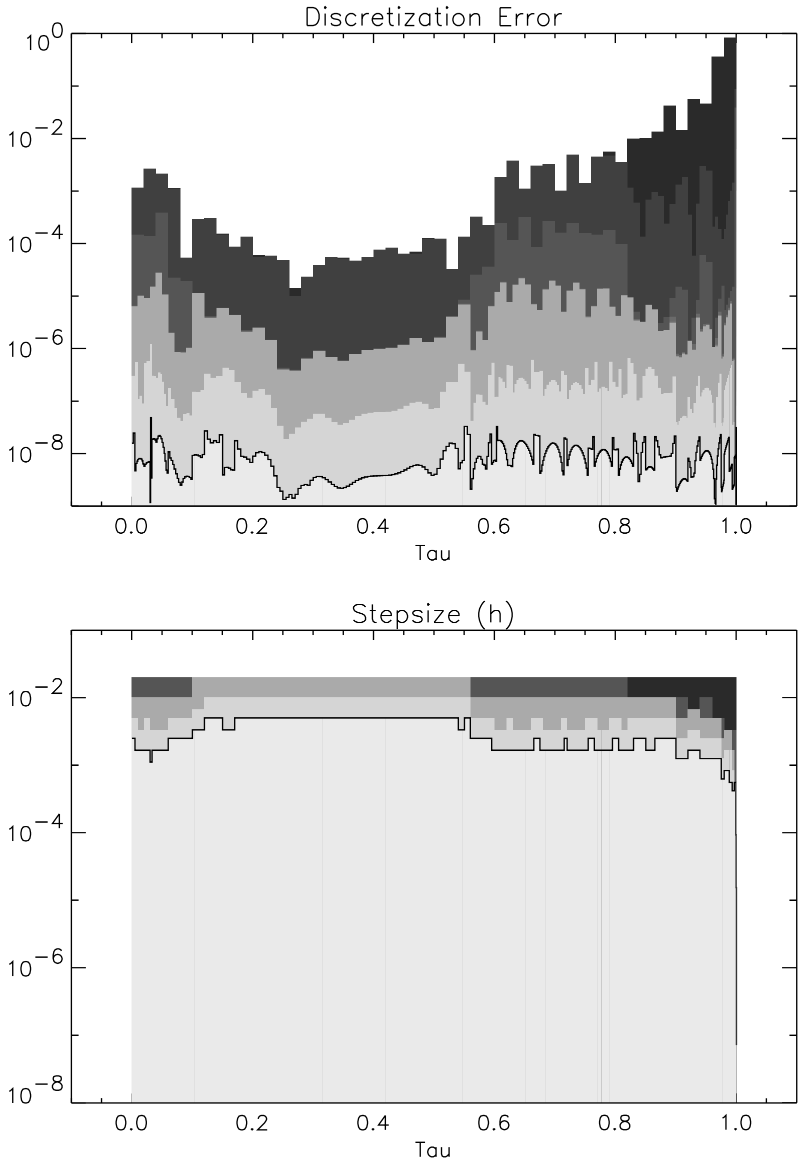


**Figure 6.4.** *Shuttle reentry—mesh refinement.*

**Example 6.2** MAX-CROSSRANGE ALTERNATE FORMULATION. It is interesting to note that the dynamics for Example 6.1 are independent of the longitude $\phi$. Observe that the longitude does not appear on the right-hand side of any of the differential equations (6.1)–(6.6). Physically this is not surprising since a spherical potential model is used to describe the earth. For convenience, in Example 6.1 the initial longitude was chosen to be $\phi = 0$. Another alternative is to simply eliminate the longitude as a state variable and the

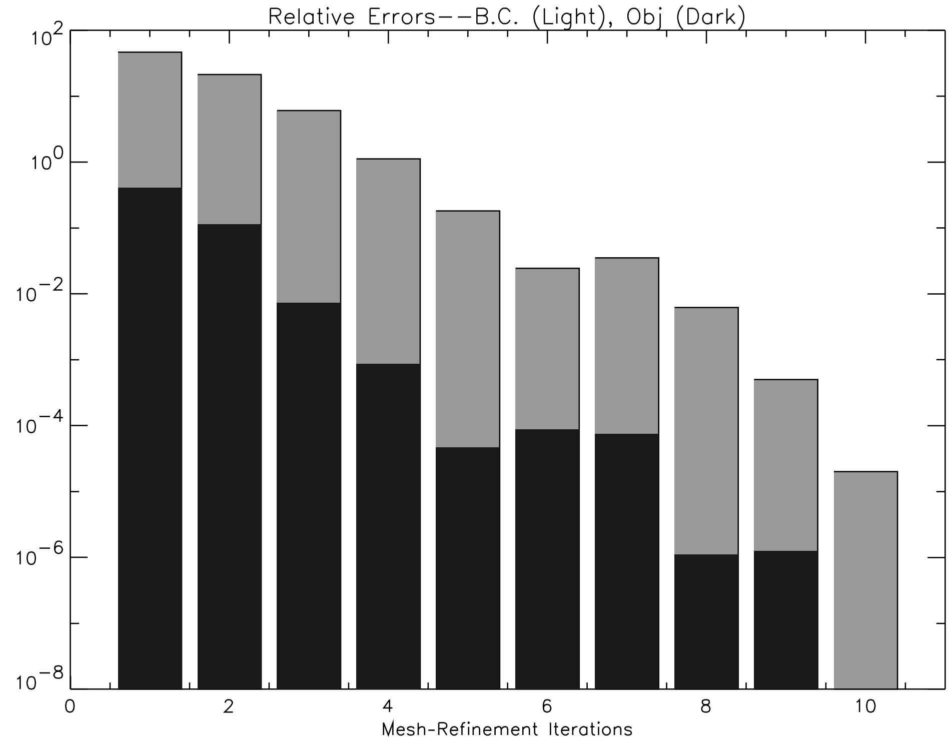


**Figure 6.5.** *Coarse grid solution errors.*

corresponding differential equation (6.2). Solutions to the problem using this formulation were presented in Section 4.13.

**Example 6.3** MINIMAX HEATING FORMULATION. Computing a reentry profile that maximizes the crossrange (6.8) is not the only criterion that can be used. As an alternative it may be desirable to minimize the peak value of the heating $q$. This can be achieved by treating the upper bound $q_U$ in (6.7) as an optimization variable. So instead of the objective (6.8) let us compute the control variables $\mathbf{u}(t)$ to minimize

$$J = q_U. \tag{6.11}$$

To preclude excessive oscillations in the solution let as also impose bounds on the rates $|\dot{\gamma}| \leq \varrho$ and $|\dot{\psi}| \leq \varsigma$ leading to the additional path constraints

$$-\varrho \leq \quad \frac{L}{mv}\cos(\beta) + \cos\gamma\left(\frac{v}{r} - \frac{g}{v}\right) \quad \leq \varrho, \tag{6.12}$$

$$-\varsigma \leq \quad \frac{1}{mv\cos\gamma}L\sin(\beta) + \frac{v}{r\cos\theta}\cos\gamma\sin\psi\sin\theta \quad \leq \varsigma, \tag{6.13}$$

where $\varrho = \varsigma = .2$ deg/sec. In contrast to Example 6.1, which maximized the final latitude, here let us consider three cases, with the fixed values $\theta(t_F) = 15, 20, 25$ deg. Table 6.2 summarizes the optimal solutions. Figure 6.6 illustrates the state variable history for each

**Table 6.2.** *Minimax heating reentry example.*

| $\theta(t_F)$ (deg) | $t_F$ (sec) | $q_U^*$ |
|---|---|---|
| 25 | 1994.44 | 49.8777 |
| 20 | 1714.81 | 38.0550 |
| 15 | 1390.80 | 27.9982 |

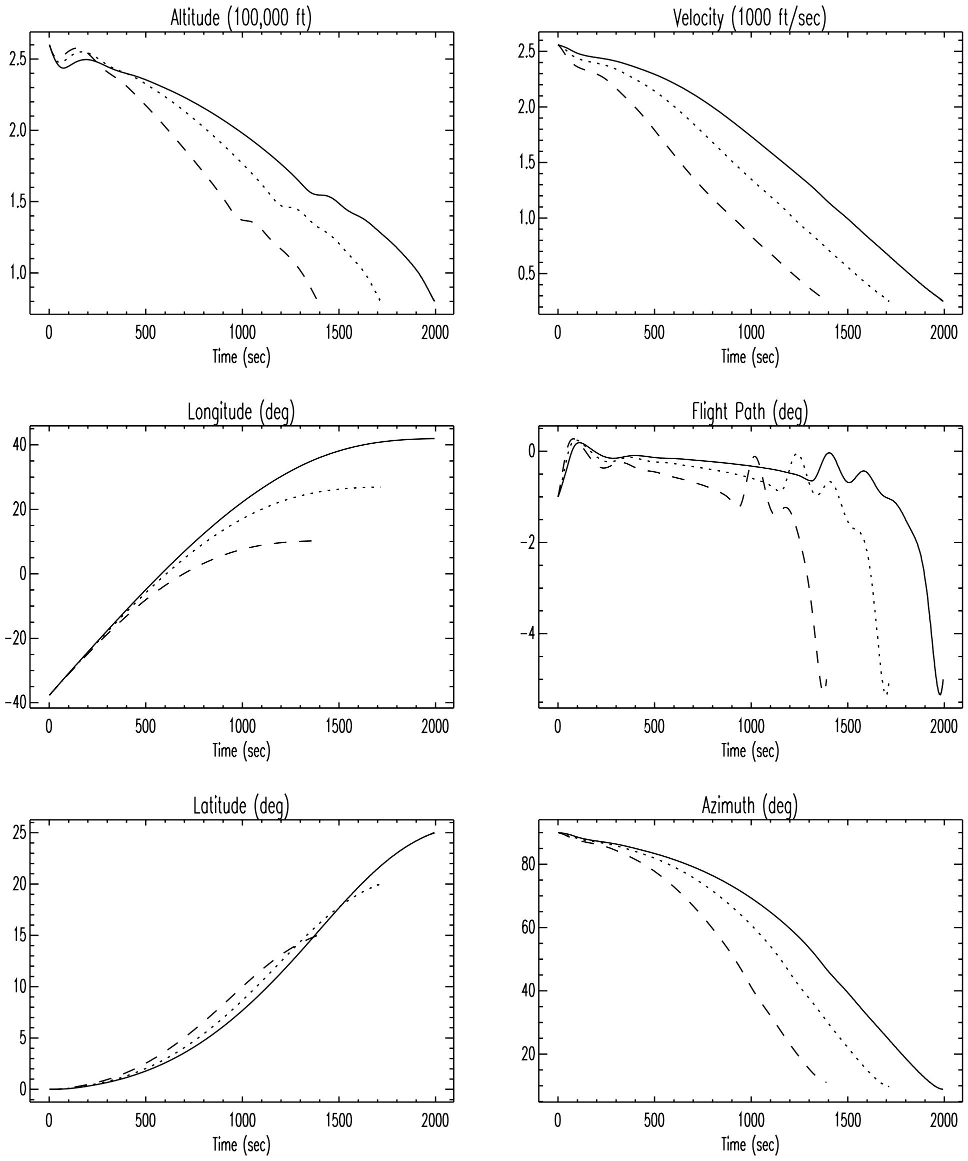


**Figure 6.6.** *Minimax heating shuttle reentry—state variables.*

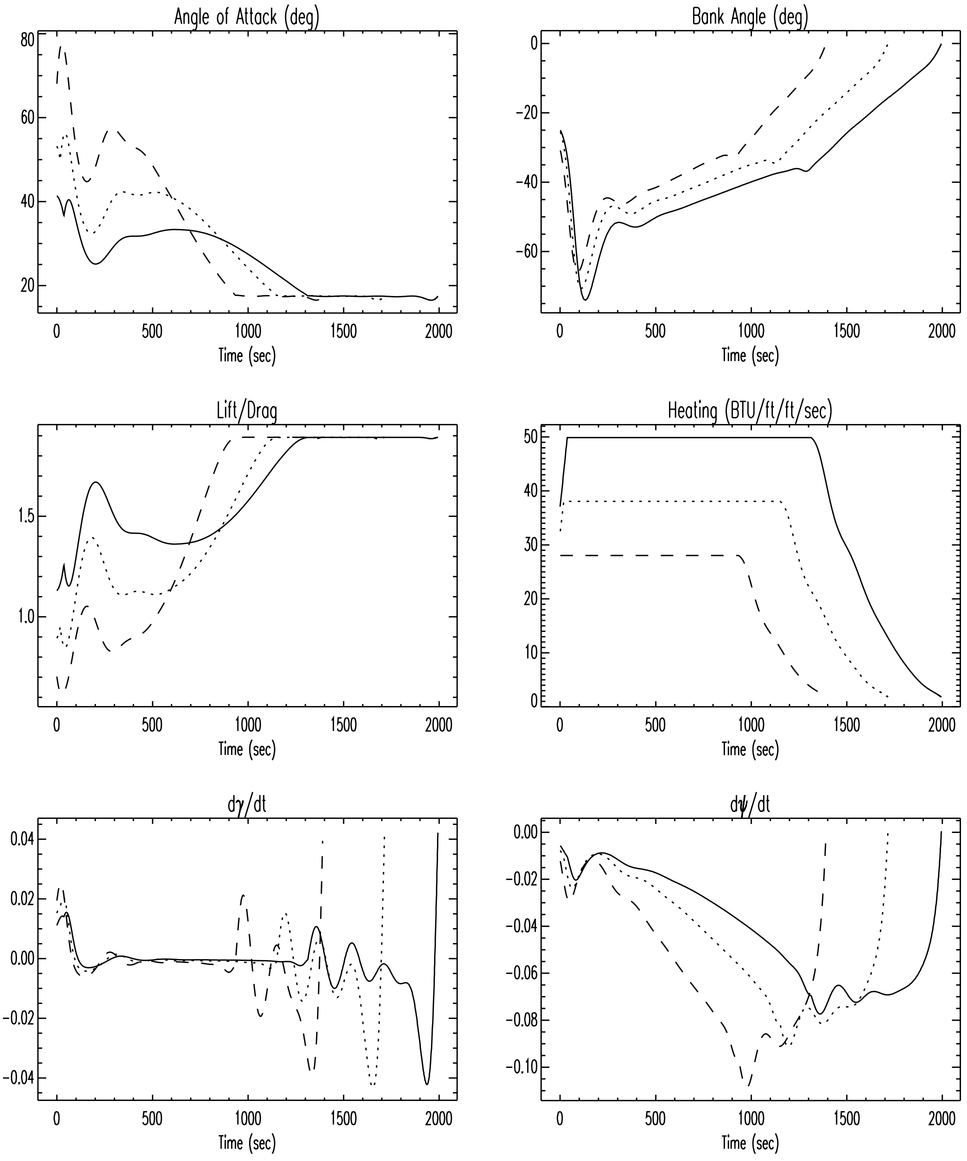


**Figure 6.7.** *Minimax heating shuttle reentry—control variables, path constraints.*

case, with a solid line used for the $\theta(t_F) = 25$ case, a dotted line used for $\theta(t_F) = 20$, and a dashed line for $\theta(t_F) = 15$. Using the same plot key, Figure 6.7 displays the optimal control and path constraint histories. One additional quantity, the lift to drag ratio $L/D$, is also displayed. Traditional engineering analysis suggests that aerodynamic efficiency is best when $L/D$ is maximized, and it is readily verified that for the aerodynamics given, the maximum value of the quantity

$$\frac{L}{D} = \frac{c_L}{c_D}$$

is $(L/D)^* = 1.89211$ which occurs when $\hat{\alpha} = 17.3919$ deg. After a transition period, the final portion of all three optimal trajectories utilizes a "max $L/D$" angle of attack.

## 6.2 Minimum Time to Climb

**Example 6.4** MINIMUM TIME TO CLIMB. The original minimum time to climb problem was presented by Bryson, Desai, and Hoffman [53] and has been the subject of many analyses since then. Although the problem is not nearly as difficult to solve as the shuttle reentry examples, it is included here because it illustrates the treatment of tabular data. The basic problem is to choose the optimal control function $\alpha(t)$ (the angle of attack) such that an airplane flies from a point on a runway to a specified final altitude as quickly as possible. In its simplest form, the planar motion of the aircraft is described by the following set of ODEs:

$$\dot{h} = v \sin\gamma, \tag{6.14}$$

$$\dot{v} = \frac{1}{m}[T(M,h)\cos\alpha - D] - \frac{\mu}{(R_e+h)^2}\sin\gamma, \tag{6.15}$$

$$\dot{\gamma} = \frac{1}{mv}[T(M,h)\sin\alpha + L] + \cos\gamma\left[\frac{v}{(R_e+h)} - \frac{\mu}{v(R_e+h)^2}\right], \tag{6.16}$$

$$\dot{w} = \frac{-T(M,h)}{I_{sp}}, \tag{6.17}$$

where $h$ is the altitude (ft), $v$ the velocity (ft/sec), $\gamma$ the flight path angle (rad), $w$ the weight (lb), $m = w/g_0$ the mass, $\mu$ the gravitational constant, and $R_e$ the radius of the earth. Furthermore, the simple bounds

$$\begin{aligned} 0 \le h \le 69000\,(\text{ft}), &\qquad 1 \le v \le 2000\,(\text{ft}),\\ -89\,(\text{deg}) \le \gamma \le 89\,(\text{deg}), &\qquad 0 \le w \le 45000\,(\text{lb}),\\ -20\,(\text{deg}) \le \alpha \le 20\,(\text{deg}) & \end{aligned}$$

are also imposed.

The aerodynamic forces on the vehicle are defined by the expressions

$$D = \frac{1}{2}C_D S\rho v^2, \tag{6.18}$$

$$L = \frac{1}{2}C_L S\rho v^2, \tag{6.19}$$

$$C_L = c_{L\alpha}(M)\alpha, \tag{6.20}$$

$$C_D = c_{D0}(M) + \eta(M)c_{L\alpha}(M)\alpha^2, \tag{6.21}$$

where $D$ is the drag, $L$ is the lift, $C_L$ and $C_D$ are the aerodynamic lift and drag coefficients, respectively, with $S$ the aerodynamic reference area of the vehicle, and $\rho$ is the atmospheric density. Although the results presented here use a cubic spline approximation to the 1962

Standard Atmosphere [46], qualitatively similar results can be achieved with a simple exponential approximation to $\rho(h)$ (cf. Example 6.1). The following constants complete the definition of the problem:

$$
\begin{array}{lcllcl}
h(0) & = & 0\ \text{(ft)}, & h(t_F) & = & 65600.0\ \text{(ft)}, \\
v(0) & = & 424.260\ \text{(ft/sec)}, & v(t_F) & = & 968.148\ \text{(ft/sec)}, \\
\gamma(0) & = & 0\ \text{(rad)}, & \gamma(t_F) & = & 0\ \text{(rad)}, \\
w(0) & = & 42000.0\ \text{(lb)}, & S & = & 530\ (\text{ft}^2), \\
I_{sp} & = & 1600.0\ \text{(sec)}, & \mu & = & 0.14076539 \times 10^{17}\ (\text{ft}^3/\text{sec}^2), \\
g_0 & = & 32.174\ (\text{ft/sec}^2), & R_e & = & 20902900\ \text{(ft)}.
\end{array}
$$

## 6.2.1 Tabular Data

As with most real aircraft, the aerodynamic and propulsive forces are specified in tabular form. For the sake of completeness, the data as they appeared in the original reference [53] are given in Tables 6.3 and 6.4. There are a number of significant points that characterize the *tabular data* representation. First, both the thrust and the aerodynamic table values are given to limited numeric precision (i.e., approximately two significant figures). Perhaps the most obvious explanation for the limited precision is that the data probably were originally obtained from experimental tests and truncated at the precision of the test equipment. Unfortunately, the statistical analysis (if any) of the original test data has long since been forgotten. Consequently, it is common to assume that the table values are "exact" and correct to full machine precision. A second difficulty, which is evident in the bivariate thrust data, is that apparently data are missing from the corners of the table. Of course, the data are "missing" because a real aircraft can never fly in these regimes (e.g., at Mach number

**Table 6.3.** *Propulsion data.*

Thrust $T(M,h)$ (thousands of lb)

| | Altitude $h$ (thousands of ft) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $M$ | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 40 | 50 | 70 |
| 0.0 | 24.2 | | | | | | | | | |
| 0.2 | 28.0 | 24.6 | 21.1 | 18.1 | 15.2 | 12.8 | 10.7 | | | |
| 0.4 | 28.3 | 25.2 | 21.9 | 18.7 | 15.9 | 13.4 | 11.2 | 7.3 | 4.4 | |
| 0.6 | 30.8 | 27.2 | 23.8 | 20.5 | 17.3 | 14.7 | 12.3 | 8.1 | 4.9 | |
| 0.8 | 34.5 | 30.3 | 26.6 | 23.2 | 19.8 | 16.8 | 14.1 | 9.4 | 5.6 | 1.1 |
| 1.0 | 37.9 | 34.3 | 30.4 | 26.8 | 23.3 | 19.8 | 16.8 | 11.2 | 6.8 | 1.4 |
| 1.2 | 36.1 | 38.0 | 34.9 | 31.3 | 27.3 | 23.6 | 20.1 | 13.4 | 8.3 | 1.7 |
| 1.4 | | 36.6 | 38.5 | 36.1 | 31.6 | 28.1 | 24.2 | 16.2 | 10.0 | 2.2 |
| 1.6 | | | | 38.7 | 35.7 | 32.0 | 28.1 | 19.3 | 11.9 | 2.9 |
| 1.8 | | | | | | 34.6 | 31.1 | 21.7 | 13.3 | 3.1 |

**Table 6.4.** *Aerodynamic data.*

| $M$ | 0 | 0.4 | 0.8 | 0.9 | 1.0 | 1.2 | 1.4 | 1.6 | 1.8 |
|---|---|---|---|---|---|---|---|---|---|
| $c_{L\alpha}$ | 3.44 | 3.44 | 3.44 | 3.58 | 4.44 | 3.44 | 3.01 | 2.86 | 2.44 |
| $c_{D0}$ | 0.013 | 0.013 | 0.013 | 0.014 | 0.031 | 0.041 | 0.039 | 0.036 | 0.035 |
| $\eta$ | 0.54 | 0.54 | 0.54 | 0.75 | 0.79 | 0.78 | 0.89 | 0.93 | 0.93 |

$M = 0$ and $h = 70000$ ft). In fact, for most experimentally obtained data, it can be expected that information will be missing for unrealistic portions of the domain. In view of these realities, it is common to linearly interpolate and never extrapolate the tabular data. Figure 6.8 illustrates a linear treatment of the thrust table.

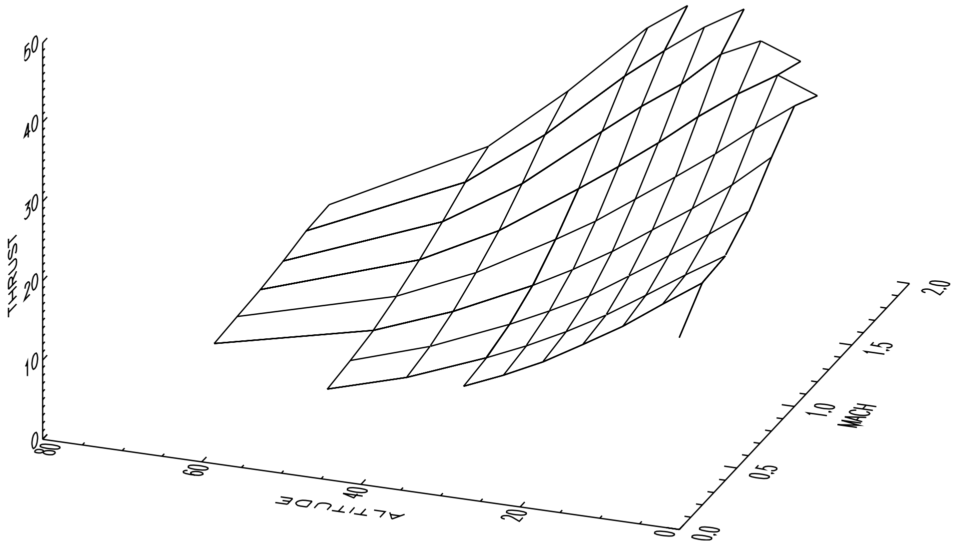


**Figure 6.8.** *Original thrust data.*

## 6.2.2 Cubic Spline Interpolation

While a linear treatment of a tabular function may be adequate for simply evaluating the functions, it is totally inappropriate if the functions are to be used within a trajectory simulation and/or optimization. The principle difficulty (as discussed in Section 1.16) stems from the fact that most numerical optimization and integration algorithms assume that the functions are continuously differentiable to at least second order. Thus, just to propagate the trajectory using an integration algorithm such as Runge–Kutta or Adams–Moulton, it is necessary that the right-hand side of the differential equations (6.14)–(6.17) have the necessary continuity. Although it is appealing to ignore a discontinuity, this is usually a poor idea (cf. [106, p. 196]). Similar requirements are imposed when a numerical optimization algorithm is used to shape the trajectory since the optimization uses second derivative (Hessian) information to construct estimates of the solution.

The most direct way to achieve the required continuity is to approximate the data by a tensor product cubic B-spline of the form

$$T(M,h) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} c_{i,j} B_i(M) B_j(h). \tag{6.22}$$

In order to use this approximation, it is necessary to compute the coefficients $c_{i,j}$. The simplest way to compute the spline coefficients is to force the approximating function to *interpolate* the data at all of the data points. However, for a unique interpolant, data are required at all points on a rectangular grid. Since data are missing in the corners of the domain, this difficulty is typically resolved by adding "fake" data in the corners using an "eyeball" approach. It is common to ignore the limited precision of the data and simply treat data as though they were of full precision. Finally, by using divided difference estimates of the derivatives at the boundary of the region, the spline coefficients are uniquely determined by solving a sparse system of linear equations. Figures 6.9 and 6.10 illustrate the cubic interpolating spline obtained using this approach.

### 6.2.3 Minimum Curvature Spline

The cubic spline interpolant does provide $C^2$ (second derivative) continuity as needed for proper behavior of integration and optimization algorithms. Unfortunately, the approximation, produced by simply interpolating the raw data, does not necessarily reflect the qualitative aspects of the data. In particular, it is common for the interpolant to introduce "wiggles" that are not actually present in the tabular data itself. This is clearly illustrated along the boundaries of the thrust surface in Figure 6.9 and especially in the aerodynamic data for $M \leq 0.8$ as shown in Figure 6.10. In the case of the latter, it is obvious that the interpolant does not reflect the fact that $\eta$ is constant for low Mach numbers. A second

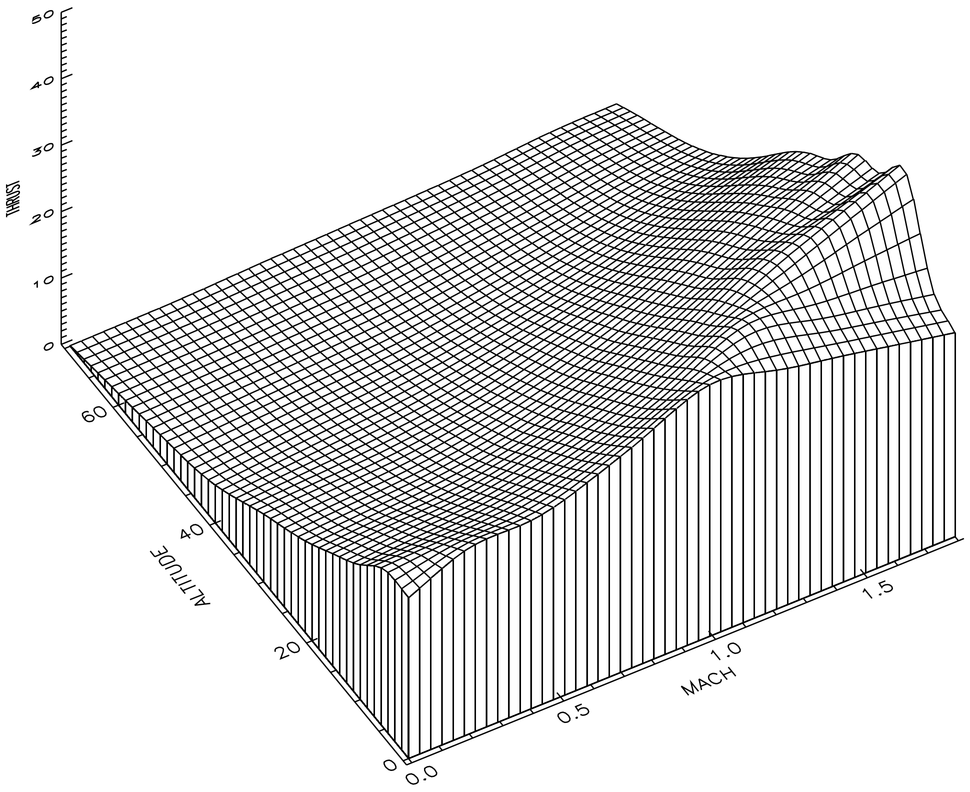


**Figure 6.9.** *Cubic spline interpolant for thrust data.*

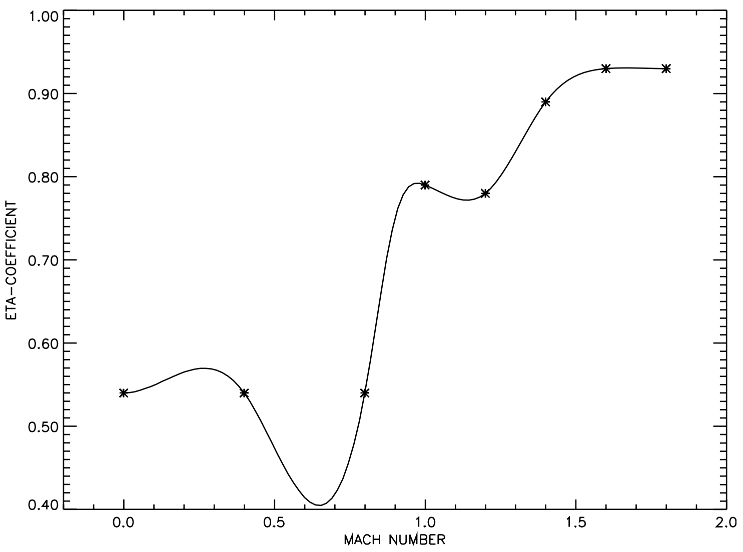


**Figure 6.10.** *Cubic spline interpolant for aerodynamic data.*

drawback of the cubic interpolant is the need for data at all points on a rectangular grid when constructing an approximation in more than one dimension.

A technique for eliminating the oscillations in the approximation is to carefully select the spline knot locations by inspection of the data, with fewer knots than data points. In this case, it is no longer possible to interpolate the data because the number of coefficients is less than the number of interpolation conditions; i.e., the system is overdetermined. However, it is reasonable to determine the spline coefficients $c_{i,j}$ such that

$$f(c_{i,j}) = \sum_{k=1}^{\ell} \left[T(M_k,h_k) - \hat{T}_k\right]^2 \tag{6.23}$$

is minimized. Furthermore, it is possible to introduce constraints on the slope of the spline approximation to reflect monotonicity of data, i.e.,

$$\left[\frac{\partial T}{\partial M}\right] \geq 0 \qquad \text{if } \left[\hat{T}_{k+1} - \hat{T}_k\right] \geq 0, \tag{6.24}$$

$$\left[\frac{\partial T}{\partial M}\right] \leq 0 \qquad \text{if } \left[\hat{T}_{k+1} - \hat{T}_k\right] \leq 0 \tag{6.25}$$

for $M \in [M_k, M_{k+1}]$. Similar constraints can be imposed to reflect monotonicity in the $h$-direction. The coefficients that satisfy these conditions can be determined by solving a sparse constrained linear least squares problem using the method described in Section 2.10.

By imposing monotonicity constraints and minimizing the error between the data and the approximating function, it is possible to achieve one of the major goals, namely constructing a $C^2$ function, which eliminates the wiggles. Unfortunately, the location of the knots in the spline approximation must be chosen such that the coefficients are *well* determined by minimizing the least square error. In fact, special care must be taken not to locate knots in regions where data are missing since this will result in a rank-deficient least squares problem. In essence, the knots must be located such that local constraint and data uniquely define the spline coefficients.

To resolve these deficiencies, we introduce a rectangular grid with $k_1$ values in the first coordinate and $k_2$ values in the second coordinate. We require that all data points lie on the rectangular grid. However, not all grid points need have data (i.e., data can be missing). Then let us consider introducing a spline with (a) double knots at the data points in order to ensure $C^2$ continuity and (b) single knots at the midpoint of each interval. Then let us determine the spline coefficients $c_{i,j}$ that minimize the "curvature"

$$f(c_{i,j}) = \sum_{k=1}^{L} \left[ \frac{\partial^2 T}{\partial M^2}(M_k, h_k) \right]^2 + \left[ \frac{\partial^2 T}{\partial h^2}(M_k, h_k) \right]^2 \tag{6.26}$$

and satisfy the data approximation constraints

$$\hat{T}_k - \epsilon \leq T(M_k, h_k) \leq \hat{T}_k + \epsilon \tag{6.27}$$

for all data points $k = 1, \dots, \ell$, where $\epsilon$ is the data precision. In order to ensure full rank in the Hessian of the objective, we evaluate the curvature at points determined by the knot-interlacing conditions [69]. We retain the slope constraints (6.24) and (6.25) on the approximation in order to reflect the monotonicity of the data. These coefficients can be determined by solving a sparse constrained linear least squares problem. The resulting approximations are illustrated in Figures 6.11 and 6.12. For this particular fit, the least squares problem had 900 variables with 988 constraints of which 77 were equalities. In

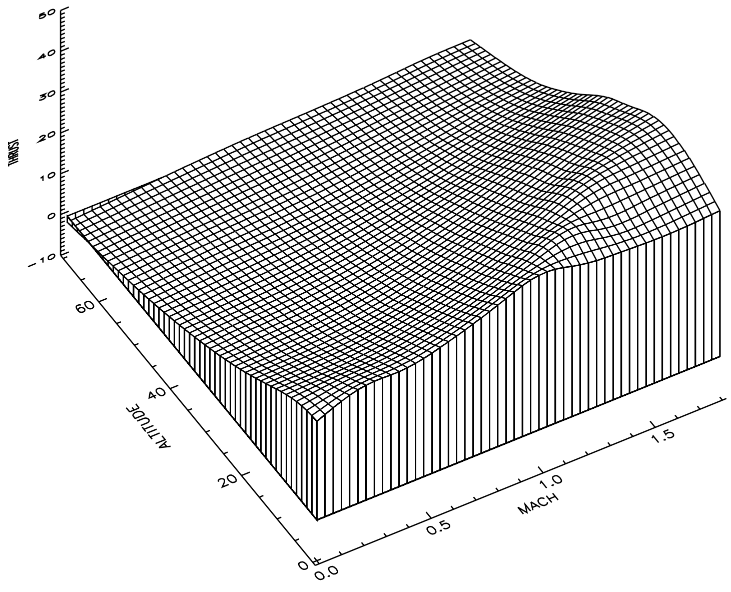


**Figure 6.11.** *Minimum curvature spline for thrust data.*

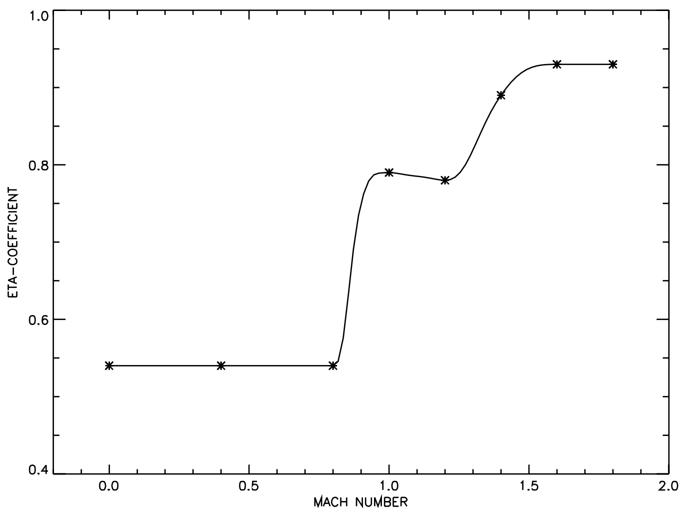


**Figure 6.12.** *Minimum curvature spline for aerodynamic data.*

general, the number of variables $n$ for a bivariate minimum curvature fit is $n = 9k_1k_2$. For the general case, the minimum curvature formulation requires imposition of $m$ constraints, where $10k_1k_2 \leq m \leq 14k_1k_2$ and the exact number depends on the number of algebraic sign changes in the slope of the data. In addition, if interpolation is required, the number of equality constraints is $m_e \leq k_1k_2$. The entire procedure described for this application has been automated for general multivariate functions with missing data as part of the $\mathbb{SOCS}$ software library.

## 6.2.4 Numerical Solution

Using the minimum curvature approximations for the tabular data, the minimum time to climb problem can be solved using the direct transcription algorithm in $\mathbb{SOCS}$. Table 6.5 summarizes the progress of the algorithm for this application using a linear initial guess for the dynamic variables. The first grid used a trapezoidal discretization (TR) with 10 grid points. The NLP problem was solved using 25 gradient evaluations (GE), 16 Hessian evaluations (HE), and a total of 523 function evaluations (FE) including the finite difference perturbations. The right-hand sides of the ODEs were evaluated 5230 times (NRHS) leading to a solution with a discretization error of $\epsilon_{\max} = 0.35$. Because the error was not sufficiently equidistributed, a second iteration using the trapezoidal discretization was performed. The HSS discretization (HS) was used for the third, fourth, and fifth refinement iterations, after which the HSC method (HC) was used for the remaining refinements. Figure 6.13 illustrates the progress of the mesh-refinement algorithm with the first refinement iteration shaded darkest and the last refinement shaded lightest. For this case, nine mesh-refinement iterations were required.

Figure 6.14 shows the solution with altitude in multiples of 10000 ft, velocity in multiples of 100 ft/sec, and weight in multiples of 10000 lb. The optimal (minimum) time for this trajectory is 324.9750302 (sec). The altitude time history demonstrates one

**Table 6.5.** *Minimum time to climb example.*

| Iter. | Disc. | $M$ | GE | HE | FE | NRHS | $\epsilon_{\max}$ | CPU (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | TR | 10 | 25 | 16 | 523 | 5230 | $0.35 \times 10^{0}$ | 2.8 |
| 2 | TR | 19 | 8 | 4 | 159 | 3021 | $0.68 \times 10^{-1}$ | 1.7 |
| 3 | HS | 19 | 8 | 5 | 174 | 6438 | $0.87 \times 10^{-2}$ | 3.4 |
| 4 | HS | 37 | 5 | 1 | 74 | 5402 | $0.51 \times 10^{-3}$ | 3.7 |
| 5 | HS | 59 | 4 | 1 | 61 | 7137 | $0.68 \times 10^{-4}$ | 7.5 |
| 6 | HC | 117 | 4 | 1 | 154 | 35882 | $0.12 \times 10^{-4}$ | 11. |
| 7 | HC | 179 | 4 | 1 | 154 | 54978 | $0.13 \times 10^{-5}$ | 16. |
| 8 | HC | 275 | 4 | 1 | 154 | 84546 | $0.14 \times 10^{-6}$ | 27. |
| 9 | HC | 285 | 3 | 1 | 129 | 73401 | $0.97 \times 10^{-7}$ | 22. |
| Total | - | - | 65 | 31 | 1582 | 276035 | - | 94.88 |

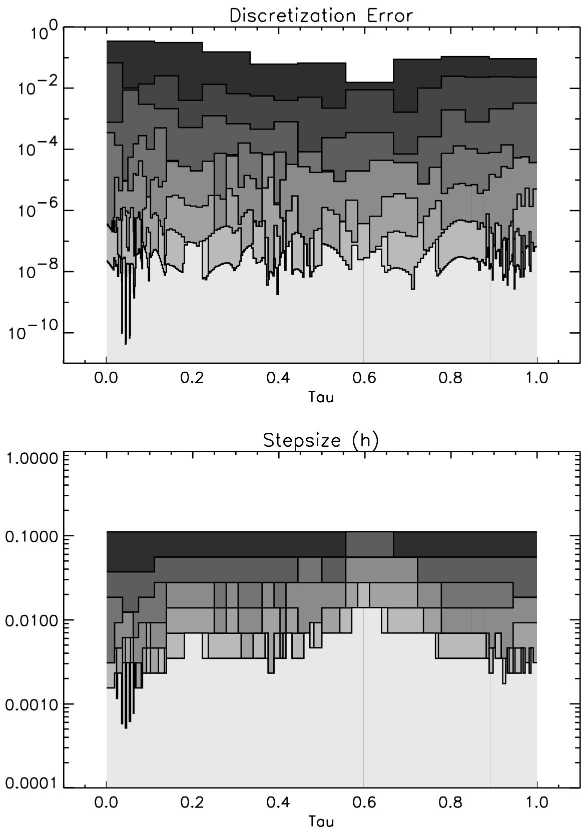


**Figure 6.13.** *Minimum time to climb—mesh refinement.*

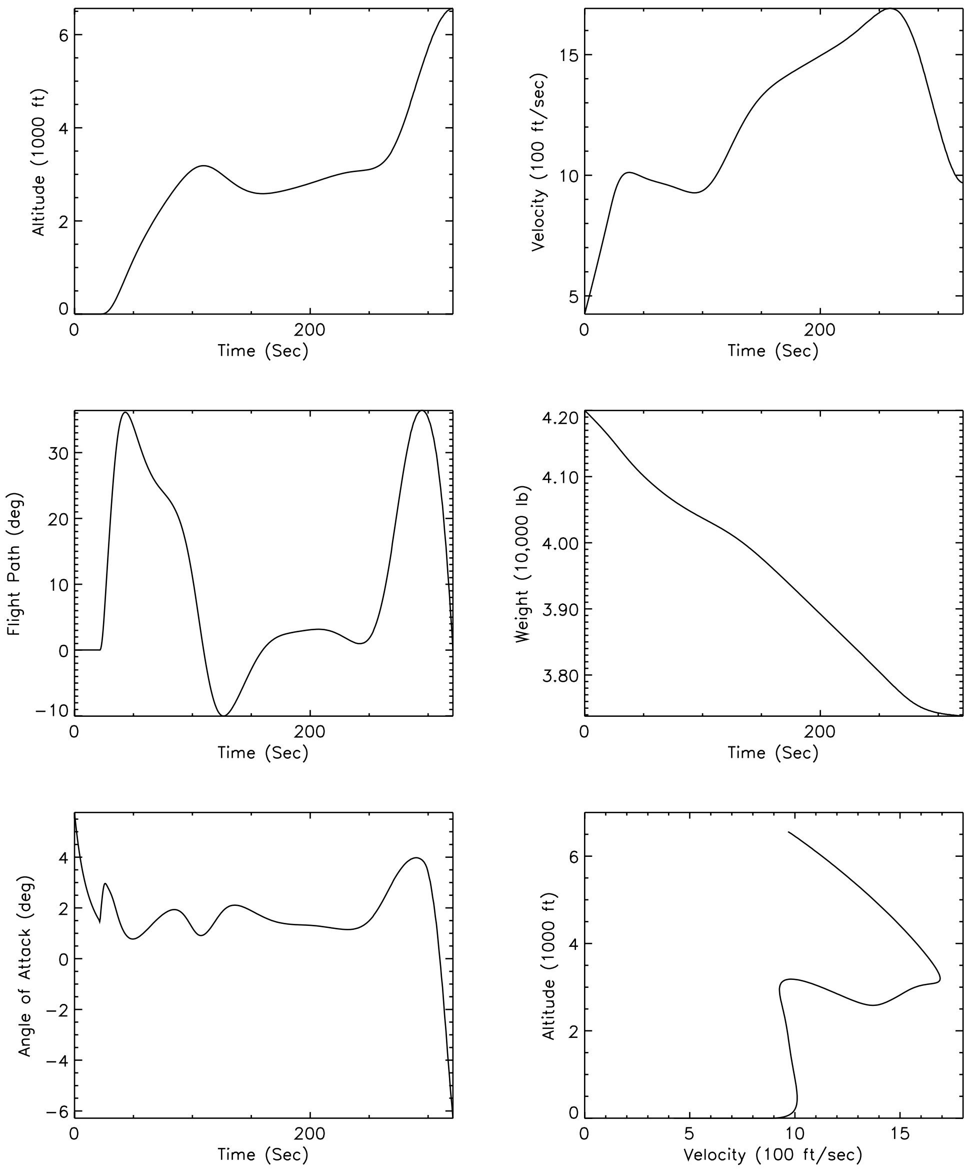


**Figure 6.14.** *Minimum time to climb solution.*

of the more amazing features of the optimal solution, namely the appearance of a *dive* midway through the minimum time to climb trajectory. When first presented in 1969, this unexpected behavior sparked considerable interest and led to the so-called energy-state approach to trajectory analysis. In particular, along the final portion of the trajectory, the energy is nearly constant, as illustrated in the plot of altitude versus velocity.

## 6.3 Low-Thrust Orbit Transfer

**Example 6.5** LOW-THRUST ORBIT TRANSFER. Constructing the trajectory for a spacecraft as it transfers from a low earth orbit to a mission orbit leads to a class of challenging optimal control examples. The dynamics are very nonlinear and, because the thrust applied to the vehicle is small in comparison to the weight of the spacecraft, the duration of the trajectory can be very long. Problems of this type have been of considerable interest in the aerospace industry [13, 15, 19, 20, 31, 74, 76, 150, 176]. Typically, the goal is to construct the optimal steering during the transfer such that the final weight is maximized (i.e., minimum fuel consumed).

The motion of a vehicle can be described by a system of second order ODEs

$$\ddot{\mathbf{r}} + \mu \frac{\mathbf{r}}{r^3} = \mathbf{a}_d, \tag{6.28}$$

where the radius $r = \|\mathbf{r}\|$ is the magnitude of the inertial position vector $\mathbf{r}$ and $\mu$ is the gravitational constant. In this formulation, we define the vector $\mathbf{a}_d$ as the *disturbing acceleration*. This representation for the equations of motion is referred to as the Gauss form of the variational equations.

The Gauss form of the equations of motion isolates the disturbing acceleration from the central force gravitational acceleration. Note that when the disturbing acceleration is zero, $\|\mathbf{a}_d\| = 0$, the fundamental system (6.28) is just a two-body problem. The solution of the two-body problem can, of course, be stated in terms of the constant orbital elements. For low-thrust trajectories, this formulation is appealing because we expect $\|\mathbf{a}_d\|$ to be "small" and, consequently, we expect that the solution can be described in terms of "almost constant" orbital elements. In order to exploit the benefits of the variational form of the differential equations (6.28), it is necessary to transform the Cartesian state into an appropriate set of orbit elements. One potential set contains the classical elements $(a, e, i, \Omega, \omega, M)$. However, these elements exhibit singularities for $e = 0$ and $i = 0$ deg or 90 deg. A set of *equinoctial* orbital elements that avoid the singularities in the classical elements has been described in [6], [49], and [74]. Kechichian developed a particular form of these equations in [122], [121], and [174]. These equations were used to solve a low-thrust earth orbit transfer problem as described in [13]. Unfortunately, this set of equinoctial elements does not accommodate orbits with $e \geq 1$. To eliminate this deficiency, a modified set of equinoctial orbit elements is described in [15] based on the work in [171].

### 6.3.1 Modified Equinoctial Coordinates

The dynamics of the system can be described in terms of the state variables

$$[\mathbf{y}^\mathsf{T}, w] = [p, f, g, h, k, L, w], \tag{6.29}$$

the control variables

$$\mathbf{u}^\mathsf{T} = [u_r, u_\theta, u_h], \tag{6.30}$$

and the unknown parameter $\tau$.

Using the modified equinoctial elements, the equations of motion for a vehicle with variable thrust can be stated as

$$\dot{\mathbf{y}} = \mathbf{A}(\mathbf{y})\boldsymbol{\Delta} + \mathbf{b}, \tag{6.31}$$

$$\dot{w} = -T\,[1 + 0.01\tau]/I_{sp}, \tag{6.32}$$

$$0 = \|\mathbf{u}\| - 1, \tag{6.33}$$

$$\tau_L \quad \le \tau \le \quad 0. \tag{6.34}$$

The equinoctial dynamics are defined by the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{2p}{q}\sqrt{\frac{p}{\mu}} & 0 \\ \sqrt{\frac{p}{\mu}}\sin L & \sqrt{\frac{p}{\mu}}\frac{1}{q}\{(q+1)\cos L + f\} & -\sqrt{\frac{p}{\mu}}\frac{g}{q}\{h\sin L - k\cos L\} \\ -\sqrt{\frac{p}{\mu}}\cos L & \sqrt{\frac{p}{\mu}}\frac{1}{q}\{(q+1)\sin L + g\} & \sqrt{\frac{p}{\mu}}\frac{f}{q}\{h\sin L - k\cos L\} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{s^2\cos L}{2q} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{s^2\sin L}{2q} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{1}{q}\{h\sin L - k\cos L\} \end{bmatrix} \tag{6.35}$$

and the vector

$$\mathbf{b}^{\mathsf{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \sqrt{\mu p}\left(\frac{q}{p}\right)^2 \end{bmatrix}, \tag{6.36}$$

where

$$q = 1 + f\cos L + g\sin L, \tag{6.37}$$

$$r = \frac{p}{q}, \tag{6.38}$$

$$\alpha^2 = h^2 - k^2, \tag{6.39}$$

$$\chi = \sqrt{h^2 + k^2}, \tag{6.40}$$

$$s^2 = 1 + \chi^2. \tag{6.41}$$

The equinoctial coordinates $\mathbf{y}$ are related to the Cartesian state $(\mathbf{r}, \mathbf{v})$ according to the expressions

$$\mathbf{r} = \begin{bmatrix} \frac{r}{s^2}\left(\cos L + \alpha^2\cos L + 2hk\sin L\right) \\ \frac{r}{s^2}\left(\sin L - \alpha^2\sin L + 2hk\cos L\right) \\ \frac{2r}{s^2}\left(h\sin L - k\cos L\right) \end{bmatrix}, \tag{6.42}$$

$$\mathbf{v} = \begin{bmatrix} -\frac{1}{s^2}\sqrt{\frac{\mu}{p}}\left(\sin L + \alpha^2\sin L - 2hk\cos L + g - 2fhk + \alpha^2 g\right) \\ -\frac{1}{s^2}\sqrt{\frac{\mu}{p}}\left(-\cos L + \alpha^2\cos L + 2hk\sin L - f + 2ghk + \alpha^2 f\right) \\ \frac{2}{s^2}\sqrt{\frac{\mu}{p}}\left(h\cos L + k\sin L + fh + gk\right) \end{bmatrix}. \tag{6.43}$$

As a result of this transformation, the disturbing acceleration vector $\mathbf{a}_d$ in (6.28) is replaced by

$$\boldsymbol{\Delta} = \boldsymbol{\Delta}_g + \boldsymbol{\Delta}_T \tag{6.44}$$

with a contribution due to oblate earth effects $\boldsymbol{\Delta}_g$ and another caused by thrust $\boldsymbol{\Delta}_T$. The disturbing acceleration is expressed in a rotating radial frame whose principle axes are defined by

$$\mathbf{Q}_r = \begin{bmatrix} \mathbf{i}_r & \mathbf{i}_\theta & \mathbf{i}_h \end{bmatrix} = \begin{bmatrix} \frac{\mathbf{r}}{\|\mathbf{r}\|} & \frac{(\mathbf{r}\times\mathbf{v})\times\mathbf{r}}{\|\mathbf{r}\times\mathbf{v}\|\|\mathbf{r}\|} & \frac{\mathbf{r}\times\mathbf{v}}{\|\mathbf{r}\times\mathbf{v}\|} \end{bmatrix}. \tag{6.45}$$

As stated, (6.31)–(6.34) are perfectly general and describe the motion of a point mass when subject to the disturbing acceleration vector $\boldsymbol{\Delta}$. Notice that when the disturbing acceleration is zero, $\boldsymbol{\Delta} = 0$, the first five equations are simply $\dot{p} = \dot{f} = \dot{g} = \dot{h} = \dot{k} = 0$, which implies that the elements are constant. It is important to note that the disturbing acceleration vector can be attributed to any perturbing force(s). A more complete derivation of the equinoctial dynamics can be found in [15].

### 6.3.2 Gravitational Disturbing Acceleration

Oblate gravity models are typically defined in a local horizontal reference frame, that is,

$$\boldsymbol{\delta}\mathbf{g} = \delta g_n \mathbf{i}_n - \delta g_r \mathbf{i}_r, \tag{6.46}$$

where

$$\mathbf{i}_n = \frac{\mathbf{e}_n - (\mathbf{e}_n^\top \mathbf{i}_r)\mathbf{i}_r}{\|\mathbf{e}_n - (\mathbf{e}_n^\top \mathbf{i}_r)\mathbf{i}_r\|} \tag{6.47}$$

defines the local north direction with $\mathbf{e}_n = (0,0,1)$. A reasonably accurate model is obtained if the tesseral harmonics are ignored and only the first four zonal harmonics are included in the geopotential function. In this case, the oblate earth perturbations to the gravitational acceleration are given by

$$\delta g_n = -\frac{\mu \cos\phi}{r^2} \sum_{k=2}^{4} \left(\frac{R_e}{r}\right)^k P_k' J_k, \tag{6.48}$$

$$\delta g_r = -\frac{\mu}{r^2} \sum_{k=2}^{4} (k+1) \left(\frac{R_e}{r}\right)^k P_k J_k, \tag{6.49}$$

where $\phi$ is the geocentric latitude, $R_e$ is the equatorial radius of the earth, $P_k(\sin\phi)$ is the $k$th-order Legendre polynomial with corresponding derivative $P_k'$, and $J_k$ are the zonal harmonic coefficients. Finally, to obtain the gravitational perturbations in the rotating radial frame, it follows that

$$\boldsymbol{\Delta}_g = \mathbf{Q}_r^\top \boldsymbol{\delta}\mathbf{g}. \tag{6.50}$$

### 6.3.3 Thrust Acceleration—Burn Arcs

To this point, the discussion has concentrated on incorporating perturbing forces due to oblate earth effects. Of course, the second major perturbation is the thrust acceleration defined by

$$\boldsymbol{\Delta}_T = \frac{g_o T[1 + .01\tau]}{w}\mathbf{u}, \tag{6.51}$$

where $T$ is the maximum thrust and $\tau_L \le \tau \le 0$ is a throttle factor. In general, the direction of the thrust acceleration vector, which is defined by the time-varying control vector $\mathbf{u}(t) = (u_r, u_\theta, u_h)$, can be chosen arbitrarily as long as the vector has unit length at all points in time. This is achieved using the path constraint (6.33). The magnitude of the thrust is, of course, related to the vehicle weight according to (6.32), where $g_0$ is the mass to weight conversion factor and the specific impulse of the motor is denoted by $I_{sp}$. Defining the thrust direction using the vector $\mathbf{u}(t)$ and path constraint $\|\mathbf{u}(t)\| = 1$ is particularly well suited for missions that involve steering over large portions of the trajectory, as illustrated in [13], because ambiguities in the pointing direction are avoided. Specifying the thrust direction by two angles (e.g., yaw and pitch), which are treated as control variables, is not unique since the angles $\alpha = \alpha_0 \pm 2k\pi$ all yield the *same* direction. In contrast, there is a *unique* set of control variables $\mathbf{u}$ corresponding to any thrust direction. This ambiguity is demonstrated in Figure 6.15.

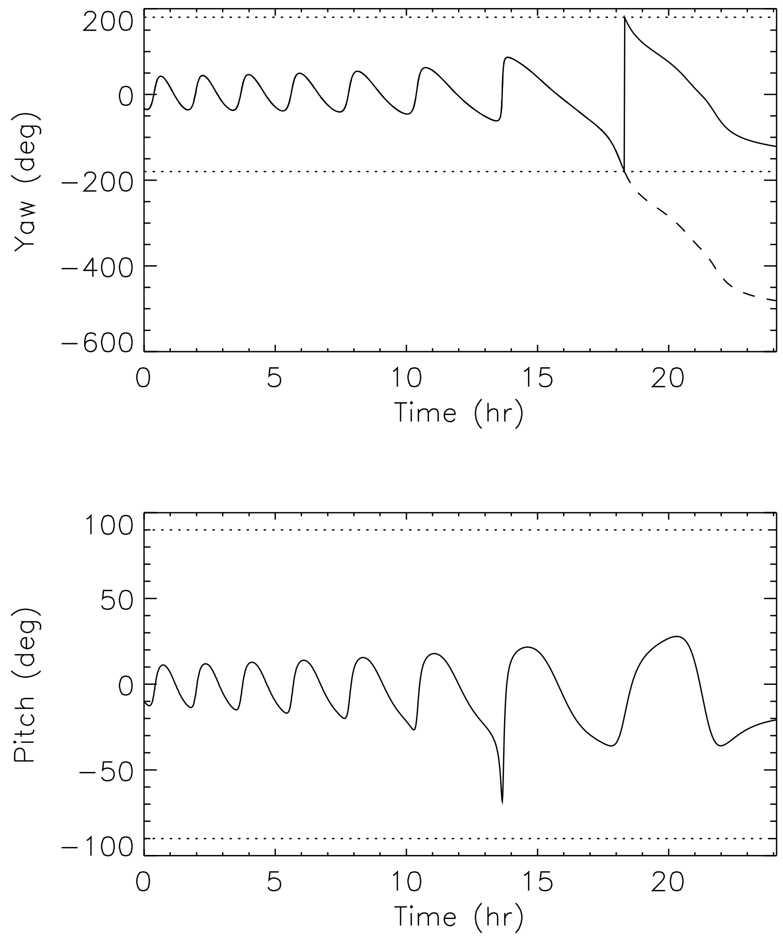


**Figure 6.15.** *Optimal control angles.*

### 6.3.4 Boundary Conditions

The standard approach for defining the boundary conditions of an orbit transfer problem is to specify the final state in terms of the instantaneous or *osculating* orbit elements at the burnout time $t_F$. The final orbit for this example has an apogee altitude of 21450 nm, a perigee altitude of 350 nm, an inclination of 63.4 deg, and an argument of perigee of 270 deg. This final orbit can be defined by the boundary conditions (all evaluated at $t = t_F$)

$$p = 40007346.015232 \text{ (ft)}, \tag{6.52}$$

$$\sqrt{f^2 + g^2} = 0.73550320568829, \tag{6.53}$$

$$\sqrt{h^2 + k^2} = 0.61761258786099, \tag{6.54}$$

$$fh + gk = 0, \tag{6.55}$$

$$gh - kf \le 0. \tag{6.56}$$

The initial orbit for this example is a "standard" space shuttle park orbit, which is circular at an altitude of 150 nm, and an inclination of 28.5 deg. This particular orbit leads to the following values for the equinoctial elements at $t = 0$:

$$\begin{array}{lll} p = 21837080.052835 \text{ (ft)}, & f = 0, & g = 0, \\ h = -0.25396764647494, & k = 0, & L = \pi \text{ (rad)}. \end{array}$$

The following constants complete the definition of the problem:

$$\begin{array}{ll} w(0) = 1 \text{ (lb)}, & g_0 = 32.174 \text{ (ft/sec}^2\text{)}, \\ I_{sp} = 450 \text{ (sec)}, & T = 4.446618 \times 10^{-3} \text{ (lb)}, \\ \mu = 1.407645794 \times 10^{16} \text{ (ft}^3\text{/sec}^2\text{)}, & R_e = 20925662.73 \text{ (ft)}, \\ J_2 = 1082.639 \times 10^{-6}, & J_3 = -2.565 \times 10^{-6}, \\ J_4 = -1.608 \times 10^{-6}, & \tau_L = -50. \end{array}$$

For convenience, we have chosen the initial weight as 1 lb, and the goal is to maximize the final weight, i.e., $w(t_F)$.

### 6.3.5 Numerical Solution

The solution to this low-thrust orbit transfer problem obtained using the implementation in $\mathbb{SOCS}$ is summarized in Figures 6.16 and 6.17. All times are plotted in hours, with the semiparameter $p$ given in millions of feet and the true longitude $L$ in multiples of 360 deg. The iteration history is summarized in Table 6.6. Figure 6.18 illustrates the behavior of the mesh-refinement algorithm. The optimal value for the final weight is $w^*(t_F) = 0.2201791266$ lb, which occurs at $t_F^* = 86810.0$ sec with an optimal throttle parameter value of $\tau^* = -9.09081$. The final trajectory profile is illustrated in Figure 6.19. It is also worthwhile to examine the behavior of the optimal control history when angles are used instead of the direction vector $\mathbf{u}(t)$. In particular, one can define

$$\begin{aligned} u_r &= -\sin\theta, \\ u_\theta &= \cos\theta\cos\psi, \\ u_h &= -\cos\theta\sin\psi \end{aligned}$$

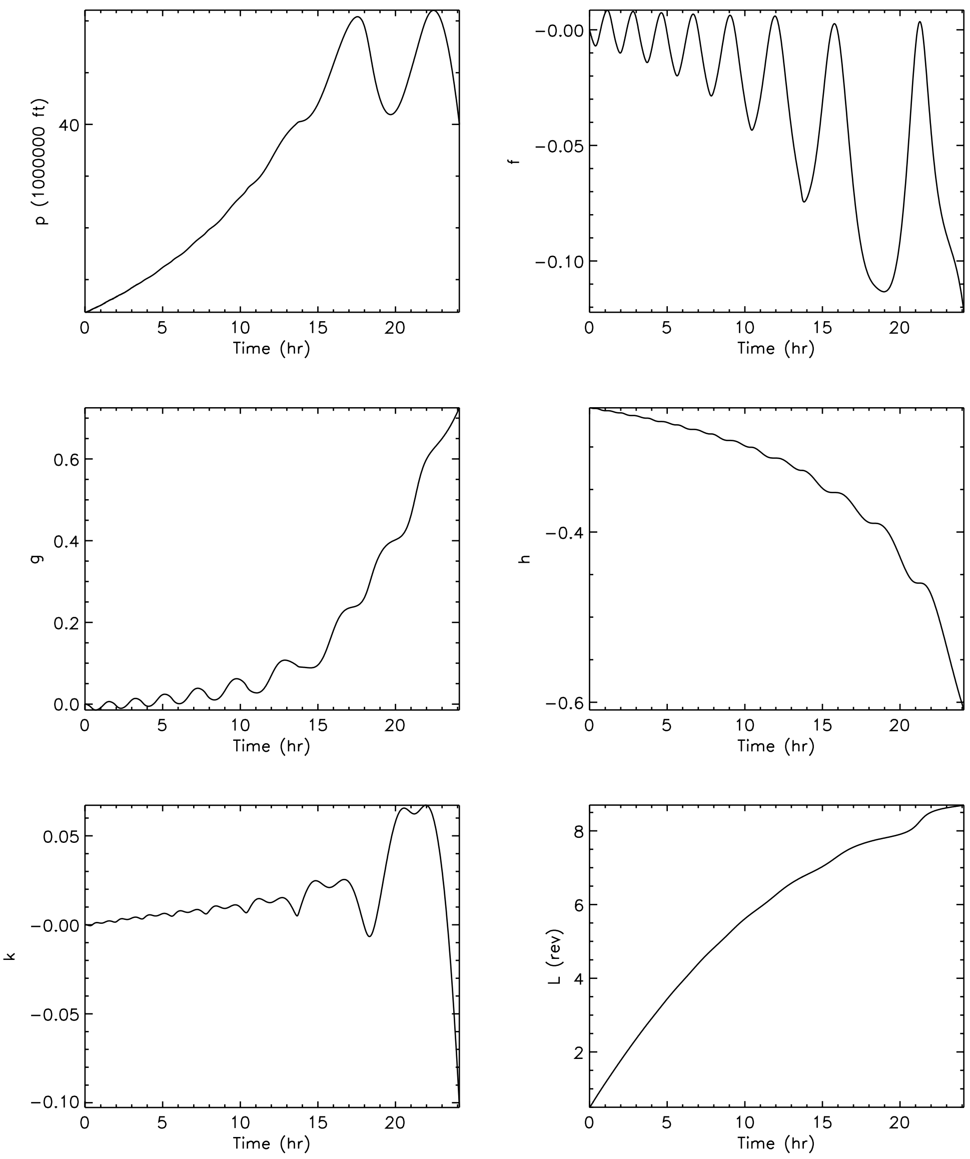


**Figure 6.16.** *Low-thrust transfer—state variables.*

using the two control angles $(\theta, \psi)$. The time history of these angles illustrated in Figure 6.15 clearly demonstrates the ambiguity in the yaw angle. In fact, if the controls were modeled using angles, it is clear that the mesh-refinement procedure would detect an apparent discontinuity caused by the yaw angle "wrapping" unless the dotted time history was followed.

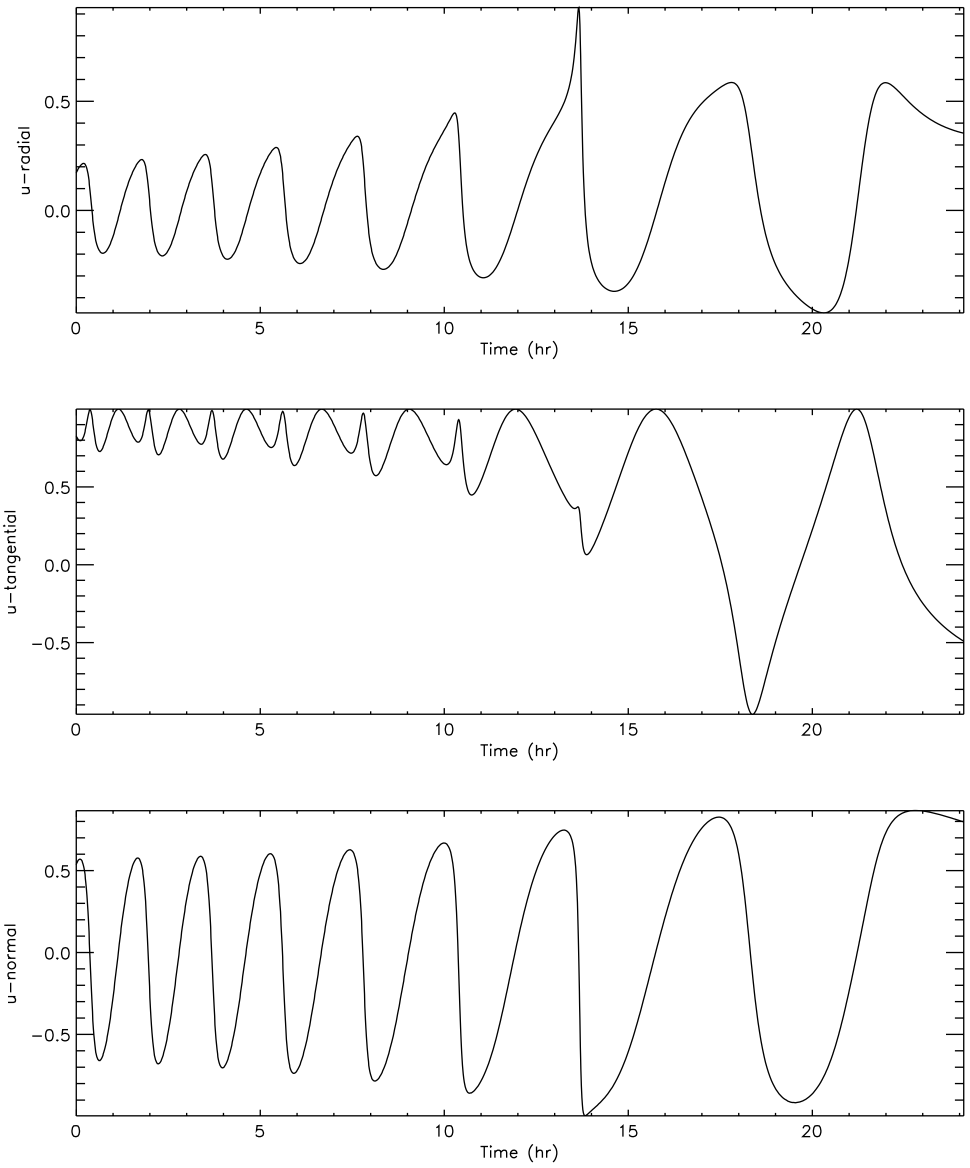


**Figure 6.17.** *Low-thrust transfer—control variables.*

## 6.4 Two-Burn Orbit Transfer

In this section, consider a problem similar to Example 6.5, with one important difference—the magnitude of the thrust. In particular, for this example the thrust $T = 1.25$ lb and the initial weight $w(0) = 1$ lb leading to a description that is very representative of most

**Table 6.6.** *Low-thrust transfer example.*

| Iter. | Disc. | $M$ | GE | HE | FE | RHS | $\epsilon_{\max}$ | CPU (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | TR | 150 | 416 | 80 | 11231 | 1684650 | $.22 \times 10^{-2}$ | $0.23 \times 10^{4}$ |
| 2 | TR | 299 | 10 | 7 | 604 | 180596 | $.26 \times 10^{-3}$ | $0.92 \times 10^{2}$ |
| 3 | HS | 299 | 11 | 8 | 682 | 407154 | $.31 \times 10^{-4}$ | $0.31 \times 10^{3}$ |
| 4 | HS | 597 | 8 | 6 | 503 | 600079 | $.15 \times 10^{-5}$ | $0.68 \times 10^{3}$ |
| 5 | HS | 966 | 6 | 3 | 292 | 563852 | $.61 \times 10^{-7}$ | $0.12 \times 10^{4}$ |
| Total | - | - | 451 | 104 | 13312 | 3436331 | - | 4562.20 |

operational launch vehicles. To make the problem more concrete, let us suppose that the vehicle begins in the same standard space shuttle park orbit as in Example 6.5, which is a 150 nm circular orbit with an inclination of 28.5 deg. The final orbit for this example is chosen to be a circular orbit with an altitude of 19323 nm and an inclination of 0 deg. This orbit is referred to as *geosynchronous* because it has a period of 24 hr, and the motion of a vehicle in this orbit is synchronized with the revolution of the earth beneath it. Thus, a satellite placed in geosynchronous orbit appears motionless when viewed from the earth, making it an attractive platform for communication systems. Since most communication satellites are placed in an orbit of this type, this particular trajectory design problem has been studied extensively, and, in contrast to Example 6.5, which is a hard problem, this may be considered an easy problem. On the other hand, because the problem is rather simple to solve, it permits us to illustrate and compare different solution methods.

The vehicle dynamics are initialized at a point in the park orbit. For convenience, the point at which the vehicle crosses the equator in a northbound direction (referred to as the ascending node) is chosen to be the initial time. After coasting for an unspecified time, the vehicle's engine is ignited. The orientation and duration of this "first burn" are also unspecified. The additional velocity added by the first burn places the vehicle into a "transfer orbit." After coasting for an unspecified time in the transfer orbit, the motor is again ignited and the "second burn" is performed. The duration of the burn and orientation of the vehicle during this time are also unspecified. However, when the second burn is completed, the vehicle must be deployed in the desired geosynchronous orbit. There are a number of ways to quantify an optimal orbit transfer. Typically, the trajectory that minimizes the fuel consumed or maximizes the final weight is preferred. An equivalent approach is to design a trajectory that minimizes the energy added by the propulsion system, and this approach is referred to as a "minimum $\Delta v$" transfer.

The motion of a vehicle can be described by the following system of first order ODEs:

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{6.57}$$

$$\dot{\mathbf{v}} = \mathbf{g} + \mathbf{T}, \tag{6.58}$$

$$\dot{w} = -T/I_{sp}. \tag{6.59}$$

These equations are equivalent to (6.28), where $\mathbf{r}$ is the inertial position vector, $\mathbf{v}$ is the inertial velocity vector, and $w$ is the weight. The gravitational acceleration is defined by $\mathbf{g}$ and the thrust acceleration is defined by $\mathbf{T}$. We use $T$ to denote the magnitude of the thrust $\|\mathbf{T}\|$ and $I_{sp}$ to denote the specific impulse.

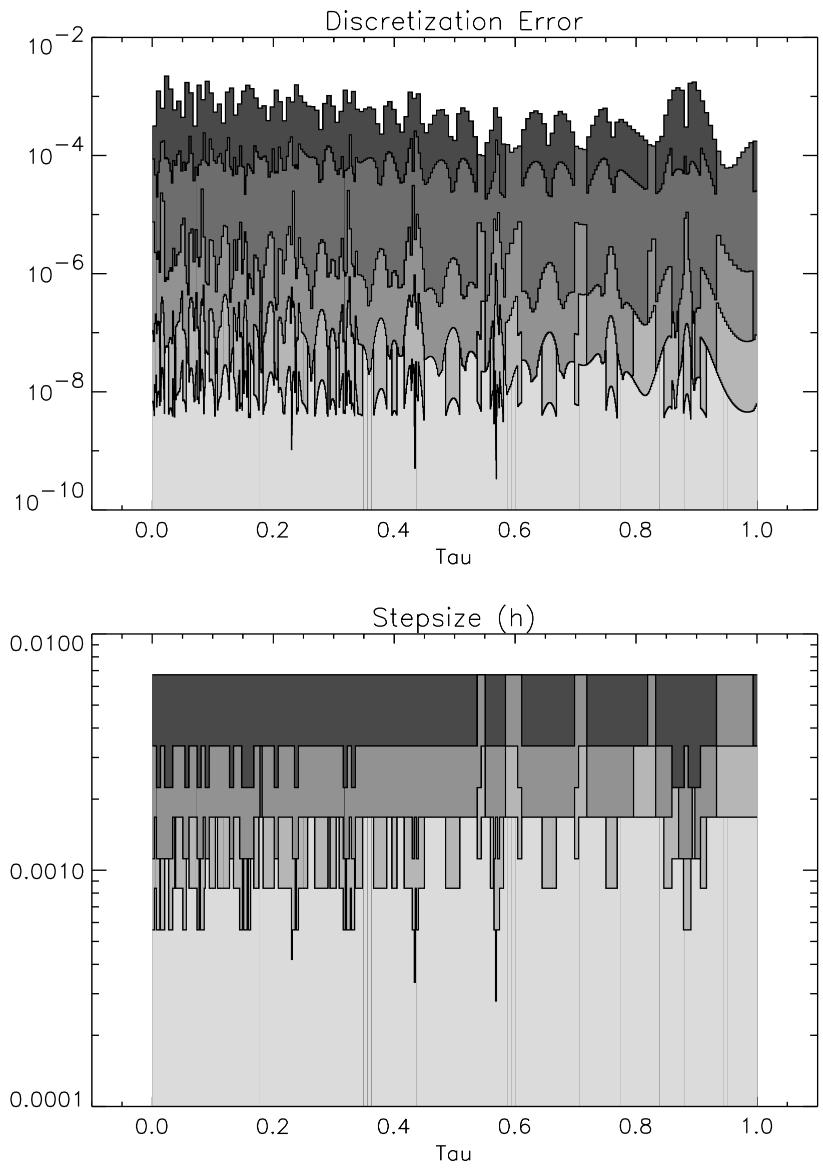


**Figure 6.18.** *Low-thrust transfer—mesh refinement.*

## 6.4.1 Simple Shooting Formulation

**Example 6.6** TWO-BURN TRANSFER. In Section 3.3, we described the *shooting method*, which is often used for solving simple BVPs such as this two-burn transfer. Early attempts to solve this problem introduced approximations to the *physics* in order to expedite the solution of (6.57)–(6.59). Although a simple shooting formulation can be obtained without approximating the physics, we will follow the historical technique.

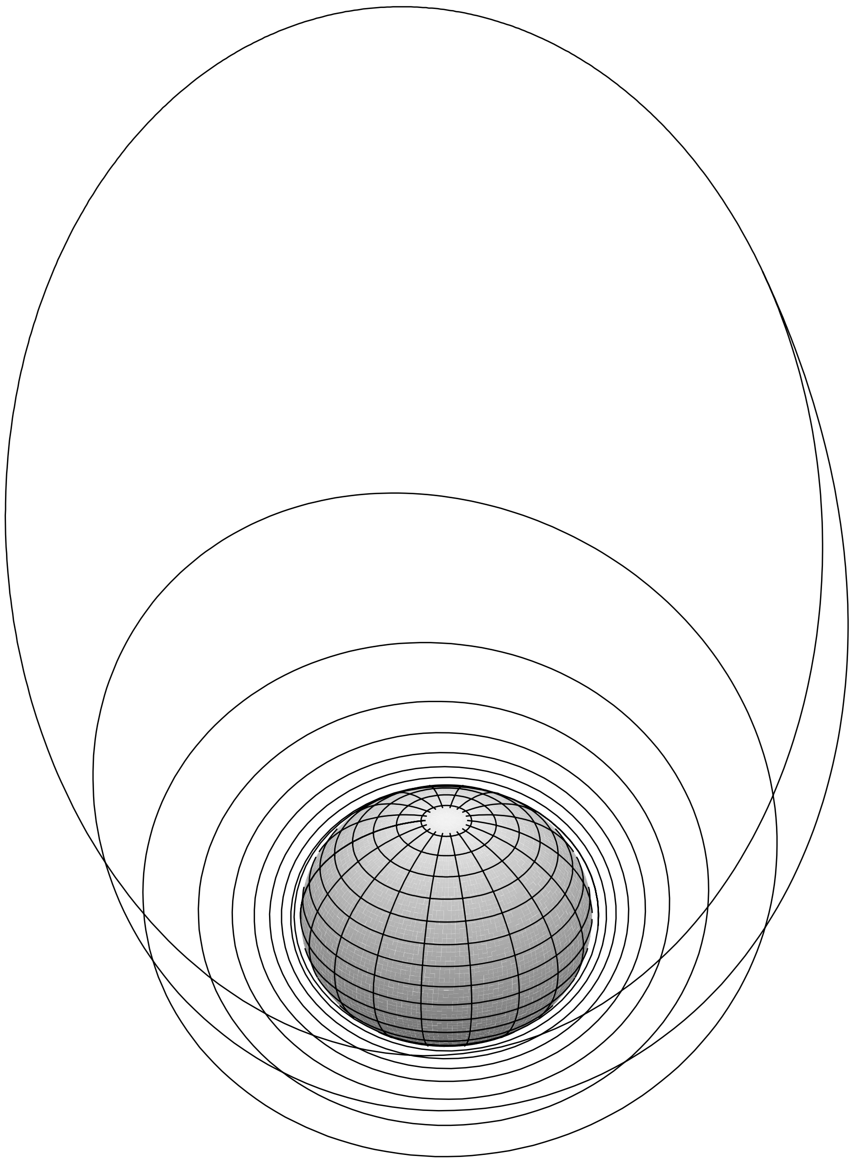


**Figure 6.19.** *Optimal low-thrust transfer.*

When the thrust $T$ is large, the duration of the burns is very short compared to the overall transfer time. Thus, it is common to assume the burns occur instantaneously. Using this approximation, the net effect of a burn is to change the *velocity* but not the *position*. This technique is called an *impulsive* $\Delta v$ approximation. To be more precise,

$$\mathbf{v}(t_2) = \mathbf{v}(t_1) + \boldsymbol{\Delta v}, \tag{6.60}$$

where $\mathbf{v}(t_1)$ is the velocity before the burn, $\mathbf{v}(t_2)$ is the velocity after the burn, $\boldsymbol{\Delta v}$ is the velocity added by the burn, and $t_1 = t_2$. The velocity change is related to the weight by the expression

$$\|\boldsymbol{\Delta v}\| = g_0 I_{sp} \ln\left[\frac{w(t_1)}{w(t_2)}\right], \tag{6.61}$$

where $g_0 = 32.174$ (ft/sec$^2$) is the mass to weight conversion factor. Also, for convenience, we can define the impulsive velocity using spherical coordinates $(\Delta v, \theta, \psi)$, where

$$\boldsymbol{\Delta v} = \mathbf{Q}_v \begin{pmatrix} \Delta v \cos\theta \cos\psi \\ \Delta v \cos\theta \sin\psi \\ \Delta v \sin\theta \end{pmatrix} \tag{6.62}$$

and the orthogonal matrix

$$\mathbf{Q}_v = \begin{bmatrix} \frac{\mathbf{v}}{\|\mathbf{v}\|} & \frac{\mathbf{v}\times\mathbf{r}}{\|\mathbf{v}\times\mathbf{r}\|} & \frac{\mathbf{v}}{\|\mathbf{v}\|} \times \left(\frac{\mathbf{v}\times\mathbf{r}}{\|\mathbf{v}\times\mathbf{r}\|}\right) \end{bmatrix} \tag{6.63}$$

defines the principle axes of an inertial velocity coordinate frame.

During coast portions of the trajectory, $\mathbf{T} = \mathbf{0}$ and the equations of motion (6.57)–(6.59) are just

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{6.64}$$
$$\dot{\mathbf{v}} = \mathbf{g}. \tag{6.65}$$

The only nonlinear quantity in these equations is the gravitational acceleration $\mathbf{g}(\mathbf{r})$. If one assumes that the earth is spherical, then $\mathbf{g}(\mathbf{r}) = -\mu\mathbf{r}/r^3$ and the differential equations (6.64)–(6.65) have an analytic solution! Thus, given the state at some time $t_0$, one can analytically propagate to another time $t_1$, i.e.,

$$\begin{bmatrix} \mathbf{r}(t_0) \\ \mathbf{v}(t_0) \end{bmatrix} \to \begin{bmatrix} \mathbf{r}(t_1) \\ \mathbf{v}(t_1) \end{bmatrix}. \tag{6.66}$$

In effect, this propagation defines a nonlinear mapping between the states at $t_0$ and $t_1$, say

$$\begin{bmatrix} \mathbf{r}(t_1) \\ \mathbf{v}(t_1) \end{bmatrix} = \mathbf{P}[\mathbf{r}(t_0), \mathbf{v}(t_0), t_1, t_0]. \tag{6.67}$$

Kepler provided the original solution of this propagation problem over 100 years ago, and a complete description of the computational procedure can be found both in Section 7.1.4 and [78]. More recently, Huffman [115] has extended the analytic propagation to include the most significant oblate earth perturbation (i.e., including the contribution of $J_2$ in (6.48) and (6.49)).

At this point, it is worthwhile to expand on a subtle but important detail regarding the meaning of analytic propagation. In order to propagate from one *time* to another, it is necessary to solve Kepler's equation. In its simplest form, one must compute $E$ such that

$$n(t - t_0) = E - e\sin E, \tag{6.68}$$

where $n$ and $e$ are constants for the orbit (the mean motion and eccentricity) and $t$ is the propagation time. Typically, this transcendental equation is solved by a Newton iteration.

The variable $E$, called the eccentric anomaly, determines the angular position of the vehicle at time $t$ on the orbit. In fact, for orbital motion, there is an angular change $\alpha$ that will satisfy the equation

$$k[\alpha, \Delta t] = 0 \tag{6.69}$$

for a specified value of $\Delta t$, but it cannot be written explicitly in the form

$$\alpha \sim k^{-1}[\Delta t].$$

Thus, if we choose to propagate the orbit by specifying the time change, we must solve a transcendental equation. But an "internal" iteration is undesirable, as discussed in Section 1.16. On the other hand, if we choose to propagate by specifying an angular change, no iteration is required! Thus, the proper formulation is to introduce the angular change $\alpha$ as an NLP variable (in addition to $t$) and then impose the Kepler equation (6.69). Section 7.1.4 revisits this issue.

The fundamental idea of combining impulsive $\Delta v$ and analytic orbit propagation was originally proposed by a German engineer, Walter Hohmann, who published the idea (in German) in Munich in 1925. Although his original paper described transfers between circular orbits with the same inclination, the term "Hohmann transfer" is now loosely applied to describe any transfer between arbitrary orbits using impulsive approximations.

The boundary conditions that define the desired geosynchronous orbit are

$$(\|\mathbf{r}\| - R_e)/\sigma = 19323 \text{ (nm)}, \tag{6.70}$$

$$\|\mathbf{v}\| = 10087.5 \text{ (ft/sec)}, \tag{6.71}$$

$$\frac{\mathbf{v}^{\mathsf{T}}\mathbf{r}}{\|\mathbf{v}\|\,\|\mathbf{r}\|} = 0, \tag{6.72}$$

$$\frac{v_3}{\|\mathbf{v}\|} = 0, \tag{6.73}$$

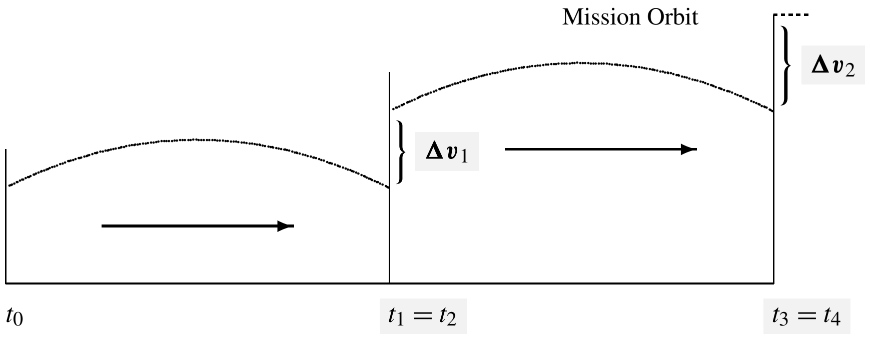
$$\frac{r_3}{\|\mathbf{r}\|} = 0, \tag{6.74}$$

where the constant $\sigma = 6076.1154855643$ ft/nm. We use the same values for $R_e$, $\mu$, and $J_2$ as in Example 6.5.

We can now define the NLP problem that must be solved. The NLP variables are

$$\mathbf{x} = \begin{bmatrix} t_1 \\ \alpha_1 \\ \Delta v_1 \\ \theta_1 \\ \psi_1 \\ t_3 \\ \alpha_2 \\ \Delta v_2 \\ \theta_2 \\ \psi_2 \end{bmatrix} = \begin{bmatrix} \text{First-burn ignition time} \\ \text{First coast angle} \\ \text{First-burn velocity} \\ \text{First-burn pitch angle} \\ \text{First-burn yaw angle} \\ \text{Second-burn ignition time} \\ \text{Second coast angle} \\ \text{Second-burn velocity} \\ \text{Second-burn pitch angle} \\ \text{Second-burn yaw angle} \end{bmatrix}. \tag{6.75}$$

When values are specified for the NLP variables, it is possible to compute the objective and constraint functions. This is illustrated schematically below:



For this direct shooting formulation, the function generator is as follows:

**Direct Shooting**

```
Input: $\mathbf{x}$
Initialize State:
    $\mathbf{r}(t_0), \mathbf{v}(t_0), t_0$
First Coast:
    Propagate through angle $\alpha_1$; from (6.67) compute $\mathbf{r}(t_1), \mathbf{v}(t_1)$.
Constraint Evaluation:
    Compute Kepler constraint from (6.69) using $\alpha_1, t_1$.
First Burn: Given $\Delta v_1, \theta_1, \psi_1$, set $\mathbf{r}(t_2) = \mathbf{r}(t_1)$, $t_2 = t_1$, and compute
    $\mathbf{v}(t_2)$ from (6.60), (6.62), and (6.63).
Second Coast:
    Propagate through angle $\alpha_2$; from (6.67) compute $\mathbf{r}(t_3), \mathbf{v}(t_3), t_3$.
Constraint Evaluation:
    Compute Kepler constraint from (6.69) using $\alpha_2, t_3$.
Second Burn: Given $\Delta v_2, \theta_2, \psi_2$, set $\mathbf{r}(t_4) = \mathbf{r}(t_3)$, $t_4 = t_3$, and compute
    $\mathbf{v}(t_4)$ from (6.60), (6.62), and (6.63).
Constraint Evaluation:
    Compute boundary conditions at $t_4$ from (6.70)–(6.74).
Terminate Trajectory
    Compute objective $F(\mathbf{x}), \mathbf{c}(\mathbf{x})$.
Output: $F(\mathbf{x}), \mathbf{c}(\mathbf{x})$
```

The goal is to minimize the objective function

$$F(\mathbf{x}) = \Delta v_1 + \Delta v_2. \tag{6.76}$$

Since none of the computed quantities for this example explicitly depends on time, this problem can also be formulated using only propagation angles. Table 6.7 presents the solutions obtained with and without time as a propagation variable (formulations A and B,

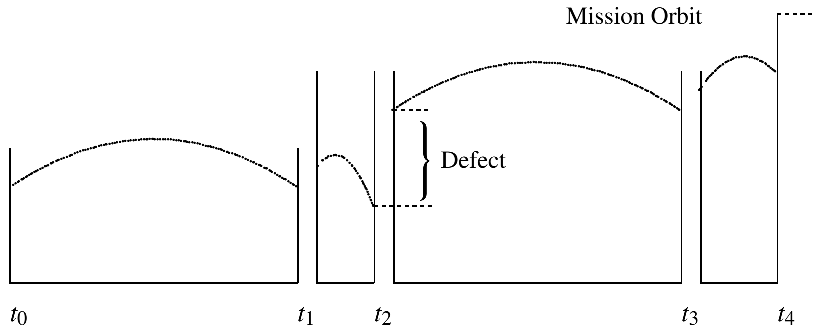
**Table 6.7.** *Minimum $\Delta v$ transfer.*

| | Formulation A | Formulation B |
|---|---|---|
| $t_1$ | 2690.41 | * |
| $\alpha_1$ | 179.667 | 179.667 |
| $\Delta v_1$ | 8049.57 | 8049.57 |
| $\theta_1$ | $0.148637 \times 10^{-2}$ | $0.146647 \times 10^{-2}$ |
| $\psi_1$ | −9.08446 | −9.08445 |
| $t_3$ | 21658.5 | * |
| $\alpha_2$ | 180.063 | 180.063 |
| $\Delta v_2$ | 5854.61 | 5854.61 |
| $\theta_2$ | $-0.136658 \times 10^{-2}$ | $-0.135560 \times 10^{-2}$ |
| $\psi_2$ | 49.7892 | 49.7892 |
| $\Delta v_1 + \Delta v_2$ | 13904.18221 | 13904.18220 |
| FE | 294 | 249 |

respectively). The only apparent difference between these approaches is that the formulation B requires fewer function evaluations (FE), which is not surprising because there are fewer variables. The equivalent weight delivered to the final orbit can be computed using (6.61) and is $w(t_F) = 0.23680470963252$ lb.

## 6.4.2 Multiple Shooting Formulation

**Example 6.7** TWO-BURN MULTIPLE SHOOTING. A very natural partition of the problem is suggested by the physics of this two-burn orbit transfer. Modeling the dynamics using four distinct phases, namely the coast in the park orbit, the first burn, the coast in the transfer orbit, and the second burn, is an obvious choice. We can also treat each of these phases as segments and apply the *multiple shooting* method as described in Section 3.4. This is illustrated below:



For the multiple shooting formulation, one must augment the NLP variables (6.75) to include the state on each side of the segment (phase) boundaries as illustrated. Thus, we include as new NLP variables the states $\mathbf{r}(t_k^-), \mathbf{v}(t_k^-)$ and $\mathbf{r}(t_k^+), \mathbf{v}(t_k^+)$ for $k = 1,2,3,4$, where $-$ denotes the quantity before the boundary and $+$ denotes the quantity after the boundary. Additional "defect" constraints must be imposed to ensure continuity across the segment boundaries and also to guarantee that the analytically propagated state is consistent with the (new) guessed values. Thus, when compared to the direct shooting method, the number of variables increases from 10 to 63. However, the number of constraints also increases from 7 to 60, so that the total number of degrees of freedom is unchanged.

One of the primary reasons to use a multiple shooting method is to improve robustness, which is demonstrated even on this simple four-segment example. As shown in Table 6.8, the multiple shooting method solved the problem with fewer gradient evaluations (2), fewer Hessian evaluations (3), and fewer function evaluations (97) when compared to the simple shooting approach. Even the modest increase in CPU time (attributed to increased problem size) is somewhat deceiving because the cost of evaluating the functions is so small for this example.

**Table 6.8.** *Shooting versus multiple shooting.*

| | Shooting | Multiple shooting | $\Delta$ |
|---|---|---|---|
| GE | 11 | 9 | −22% |
| HE | 5 | 2 | −150% |
| FE | 294 | 197 | −49% |
| CPU (sec) | 0.699127 | 0.872283 | +20% |

## 6.4.3 Collocation Formulation

**Example 6.8** Two-Burn Collocation. Both Example 6.6 and Example 6.7 used approximate *physics*, that is, simplified orbit dynamics and impulsive burn approximations. For comparison, let us now solve the same orbit transfer without making these simplifying assumptions. We still pose the problem using four phases. However, in this example we consider finite-duration burn phases with optimal steering. Oblate gravitational accelerations will be used throughout, and we will describe the dynamics using the previously discussed equinoctial coordinates. During the coast phases, from (6.31), the dynamics are given by

$$\dot{\mathbf{y}} = \mathbf{A}(\mathbf{y})\boldsymbol{\Delta}_g + \mathbf{b}, \tag{6.77}$$

where $\mathbf{A}(\mathbf{y})$ is defined by (6.35), $\mathbf{b}$ by (6.36), and $\boldsymbol{\Delta}_g$ by (6.50). During the burn phases,

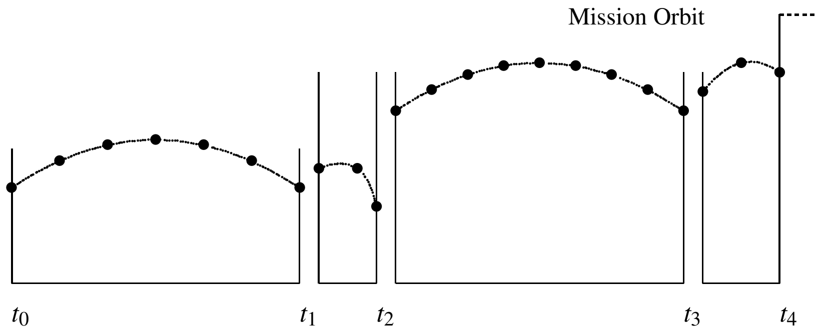
$$\dot{\mathbf{y}} = \mathbf{A}(\mathbf{y})\boldsymbol{\Delta} + \mathbf{b}, \tag{6.78}$$

$$\dot{w} = -T/I_{sp}, \tag{6.79}$$

and the total perturbation is given by (6.44). To be consistent with Examples 6.6 and 6.7, we define the orientation of the thrust using pitch and yaw angles in the inertial velocity frame and then transform them to the radial frame, that is,

$$\boldsymbol{\Delta}_T = \mathbf{Q}_r^{\mathsf{T}}\mathbf{Q}_v \begin{bmatrix} T\cos\theta\cos\psi \\ T\cos\theta\sin\psi \\ T\sin\theta \end{bmatrix}, \tag{6.80}$$

where $\mathbf{Q}_v$ is defined by (6.63) and $\mathbf{Q}_r$ is given by (6.45). Since the dynamics are represented using equinoctial coordinates, it is convenient to also specify the boundary conditions in these coordinates. Thus, the geosynchronous conditions (6.70)–(6.74) are equivalent to having $p = 19323/\sigma + R_e$ and $f = g = h = k = 0$. Constraints are also used to link the equinoctial states across the phase boundaries as with the multiple shooting formulation. The weight at the end of phase 2 is also constrained to equal the weight at the beginning of phase 4. The situation is illustrated below:



SOCS was used to solve the problem beginning with a trapezoidal discretization and 10 grid points per phase. The first NLP has 307 variables and 268 active constraints, or 39 degrees of freedom. Figure 6.20 displays the sparsity pattern of the Jacobian and Hessian matrices, corresponding to the initial trapezoidal discretization with 10 grid points per phase. Note that, because the Hessian is symmetric, only the lower-triangular portion is

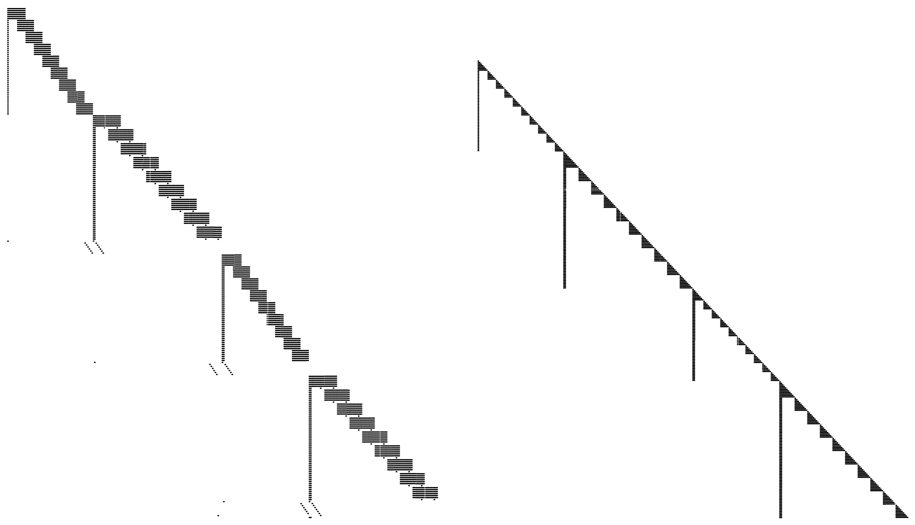

(a) Jacobian (b) Hessian

**Figure 6.20.** *Sparsity patterns.*

shown. The iteration history is summarized in Table 6.9. The second and third columns in the table give a phase-by-phase breakdown of the discretization method (Disc.) and number of grid points ($M$), respectively. Trapezoidal discretization is denoted by T and separated Simpson by H. Observe that the mesh-refinement algorithm tends to concentrate a large number of points in the early portions of the trajectory because the oblate earth perturbations are more significant in this region. The final mesh-refinement iteration required solving a sparse NLP problem with 5149 variables, 4714 active constraints, and 435 degrees of freedom.

**Table 6.9.** *Two-burn orbit transfer performance summary.*

| Iter. | Disc. | $M$ | FE | $\epsilon_{\max}$ | $T$ (sec) |
|---|---|---|---|---|---|
| 1 | (T,T,T,T) | (10,10,10,10) | 2164 | $0.26 \times 10^{0}$ | $0.41 \times 10^{2}$ |
| 2 | (H,T,T,T) | (10,19,16,19) | 604 | $0.53 \times 10^{-2}$ | $0.20 \times 10^{2}$ |
| 3 | (H,H,H,H) | (19,19,16,19) | 526 | $0.14 \times 10^{-2}$ | $0.32 \times 10^{2}$ |
| 4 | (H,H,H,H) | (37,37,31,37) | 137 | $0.96 \times 10^{-5}$ | $0.24 \times 10^{2}$ |
| 5 | (H,H,H,H) | (73,73,61,37) | 113 | $0.36 \times 10^{-6}$ | $0.35 \times 10^{2}$ |
| 6 | (H,H,H,H) | (145,73,121,37) | 113 | $0.59 \times 10^{-7}$ | $0.49 \times 10^{2}$ |
| Total | - | 376 | 3657 | | 201.84 |

Figure 6.21 illustrates the optimal two-burn transfer to geosynchronous orbit. The optimal steering angles for both burns are plotted in Figure 6.22 with the times normalized to the beginning of the burn. It is interesting to observe that the optimal burn times, i.e., the lengths of phases 2 and 4, are $t_2 - t_1 = 141.47$ (sec) and $t_4 - t_3 = 49.40$ (sec). When compared to the total mission time of 21683.5 (sec), these burn times are quite short, hence justifying the impulsive approximation. It is also interesting to note that the "true" optimal objective $w(t_F) = 0.2367248713$ lb is only 0.033% less than the impulsive burn analytic coast approximation computed in Example 6.6. However, it is necessary to use the optimal steering in Figure 6.22 to achieve this performance.

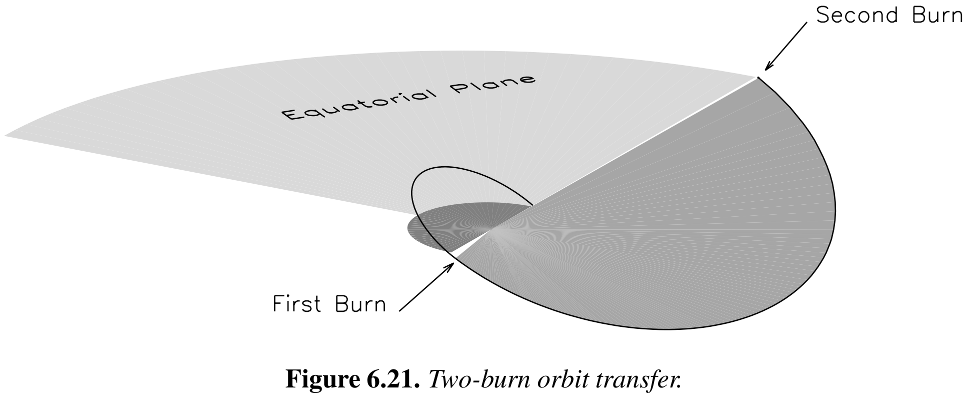
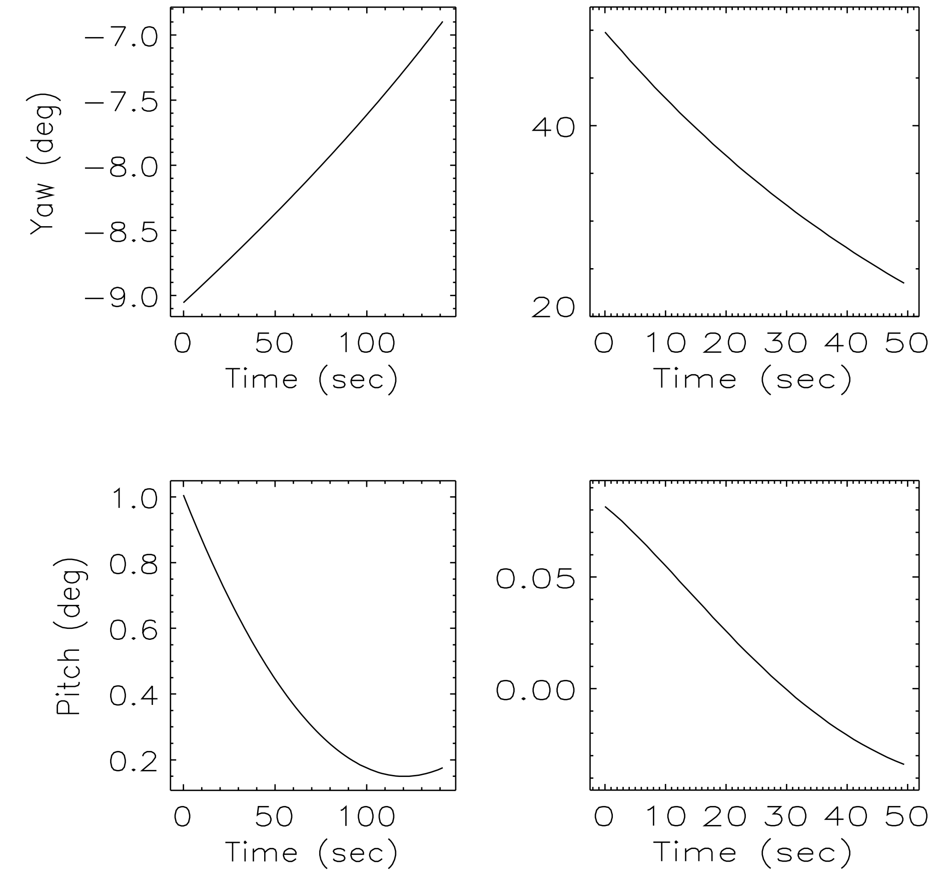


**Figure 6.21.** *Two-burn orbit transfer.*



(a) First burn (b) Second burn

**Figure 6.22.** *Optimal control angles.*

## 6.5 Hang Glider

**Example 6.9** RANGE MAXIMIZATION OF A HANG GLIDER. Originally posed by Bulirsch et al. [58], this problem describes the optimal control of a hang glider in the presence of a specified thermal updraft. It is particularly sensitive to the accuracy of the mesh, a difficulty which was resolved in the reference by exploiting a combination of direct and indirect methods.

The state variables are $\mathbf{y}^{\mathsf{T}}(t) = (x, y, v_x, v_y)$, where $x$ is the horizontal distance (meters), $y$ the altitude (meters), $v_x$ the horizontal velocity (meters/sec), and $v_y$ the vertical velocity (meters/sec). The control variable is $u(t) = C_L$, the aerodynamic lift coefficient. The final time $t_F$ is free and the final range $x_F$ is to be maximized. The state equations which describe the planar motion for the hang glider are

$$\dot{x} = v_x, \tag{6.81}$$

$$\dot{y} = v_y, \tag{6.82}$$

$$\dot{v_x} = \frac{1}{m}(-L\sin\eta - D\cos\eta), \tag{6.83}$$

$$\dot{v_y} = \frac{1}{m}(L\cos\eta - D\sin\eta - W), \tag{6.84}$$

where the quadratic drag polar

$$C_D(C_L) = C_0 + kC_L^2 \tag{6.85}$$

with

$$\begin{aligned}
D &= \frac{1}{2}C_D\rho S v_r^2, & L &= \frac{1}{2}C_L\rho S v_r^2, \\
X &= \left(\frac{x}{R} - 2.5\right)^2, & u_a(x) &= u_M(1-X)\exp[-X], \\
V_y &= v_y - u_a(x), & v_r &= \sqrt{v_x^2 + V_y^2}, \\
\sin\eta &= \frac{V_y}{v_r}, & \cos\eta &= \frac{v_x}{v_r}.
\end{aligned}$$

Note the results presented in [58] correspond to a value of 3.5 instead of 2.5 in the expression for $X$. The following constants complete the definition of the problem:

$$\begin{aligned}
u_M &= 2.5, & m &= 100\,(\text{kg}), \\
R &= 100, & S &= 14\,(\text{m}^2), \\
C_0 &= .034, & \rho &= 1.13\,(\text{kg/m}^3), \\
k &= .069662, & g &= 9.80665\,(\text{m/sec}^2),
\end{aligned}$$

where $W = mg$. The lift coefficient is bounded

$$0 \le C_L \le 1.4 \tag{6.86}$$

and the following boundary conditions are imposed:

$$\begin{aligned}
x(0) &= 0\,(\text{m}), & x(t_F) &: \quad \text{free}, \\
y(0) &= 1000\,(\text{m}), & y(t_F) &= 900\,(\text{m}), \\
v_x(0) &= 13.227567500\,(\text{m/sec}), & v_x(t_F) &= 13.227567500\,(\text{m/sec}), \\
v_y(0) &= -1.2875005200\,(\text{m/sec}), & v_y(t_F) &= -1.2875005200\,(\text{m/sec}).
\end{aligned}$$

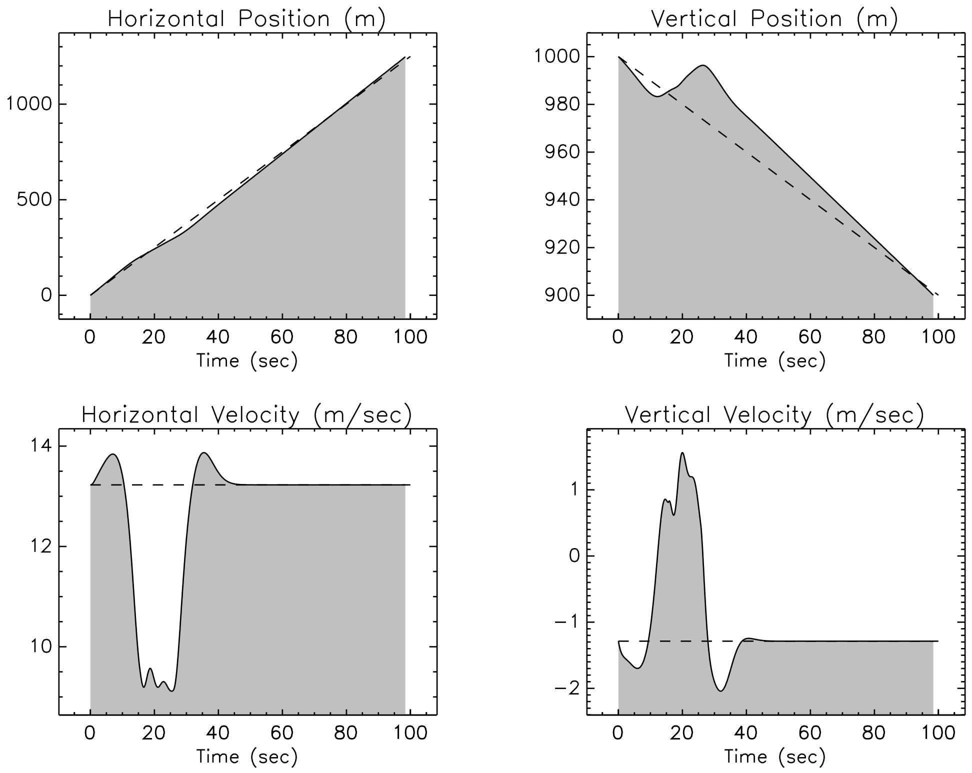


**Figure 6.23.** *Hang glider, states.*

The initial guess was computed using linear interpolation between the boundary conditions, with $x(t_F) = 1250$, and $C_L(0) = C_L(t_F) = 1$.

The optimal state histories are displayed in Figure 6.23, with the initial guess plotted as a dashed line. Figure 6.24 shows the optimal control. The mesh-refinement history is summarized in Table 6.10 and plotted in Figure 6.25. Observe that eight mesh-refinement iterations were required before the discretization error $\epsilon$ was reduced significantly. This behavior is consistent with the sensitivity reported in [58], which led them to use a hybrid technique. The maximum range is $x^*(t_F) = 1248.031026$ (m).

## 6.6 Abort Landing in the Presence of Windshear

**Example 6.10** ABORT LANDING IN THE PRESENCE OF WINDSHEAR. The dynamic behavior of an aircraft landing in the presence of a windshear was first formulated as an optimal control problem by Miele,[4] Wang, and Melvin [134]. A number of other authors

[4]Dr. Angelo Miele was formerly Director of Astrodynamics and Flight Mechanics at Boeing Scientific Research Laboratory (BSRL), and is now Foyt Professor Emeritus at Rice University.

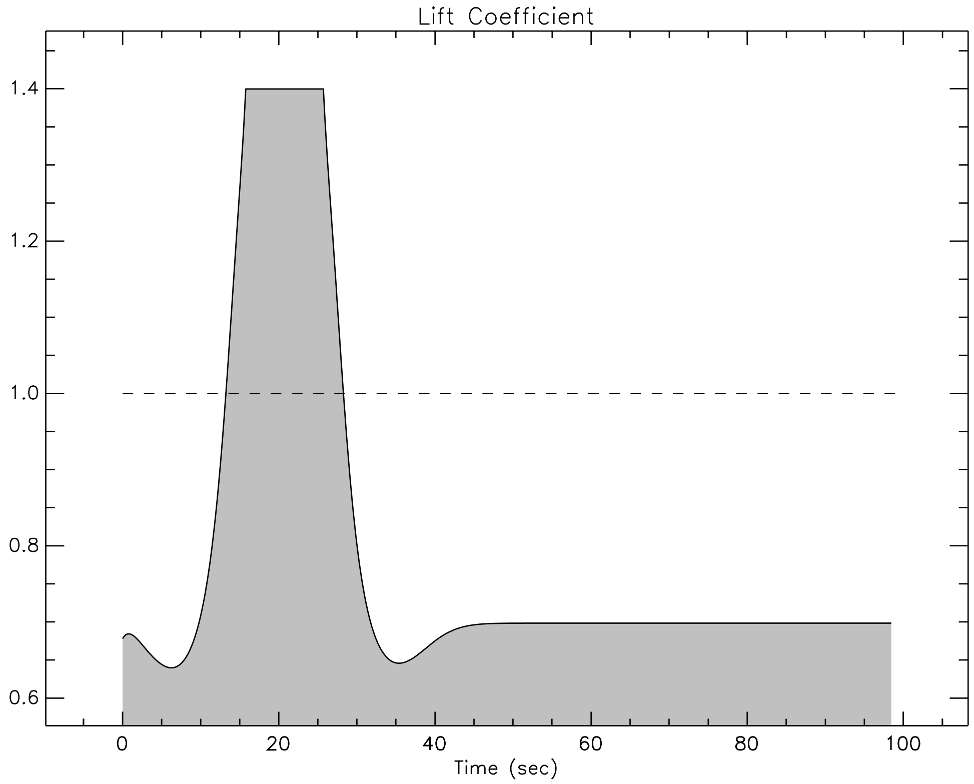


**Figure 6.24.** *Hang glider control $C_L(t)$.*

**Table 6.10.** *Hang glider mesh-refinement summary.*

| $k$ | $M$ | NGC | NHC | NFE | NRHS | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|
| 1 | 25 | 23 | 16 | 383 | 14207 | 2.0200 | $1.00\times10^{-1}$ |
| 2 | 49 | 30 | 28 | 605 | 38163 | $3.5669\times10^{1}$ | $2.90\times10^{-1}$ |
| 3 | 49 | 31 | 29 | 1946 | 204480 | $1.2682\times10^{1}$ | $7.00\times10^{-1}$ |
| 4 | 97 | 56 | 54 | 3596 | 722034 | 5.2137 | 2.13 |
| 5 | 106 | 18 | 16 | 1088 | 259858 | 5.5682 | $8.30\times10^{-1}$ |
| 6 | 113 | 33 | 31 | 2078 | 499588 | 1.4190 | 1.67 |
| 7 | 134 | 22 | 20 | 1352 | 398490 | 1.2231 | 1.30 |
| 8 | 145 | 17 | 15 | 1022 | 335652 | 1.2494 | 1.15 |
| 9 | 151 | 17 | 15 | 1022 | 349236 | $1.7212\times10^{-2}$ | 1.20 |
| 10 | 277 | 21 | 19 | 1286 | 777980 | $1.6366\times10^{-2}$ | 3.22 |
| 11 | 291 | 9 | 6 | 449 | 330507 | $3.4776\times10^{-5}$ | 1.23 |
| 12 | 567 | 4 | 2 | 164 | 312058 | $5.2638\times10^{-7}$ | 3.02 |
| 13 | 739 | 3 | 1 | 98 | 306696 | $2.0964\times10^{-8}$ | 4.04 |
| Total | 739 | 284 | 252 | 15089 | 4548949 |  | $2.088\times10^{+1}$ |

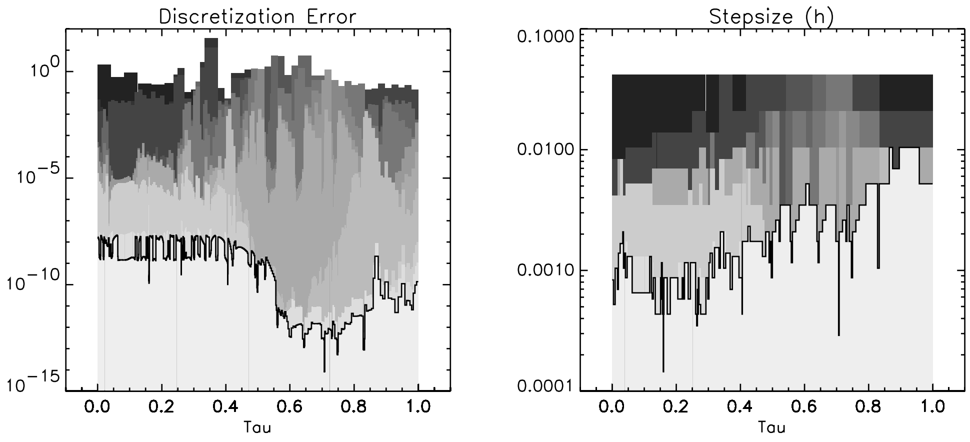


**Figure 6.25.** *Hang glider mesh-refinement history.*

investigated the problem including Bulirsch, Montrone, and Pesch [56, 57], who introduce the problem as follows:

> One of the most dangerous situations for a passenger aircraft in take-off and landing is caused by the presence of low altitude windshears. This meteorological phenomenon, which is more common in subtropical regions, is usually associated with high ground temperatures leading to a so-called downburst. This downburst involves a column of descending air which spreads horizontally near the ground. Even for a highly skilled pilot, an inadvertent encounter with a windshear can be a fatal problem, since the aircraft might encounter a headwind followed by a tailwind, both coupled with a downdraft. The transition from headwind to tailwind yields an acceleration so that the resulting windshear inertia force can be as large as the drag of the aircraft, and sometimes as large as the thrust of the engines. This explains why the presence of low altitude windshears is a threat to safety in aviation. Some 30 aircraft accidents over the past 20 years have been attributed to windshear, and this attests to the perilousness of this occurrence. Among these accidents, the most disastrous ones happened in 1982 in New Orleans, where 153 people were killed, and in 1985 in Dallas, where 137 people were killed.

### 6.6.1 Dynamic Equations

Following the development in [56] the dynamic behavior of the aircraft is described by

$$\dot{x} = v\cos\gamma + w_x, \tag{6.87}$$

$$\dot{h} = v\sin\gamma + w_h, \tag{6.88}$$

$$\dot{v} = \frac{1}{m}[T\cos(\alpha+\delta) - D] - g\sin\gamma - (\dot{w}_x\cos\gamma + \dot{w}_h\sin\gamma), \tag{6.89}$$

$$\dot{\gamma} = \frac{1}{mv}[T\sin(\alpha+\delta) + L] - \frac{g}{v}\cos\gamma + (\dot{w}_x\sin\gamma - \dot{w}_h\cos\gamma), \tag{6.90}$$

where the state variables are the horizontal distance $x$, the altitude $h$, the relative velocity $v$, and the relative flight path angle $\gamma$. The thrust and aerodynamic forces are defined by

$$T = \beta T_*, \tag{6.91}$$

$$T_* = a_0 + a_1 v + a_2 v^2, \tag{6.92}$$

$$D = \frac{1}{2} C_D \rho S v^2, \tag{6.93}$$

$$C_D(\alpha) = b_0 + b_1\alpha + b_2\alpha^2, \tag{6.94}$$

$$L = \frac{1}{2} C_L \rho S v^2, \tag{6.95}$$

$$C_L(\alpha) = \begin{cases} c_0 + c_1\alpha, & \alpha \le \alpha_*, \\ c_0 + c_1\alpha + c_2(\alpha - \alpha_*)^2, & \alpha_* \le \alpha \le \alpha_{max}, \end{cases} \tag{6.96}$$

$$\beta(t) = \begin{cases} \beta_0 + \dot\beta_0 t, & 0 \le t \le t_\beta, \\ 1, & t_\beta \le t \le t_F, \end{cases} \tag{6.97}$$

where the thrust, drag, and lift are denoted by $T$, $D$, and $L$, respectively. The power setting $\beta$ is specified as in [134]. It is hypothesized that upon sensing a downdraft, the pilot increases power at a constant rate until reaching a maximum value at time $t_\beta = (1 - \beta_0)/\dot\beta_0$ and thereafter holds it constant. The windshear is modeled as follows:

$$w_x = A(x), \tag{6.98}$$

$$w_h = \frac{h}{h_*} B(x) \tag{6.99}$$

with

$$A(x) = \begin{cases} -50 + ax^3 + bx^4, & 0 \le x \le 500, \\ \frac{1}{40}(x - 2300), & 500 \le x \le 4100, \\ 50 - a(4600 - x)^3 - b(4600 - x)^4, & 4100 \le x \le 4600, \\ 50, & 4600 \le x, \end{cases} \tag{6.100}$$

$$B(x) = \begin{cases} dx^3 + ex^4, & 0 \le x \le 500, \\ -51 \exp[-c(x - 2300)^4], & 500 \le x \le 4100, \\ d(4600 - x)^3 - e(4600 - x)^4, & 4100 \le x \le 4600, \\ 0, & 4600 \le x. \end{cases} \tag{6.101}$$

Table 6.11 summarizes the various parameters as given in [56] needed to complete the model of a Boeing 727 airplane.

**Table 6.11.** *Dynamic model parameters.*

| | | | |
|---|---|---|---|
| $t_F$ | 40 sec | $u_{max}$ | 3 deg/sec |
| $\alpha_{max}$ | 17.2 deg | $\rho$ | $.2203\times 10^{-2}$ lb sec$^2$ ft$^{-4}$ |
| $S$ | $.1560\times 10^4$ ft$^2$ | $g$ | $3.2172\times 10^1$ ft sec$^{-2}$ |
| $mg$ | 150000 lb | $\delta$ | 2 deg |
| $a_0$ | $.4456\times 10^5$ lb | $a_1$ | $-.2398\times 10^2$ lb sec/ft |
| $a_2$ | $.1442\times 10^{-1}$ lb sec$^2$ ft$^{-2}$ | $\beta_0$ | .3825 |
| $\dot{\beta}_0$ | .2 sec$^{-1}$ | $b_0$ | .1552 |
| $b_1$ | .12369 rad$^{-1}$ | $b_2$ | 2.4203 rad$^{-2}$ |
| $c_0$ | .7125 | $c_1$ | 6.0877 rad$^{-1}$ |
| $c_2$ | $-9.0277$ rad$^{-2}$ | $a_*$ | 12 deg |
| $h_*$ | 1000 ft | $a$ | $6\times 10^{-8}$ sec$^{-1}$ ft$^{-2}$ |
| $b$ | $-4\times 10^{-11}$ sec$^{-1}$ ft$^{-3}$ | $c$ | $-\ln(25/30.6)\times 10^{-12}$ ft$^{-4}$ |
| $d$ | $-8.02881\times 10^{-8}$ sec$^{-1}$ ft$^{-2}$ | $e$ | $6.28083\times 10^{-11}$ sec$^{-1}$ ft$^{-3}$ |
| $x_0$ | 0 ft | $\gamma_0$ | $-2.249$ deg |
| $h_0$ | 600 ft | $\alpha_0$ | 7.353 deg |
| $v_0$ | 239.7 ft/sec | $\gamma_F$ | 7.431 deg |

### 6.6.2 Objective Function

The goal of this abort landing problem is to avoid having the airplane crash. To achieve this we introduce two things: an optimization parameter $h_{min}$, which is the minimum altitude that occurs during the landing, and a new path inequality constraint

$$h(t) \geq h_{min}. \tag{6.102}$$

The optimization objective is to *maximize*

$$F = h_{min} \tag{6.103}$$

subject to the path inequality (6.102). This *maximin* problem guarantees the aircraft will be as high above the ground as possible. The results obtained using $\mathbb{SOCS}$ treat this formulation directly.

In contrast, both [134] and [56] treat the objective differently. Because of the relationship between the Hölder and Chebyshev norm

$$\lim_{k\to\infty}\left[\int_0^{t_F}[h_r - h(t)]^{2k}dt\right]^{\frac{1}{2k}} = \max_{0\leq t\leq t_f}[h_r - h(t)],$$

where $h_r = 1000$ ft is a reference altitude. Specifically, Miele, Wang, and Melvin [134] simply choose a large value for $k$ and minimize

$$F_M = \int_0^{t_F}[h_r - h(t)]^6 dt.$$

Unfortunately this approach is numerically unattractive because of the scale of the objective function. In order to address the numeric difficulties Bulirsch, Montrone, and Pesch [56] introduce a new state variable and path inequality constraint

$$\dot{\zeta} = 0,$$
$$0 \geq h_r - h(t) - \zeta(t)$$

and then postulate a composite objective function

$$F_B = \Lambda \int_0^{t_F} [h_r - h(t)]^6 dt + \Theta \zeta(t_F).$$

The parameter $\Lambda$ is gradually changed from one to zero, while $\Theta$ is increased from zero to one using a homotopy strategy described in [57].

### 6.6.3 Control Variable

The control variable for this problem is the angle of attack $\alpha(t)$, which is subject to the constraints

$$\alpha(t) \leq \alpha_{max}, \tag{6.104}$$
$$|\dot{\alpha}(t)| \leq u_{max}. \tag{6.105}$$

Using the approach described in Section 4.10 it is possible to directly impose the rate constraint (6.105).

In contrast, the angle of attack is treated as a state variable in [56] and the rate is treated as a control. This introduces an additional differential equation

$$\dot{\alpha} = u$$

in which case the rate constraint (6.105) can be treated as simple bounds $-u_{max} \leq u \leq u_{max}$ on the (new) control. Unfortunately, this transformation introduces two other more deleterious effects. The new control $u$ appears linearly in the problem, whereas the "real" control $\alpha$ appears nonlinearly, and consequently singular arcs can (and do) occur. Furthermore, it increases the order of the path constraints. When $u$ is the control, the solution has a boundary arc with respect to the first order state constraint and touch points with respect to the third order state constraints.

### 6.6.4 Model Data

The mathematical description of a complicated physical system usually entails specification of tabular data. In order to construct a mathematical model with the appropriate continuity, one approach is to construct a B-spline approximation as described in Section 6.2.3. This becomes particularly important when using an indirect method to solve the optimal control problem as in [56] because discontinuities introduce "interior-point conditions" and as they state

> ...a reduction of the matching points from 8 points in Miele's model to 3 points in our model, makes the derivation of the necessary conditions of optimal control theory much easier. Since Miele's model uses cubic splines, the wind components of his model are $C^2$ functions everywhere.

While smoothing the model data is one way to treat discontinuous functions, there is another approach which can be employed. Let us model the problem using a multiple phase structure. Since discontinuous behavior can occur at a phase boundary, we can define a phase structure specifically to treat the different regions in the problem data. In particular let us model the problem using five distinct phases. Each phase is characterized by a unique set of boundary conditions and right-hand-side functions as follows:

**Phase 1** $\boxed{0 = t_I^{(1)} \le t \le t_F^{(1)}}$

$$\begin{aligned} x[t_I^{(1)}] &= x_0, & h[t_I^{(1)}] &= h_0, \\ v[t_I^{(1)}] &= v_0, & \gamma[t_I^{(1)}] &= \gamma_0, \\ \alpha[t_I^{(1)}] &= \alpha_0, & x[t_F^{(1)}] &= 500, \\ A(x) &= -50 + ax^3 + bx^4, & B(x) &= dx^3 + ex^4, \\ \beta(t) &= \beta_0 + \dot{\beta}_0 t. \end{aligned}$$

**Phase 2** $\boxed{t_I^{(2)} \le t \le t_F^{(2)} = t_\beta}$

$$\begin{aligned} x[t_I^{(2)}] &= 500, & \beta(t) &= \beta_0 + \dot{\beta}_0 t, \\ A(x) &= -50 + ax^3 + bx^4, & B(x) &= dx^3 + ex^4. \end{aligned}$$

**Phase 3** $\boxed{t_\beta = t_I^{(3)} \le t \le t_F^{(3)}}$

$$\begin{aligned} x[t_F^{(3)}] &= 4100, & \beta(t) &= 1, \\ A(x) &= \frac{1}{40}(x - 2300), & B(x) &= -51 \exp[-c(x-2300)^4]. \end{aligned}$$

**Phase 4** $\boxed{t_I^{(4)} \le t \le t_F^{(4)}}$

$$\begin{aligned} x[t_I^{(4)}] &= 4100, & x[t_F^{(4)}] &= 4600, \\ A(x) &= 50 - a(4600-x)^3 - b(4600-x)^4, & B(x) &= d(4600-x)^3 - e(4600-x)^4, \\ \beta(t) &= 1. \end{aligned}$$

**Phase 5** $\boxed{t_I^{(5)} \le t \le t_F^{(5)} = t_F}$

$$\begin{aligned} x[t_I^{(5)}] &= 4600, & \gamma(t_F^{(5)}) &= \gamma_F, \\ A(x) &= 50, & B(x) &= 0, \\ \beta(t) &= 1. \end{aligned}$$

Observe that the throttle parameter (6.97) is a monotonic function of time, and the wind model (6.100)–(6.101) is a monotonic function of distance $x$ which in turn is a monotonic function of time. Consequently there is a natural "interleaving" of the discontinuous regions that is explicitly identified by the phase structure. Note also that the discontinuous behavior in the second derivative of the lift coefficient (6.96) does not present any difficulties because it is a function of the control $\alpha$, which can be discontinuous.

### 6.6.5 Computational Results

The overall problem can be posed using five distinct phases. On all five phases, there are four state variables $\mathbf{y}^{\mathsf{T}} = (x, h, v, \gamma)$, one control $u = \alpha$, and one parameter $p = h_{min}$. Each phase must satisfy the differential equations (6.87)–(6.90) and the path constraint (6.102). On all phases, the control variable must satisfy the simple bounds (6.104) and the rate constraints (6.105). At all interior phase boundaries we require continuity in the state, control, parameter, and time, unless the values are fixed by the boundary conditions. Within each individual phase, the right-hand-side functions are defined by the explicit phase structure. And finally the goal is to maximize the parameter $p = h_{min}$ (in the last phase).

Using a linear guess for the dynamic variables, with 10 grid points in each phase, the optimal control problem was solved using $\mathbb{SOCS}$ and the computational performance is summarized in Table 6.12. The requested resolution for the mesh-refinement procedure was $\epsilon_{\max} \le 10^{-7}$, which was obtained after seven mesh-refinement iterations. The trapezoidal (TR) discretization was used for the early iterations, followed by either the compressed Hermite–Simpson (HC) or the separated Hermite–Simpson (HS) method, as indicated in the second column of the table. The number of grid points is given in the third column, the number of QP subproblems in the fourth column, and the accuracy achieved in the fifth column. The total CPU time for each iteration is given in the last column, with the entire solution obtained in 9.00 sec.

**Table 6.12.** *Windshear performance summary.*

| Iter. | Disc. | $M$ | NQP | $\epsilon_{\max}$ | CPU (sec) |
|---|---|---|---|---|---|
| 1 | (TR,TR,TR,TR,TR) | (10,10,10,10,10) | 10 | $2.29 \times 10^{-2}$ | $8.0 \times 10^{-2}$ |
| 2 | (TR,HC,TR,HC,TR) | (19,10,19,10,19) | 11 | $2.18 \times 10^{-3}$ | $1.6 \times 10^{-1}$ |
| 3 | (HS,HC,HS,HC,HS) | (37,10,37,19,37) | 9 | $3.02 \times 10^{-4}$ | $3.1 \times 10^{-1}$ |
| 4 | (HS,HS,HS,HS,HS) | (67,19,73,47,73) | 7 | $2.69 \times 10^{-5}$ | $6.6 \times 10^{-1}$ |
| 5 | (HS,HS,HS,HS,HS) | (67,19,115,47,145) | 19 | $2.67 \times 10^{-6}$ | $4.8 \times 10^{0}$ |
| 6 | (HS,HS,HS,HS,HS) | (67,19,229,47,145) | 7 | $1.32 \times 10^{-7}$ | $1.9 \times 10^{0}$ |
| 7 | (HC,HC,HC,HC,HC) | (34,10,120,24,73) | 4 | $9.87 \times 10^{-8}$ | $1.0 \times 10^{0}$ |
| Total | - | 261 | 67 | | 9.00 |

Figure 6.26 illustrates the optimal trajectory, and the optimal time histories for the four state variables are plotted in Figure 6.27. The phase boundaries are delineated by dotted vertical lines, and the lower bound on the altitude is shown with a dashed horizontal line. The optimal value achieved was $F^* = 491.85230$ (ft). The illustration suggests that there is a region in Phase 3 during which the altitude is on the minimum altitude boundary and a second touch point in Phase 5. In fact, during the extended boundary arc in Phase 3, the state inequality is *not* in the active set at all grid points. Instead there is a very

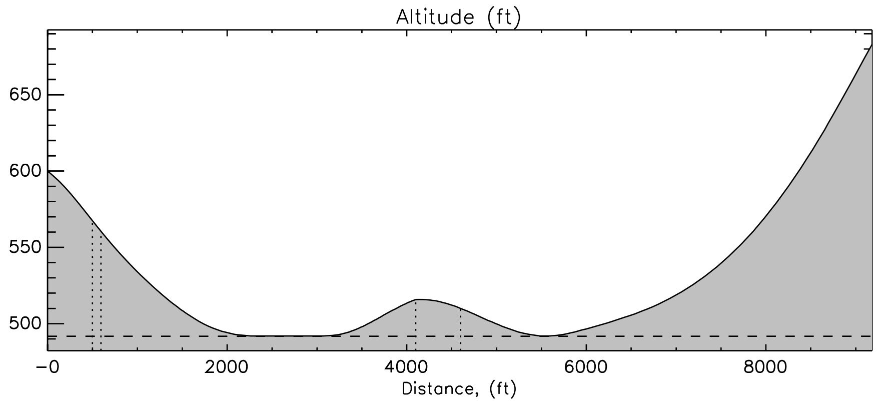


**Figure 6.26.** *Optimal trajectory profile.*

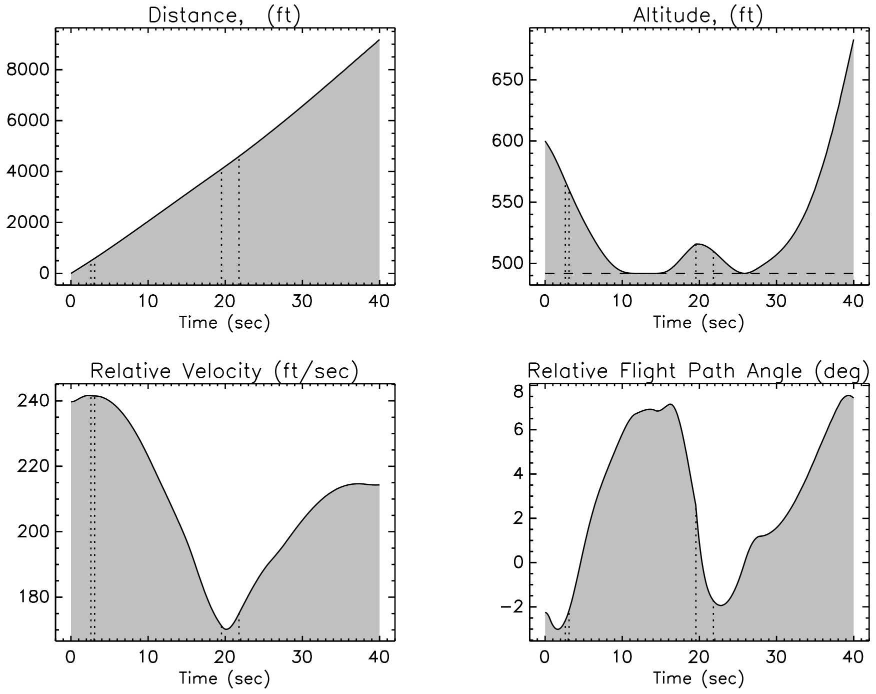


**Figure 6.27.** *Optimal state profile.*

small oscillation in the altitude, all within the mesh-refinement tolerance. This behavior is similar to that discussed in Section 4.12 (cf. Figure 4.29). Figure 6.28 illustrates the optimal control required to maximize the objective, as well as the corresponding angle of attack rate.

It is worthwhile to compare the results presented here with those found in [56]. Qualitatively the results are very similar, with the optimal trajectory having an arc that follows

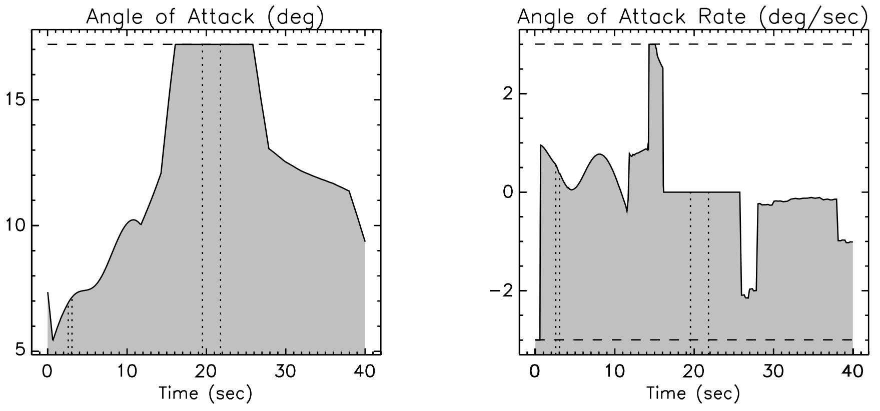


**Figure 6.28.** *Optimal control profile.*

the minimum altitude, and a subsequent single touch point. The angle of attack profile also compares favorably. Nevertheless, the solution techniques differ in three ways:

1. treatment of the objective function,

2. definition of the control variable,

3. treatment of the physical model data.

In particular, in [56], two additional states are introduced: one because $\dot{\alpha}$ is treated as the control, and a second to represent the minimum altitude. An indirect multiple shooting method was used to solve this problem, which required a sophisticated homotopy strategy in order to identify the correct switching structure and also deal with singular arcs. In contrast, the approach described here does not introduce additional state variables, and as such does not require the same derivative continuity in the angle of attack. Finally, our approach does not increase the index of the DAE, i.e., the order of the state constraints, and avoids the computational complexity associated with picking the correct arc structure as required by an indirect method.

## 6.7 Space Station Attitude Control

**Example 6.11** SPACE STATION ATTITUDE CONTROL. In his Master's thesis on the pseudospectral method, Pietz [142] presents results for an application that arises when trying to control the attitude of the International Space Station. A complete discussion of the problem can be found in Chapter 4 of the thesis. For convenience we present a brief summary of the optimal control problem. The state of the system is defined by three vectors, namely $\boldsymbol{\omega}(t)$, which is the angular velocity of the spacecraft with respect to an inertial reference frame, $\mathbf{r}(t)$, which are the Euler–Rodriguez parameters used to define the vehicle attitude, and $\mathbf{h}(t)$, which is the angular momentum of the control moment gyroscopes (CMGs). The torque $\mathbf{u}(t)$ is used to control the system. The dynamics are given by the differential-algebraic system

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}\left\{\boldsymbol{\tau}_{gg}(\mathbf{r}) - \boldsymbol{\omega}(t)^{\otimes}\left[\mathbf{J}\boldsymbol{\omega}(t) + \mathbf{h}(t)\right] - \mathbf{u}(t)\right\}, \tag{6.106}$$

$$\dot{\mathbf{r}} = \frac{1}{2}\left[\mathbf{r}(t)\mathbf{r}^{\mathsf{T}}(t) + \mathbf{I} + \mathbf{r}(t)^{\otimes}\right]\left[\boldsymbol{\omega}(t) - \boldsymbol{\omega}_0(\mathbf{r})\right], \tag{6.107}$$

$$\dot{\mathbf{h}} = \mathbf{u}(t), \tag{6.108}$$

$$0 \le h_{max} - \|\mathbf{h}\|. \tag{6.109}$$

The dynamics utilize the skew-symmetric cross product operator defined by

$$\mathbf{a}^{\otimes} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{6.110}$$

The gravity gradient torque is given by

$$\boldsymbol{\tau}_{gg}(\mathbf{r}) = 3\omega_{orb}^2 \mathbf{C}_3^{\otimes}\mathbf{J}\mathbf{C}_3, \tag{6.111}$$

where $\mathbf{C}_3$ is the third column of the rotation matrix

$$\mathbf{C} = \mathbf{I} + \frac{2}{1 + \mathbf{r}^{\mathsf{T}}\mathbf{r}}\left(\mathbf{r}^{\otimes}\mathbf{r}^{\otimes} - \mathbf{r}^{\otimes}\right). \tag{6.112}$$

Also

$$\boldsymbol{\omega}_0(\mathbf{r}) = -\omega_{orb}\mathbf{C}_2. \tag{6.113}$$

The state at the initial time is prescribed leading to the boundary conditions

$$\boldsymbol{\omega}(t_0) = \overline{\boldsymbol{\omega}}_0, \tag{6.114}$$

$$\mathbf{r}(t_0) = \overline{\mathbf{r}}_0, \tag{6.115}$$

$$\mathbf{h}(t_0) = \overline{\mathbf{h}}_0. \tag{6.116}$$

At the final time it is desirable to reach an orientation that can be maintained without utilizing additional control torque. This torque equilibrium attitude (TEA) is achieved by setting the right-hand sides of (6.106) and (6.107) to zero, with $\mathbf{u} = \mathbf{0}$, leading to the boundary conditions

$$\mathbf{0} = \mathbf{J}^{-1}\left\{\boldsymbol{\tau}_{gg}(\mathbf{r}(t_f)) - \boldsymbol{\omega}(t_f)^{\otimes}\left[\mathbf{J}\boldsymbol{\omega}(t_f) + \mathbf{h}(t_f)\right]\right\}, \tag{6.117}$$

$$\mathbf{0} = \frac{1}{2}\left[\mathbf{r}(t_f)\mathbf{r}^{\mathsf{T}}(t_f) + \mathbf{I} + \mathbf{r}(t_f)^{\otimes}\right]\left[\boldsymbol{\omega}(t_f) - \boldsymbol{\omega}_0(\mathbf{r}(t_f))\right]. \tag{6.118}$$

The goal is to choose the control vector $\mathbf{u}(t)$ to minimize the magnitude of the final momentum $\|\mathbf{h}\|$, or equivalently to minimize

$$F = \mathbf{h}^{\mathsf{T}}(t_f)\mathbf{h}(t_f). \tag{6.119}$$

Note that Pietz [142] treats $\|\mathbf{h}(t_f)\|$ as the objective function; however, this quantity is not differentiable at $\|\mathbf{h}(t_f)\| = 0$. In contrast, the objective given by (6.119) is differentiable everywhere and leads to an equivalent solution.

The problem description is completed by specifying the following parameters:

$$t_0 = 0, \qquad t_f = 1800,$$

$$h_{max} = 10000, \qquad \omega_{orb} = .06511\frac{\pi}{180},$$

| $\overline{\boldsymbol{\omega}}_0$ | $\overline{\mathbf{r}}_0$ | $\overline{\mathbf{h}}_0$ |
|---|---|---|
| $-9.5380685844896\times10^{-6}$ | $2.9963689649816\times10^{-3}$ | 5000 |
| $-1.1363312657036\times10^{-3}$ | $1.5334477761054\times10^{-1}$ | 5000 |
| $5.3472801108427\times10^{-6}$ | $3.8359805613992\times10^{-3}$ | 5000 |

$$\mathbf{J} = \begin{pmatrix} 2.80701911616\times10^{7} & 4.822509936\times10^{5} & -1.71675094448\times10^{7} \\ 4.822509936\times10^{5} & 9.5144639344\times10^{7} & 6.02604448\times10^{4} \\ -1.71675094448\times10^{7} & 6.02604448\times10^{4} & 7.6594401336\times10^{7} \end{pmatrix}.$$

The optimal control problem defined by (6.106)–(6.119) was solved using the $\mathbb{SOCS}$ software. Mesh refinement was performed to ensure the discretization error was below $10^{-7}$, which produces a solution with at least eight significant figures of accuracy. Constant values were used as initial guesses for all of the dynamic variables in the problem.

Unfortunately the original formulation suggested by Pietz [142] does not have a *unique* solution. In particular there are many control histories that will produce $F^* = \mathbf{h}^{*\mathsf{T}}(t_f)\mathbf{h}^*(t_f) = 0$. From an algorithmic perspective, when a nonunique problem is solved the standard recourse for the underlying nonlinear program is to *force* the projected Hessian matrix to be positive definite. Within $\mathbb{SOCS}$ this is achieved by introducing a nonzero Levenberg parameter to form a modified Hessian matrix. For the SNOPT algorithm used to obtain the results in Pietz [142], a quasi-Newton Hessian approximation is used which is forced to be positive definite. Thus in all cases the computational algorithms produce a solution; however, none of them is unique!

With this insight, the problem can be reformulated. One possible approach is to enforce the additional boundary conditions

$$\mathbf{h}(t_f) = \mathbf{0} \tag{6.120}$$

and then introduce a different objective that uniquely defines the optimal trajectory, e.g., minimize

$$F = 10^{-6}\int_{t_0}^{t_f} \mathbf{u}^{\mathsf{T}}(t)\mathbf{u}(t)dt. \tag{6.121}$$

This can be interpreted as a "minimum energy" criterion. Using this modified formulation, the problem was solved using $\mathbb{SOCS}$ and the solution is illustrated with a shaded region and solid line in Figures 6.29–6.31. The optimal value of the objective function is $F^* = 3.586883988$. It is also interesting that the computational behavior of the $\mathbb{SOCS}$ algorithm for this problem was quite well behaved as summarized in Table 6.13.

The thesis of Bhatt [41] extends the dynamic model proposed by Pietz to incorporate multibody effects and aerodynamic torques. Bhatt also suggests minimizing the peak CMG momentum as an objective function; i.e., instead of using (6.121) let us minimize

$$F = h_{max}, \tag{6.122}$$

which is the upper bound appearing in (6.109). We find that the solution to this minimax problem yields $F^* = 8660.2540$ (ft-lbf-sec), which is an improvement over the saturation limit $h_{max} = 10000$. However, to achieve this value the behavior of the other states and control become significantly less smooth as illustrated by the dotted lines in Figures 6.29–6.31.

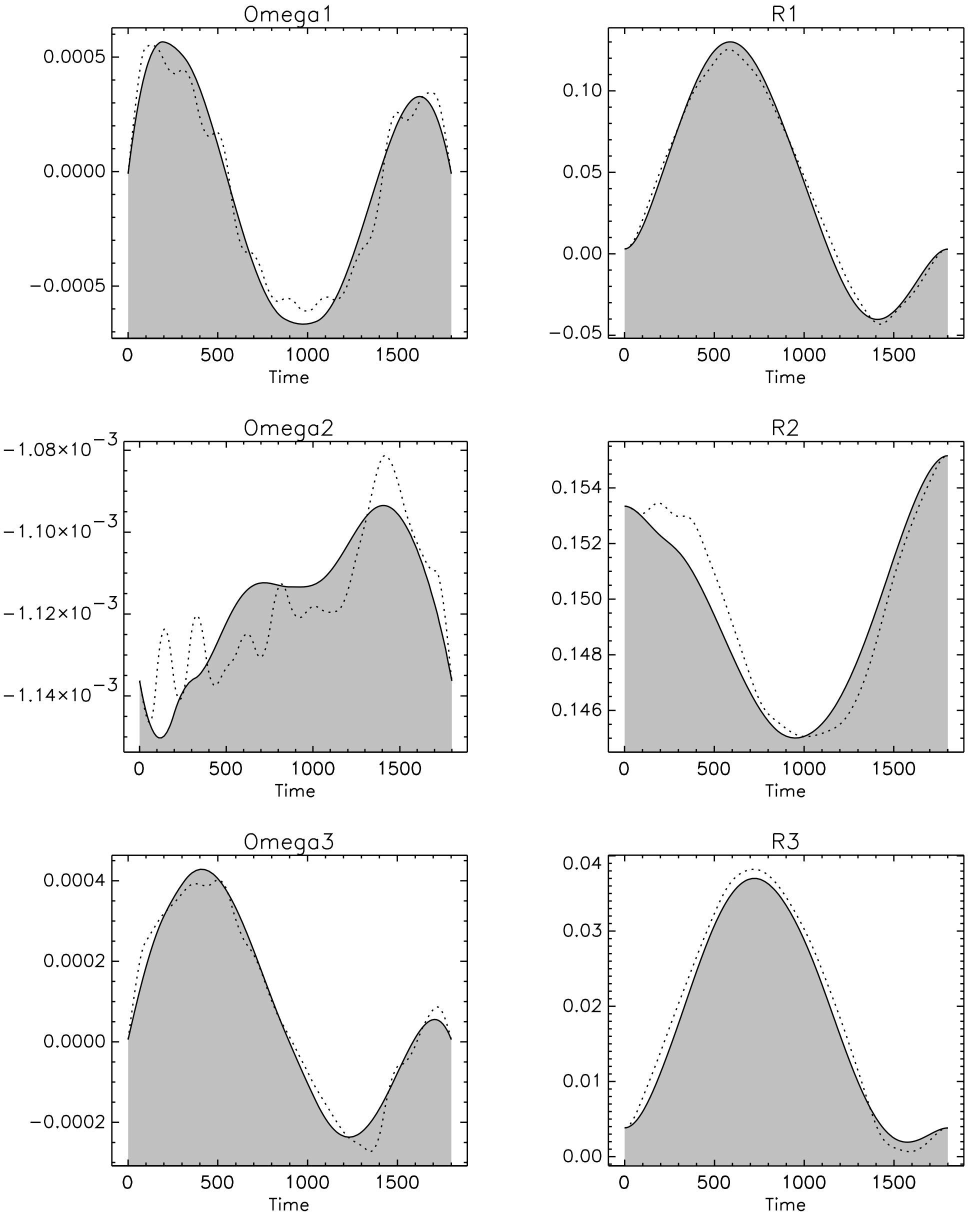


**Figure 6.29.** *ISS momentum dumping; states $\omega_k(t)$, $r_k(t)$.*

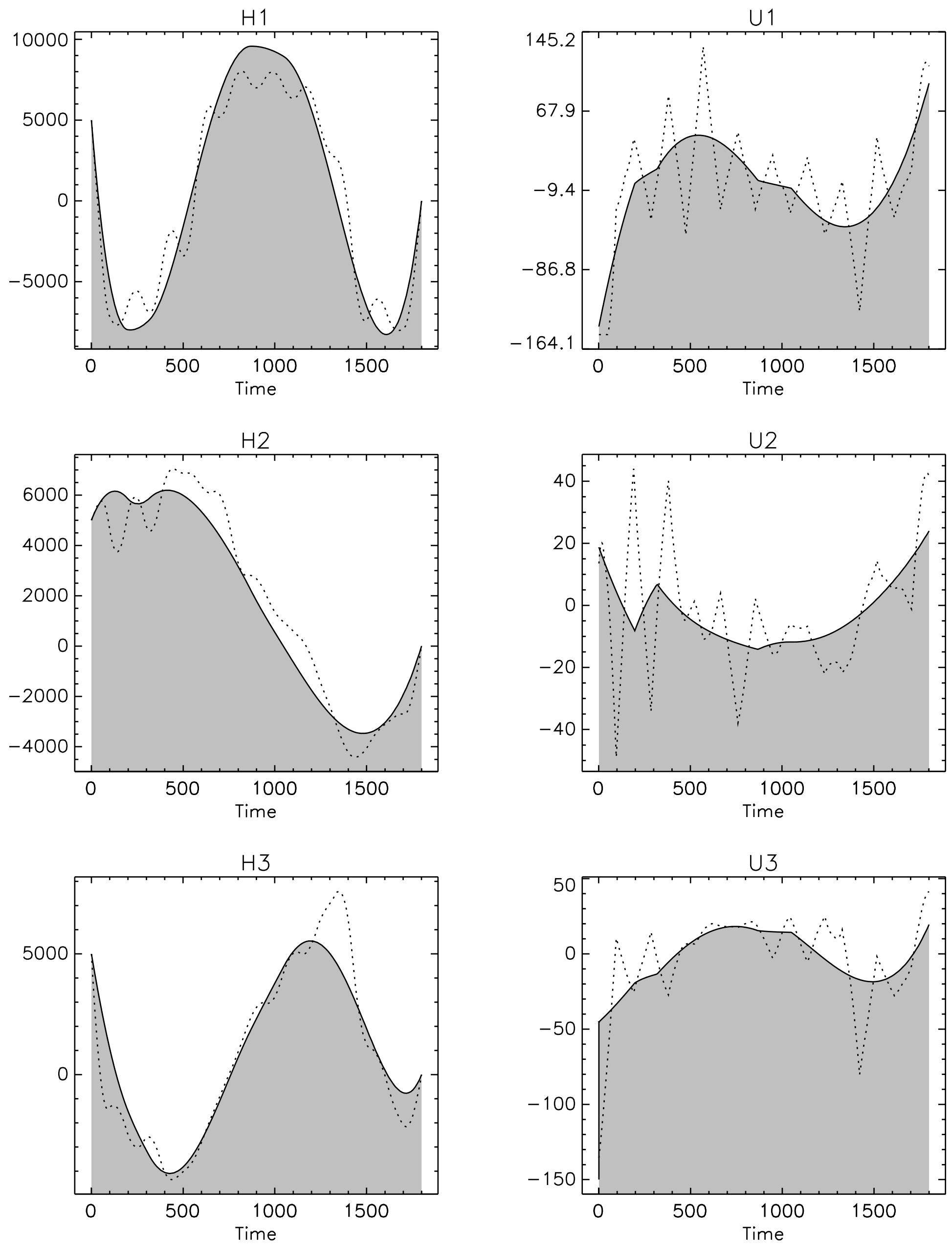


**Figure 6.30.** *ISS momentum dumping; states $h_k(t)$, controls $u_k(t)$.*

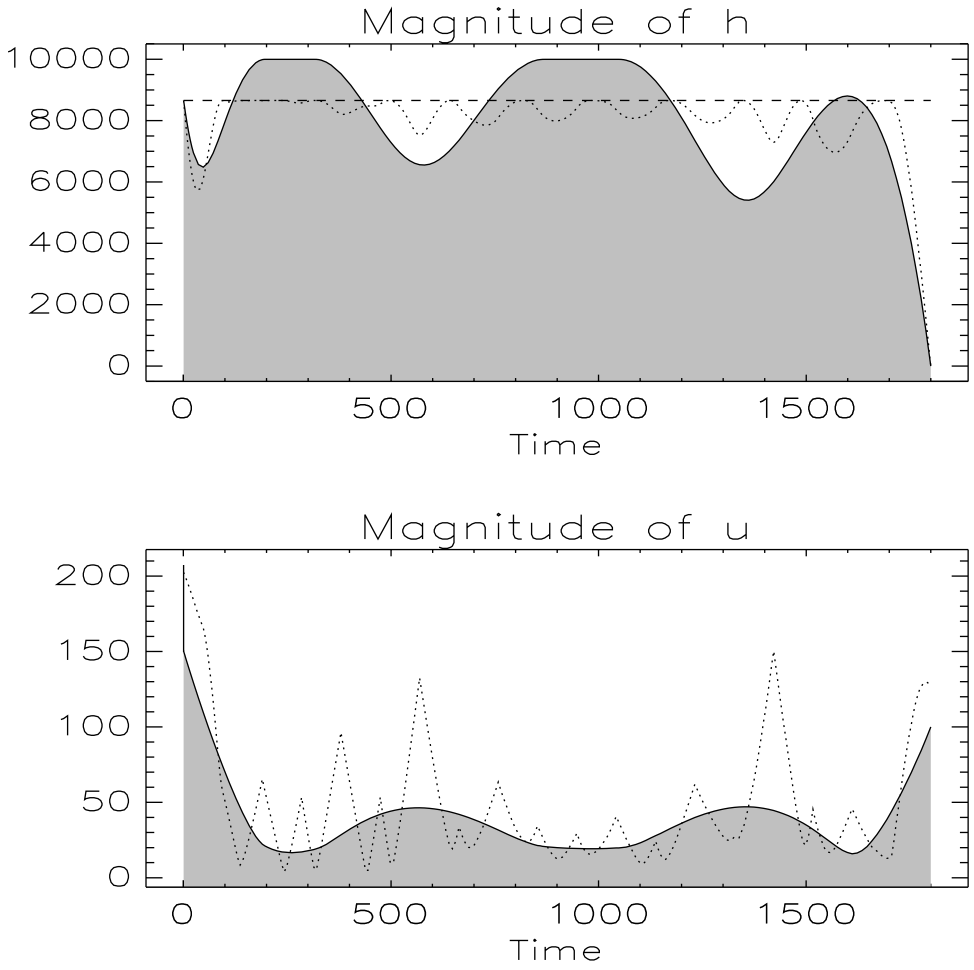


**Figure 6.31.** *ISS momentum dumping; magnitudes* $\|\mathbf{h}(t)\|$ *and* $\|\mathbf{u}(t)\|$.

**Table 6.13.** *Space station mesh-refinement summary.*

| $k$ | $M$ | NGC | NHC | NFE | NRHS | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|
| 1 | 20 | 13 | 2 | 365 | 12537 | $2.0720\times10^{-2}$ | $9.0000\times10^{-2}$ |
| 2 | 39 | 6 | 2 | 270 | 20658 | $2.1309\times10^{-3}$ | $1.0000\times10^{-1}$ |
| 3 | 39 | 9 | 2 | 1170 | 113588 | $4.2938\times10^{-5}$ | $3.7000\times10^{-1}$ |
| 4 | 77 | 4 | 1 | 544 | 128606 | $9.0924\times10^{-6}$ | $4.5000\times10^{-1}$ |
| 5 | 87 | 4 | 1 | 544 | 149154 | $1.3774\times10^{-6}$ | $5.4000\times10^{-1}$ |
| 6 | 169 | 3 | 1 | 489 | 267935 | $5.1763\times10^{-8}$ | $1.0500\times10^{0}$ |
| Total | 169 | 39 | 9 | 3382 | 692478 | | $2.6000\times10^{0}$ |

$k$ Mesh-Refinement Number; $M$ Number of Mesh Points
NGC Number of Gradient Calls; NHC Number of Hessian Calls
NFE Number of Function Evaluations; NRHS Number of Right-Hand-Side Evaluations
$\epsilon$ Discretization Error; Time (sec) CPU Time

## 6.8 Reorientation of an Asymmetric Rigid Body

**Example 6.12** REORIENTATION OF RIGID BODY. Fleming, Sekhavat, and Ross [80] describe an application of optimal control techniques to the attitude control of a spacecraft. The dynamics are given by the following system of ODEs:

$$\dot{q}_1 = \frac{1}{2}\left[\omega_1 q_4 - \omega_2 q_3 + \omega_3 q_2\right], \tag{6.123a}$$

$$\dot{q}_2 = \frac{1}{2}\left[\omega_1 q_3 + \omega_2 q_4 - \omega_3 q_1\right], \tag{6.123b}$$

$$\dot{q}_3 = \frac{1}{2}\left[-\omega_1 q_2 + \omega_2 q_1 + \omega_3 q_4\right], \tag{6.123c}$$

$$\dot{q}_4 = \frac{1}{2}\left[-\omega_1 q_1 - \omega_2 q_2 - \omega_3 q_3\right], \tag{6.123d}$$

$$\dot{\omega}_1 = \frac{u_1}{I_x} - \left(\frac{I_z - I_y}{I_x}\right)\omega_2\omega_3, \tag{6.123e}$$

$$\dot{\omega}_2 = \frac{u_2}{I_y} - \left(\frac{I_x - I_z}{I_y}\right)\omega_1\omega_3, \tag{6.123f}$$

$$\dot{\omega}_3 = \frac{u_3}{I_z} - \left(\frac{I_y - I_x}{I_z}\right)\omega_1\omega_2. \tag{6.123g}$$

In this formulation the orientation is defined using the four-parameter set

$$\mathbf{q}^{\mathsf{T}} = (q_1, q_2, q_3, q_4) = (\widetilde{\mathbf{q}}, q_4), \tag{6.124}$$

called Euler parameters or quaternions [116, pp. 18–31], where $\|\mathbf{q}\| = 1$. The angular velocity is defined by the vector $\boldsymbol{\omega}^{\mathsf{T}} = (\omega_1, \omega_2, \omega_3)$. For their example, the authors model the behavior of the NASA X-ray Timing Explorer (XTE) spacecraft in which case the moments of inertia are given by

$$I_x = 5621 \quad \text{Kg}\cdot\text{m}^2, \qquad I_y = 4547 \quad \text{Kg}\cdot\text{m}^2, \qquad I_z = 2364 \quad \text{Kg}\cdot\text{m}^2$$

and the control torques $\mathbf{u} = (u_1, u_2, u_3)$ are limited by

$$\|\mathbf{u}\|_\infty \le 50 \quad \text{N}\cdot\text{m}. \tag{6.125}$$

When using this formulation the differential variables are $\mathbf{y}^{\mathsf{T}} = (\mathbf{q}^{\mathsf{T}}, \boldsymbol{\omega}^{\mathsf{T}})$ and the algebraic variables are $\mathbf{u}$.

Since the quaternions must have norm one, an alternative is to omit differential equation (6.123d) from the ODE system (6.123a)–(6.123g) and replace it with an algebraic constraint leading to the DAE system

$$\dot{q}_1 = \frac{1}{2}\left[\omega_1 q_4 - \omega_2 q_3 + \omega_3 q_2\right], \tag{6.126a}$$

$$\dot{q}_2 = \frac{1}{2}\left[\omega_1 q_3 + \omega_2 q_4 - \omega_3 q_1\right], \tag{6.126b}$$

$$\dot{q}_3 = \frac{1}{2}\left[-\omega_1 q_2 + \omega_2 q_1 + \omega_3 q_4\right], \tag{6.126c}$$

$$\dot{\omega}_1 = \frac{u_1}{I_x} - \left(\frac{I_z - I_y}{I_x}\right)\omega_2\omega_3, \tag{6.126d}$$

$$\dot{\omega}_2 = \frac{u_2}{I_y} - \left(\frac{I_x - I_z}{I_y}\right)\omega_1\omega_3, \tag{6.126e}$$

$$\dot{\omega}_3 = \frac{u_3}{I_z} - \left(\frac{I_y - I_x}{I_z}\right)\omega_1\omega_2, \tag{6.126f}$$

$$0 = \|\mathbf{q}\| - 1. \tag{6.126g}$$

When posed in this manner the differential variables are $\mathbf{y}^{\mathsf{T}} = (\widetilde{\mathbf{q}}^{\mathsf{T}}, \boldsymbol{\omega}^{\mathsf{T}})$, and the algebraic variables are $\widehat{\mathbf{u}} = (q_4, \mathbf{u})$. Observe that the quaternion $q_4$ is treated as an algebraic state. The goal is to reorient the spacecraft by executing a 150 deg roll maneuver about the x-axis in minimum time. The initial and final states are specified by the boundary conditions[5]

$$\mathbf{q}^{\mathsf{T}}(0) = (0,0,0,1), \quad \mathbf{q}^{\mathsf{T}}(t_F) = \left(\sin\frac{\phi}{2}, 0, 0, \cos\frac{\phi}{2}\right), \quad \boldsymbol{\omega}(0) = \mathbf{0}, \quad \boldsymbol{\omega}(t_F) = \mathbf{0}, \tag{6.127}$$

where $\phi = 150°$ is the Euler axis rotation angle.

## 6.8.1 Computational Issues

As posed the problem illustrates a number of subtle computational issues. First, there are many local solutions to this problem all with the same value of the optimal time, namely $t_F^* = 28.630408$ (sec). Two typical solutions are plotted in Figures 6.32–6.34, and numerical experimentation suggests there are other solutions as well. Since the controls appear linearly in the problem, the solution is "bang-bang." However, the optimality conditions do not determine the particular *sequence* of boundary arcs. For example, notice that in Figure 6.34 the solution plotted as a solid line begins with $\mathbf{u}(0) = (+50, -50, +50)$. In contrast the second solution plotted with a dotted line begins with $\mathbf{u}(0) = (+50, -50, -50)$. The controls must be on either the upper or lower bound in order to satisfy the maximum principle; however, there is no restriction on *which bound!* Clearly, different combinations can lead to different arc sequences, which are all potential local optimizers.

Second, it is not clear what (if any) difference there is between the ODE formulation (6.123a)–(6.123g) and the DAE formulation (6.126a)–(6.126g). To address this let us consider the computational behavior on one particular case, in which we (arbitrarily) add the additional boundary conditions $\mathbf{u}^{\mathsf{T}}(0) = (+50, -50, +50)$ to the required conditions (6.127). For this experiment let us solve the problem using the standard mesh-refinement

[5]Boundary values correct a typographical error in [80].

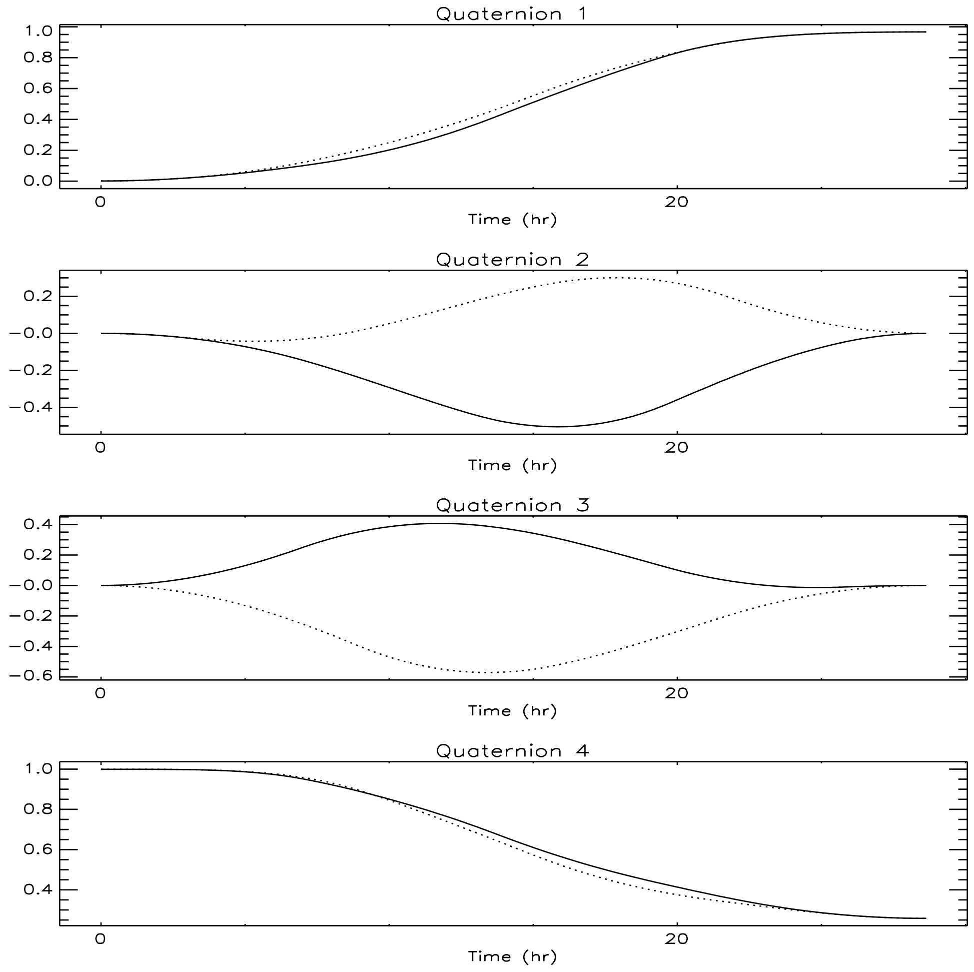


**Figure 6.32.** *Optimal quaternion history.*

procedure, beginning with a trapezoidal discretization, and a mesh-refinement tolerance of $\epsilon_{\max} = 10^{-7}$. As an initial guess we use a straight line between the initial and final conditions (with $\mathbf{u}^{(0)}(t_F) = \mathbf{0}$) and a guessed value $t_F^{(0)} = 30$. Table 6.14 summarizes the results of six different cases. For the first four cases the initial grid consisted of 20 equally distributed points, and the final two cases utilize an initial grid with 50 points. Cases 1 and 2 summarize the results using the DAE problem formulation with the SQP and Barrier NLP algorithms, respectively. The computational results strongly suggest that the DAE problem is superior. When the problem is formulated as an ODE (using (6.123a)–(6.123g)) we observe either ill-conditioning and/or slow convergence, regardless of the choice of NLP algorithm. The difficulty can be attributed to a fundamental numerical problem associated with quaternions. The DAE formulation explicitly ensures satisfaction of the normalization condition $\|\mathbf{q}(t)\| = 1$ at every grid point, even when the mesh is coarse. In contrast to en-

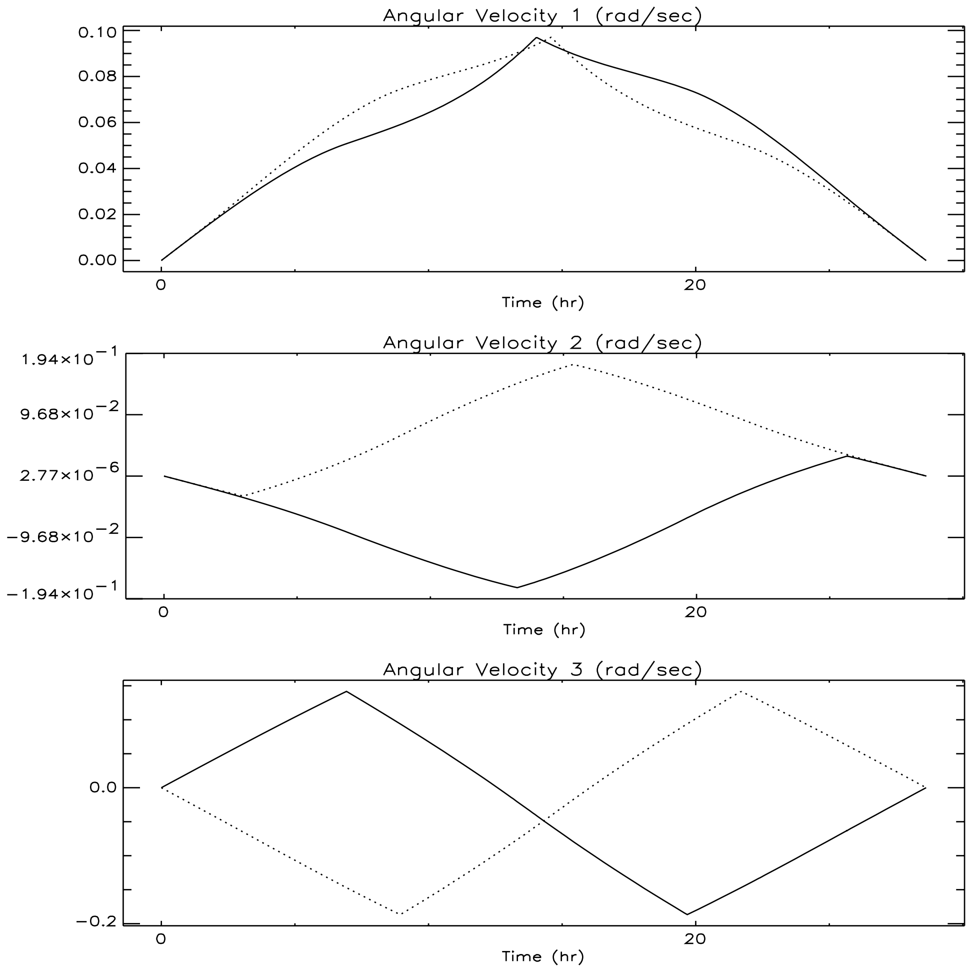


**Figure 6.33.** *Optimal velocity history.*

sure the normalization holds when using the ODE formulation, it is necessary to integrate the ODE *very* accurately. In other words, one cannot expect $\|\mathbf{q}(t)\| = 1$ unless the ODEs (6.123a)–(6.123d) are solved accurately. In particular, even if $\|\mathbf{q}(0)\| = 1$, we must expect that integration error will cause $\|\mathbf{q}(t_F)\| \neq 1$, thereby leading to an ill-conditioned BVP.

It is instructive to examine the behavior of the normalization condition $\|\mathbf{q}\| = 1$ after a single step. Specifically consider a single trapezoidal step

$$0 = \mathbf{y}_1 - \mathbf{y}_0 - \frac{h}{2}(\mathbf{f}_0 + \mathbf{f}_1), \tag{6.128}$$

where $\mathbf{y}^\mathsf{T} = (\mathbf{q}^\mathsf{T}, \boldsymbol{\omega}^\mathsf{T})$ and $\mathbf{f}$ is defined by the right-hand sides of the ODE system (6.123a)–(6.123g). Let us assume $\mathbf{y}_0^\mathsf{T} = (0,0,0,1,0,0,0)$ as in (6.127) and further assume

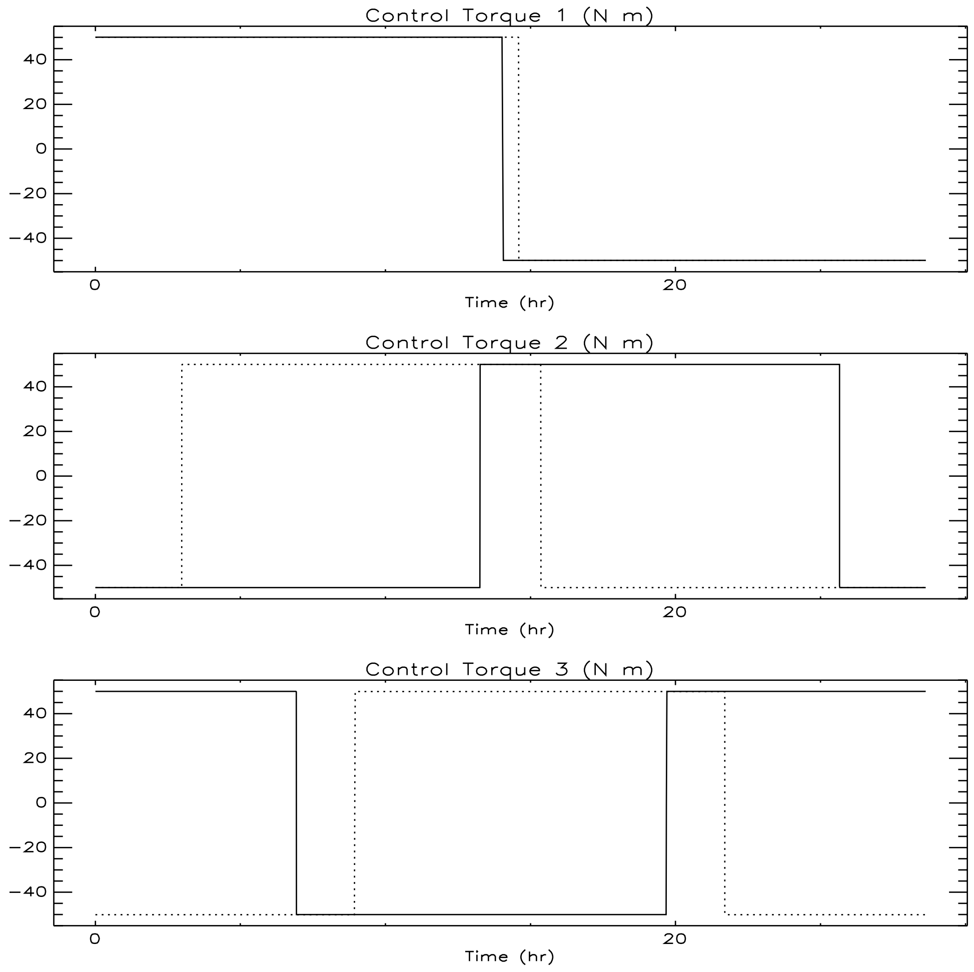


**Figure 6.34.** *Optimal control torque history.*

$\mathbf{u}^{\mathsf{T}} = (+50, -50, +50)$ over the entire step. Clearly $\|\mathbf{q}(0)\| = 1$. Figure 6.35 illustrates the normalization error in the first four components of $\mathbf{y}$ for a range of stepsizes. Clearly, the numerical integration does not preserve normalization *even over a single step.* Unfortunately, this result has catastrophic implications when present in the boundary value context. In particular, if the (correctly defined) boundary conditions require $\|\mathbf{q}(0)\| = 1$ and $\|\mathbf{q}(h)\| = 1$, this cannot be achieved unless the numerical integration scheme has zero error! As a consequence when a direct transcription method is used to solve the BVP, the constraints are degenerate, leading to a rank-deficient Jacobian matrix, and the overall success or failure of the method will depend on how the NLP treats rank deficiency. Since this difficulty is not unique to the trapezoidal discretization, or, for that matter, the direct transcription method, it is preferable to simply pose the problem as a DAE.

**Table 6.14.** *Performance comparison.*

| Formulation | NLP | Ref. Iter. | Grid Pts. | Func. Eval. | Time (sec) |
|---|---|---|---|---|---|
| DAE | SQP | 9 | 136 | 10572 | 25.01 |
| DAE | Barrier | 9 | 130 | 37407 | 19.52 |
| ODE | SQP | 8[a] | 353 | 28999 | 121.9 |
| ODE | Barrier | 1[b] | 20 | 3243 | .58 |
| ODE | SQP | 9[c] | 349 | 17340 | 131.02 |
| ODE | Barrier | 1[d] | 50 | 3597 | 1.56 |

[a] Abnormal Termination (Incorrect Inertia Ill-Conditioning
[b] Abnormal Termination (Ill-Conditioning) Final Objective $t_F^* = 28.877792$.
[c] Optimal Objective $t_F^* = 28.661727$.
[d] Abnormal Termination; Final Objective $t_F^* = 28.664134$.

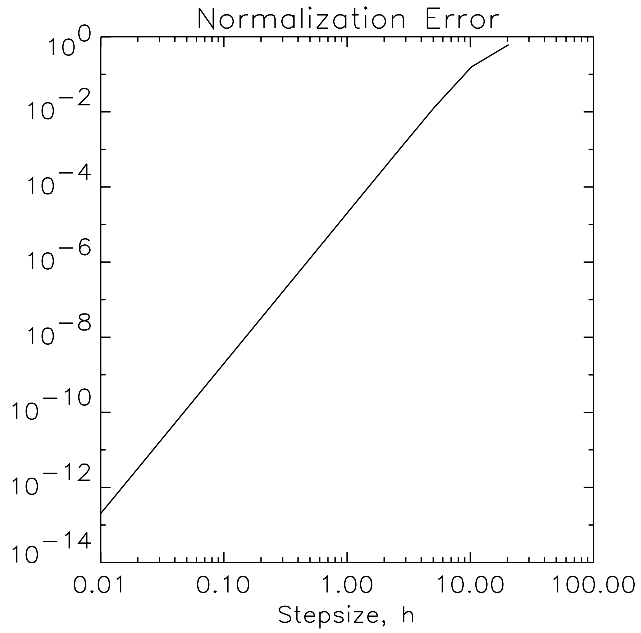


**Figure 6.35.** *Stepwise normalization error growth* $1 - \|\mathbf{q}(h)\|$.

## 6.9 Industrial Robot

**Example 6.13** Industrial Robot. An interesting example describing the motion of an industrial robot is presented in [169]. The machine is shown in Figure 6.36 and is called the Manutec r3 [138]. It has six degrees of freedom, although only three degrees of freedom are considered in this formulation since they adequately describe the position of the tool center point. The dynamics are described by the second order system

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{f}(\dot{\mathbf{q}}, \mathbf{q}, \mathbf{u}), \tag{6.129}$$

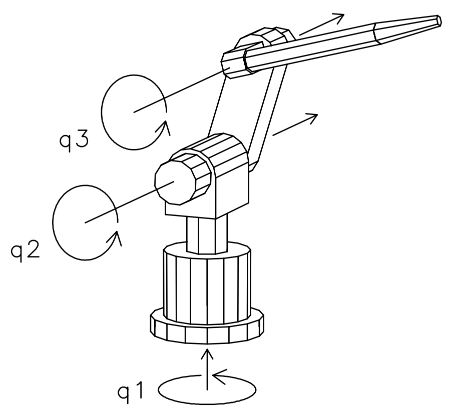


**Figure 6.36.** *Manutec r3 robot.*

where the vector $\mathbf{q} = [q_1(t), q_2(t), q_3(t)]^{\mathsf{T}}$ defines the relative angles between the robot arms. The torque voltages $\mathbf{u} = [u_1(t), u_2(t), u_3(t)]^{\mathsf{T}}$ are applied to control the robot motors. The symmetric positive definite mass matrix $\mathbf{M}(\mathbf{q})$ involves very complicated expressions of the state $\mathbf{q}$, as does the function $\mathbf{f}$, which defines the moments caused by Coriolis, centrifugal, and gravitational forces. A more complete description of the quantities in this *multibody system* can be found in [169]. If the angular velocities are denoted by $\mathbf{v}$, then (6.129) can be written as the first order system

$$\dot{\mathbf{q}} = \mathbf{v}, \tag{6.130}$$
$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} = \mathbf{f}(\mathbf{v}, \mathbf{q}, \mathbf{u}). \tag{6.131}$$

Because the matrix $\mathbf{M}(\mathbf{q})$ has a special tree structure, it is possible to construct the inverse using an $\mathcal{O}(n)$ algorithm ($n = 3$), as described in [138]. Thus, we obtain the semiexplicit first order system

$$\dot{\mathbf{q}} = \mathbf{v}, \tag{6.132}$$
$$\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{q})\mathbf{f}(\mathbf{v}, \mathbf{q}, \mathbf{u}). \tag{6.133}$$

It should be emphasized that the right-hand side of (6.133) involves *very complicated* expressions that are often generated automatically by simulation software systems. The computational results use software graciously provided by O. von Stryk to compute these differential equations.

The goal is to construct a trajectory for the robot tool tip that goes from one specified location to another (a "point-to-point" path). Also, we want the tool to begin and end the trajectory at rest, i.e., with zero velocity. Thus, we have the boundary conditions

$$\mathbf{q}(0) = (0, -1.5, 0)^{\mathsf{T}} \text{ (rad)}, \qquad \mathbf{v}(0) = \mathbf{0} \text{ (rad/sec)},$$
$$\mathbf{q}(t_F) = (1, -1.95, 1)^{\mathsf{T}} \text{ (rad)}, \qquad \mathbf{v}(t_F) = \mathbf{0} \text{ (rad/sec)}.$$

This particular tool also has physical limits that restrict the state and control variables.

Specifically,

$$\begin{aligned}
\|\mathbf{u}(t)\|_\infty &\le 7.5 \text{ (V)},\\
|q_1(t)| &\le 2.97 \text{ (rad)},\\
|q_2(t)| &\le 2.01 \text{ (rad)},\\
|q_3(t)| &\le 2.86 \text{ (rad)},\\
|v_1(t)| &\le 3.0 \text{ (rad/sec)},\\
|v_2(t)| &\le 1.5 \text{ (rad/sec)},\\
|v_3(t)| &\le 5.2 \text{ (rad/sec)}.
\end{aligned}$$

Three different objective functions are considered in [169], the first two being minimum time

$$J = t_F \tag{6.134}$$

and minimum energy

$$J = \int_0^{t_F} \mathbf{u}^\mathsf{T}(t)\mathbf{u}(t)dt \tag{6.135}$$

for a fixed final time $t_F$.

The minimum energy problem is relatively straightforward and of significant practical value. However, a number of the concepts described in this book are illustrated by the minimum time problem, and so it will be the primary focus of the discussion. This example is useful because, as the authors suggest in their abstract,

> The highly accurate solutions reported in this paper may also serve as benchmark problems for other methods.

What is most admirable is *how* the authors obtained the solutions using an indirect multiple shooting method. As stated in [169],

> The main drawbacks when applying the multiple shooting method in the numerical solution of optimal control problems are, 1. the derivation of the necessary conditions (e.g., the adjoint differential equations), 2. the estimation of the optimal switching structure, and 3. the estimation of an appropriate initial guess of the unknown state and adjoint variables $x(t), \lambda(t), \eta(t)$ in order to start the iteration process.

Briefly, their approach was to derive the adjoint equations using the software tool Maple, which produced over 4000 lines of FORTRAN code. Then a direct transcription method (with a coarse mesh) was used to construct both an initial guess for the switching structure and estimates for the state and adjoint variables. Finally, this information was used to initialize an indirect multiple shooting method. Our goal here is to solve the same problem *without* deriving the adjoint equations and guessing the adjoint variables.

As posed, the minimum time problem presents two major difficulties. First, the control variable appears linearly in the differential equations. Consequently, it is most likely that the control will be bang-bang, as discussed in Example 4.11. Singular subarcs, as described in Example 4.9, are also possible. However, for the specific boundary conditions given here, they do not appear. A more complete analysis can be found in [169]. Second, when the limits on the angular velocity are active, the resulting DAE has index two. Since the Jacobian matrix is singular on the state constraints, the control variable is not well defined by the usual necessary conditions, as illustrated by Example 4.10. The approach we will follow is to first estimate the switching structure and then solve a multiphase problem with index reduction on the phases corresponding to the active state boundaries.

The first step is to compute an estimate for the switching structure. To do this, let us solve a modified version of the minimum time problem. Specifically, minimize

$$J = t_F + \rho \int_0^{t_F} \mathbf{u}^{\mathsf{T}}(t)\mathbf{u}(t)dt, \tag{6.136}$$

where the "small" parameter $\rho = 1 \times 10^{-5}$ is chosen to regularize the problem. By introducing this quadratic regularization, the control becomes uniquely defined and the solution should be "close" to the minimum time problem. It is not necessary (or desirable) to solve this modified problem to a high degree of precision because the primary goal is to construct the appropriate switching structure. Table 6.15 summarizes the results, which were intentionally terminated after seven mesh-refinement iterations. In fact, the computed approximation to the minimum time is 0.495196288 sec, which agrees with the true optimum value 0.49518904 sec to four significant figures. Although there is very little "visible" difference in the objective function, there is a visible difference in the control history. It is interesting to note that the mesh-refinement algorithm presented in Section 4.6.9 selected the separated Simpson discretization because the function evaluations were relatively expensive.

**Table 6.15.** *Minimum time with regularization.*

| Iter. | Disc. | $M$ | GE | HE | FE | RHS | $\epsilon_{\max}$ | CPU (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | TR | 10 | 16 | 10 | 619 | 6190 | $0.11 \times 10^{-1}$ | $0.27 \times 10^{1}$ |
| 2 | TR | 19 | 10 | 8 | 469 | 8911 | $0.86 \times 10^{-2}$ | $0.44 \times 10^{1}$ |
| 3 | HS | 19 | 7 | 5 | 304 | 11248 | $0.23 \times 10^{-2}$ | $0.14 \times 10^{2}$ |
| 4 | HS | 37 | 7 | 5 | 304 | 22192 | $0.20 \times 10^{-2}$ | $0.37 \times 10^{2}$ |
| 5 | HS | 46 | 8 | 6 | 359 | 32669 | $0.36 \times 10^{-3}$ | $0.57 \times 10^{2}$ |
| 6 | HS | 84 | 7 | 5 | 304 | 50768 | $0.50 \times 10^{-4}$ | $0.19 \times 10^{3}$ |
| 7 | HS | 97 | 5 | 3 | 194 | 37442 | $0.13 \times 10^{-4}$ | $0.19 \times 10^{3}$ |
| Total |  | 97 | 60 | 42 | 2553 | 169420 |  | 493.53 |

The solution to the relaxed problem is illustrated by the dashed line in Figure 6.37. The relative accuracy of the solution is dominated by the discretization error in the neighborhood of the discontinuities. However, instead of trying to improve the accuracy by adding grid points, it is better to explicitly introduce this discontinuous behavior using separate phases. Examination of the solution provides an estimate for the switching structure. Thus, we are led to model the behavior using nine distinct phases with each phase characterized by a different set of active path constraints. Table 6.16 summarizes the phase-by-phase situation. Thus, in phase 1, the first three angular velocities $v_1, v_2$, and $v_3$ are free, whereas the first two control variables $u_1$ and $u_2$ are at their minimum values, while the final control $u_3$ is a maximum.

To illustrate how the multiple-phase structure is exploited, let us consider the dynamics in phase 3. In phase 3, state variable $v_2$ is on its lower bound and state variable $v_3$ is on

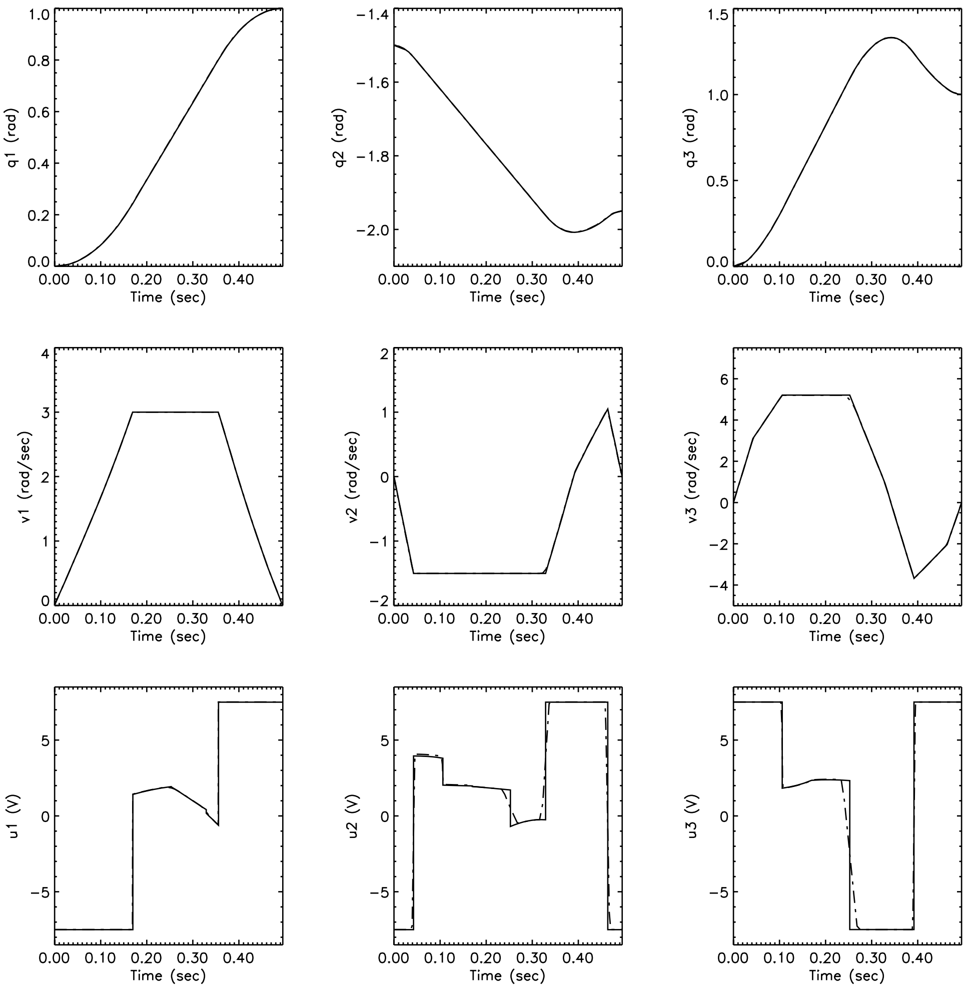


**Figure 6.37.** *Robot solution.*

**Table 6.16.** *Minimum time switching structure.*

| Phase | $v_1$ | $v_2$ | $v_3$ | $u_1$ | $u_2$ | $u_3$ |
|---|---|---|---|---|---|---|
| 1 | free | free | free | min | min | max |
| 2 | free | min | free | min | free | max |
| 3 | free | min | max | min | free | free |
| 4 | max | min | max | free | free | free |
| 5 | max | min | free | free | free | min |
| 6 | max | free | free | free | max | min |
| 7 | free | free | free | max | max | min |
| 8 | free | free | free | max | max | max |
| 9 | free | free | free | max | min | max |

its upper bound. Thus, we must impose the algebraic constraints

$$0 = v_2(t) - v_{2L}, \tag{6.137}$$
$$0 = v_3(t) - v_{3U}, \tag{6.138}$$

where the lower bound $v_{2L} = -1.5$ and the upper bound $v_{3U} = 5.2$. Neither (6.137) nor (6.138) explicitly contains a control variable and, as such, both are index-two path constraints. To reduce the index, we must differentiate with respect to time giving

$$0 = \dot{v}_2(t), \tag{6.139}$$
$$0 = \dot{v}_3(t). \tag{6.140}$$

Thus, from (6.132) and (6.133) we must have

$$\dot{q}_1 = v_1(t), \tag{6.141}$$
$$\dot{q}_2 = v_{2L}, \tag{6.142}$$
$$\dot{q}_3 = v_{3U}, \tag{6.143}$$
$$\dot{v}_1 = \left\{ \mathbf{M}^{-1}(\mathbf{q})\mathbf{f}(\mathbf{v},\mathbf{q},\mathbf{u}) \right\}_1, \tag{6.144}$$
$$0 = \left\{ \mathbf{M}^{-1}(\mathbf{q})\mathbf{f}(\mathbf{v},\mathbf{q},\mathbf{u}) \right\}_2, \tag{6.145}$$
$$0 = \left\{ \mathbf{M}^{-1}(\mathbf{q})\mathbf{f}(\mathbf{v},\mathbf{q},\mathbf{u}) \right\}_3. \tag{6.146}$$

Since $\mathbf{v} = (v_1(t), v_{2L}, v_{3U})$ and $\mathbf{u} = (u_{1L}, u_2(t), u_3(t))$, the dynamics in phase 3 are completely defined by this (index-one) DAE system. Thus, phase 3 can be modeled using four (not six) differential equations and two (not three) algebraic equations. The differential (state) variables in this phase are

$$(q_1(t), q_2(t), q_3(t), v_1(t))$$

and the algebraic (control) variables are $(u_2(t), u_3(t))$. The beginning and end of the phase are not specified, so we must treat the values $t_I^{(3)}$ and $t_F^{(3)}$ as additional NLP variables, where phase 3 is defined on the domain $t_I^{(3)} \le t \le t_F^{(3)}$. The phase description is complete when boundary conditions are specified. Continuity at the beginning and end of the phase is enforced by imposing the boundary conditions

$$\begin{aligned}
t_F^{(2)} &= t_I^{(3)}, & t_F^{(3)} &= t_I^{(4)},\\
q_1(t_F^{(2)}) &= q_1(t_I^{(3)}), & q_1(t_F^{(3)}) &= q_1(t_I^{(4)}),\\
q_2(t_F^{(2)}) &= q_2(t_I^{(3)}), & q_2(t_F^{(3)}) &= q_2(t_I^{(4)}),\\
q_3(t_F^{(2)}) &= q_3(t_I^{(3)}), & q_3(t_F^{(3)}) &= q_3(t_I^{(4)}),\\
v_1(t_F^{(2)}) &= v_1(t_I^{(3)}), & v_1(t_F^{(3)}) &= v_1(t_I^{(4)}).
\end{aligned}$$

To ensure the proper transition for the state-constrained variables between phases 2, 3, and 4, at the end of phase 2 we must impose the condition $v_3(t_F^{(2)}) = v_{3U}$ and, at the end of phase 3, it is necessary that $v_1(t_F^{(3)}) = v_{1U}$. Observe that no continuity conditions are imposed on the control variables. In fact, one benefit of the multiphase treatment of this problem is that it does permit accurate modeling of the control discontinuities.

It is worth noting that the technique presented here models phase 3 with a reduced set of state and control variables. Computationally, this is advantageous because the size of the transcribed problem has been reduced. However, it may be more convenient to implement a problem that has the same number of states and controls on each phase. This can be achieved by adding the required path equality constraints on each phase and then simply solving a larger NLP problem. This method is outlined in the doctoral thesis of O. von Stryk.

For brevity, we omit an explicit description of the approach for all nine phases. However, it should be clear that with a known switching structure it is relatively straightforward to implement the correct conditions in a computational tool such as $\mathbb{SOCS}$. When this information is explicitly incorporated into a nine-phase formulation, the resulting nonlinear program is well-posed and leads to the solution illustrated by the solid line in Figure 6.37. Table 6.17, which summarizes the behavior of the algorithm, clearly demonstrates the benefit of incorporating the switching structure into the formulation.

**Table 6.17.** *Minimum time with switching structure.*

| Iter. | Disc. | $M$ | GE | HE | FE | RHS | $\epsilon_{\max}$ | CPU (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | TR | 45 | 4 | 0 | 34 | 1530 | $0.18 \times 10^{-4}$ | $0.34 \times 10^{1}$ |
| 2 | HC | 45 | 2 | 0 | 34 | 2754 | $0.25 \times 10^{-6}$ | $0.22 \times 10^{1}$ |
| 3 | HC | 57 | 1 | 0 | 18 | 1890 | $0.66 \times 10^{-7}$ | $0.23 \times 10^{1}$ |
| Total | | 57 | 7 | 0 | 86 | 6174 | | 7.81 |

As a final point of interest, it is worth comparing the accuracy of the solutions computed using the direct transcription approach with those obtained by the benchmark indirect multiple shooting approach [169]. The optimal objective function values are given in Table 6.18 for both methods, with the significant figures that agree underlined. The results agree to seven significant figures in both cases, which also validates the reliability of the discretization error $\epsilon_{\max} \leq 1 \times 10^{-7}$. More important is the fact that the direct transcription results were obtained without computing the adjoint equations and estimating values for the adjoint variables. This is especially encouraging since it suggests that accurate results can be obtained for many practical applications even when it is too complicated to form the adjoint equations.

**Table 6.18.** *Accuracy comparison.*

| Problem | Indirect multiple shooting | Direct transcription |
|---|---|---|
| Min. energy | $\underline{20.40424}7$ | $\underline{20.40424}581$ |
| Min. time | $\underline{0.4951890}4$ | $\underline{0.4951890}37$ |

## 6.10 Multibody Mechanism

**Example 6.14** ANDREW'S SQUEEZER MECHANISM. Hairer and Wanner [107, pp. 530–542] describe a very nice example of a *multibody system* called "Andrew's squeezer mechanism" and have graciously supplied a software implementation of the relevant equations. The problem is used as a benchmark for testing a number of different multibody simulation codes as described in [154]. For the sake of brevity, we omit a detailed description of the problem and refer the reader to [107]. The multibody dynamics are described by a second order system similar to (6.129):

$$\mathbf{M}(\mathbf{p})\ddot{\mathbf{p}} = \mathbf{f}(\dot{\mathbf{p}},\mathbf{p},u) - \mathbf{G}^{\mathsf{T}}(\mathbf{p})\boldsymbol{\lambda}, \tag{6.147}$$

$$\mathbf{0} = \mathbf{g}(\mathbf{p}). \tag{6.148}$$

For this example, the "position" vector $\mathbf{p} = (\beta,\Theta,\gamma,\Phi,\delta,\Omega,\varepsilon)^{\mathsf{T}}$ consists of seven angles that define the orientation of the seven-body mechanism. The bodies are linked together via the algebraic constraints (6.148). The symmetric *mass matrix* **M** is tridiagonal, and the derivation of these equations is often facilitated by noting that

$$m_{ij} = \frac{\partial^2 T}{\partial \dot{p}_i \partial \dot{p}_j},$$

where $T$ is the kinetic energy of the system. The mechanism is driven by a motor whose drive torque is given by $u$. In [107], the torque $u_0 = 0.033$ Nm is treated as a constant, in which case this is simply an IVP. For the sake of illustration, let us assume that the drive torque is a control variable that can be adjusted as a function of time. Let us impose bounds

$$0 \le u(t) \le 0.066 \qquad (= 2 \times 0.033) \tag{6.149}$$

and then try to compute the control such that

$$J = \frac{1}{t_F u_0^2} \int_0^{t_F} u^2(t)dt \tag{6.150}$$

is minimized over the time domain $0 \le t \le t_F = 0.03$ ms.

First, let us convert (6.147) from a second order implicit form to a first order semiexplicit DAE system giving

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{6.151}$$

$$\dot{\mathbf{v}} = \mathbf{q}, \tag{6.152}$$

$$\mathbf{0} = \mathbf{M}(\mathbf{p})\mathbf{q} - \mathbf{f}(\mathbf{v},\mathbf{p},u) + \mathbf{G}^{\mathsf{T}}(\mathbf{p})\boldsymbol{\lambda}, \tag{6.153}$$

$$\mathbf{0} = \mathbf{g}(\mathbf{p}). \tag{6.154}$$

In this formulation, **q**, $\boldsymbol{\lambda}$, and $u$ are the algebraic variables **u**, whereas **p** and **v** are the differential variables **y**. Observe that the complete set of algebraic variables denoted by **u** includes the "real" control $u$ as well as the other algebraic variables **q** and $\boldsymbol{\lambda}$. If we differentiate (6.154) once with respect to time, we find

$$\mathbf{0} = \frac{d}{dt}\left[\mathbf{g}(\mathbf{p})\right] = \mathbf{G}(\mathbf{p})\dot{\mathbf{p}} = \mathbf{G}(\mathbf{p})\mathbf{v}. \tag{6.155}$$

A second differentiation with respect to time yields

$$\mathbf{0} = \frac{d^2}{dt^2}\left[\mathbf{g}(\mathbf{p})\right] = \frac{d}{dt}\left[\mathbf{G}(\mathbf{p})\mathbf{v}\right] = \dot{\mathbf{G}}(\mathbf{p})\mathbf{v} + \mathbf{G}(\mathbf{p})\dot{\mathbf{v}} = \mathbf{g}_{pp}(\mathbf{p})(\mathbf{v},\mathbf{v}) + \mathbf{G}(\mathbf{p})\mathbf{q}. \tag{6.156}$$

Since the algebraic variable $\mathbf{q}$ appears in this *acceleration-level* constraint, no additional derivatives are needed. Observe that in its original form the problem is index three, and this illustrates the process of *index reduction*. Collecting results yields the index-one DAE system

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{6.157}$$

$$\dot{\mathbf{v}} = \mathbf{q}, \tag{6.158}$$

$$\mathbf{0} = \mathbf{M}(\mathbf{p})\mathbf{q} - \mathbf{f}(\mathbf{v},\mathbf{p},u) + \mathbf{G}^{\mathrm{T}}(\mathbf{p})\boldsymbol{\lambda}, \tag{6.159}$$

$$\mathbf{0} = \mathbf{g}_{pp}(\mathbf{p})(\mathbf{v},\mathbf{v}) + \mathbf{G}(\mathbf{p})\mathbf{q}. \tag{6.160}$$

Since the original problem was index three, some care must be exercised when defining initial conditions to ensure that they are consistent. First, the initial conditions must satisfy (6.154), so following [107] we choose one of the position variables $\Theta(0) = 0$ and then compute the remaining six such that (6.154) is satisfied. The velocity-level constraint $\mathbf{0} = \mathbf{G}(\mathbf{p})\mathbf{v}$ is satisfied if we put $\mathbf{v}(0) = \mathbf{0}$. Finally, it remains to specify initial values for the algebraic variables $\mathbf{q}$, $\boldsymbol{\lambda}$, and $u$ such that (6.159) and (6.160) hold. Rewriting these equations gives

$$\begin{bmatrix} \mathbf{M}(\mathbf{p}) & \mathbf{G}^{\mathrm{T}}(\mathbf{p}) \\ \mathbf{G}(\mathbf{p}) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\mathbf{v},\mathbf{p},u) \\ -\mathbf{g}_{pp}(\mathbf{p})(\mathbf{v},\mathbf{v}) \end{bmatrix}. \tag{6.161}$$

Thus, for given values of the position $\mathbf{p}$, velocity $\mathbf{v}$, and control $u$, we can solve for the corresponding values of $\mathbf{q}$ and $\boldsymbol{\lambda}$. The initial values computed this way are consistent and *for a constant u* are sufficient to completely determine the solution to the IVP. However, when the torque $u$ is allowed to vary with time, there are many solutions, and so, for comparison, we will impose the boundary condition $\beta(t_F) = p_1(t_F) = 15.8106$ rad. This specified value for the angle $\beta$ is the same as the final value achieved when a constant torque is used. In other words, we will match the final state for the constant-torque IVP solution with the optimal control solution. Thus, the goal of the optimal control is to reach the same state in the same time while expending less "energy" (6.150).

The direct transcription method requires an initial guess for the state and control time functions. For most applications (including most other examples in this book), it suffices to supply a linear initial guess. In fact, the default procedure used by $\mathbb{SOCS}$ to compute the state at grid point $k$ is

$$\mathbf{y}_k = \mathbf{y}_1 + \frac{(k-1)}{(M-1)}(\mathbf{y}_M - \mathbf{y}_1). \tag{6.162}$$

The user must specify the state at the initial grid point, $\mathbf{y}_1$, the state at the final grid point, $\mathbf{y}_M$, the initial and final times, $t_I$ and $t_F$, and the number of grid points, $M$. A similar expression is used to define the control $\mathbf{u}_k$ and the grid time values $t_k$. However, it is often possible to construct a much better initial guess for the state and control variables using special information about the problem and, indeed, this is so for this multibody example. One obvious approach is to fix $u(t) = u_0$ and then integrate the index-one DAE system (6.157)–(6.160) using software for solving a DAE IVP such as DASSL [140, 141] or RADAU5 [107]. The second approach (also tested in [107]) is to apply an ODE method to the differential equations (6.157), (6.158). This technique requires an explicit expression for the vector $\mathbf{q}$ appearing on the right-hand side of (6.158). Fortunately, by solving (6.161),

$$\begin{bmatrix} \mathbf{q} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{M}(\mathbf{p}) & \mathbf{G}^{\mathrm{T}}(\mathbf{p}) \\ \mathbf{G}(\mathbf{p}) & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{f}(\mathbf{v},\mathbf{p},u_0) \\ -\mathbf{g}_{pp}(\mathbf{p})(\mathbf{v},\mathbf{v}) \end{bmatrix}, \tag{6.163}$$

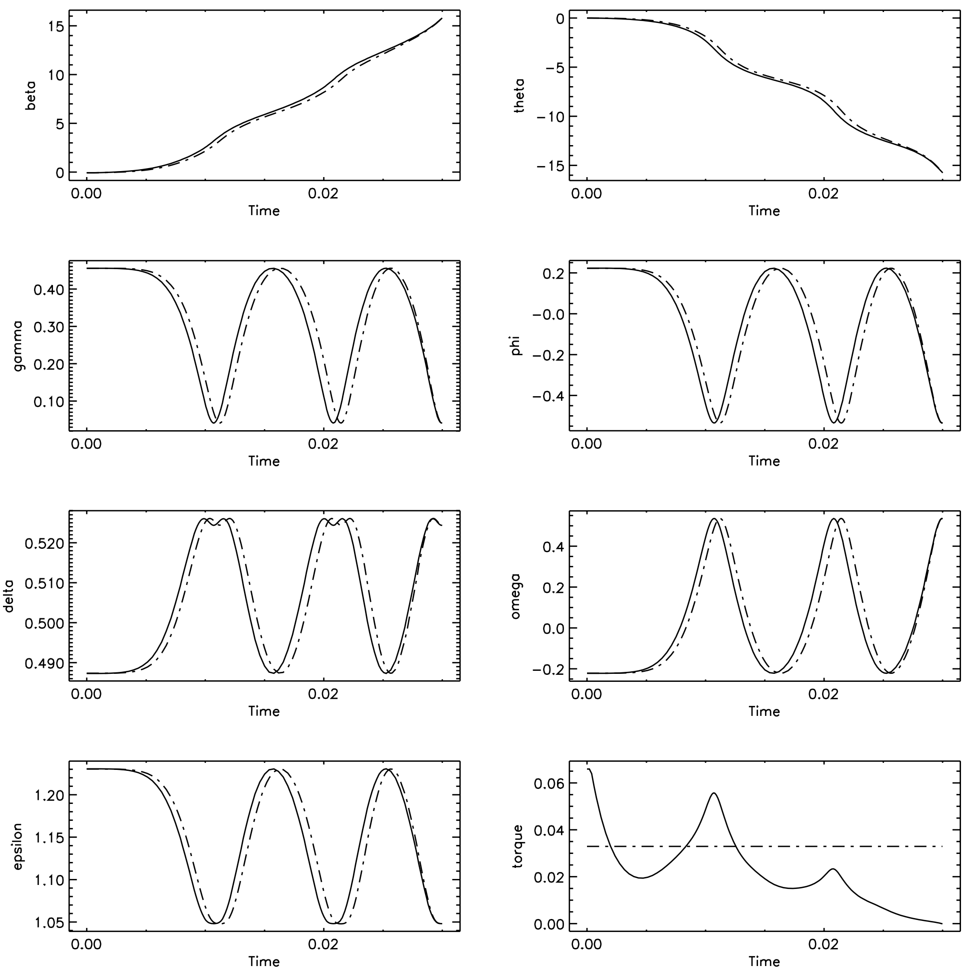


**Figure 6.38.** *Squeezer mechanism solution.*

one obtains the requisite explicit expression for $\mathbf{q} = \mathbf{q}(\mathbf{v}, \mathbf{p}, u_0)$. Of course, in practice we just solve (6.161) and do not explicitly compute the matrix inverse. An Adams predictor-corrector method was used to integrate the resulting ODE system, although any other IVP method could be used.

Figure 6.38 illustrates the solution obtained using SOCS. The minimum energy results are plotted with a solid line and the constant-torque reference trajectories using a dashed line. The minimum energy $J^* = 0.6669897295$ compared with a value $J = 1$ for the constant-torque reference trajectory. The initial guess was constructed by evaluating the integrated profile at 20 equally spaced grid points. The final solution was obtained after four mesh-refinement iterations; the algorithm performance is summarized in Table 6.19. It is interesting to note that the preferred discretization selected by the mesh-refinement algorithm, as described in Section 4.6.9, was the HSS (HS). To more fully appreciate this behavior, the same problem was solved four different ways—using two different discretiza-

**Table 6.19.** *Minimum energy squeezer mechanism.*

| Iter. | Disc. | $M$ | GE | HE | FE | RHS | $\epsilon_{\max}$ | CPU (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | HS | 20 | 29 | 15 | 2585 | 100815 | $0.66 \times 10^{-4}$ | $0.15 \times 10^{2}$ |
| 2 | HS | 39 | 18 | 15 | 2381 | 183337 | $0.98 \times 10^{-5}$ | $0.22 \times 10^{2}$ |
| 3 | HS | 72 | 10 | 7 | 1154 | 165022 | $0.27 \times 10^{-5}$ | $0.18 \times 10^{2}$ |
| 4 | HS | 143 | 7 | 2 | 456 | 129960 | $0.24 \times 10^{-7}$ | $0.20 \times 10^{2}$ |
| Total | | 143 | 64 | 39 | 6576 | 579134 | | 74.15 |

tions, with and without right-hand-side sparsity. For the sake of discussion, we refer to these as Methods A–D and they are summarized below:

| Method | Discretization | Separability | RHS sparsity |
|---|---|---|---|
| A | HS | Yes | Yes |
| B | HS | Yes | No |
| C | HC | No | Yes |
| D | HC | No | No |

An examination of the data in Table 6.20 explains why Method A (HSS exploiting right-hand-side sparsity) is the preferred approach. First, the number of index sets was 16. This can be attributed both to the separability of the discretization and the right-hand-side sparsity as illustrated by the template (cf. (4.115))

$$\mathcal{T} = \text{struct}\left[\begin{array}{c|c} \dfrac{\partial \mathbf{f}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \dfrac{\partial \mathbf{g}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{u}} \\ \dfrac{\partial \mathbf{w}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{w}}{\partial \mathbf{u}} \end{array}\right] = \left[\begin{array}{cccccccccccccccccccccccccccc}
&&&&&&&\times \\
&&&&&&&&\times \\
&&&&&&&&&\times \\
&&&&&&&&&&\times \\
&&&&&&&&&&&\times \\
&&&&&&&&&&&&\times \\
&&&&&&&&&&&&&\times \\
&&&&&&&&&&&&&&\times \\
&&&&&&&&&&&&&&&\times \\
&&&&&&&&&&&&&&&&\times \\
&&&&&&&&&&&&&&&&&\times \\
&&&&&&&&&&&&&&&&&&\times \\
&&&&&&&&&&&&&&&&&&&\times \\
&&&&&&&&&&&&&&&&&&&&\times \\
\times&\times&&&&&&\times&\times&&&&&&\times&\times&&&&&&\times&\times&\times&\times&\times&\times&\times \\
\times&\times&&&&&&\times&&&&&&&\times&\times&&&&&&\times&\times&\times&\times&\times&\times \\
&&\times&&&&&&&&&&&&&&\times&&&&&\times&\times \\
&&&\times&\times&&&&&&&\times&&&&&&\times&\times&&&&&\times&\times \\
&&&\times&\times&&&&&&\times&\times&&&&&&\times&\times&&&&&\times&\times \\
&&&&&\times&\times&&&&&&&\times&&&&&&\times&\times&&&&&\times&\times \\
&&&&&\times&\times&&&&&&\times&\times&&&&&&\times&\times&&&&&\times&\times \\
\times&\times&\times&&&&&\times&\times&\times&&&&&\times&\times&\times \\
\times&\times&\times&&&&&\times&\times&\times&&&&&\times&\times&\times \\
\times&\times&&\times&\times&&&\times&\times&&\times&\times&&&\times&\times&&\times&\times \\
\times&\times&&\times&\times&&&\times&\times&&\times&\times&&&\times&\times&&\times&\times \\
\times&\times&&&&\times&\times&\times&\times&&&&\times&\times&\times&\times&&&&\times&\times \\
\times&\times&&&&\times&\times&\times&\times&&&&\times&\times&\times&\times&&&&\times&\times \\
&&&&&&&&&&&&&&&&&&&&&&&&&&&\times
\end{array}\right]. \qquad (6.164)$$

When right-hand-side sparsity is not exploited (Method B), the number of index sets increases to 28. Furthermore, when discretization separability is not exploited, the number of index sets becomes larger still. Because the number of index sets for the HSS method is much smaller, the number of function evaluations needed to compute the Jacobian and Hessian is also significantly less. The net result is that the right-hand sides of the DAEs were

**Table 6.20.** *Discretization performance comparison.*

| | A | B | C | D |
|---|---|---|---|---|
| Index sets | 16 | 28 | 38 | 70 |
| FE per Hess./Jac. | 152 | 434 | 779 | 2555 |
| Total RHS eval. | 579134 | 1628010 | 2879084 | 8399250 |
| Largest NLP | 7980 | 9044 | 5992 | 6790 |
| Mesh-ref. iter. | 4 | 5 | 4 | 5 |
| Total CPU time | 74.15 | 179.84 | 151.43 | 400.37 |
| % Time for FE | 23% | 34% | 55% | 70% |

evaluated a significantly smaller number of times. Notice that the final NLP was larger for the HSS discretization (Methods A and B). Nevertheless, Method A is over 5.4 times faster than the HSC discretization without right-hand-side sparsity (Method D). In simple terms, for this problem it is better to solve a larger NLP problem because the derivatives can be computed more efficiently!

The solution to this problem exhibits another phenomenon associated with the numerical solution of high-index DAEs. Figure 6.39 plots the time history of the errors in the path constraints. Specifically, it displays the acceleration-level error $\|\mathbf{g}_{pp}(\mathbf{p})(\mathbf{v},\mathbf{v})+\mathbf{G}(\mathbf{p})\mathbf{q}\|$, the velocity-level error $\|\mathbf{G}(\mathbf{p})\mathbf{v}\|$, and the position error $\|\mathbf{g}(\mathbf{p})\|$. The acceleration error is acceptably small: $\|\frac{d^2\mathbf{g}(t)}{dt^2}\| \sim \mathcal{O}(10^{-11})$. However, both the position and the velocity errors "drift" significantly from zero.

## 6.11 Kinematic Chain

**Example 6.15** KINEMATIC CHAIN. Büskens and Gerdts (see [92]) present an example that requires control of a *multibody system.* The problem is interesting because it can be made arbitrarily large and requires the treatment of an index-2 DAE system. The multibody dynamics are described by a system similar to (6.147)–(6.148):

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{6.165}$$

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{f}(\mathbf{p},\mathbf{v}) - \mathbf{C}^{\mathsf{T}}(\mathbf{p})\boldsymbol{\lambda} + \mathbf{K}\mathbf{u}, \tag{6.166}$$

$$\mathbf{0} = \mathbf{c}(\mathbf{p}). \tag{6.167}$$

Figure 6.40 illustrates a chain with $\nu$ links, which can be described by the position vector

$$\mathbf{p}^{\mathsf{T}} = \left(\mathbf{p}_1,\ldots,\mathbf{p}_{\nu+1}\right)^{\mathsf{T}}. \tag{6.168}$$

The location of the individual joints can be represented by four Cartesian components

$$\mathbf{p}_k = \begin{pmatrix}\mathbf{x}_k\\ \mathbf{d}_k\end{pmatrix}, \qquad k = 1,\ldots,\nu, \tag{6.169}$$

$$\mathbf{p}_{\nu+1} = \mathbf{x}_{\nu+1}, \tag{6.170}$$

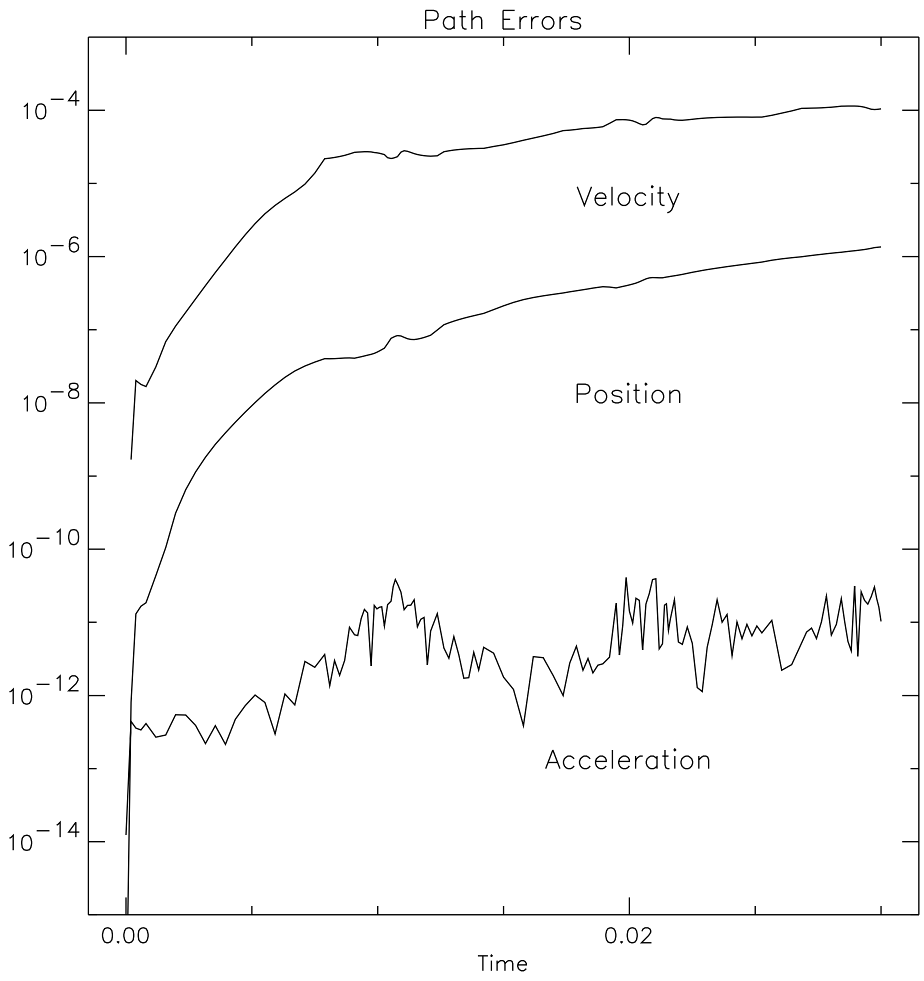


**Figure 6.39.** *Path-constraint errors.*

where the 2-vector $\mathbf{x}_k = (x_k, y_k)$ defines the horizontal and vertical position of link $k$, and the 2-vector $\mathbf{d}_k$ points from joint $k$ to joint $(k+1)$. The *holonomic constraints* (6.167)

$$\mathbf{c}(\mathbf{p}) = \begin{bmatrix} \mathbf{c}_1(\mathbf{p}_1, \mathbf{x}_2) \\ \vdots \\ \mathbf{c}_\nu(\mathbf{p}_\nu, \mathbf{x}_{\nu+1}) \end{bmatrix} \tag{6.171}$$

restrict the motion of the individual links. Specifically links of length $l_k$ must satisfy the conditions

$$\mathbf{c}_k(\mathbf{p}_k, \mathbf{x}_{k+1}) = \begin{bmatrix} \frac{1}{2}\left(\|\mathbf{d}_k\|^2 - l_k^2\right) \\ \mathbf{x}_k + \mathbf{d}_k - \mathbf{x}_{k+1} \end{bmatrix} = \mathbf{0}. \tag{6.172}$$

As written, (6.165)–(6.167) is an index-3 DAE system, and as such it is desirable to reduce

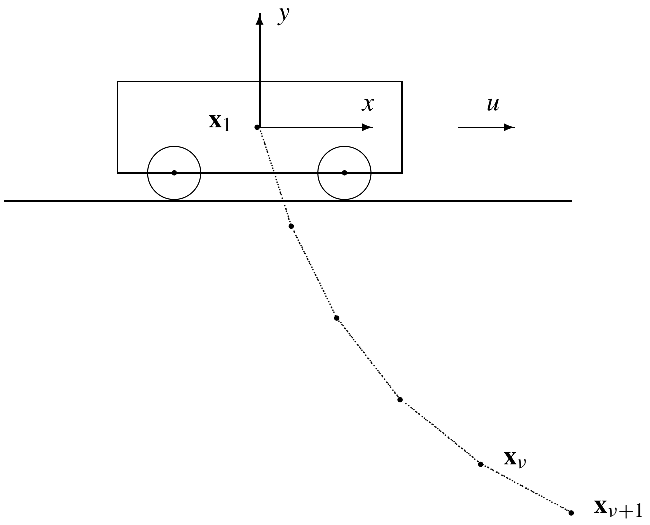


**Figure 6.40.** *Kinematic chain.*

the index by differentiating (6.167). In particular we must have

$$\mathbf{0} = \frac{d}{dt}\left[\mathbf{c}(\mathbf{p})\right] = \mathbf{C}(\mathbf{p})\dot{\mathbf{p}} = \mathbf{C}(\mathbf{p})\mathbf{v}. \tag{6.173}$$

The velocity constraint is represented in terms of the matrix

$$\mathbf{C}(\mathbf{p}) = \begin{bmatrix} \mathbf{C}_1(\mathbf{p}_1) & \mathbf{P}_1 & & \\ & \ddots & \ddots & \\ & & \mathbf{C}_\nu(\mathbf{p}_\nu) & \mathbf{P}_\nu \end{bmatrix}, \tag{6.174}$$

where

$$\mathbf{C}_k(\mathbf{p}_k) = \begin{bmatrix} (0,0) & \mathbf{d}_k^\top \\ \mathbf{I}_2 & \mathbf{I}_2 \end{bmatrix} \tag{6.175}$$

with

$$\mathbf{P}_k = \begin{bmatrix} \mathbf{0}_2 & \mathbf{0}_2 \\ -\mathbf{I}_2 & \mathbf{0}_2 \end{bmatrix}, \qquad k = 1,\ldots,(\nu-1), \qquad \mathbf{P}_\nu = \begin{bmatrix} \mathbf{0}_2 \\ -\mathbf{I}_2 \end{bmatrix} \tag{6.176}$$

and

$$\mathbf{0}_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \mathbf{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

An index-one DAE can be obtained by again differentiating

$$\mathbf{0} = \frac{d}{dt}\left[\mathbf{C}(\mathbf{p})\mathbf{v}\right] = \dot{\mathbf{C}}(\mathbf{p})\mathbf{v} + \mathbf{C}(\mathbf{p})\dot{\mathbf{v}}. \tag{6.177}$$

From (6.174) it follows that

$$\dot{\mathbf{C}}(\mathbf{p}) = \begin{bmatrix} \dot{\mathbf{C}}_1(\mathbf{p}_1) & \dot{\mathbf{P}}_1 & & & \\ & \ddots & \ddots & & \\ & & & \dot{\mathbf{C}}_\nu(\mathbf{p}_\nu) & \dot{\mathbf{P}}_\nu \end{bmatrix}, \tag{6.178}$$

where

$$\dot{\mathbf{C}}_k(\mathbf{p}_k) = \begin{bmatrix} (0,0) & \dot{\mathbf{d}}_k^\top \\ \mathbf{0}_2 & \mathbf{0}_2 \end{bmatrix} = \begin{bmatrix} (0,0) & (v_{k,3}, v_{k,4}) \\ \mathbf{0}_2 & \mathbf{0}_2 \end{bmatrix} \tag{6.179}$$

and $\dot{\mathbf{P}}_k = \mathbf{0}$.

Finally, by adding an additional algebraic variable $\mathbf{q}$ and path constraint, (6.166) can be written in semiexplicit form. Collecting results the dynamics are defined by the index-one DAE system

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{6.180}$$

$$\dot{\mathbf{v}} = \mathbf{q}, \tag{6.181}$$

$$\mathbf{0} = \mathbf{Mq} - \mathbf{f}(\mathbf{p}, \mathbf{v}, \mathbf{u}) + \mathbf{C}^\top(\mathbf{p})\boldsymbol{\lambda} - \mathbf{Ku}, \tag{6.182}$$

$$\mathbf{0} = \dot{\mathbf{C}}\mathbf{v} + \mathbf{Cq}. \tag{6.183}$$

The problem description is completed by defining the quantities in (6.182). First, the mass matrix is given by

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_1 & & & & \\ & \mathbf{M}_2 & & & \\ & & \ddots & & \\ & & & \mathbf{M}_\nu & \\ & & & & \mathbf{0}_2 \end{bmatrix} \tag{6.184}$$

with

$$\mathbf{M}_1 = \left(2 + \nu^{-1}\right) \begin{bmatrix} 1 & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & \frac{1}{12} \end{bmatrix} \tag{6.185}$$

and

$$\mathbf{M}_k = \nu^{-1} \begin{bmatrix} \mathbf{I}_2 & \frac{1}{2}\mathbf{I}_2 \\ \frac{1}{2}\mathbf{I}_2 & \frac{1}{3}\mathbf{I}_2 \end{bmatrix}, \qquad k = 2, \dots, \nu. \tag{6.186}$$

The force vector is given by

$$\mathbf{f}^\top(\mathbf{p}, \mathbf{v}) = (\mathbf{f}_1^\top, \mathbf{f}_2^\top, \dots, \mathbf{f}_\nu^\top, 0, 0) \tag{6.187}$$

with

$$\mathbf{f}_1^\top = (0, 0, 0, 0) \tag{6.188}$$

and

$$\mathbf{f}_k^{\mathsf{T}} = -g\nu^{-1}\left(0,1,0,\frac{1}{2}\right), \qquad k = 2,\dots,\nu. \tag{6.189}$$

The constants correspond to a chain with a total mass of 1 kg, and links of length $l_k = \nu^{-1}$, for a total length of 1 m, where $g = 9.81$. There is a single control $\mathbf{u}(t) = u(t)$ corresponding to a force in the horizontal direction at link one. The matrix $\mathbf{K}$ is diagonal with

$$K_{i,i} = \begin{cases} 1, & i = 1, \\ 0, & i = 2,\dots,4\nu+2. \end{cases} \tag{6.190}$$

The chain is initialized in a horizontal stretched out position

$$\mathbf{p}_k(t_I) = \begin{pmatrix} \mathbf{x}_k \\ \mathbf{d}_k \end{pmatrix} = \begin{pmatrix} (k-1)l_k \\ 0 \\ l_k \\ 0 \end{pmatrix}, \qquad k = 1,\dots,\nu, \tag{6.191}$$

$$\mathbf{p}_{\nu+1}(t_I) = \mathbf{x}_{\nu+1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6.192}$$

with zero initial velocity $\mathbf{v}(t_I) = \mathbf{0}$. As an objective Büskens and Gerdts suggest minimizing

$$\begin{aligned} F &= \int_{t_I}^{t_F} \alpha \mathbf{x}_1^{\mathsf{T}}(t)\mathbf{x}_1(t) + \alpha^{-1}\mathbf{u}^{\mathsf{T}}(t)\mathbf{u}(t)dt \\ &= \alpha \int_{t_I}^{t_F} x_1^2(t)dt + \alpha \int_{t_I}^{t_F} y_1^2(t)dt + \frac{1}{\alpha}\int_{t_I}^{t_F} u^2(t)dt \end{aligned} \tag{6.193}$$

with $\alpha = 1000$ over the time interval $t_I = 0$ to $t_F = 1$.

To begin the optimization process a reasonable initial guess can be constructed by exploiting the algebraic equations (6.182) and (6.183) which can be written as

$$\begin{bmatrix} \mathbf{M} & \mathbf{C}^{\mathsf{T}} \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f} + \mathbf{K}\mathbf{u} \\ -\dot{\mathbf{C}}\mathbf{v} \end{bmatrix}. \tag{6.194}$$

As an initial guess we choose $\mathbf{u}^{(0)}(t) = \mathbf{0}$. We then numerically integrate the ODE (not DAE) system (6.180)–(6.181) from the given initial conditions (6.191)–(6.192). To solve this IVP at each integration step it is necessary to compute $\mathbf{q}(t)$, which can be computed by solving the linear system (6.194). This procedure will construct an initial time history for all of the dynamic variables that satisfies the DAE system (6.180)–(6.183).

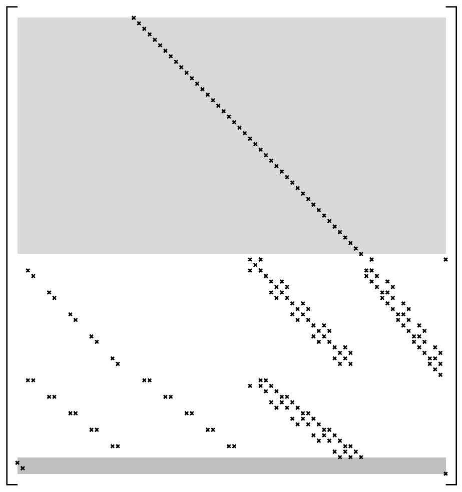
Computational results are presented for a chain with five links ($\nu = 5$), and Table 6.21 summarizes a number of key parameters for this case.

**Table 6.21.** *Chain problem parameters.*

| | |
|---|---|
| Differential Variables | 44 |
| Algebraic Variables | 38 |
| Differential Equations | 44 |
| Algebraic Equations | 37 |
| Quadrature Functions | 3 |
| Max. Nonzeros in Row of RHS Template | 6 |
| Evaluations for Jacobian (HS) | 12 |
| Evaluations for Hessian (HS) | 27 |
| Evaluations for Jacobian (HC) | 21 |
| Evaluations for Hessian (HC) | 42 |

For this example the right-hand-side sparsity template (cf. (4.89)) is of the form

$\mathcal{T} =$



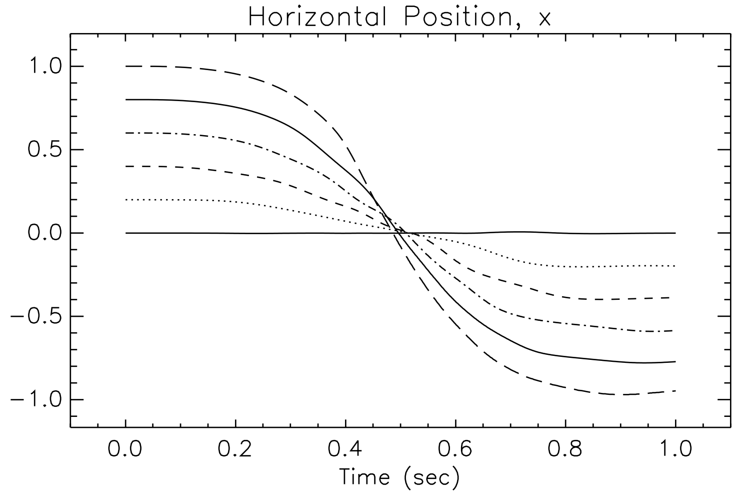
.

The first 44 rows correspond to the differential equations (6.180)–(6.181) and are shaded a light gray. The next 37 rows correspond to the algebraic equations (6.182)–(6.183) and are unshaded. The final three rows, shaded a dark gray, correspond to the individual terms in the quadrature objective function (6.193). Observe that this integral has been written as the sum of three terms in order to exploit separability. It is critical to exploit the fact that the right-hand-side template has at most six nonzero elements in any row. In fact, if right-hand-side sparsity is ignored for this DAE system of order 82, a central difference Jacobian and Hessian will require 3485 function evaluations, which is significantly more than the values in Table 6.21. This point is further emphasized by reviewing the mesh-refinement summary in Table 6.22. Notice that the entire problem, including five mesh-refinement iterations, 60 gradient evaluations, and 29 Hessian evaluations, required only 4937 function evaluations. For this example the first refinement iteration utilized 25 equidistributed grid points and the HSS discretization. Subsequent iterations used an HSC discretization with the grid distribution constructed by the refinement algorithm.

Figure 6.41 shows the time history for all of the link position state variables, and the optimal control time history is illustrated in Figure 6.42. The optimal objective function is $F^* = .6447976339 \times 10^{-1}$.

**Table 6.22.** *Chain mesh-refinement summary.*

| $k$ | $M$ | $n$ | NGC | NHC | NFE | NRHS | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | 25 | 4018 | 35 | 13 | 586 | 37518 | $2.8163\times10^{-3}$ | $1.8730\times10^{1}$ |
| 2 | 49 | 5842 | 9 | 6 | 1626 | 188504 | $1.2506\times10^{-4}$ | $1.8970\times10^{1}$ |
| 3 | 97 | 11602 | 6 | 4 | 1077 | 265747 | $1.1385\times10^{-5}$ | $3.7610\times10^{1}$ |
| 4 | 193 | 23122 | 6 | 4 | 1077 | 523411 | $5.0244\times10^{-7}$ | $6.8140\times10^{1}$ |
| 5 | 385 | 46162 | 4 | 2 | 571 | 645353 | $3.2910\times10^{-8}$ | $9.1150\times10^{1}$ |
| Total | 385 |  | 60 | 29 | 4937 | 1660533 |  | $2.3460\times10^{2}$ |



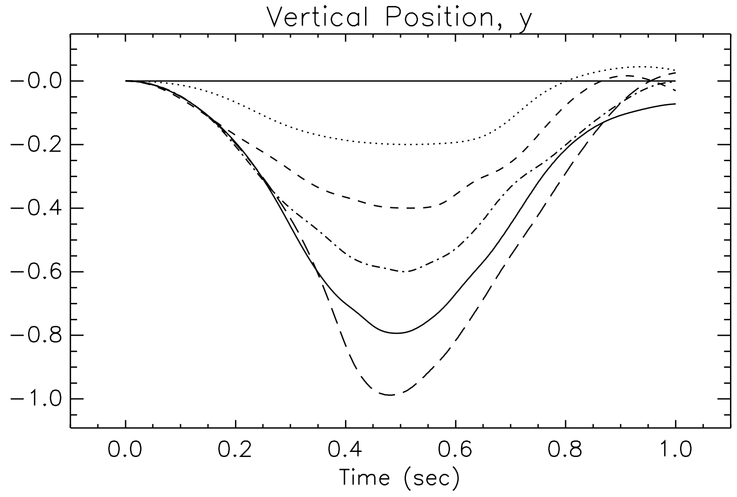


**Figure 6.41.** *Kinematic chain, states* $\mathbf{x}_k = (x_k, y_k)$ *for* $k = 1, \ldots, 6$.

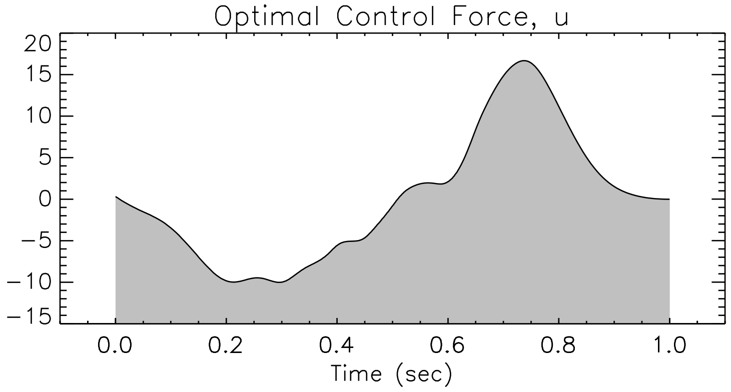


**Figure 6.42.** *Kinematic chain, optimal control.*

Solving this problem using the direct transcription method implemented in $\mathbb{SOCS}$ is relatively straightforward. In contrast, if a shooting or multiple shooting method is used, one must address a particular issue encountered for all DAE optimization problems. Specifically when the system dynamics are described by the DAEs

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{u}, t), \tag{6.195}$$

$$\mathbf{0} = \mathbf{g}(\mathbf{y}, \mathbf{u}, t) \tag{6.196}$$

the shooting method must incorporate software for solving a DAE IVP such as DASSL [140, 141] or RADAU5 [107]. To start the integration it is necessary to have *consistent initial conditions*; that is, at the beginning of the propagation it is necessary that

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t_I), \mathbf{u}(t_I), t_I]. \tag{6.197}$$

However, when one or more of the values $\mathbf{y}(t_I)$, $\mathbf{u}(t_I)$, or $t_I$ is modified as part of an optimization iteration, it is quite likely that at the perturbed point $(\widetilde{\mathbf{y}}(t_I), \widetilde{\mathbf{u}}(t_I), \tilde{t}_I) \neq (\mathbf{y}(t_I), \mathbf{u}(t_I), t_I)$

$$\mathbf{0} \neq \mathbf{g}[\widetilde{\mathbf{y}}(t_I), \widetilde{\mathbf{u}}(t_I), \tilde{t}_I]. \tag{6.198}$$

Unfortunately, when this happens the DAEs (6.195)–(6.196) cannot be propagated, and the process fails. Gerdts [92] discusses two different remedies for this dilemma. It is worth emphasizing that having inconsistent initial conditions is simply not an issue with the direct transcription method.

## 6.12 Dynamic MPEC

**Example 6.16** DYNAMIC MPEC. When formulating the behavior of physical systems it is sometimes useful to describe the dynamics using differential equations with discontinuous right-hand sides. To illustrate this situation let us consider a simple example originally suggested by Stewart and Anitescu [162] and discussed by Baumrucker, Renfro, and Biegler [7].[6] The goal is to minimize the objective

$$F = [y(2) - 5/3]^2 + \int_0^2 y^2(t)dt \tag{6.199}$$

and satisfy the simple differential equation

$$\dot{y} = 2 - sgn(y), \tag{6.200}$$

defined for $0 \le t \le 2$, with initial condition $y(0) = y_0$, where the signum or sign function is given by (1.126). If the state $y$ represents the velocity of a body moving in contact with a surface, then the term $sgn(y)$ describes the friction force, and the example represents the situation found in contact and friction problems in computational mechanics. As stated there is no control variable and the solution is uniquely defined by the initial condition. However, as a practical matter the numerical solution of (6.200) is problematic because $sgn(y)$ is discontinuous.

Now, if $y(0) = y_0 = -1$, it is easy to demonstrate for this very simple example that the dynamics are described by

$$\dot{y} = 3, \qquad 0 \le t \le 1/3, \tag{6.201a}$$
$$\dot{y} = 1, \qquad 1/3 \le t \le 2. \tag{6.201b}$$

Clearly the best way to treat the discontinuity is to introduce a separate phase corresponding to each region. The location of the phase boundary can be determined by introducing a single variable $t_s$ and then imposing a single continuity constraint $y(t_s^-) = y(t_s^+)$ between the phases. Unfortunately, for realistic problems with more complicated nonlinear dynamics it is seldom so easy to identify the phase structure. Instead the discontinuous behavior may occur many times during the process. Furthermore both the number and location of the discontinuities can change from one optimization iteration to the next.

Let us consider an alternate approach. Since $sgn[y(t)]$ is a discontinuous function of time, let us replace it by an algebraic variable

$$s(t) = sgn[y(t)], \tag{6.202}$$

which, unlike a state variable, can also behave discontinuously. Suppose at *every time t* we compute the variable $s(t) = s$ to minimize

$$\phi(s) = -sy \tag{6.203}$$

[6]This topic was the subject of a technical interchange at Argonne National Laboratory on October 25, 2004, hosted by Sven Leyffer, with participation by Larry Biegler, Shiva Kameswaran, and Juan J. Arrieta-Camacho (Carnegie Mellon University) and Steve Campbell (North Carolina State University).

subject to the constraints

$$s+1 \geq 0, \tag{6.204}$$

$$1-s \geq 0. \tag{6.205}$$

This is an inner level NLP just as first introduced in (1.128a)–(1.128b), and since it must be solved for each time $t$ the overall approach is referred to as a *dynamic MPEC*.

Let us now formulate an optimal control problem to reflect these conditions. For this example we have a single differential variable $y(t)$ and three algebraic variables $\mathbf{u}^\mathsf{T}(t) = [s(t), p(t), q(t)]$. We must determine these quantities to minimize the objective function

$$F = [y(2)-5/3]^2 + \int_0^2 y^2(t) + \rho\{p(t)[s(t)+1] + q(t)[1-s(t)]\}dt \tag{6.206}$$

for $\rho > 0$ and satisfy the dynamic constraints

$$\dot{y} = 2 - s(t), \tag{6.207}$$

$$0 = -y(t) - p(t) + q(t), \tag{6.208}$$

$$-1 \leq s(t) \quad \leq \quad 1, \tag{6.209}$$

$$0 \leq p(t), \tag{6.210}$$

$$0 \leq q(t) \tag{6.211}$$

with boundary condition $y(0) = y_0$. Observe that the complementarity conditions corresponding to (1.130)–(1.131)

$$0 \leq p \perp (s+1) \geq 0, \tag{6.212}$$

$$0 \leq q \perp (1-s) \geq 0 \tag{6.213}$$

are treated using an exact penalty function in the modified objective (6.206). When viewed this way, the functions $p(t)$ and $q(t)$ are just the time varying Lagrange multipliers for the inner level optimization problem (6.203)–(6.205). By using the integral form, the $\mathbb{SOCS}$ mesh-refinement procedure will automatically locate the grid points to achieve the required accuracy. The solution produced by $\mathbb{SOCS}$ is plotted in Figure 6.43. Table 6.23 summarizes the behavior of $\mathbb{SOCS}$ to solve this problem. The first two refinement iterations used a trapezoidal discretization and the last five were forced to be (compressed) Hermite–Simpson. The default $\mathbb{SOCS}$ approach would use (separated) Simpson for the last five refinements, and is a bit faster, but has the same grid configuration. Although this was done as a one-phase problem it appears that the switch occurs very close to $t = 1/3$ with almost all of the grid points clustered in this region as illustrated in Figure 6.44. For $x(0) = -1$

the optimal objective function value was $\phi^* = 1.6551964$, using the exact penalty value $\rho = 10^3$.

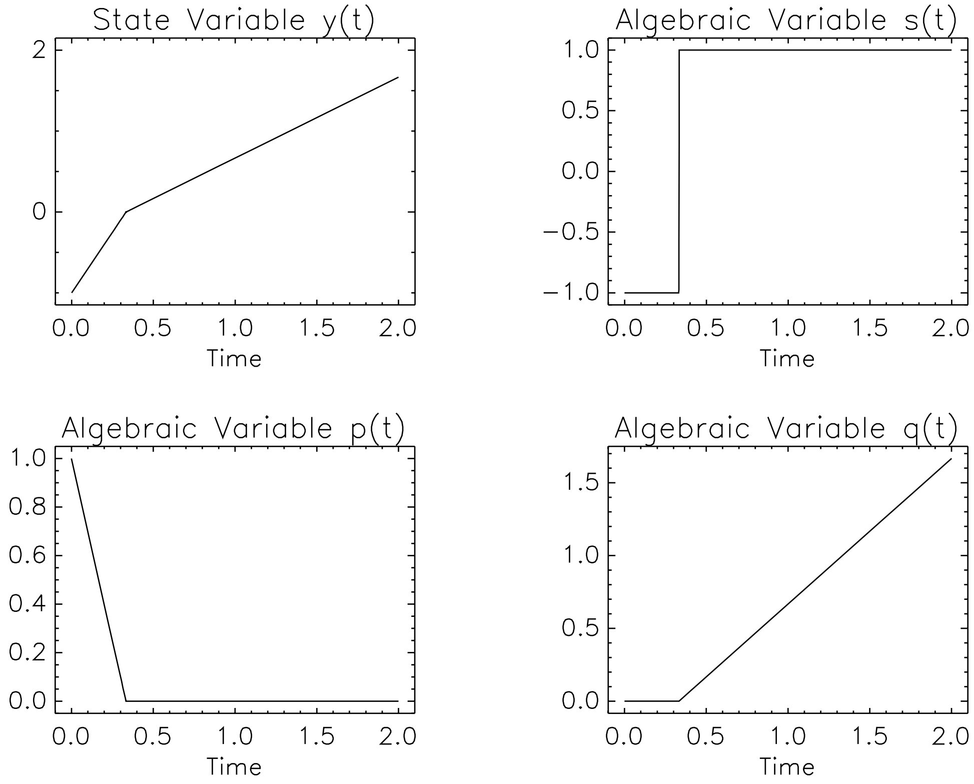


**Figure 6.43.** *MPEC solution.*

**Table 6.23.** *Mesh-refinement summary.*

| $k$ | $M$ | NGC | NHC | NFE | NRHS | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|
| 1 | 10 | 6 | 3 | 45 | 2117 | $2.4691\times10^{-1}$ | $1.0\times10^{-2}$ |
| 2 | 16 | 11 | 9 | 101 | 4159 | $5.8964\times10^{-3}$ | $3.0\times10^{-2}$ |
| 3 | 16 | 8 | 6 | 503 | 21667 | $1.8081\times10^{-3}$ | $5.0\times10^{-2}$ |
| 4 | 24 | 8 | 6 | 503 | 32051 | $1.2141\times10^{-4}$ | $8.0\times10^{-2}$ |
| 5 | 30 | 41 | 39 | 3077 | 191545 | $6.9649\times10^{-6}$ | $6.1\times10^{-1}$ |
| 6 | 36 | 5 | 3 | 269 | 30749 | $1.7732\times10^{-7}$ | $1.8\times10^{-1}$ |
| 7 | 41 | 5 | 3 | 269 | 34819 | $4.3035\times10^{-}8$ | $2.0\times10^{-1}$ |
| Total | 41 | 84 | 69 | 4767 | 317107 | | 1.16 |

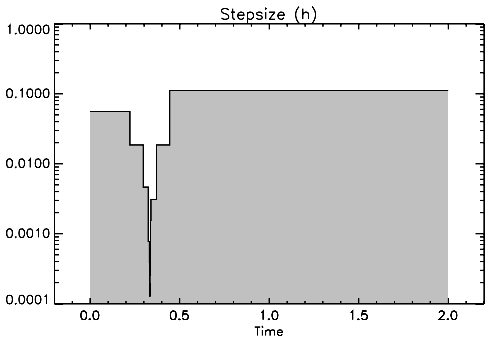


**Figure 6.44.** *MPEC grid distribution.*

## 6.13 Free-Flying Robot

**Example 6.17** FREE-FLYING ROBOT. Sakawa [153] presents an example that describes the motion of a free-flying robot equipped with a propulsion system. The inertial coordinates of the center of gravity are denoted by $(x_1, x_2)$ and the corresponding velocity by $(v_1, v_2)$. The thrust direction is denoted by $\theta$ and the angular velocity by $\omega$. The thrust from two engines denoted by $T_1$ and $T_2$ serve as the control variables:

$$\dot{x}_1 = v_1, \tag{6.214}$$
$$\dot{x}_2 = v_2, \tag{6.215}$$
$$\dot{\theta} = \omega, \tag{6.216}$$
$$\dot{v}_1 = [T_1 + T_2]\cos\theta, \tag{6.217}$$
$$\dot{v}_2 = [T_1 + T_2]\sin\theta, \tag{6.218}$$
$$\dot{\omega} = \alpha T_1 - \beta T_2. \tag{6.219}$$

Bounds are also imposed on the thrust magnitudes:

$$|T_1(t)| \le 1, \qquad |T_2(t)| \le 1. \tag{6.220}$$

As boundary conditions for the state $\mathbf{y} = (x_1, x_2, \theta, v_1, v_2, \omega)^\mathsf{T}$ we require

$$y(0) = \left(-10, -10, \frac{\pi}{2}, 0, 0, 0\right)^\mathsf{T}, \tag{6.221}$$
$$y(t_F) = (0,0,0,0,0,0)^\mathsf{T}. \tag{6.222}$$

Although the dynamics are relatively straightforward this problem is illustrative because of the objective function. The objective proposed by Sakawa was

$$F = \frac{1}{2}\sigma \|\mathbf{y}(t_F) - \mathbf{y}_F\|^2 + \gamma \int_0^{t_F} (|T_1| + |T_2|)\,dt. \tag{6.223}$$

The first term can be omitted by simply imposing the final condition (6.222) directly. Setting $\gamma = 1$, $\alpha = \beta = .2$, and $t_F = 12$ completes the problem definition, and the objective function becomes

$$F = \int_0^{12} (|T_1| + |T_2|)\, dt. \tag{6.224}$$

Unfortunately, the objective function as written has discontinuous derivatives because it involves the absolute value function. An approach for treating absolute values motivated by the MPEC formulation (1.133c)–(1.133e) was described in Example 1.13, and the same technique can be utilized here. The "trick" is to express each control in terms of its positive and negative component:

$$T_1 = u_1 - u_2, \tag{6.225}$$
$$T_2 = u_3 - u_4 \tag{6.226}$$

with the restriction that $\mathbf{u} = (u_1, u_2, u_3, u_4)^{\mathsf{T}} \geq \mathbf{0}$. In effect, each "real" control is split into two controls, but in so doing one can write

$$|T_1| = u_1 + u_2, \tag{6.227}$$
$$|T_2| = u_3 + u_4. \tag{6.228}$$

After making this transformation the modified formulation requires minimizing the differentiable objective function

$$F = \int_0^{12} (u_1 + u_2 + u_3 + u_4)\, dt \tag{6.229}$$

while solving the DAE system

$$\dot{y}_1 = y_4, \tag{6.230}$$
$$\dot{y}_2 = y_5, \tag{6.231}$$
$$\dot{y}_3 = y_6, \tag{6.232}$$
$$\dot{y}_4 = [u_1 - u_2 + u_3 - u_4] \cos y_3, \tag{6.233}$$
$$\dot{y}_5 = [u_1 - u_2 + u_3 - u_4] \sin y_3, \tag{6.234}$$
$$\dot{y}_6 = \alpha(u_1 - u_2) - \beta(u_3 - u_4), \tag{6.235}$$
$$1 \geq u_1 + u_2, \tag{6.236}$$
$$1 \geq u_3 + u_4 \tag{6.237}$$

with $u_k \geq 0$ for $k = 1, 2, 3, 4$.

The optimal objective function is $F^* = 7.910154646$. The optimal state histories are illustrated in Figure 6.45. The time histories for the controls $\mathbf{u}$ are plotted in Figure 6.46 as well as the time history for the "real" controls $T_1$ and $T_2$. It is not surprising that the control history has a "bang-bang" appearance since the controls appear linearly in the problem. Note also that the final mesh distribution displayed in Figure 6.47 has very small steps in the vicinity of the step discontinuities, which is necessary to achieve the requested accuracy. This step contraction requires 12 iterations in the mesh-refinement procedure as summarized in Table 6.24.

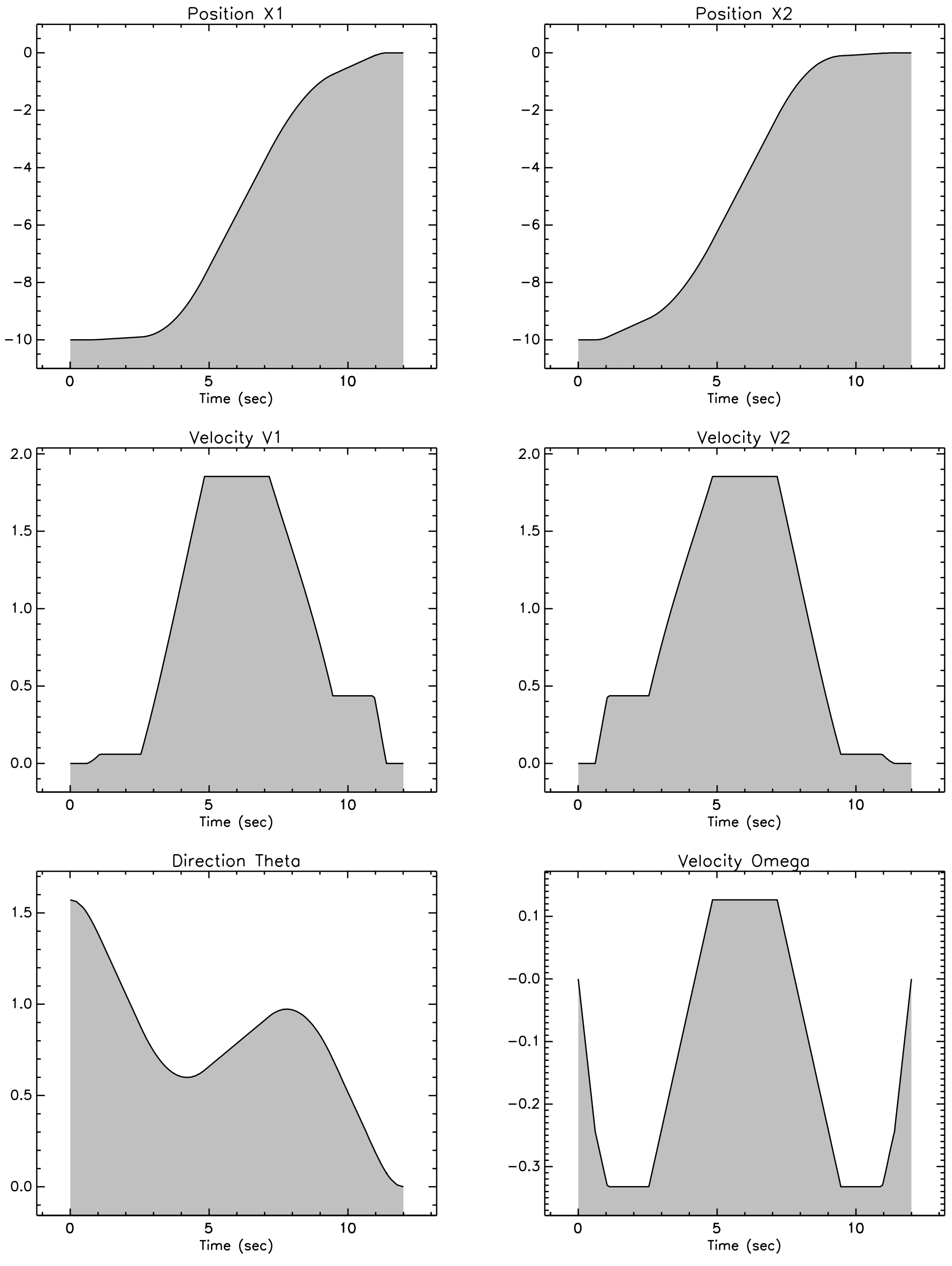


**Figure 6.45.** *Free-flying robot states.*

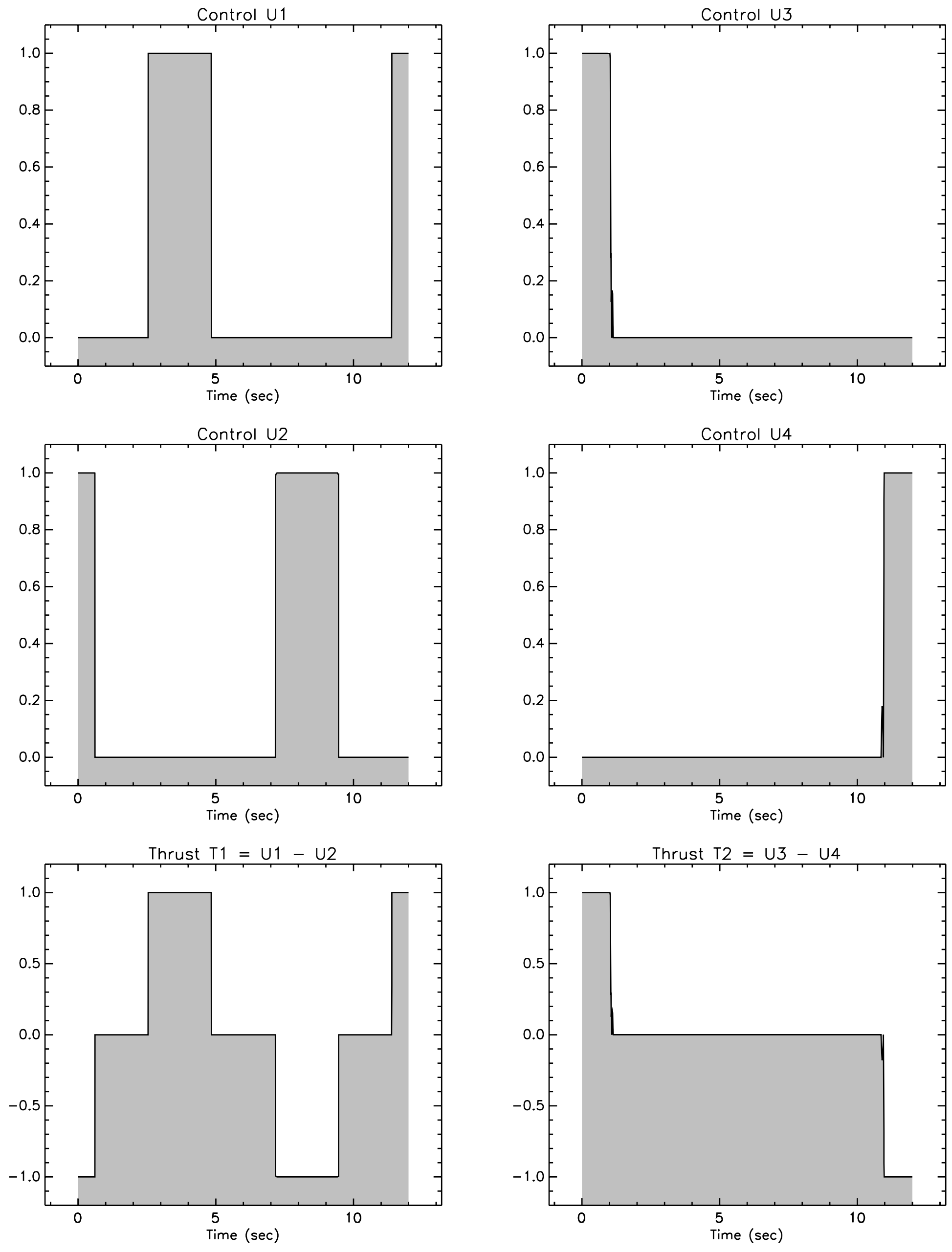


**Figure 6.46.** *Free-flying robot controls.*

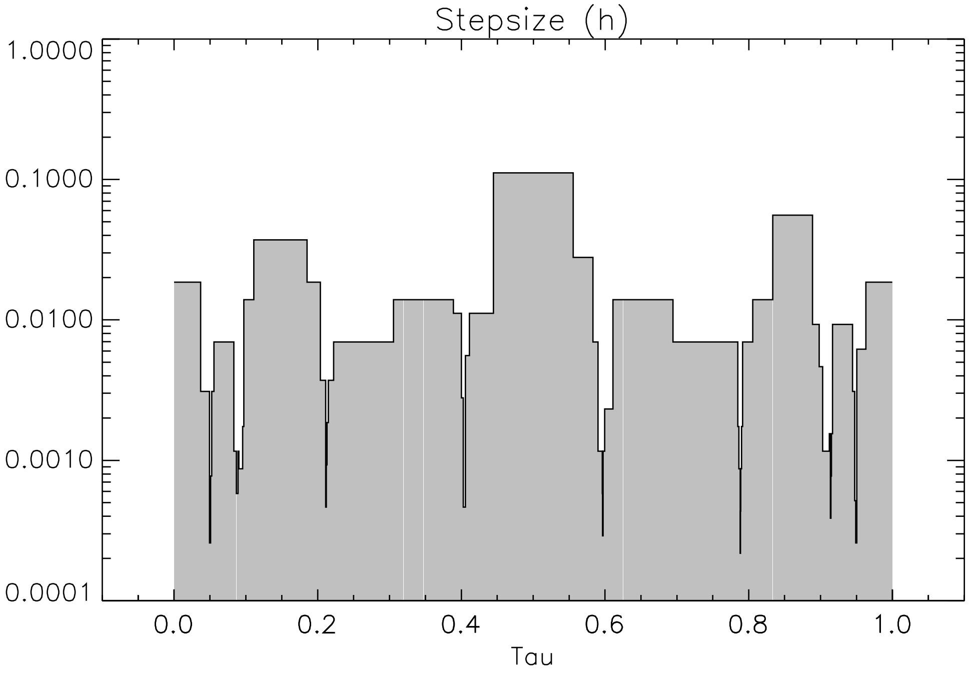


**Figure 6.47.** *Free-flying robot final grid distribution.*

**Table 6.24.** *Free-flying robot mesh-refinement summary.*

| $k$ | $M$ | NGC | NHC | NFE | NRHS | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|
| 1 | 10 | 16 | 9 | 236 | 4417 | $3.8980\times10^{-2}$ | $9.0000\times10^{-2}$ |
| 2 | 10 | 10 | 8 | 1274 | 28968 | $1.4679\times10^{-2}$ | $1.2000\times10^{-1}$ |
| 3 | 19 | 11 | 9 | 1427 | 62133 | $3.5083\times10^{-3}$ | $3.8000\times10^{-1}$ |
| 4 | 37 | 28 | 26 | 4028 | 312522 | $3.6559\times10^{-4}$ | $2.4200\times10^{+0}$ |
| 5 | 53 | 40 | 38 | 5864 | 641114 | $3.3852\times10^{-4}$ | $5.2700\times10^{+0}$ |
| 6 | 69 | 20 | 18 | 2804 | 416454 | $1.8255\times10^{-5}$ | $3.2700\times10^{+0}$ |
| 7 | 82 | 15 | 13 | 2039 | 370251 | $3.6540\times10^{-6}$ | $6.0300\times10^{+0}$ |
| 8 | 103 | 44 | 42 | 6476 | 1375062 | $2.8825\times10^{-5}$ | $1.6500\times10^{+1}$ |
| 9 | 115 | 17 | 15 | 2345 | 589151 | $4.4139\times10^{-6}$ | $9.1200\times10^{+0}$ |
| 10 | 134 | 20 | 18 | 2804 | 808226 | $1.6770\times10^{-6}$ | $1.2330\times10^{+1}$ |
| 11 | 142 | 7 | 5 | 815 | 293347 | $3.1444\times10^{-7}$ | $9.7000\times10^{+0}$ |
| 12 | 159 | 5 | 3 | 509 | 230675 | $9.8315\times10^{-8}$ | $1.1620\times10^{+1}$ |
| Total | 159 | 233 | 204 | 30621 | 5132320 | | $7.6850\times10^{+1}$ |

## 6.14 Kinetic Batch Reactor

**Example 6.18** KINETIC BATCH REACTOR. In his doctoral thesis Leineweber [130] presents a problem originally given by Caracotsios and Stewart [62] that describes

> an optimal control problem which has several interesting features: stiff nonlinear DAE's, two model stages, a nonlinear inequality path constraint, equality and inequality boundary conditions, and unspecified terminal time. The example in its original form was given by the Dow Chemical Company as a challenging test problem for parameter estimation software .... The desired product $AB$ is formed in the reaction
>
> $$HA + 2BM \longrightarrow AM + MBMH.$$
>
> (For proprietary reasons, the true nature of the reacting species has been disguised.)

Leineweber presents a kinetic model of the batch reactor system in terms of both differential and algebraic states denoted by $x_j$ and $z_j$, respectively. A detailed explanation of the chemical process is omitted; however, Table 6.25 summarizes the model presented in [130]. All of the species concentrations (enclosed in brackets [.]) are given in gmol per kg of the reaction mixture. For the formulation given here the differential and algebraic variables are denoted by $y_j$ and $u_j$, respectively, which correspond to the original notation as shown in Table 6.25.

**Table 6.25.** *Batch reactor dynamic variables.*

| | | Description | |
|---|---|---|---|
| $y_1$ | $x_0$ | Differential State | $[HA]+[A^-]$ |
| $y_2$ | $x_1$ | Differential State | $[BM]$ |
| $y_3$ | $x_2$ | Differential State | $[HABM]+[ABM^-]$ |
| $y_4$ | $x_3$ | Differential State | $[AB]$ |
| $y_5$ | $x_4$ | Differential State | $[MBMH]+[MBM^-]$ |
| $y_6$ | $x_5$ | Differential State | $[M^-]$ |
| $u_1$ | $z_0$ | Algebraic State | $-\log([H^+])$ |
| $u_2$ | $z_1$ | Algebraic State | $[A^-]$ |
| $u_3$ | $z_2$ | Algebraic State | $[ABM^-]$ |
| $u_4$ | $z_3$ | Algebraic State | $[MBM^-]$ |
| $u_5$ | $T$ | Reaction Temperature | |

The kinetic model is stated in terms of six differential mass balance equations

$$\dot{y}_1 = -k_2 y_2 u_2, \tag{6.238}$$

$$\dot{y}_2 = -k_1 y_2 y_6 + k_{-1} u_4 - k_2 y_2 u_2, \tag{6.239}$$

$$\dot{y}_3 = k_2 y_2 u_2 + k_3 y_4 y_6 - k_{-3} u_3, \tag{6.240}$$

$$\dot{y}_4 = -k_3 y_4 y_6 + k_{-3} u_3, \tag{6.241}$$

$$\dot{y}_5 = k_1 y_2 y_6 - k_{-1} u_4, \tag{6.242}$$

$$\dot{y}_6 = -k_1 y_2 y_6 + k_{-1} u_4 - k_3 y_4 y_6 + k_{-3} u_3, \tag{6.243}$$

an electroneutrality condition

$$0 = p - y_6 + 10^{-u_1} - u_2 - u_3 - u_4, \tag{6.244}$$

and three equilibrium conditions

$$0 = u_2 - K_2 y_1/(K_2 + 10^{-u_1}), \tag{6.245}$$

$$0 = u_3 - K_3 y_3/(K_3 + 10^{-u_1}), \tag{6.246}$$

$$0 = u_4 - K_1 y_5/(K_1 + 10^{-u_1}), \tag{6.247}$$

where

$$k_1 = \hat{k}_1 \exp(-\beta_1/u_5), \tag{6.248}$$

$$k_{-1} = \hat{k}_{-1} \exp(-\beta_{-1}/u_5), \tag{6.249}$$

$$k_2 = \hat{k}_2 \exp(-\beta_2/u_5), \tag{6.250}$$

$$k_3 = k_1, \tag{6.251}$$

$$k_{-3} = \frac{1}{2} k_{-1}. \tag{6.252}$$

The values for the parameters $\hat{k}_j$ (kg/gmol/hr), $\beta_j$ (K), and $K_j$ (gmol/kg) are

| | | |
|---|---|---|
| $\hat{k}_1 = 1.3708 \times 10^{12}$, | $\beta_1 = 9.2984 \times 10^3$, | $K_1 = 2.575 \times 10^{-16}$, |
| $\hat{k}_{-1} = 1.6215 \times 10^{20}$, | $\beta_{-1} = 1.3108 \times 10^4$, | $K_2 = 4.876 \times 10^{-14}$, |
| $\hat{k}_2 = 5.2282 \times 10^{12}$, | $\beta_2 = 9.5999 \times 10^3$, | $K_3 = 1.7884 \times 10^{-16}$. |

The reaction temperature $T$ (K) is treated as the control variable and is limited to

$$293.15 \le u_5(t) \le 393.15 \tag{6.253}$$

for the duration of the process $0 \le t \le t_F$. Leineweber introduces a piecewise linear approximation for $u_5(t)$. In contrast, we will treat $u_5(t)$ like all other algebraic variables, and consequently the representation will be modified during the mesh-refinement procedure. Presumably this is a more accurate treatment for the reaction temperature control. The initial catalyst concentration, which Leineweber denotes by $[Q^+]$, is treated as the design parameter $p$ (gmol/kg) appearing in (6.244). This parameter is included as an optimization variable and is restricted by the bounds

$$0 \le p \le .0262. \tag{6.254}$$

At the initial time $t = 0$ the parameter is related to the corresponding differential state through the point constraint

$$\psi = y_6(0) - p = 0 \tag{6.255}$$

and the remaining states are fixed by the boundary conditions

$$y_1(0) = 1.5776, \tag{6.256}$$

$$y_2(0) = 8.32, \tag{6.257}$$

$$y_3(0) = y_4(0) = y_5(0) = 0. \tag{6.258}$$

At the free final time $t_F$, it is required that

$$y_4(t_F) \geq 1. \tag{6.259}$$

In order to restrict the rate of product formation during the initial 25% of the process time, Leineweber introduces the nonlinear inequality path constraint

$$y_4(t) \leq at^2 \qquad \text{for} \quad 0 \leq t \leq (t_F/4), \tag{6.260}$$

where $a = 2$ (gmol/kg/hr$^2$). And finally, the desired objective is to minimize the quantity

$$F = \gamma_1 t_F + \gamma_2 p, \tag{6.261}$$

where $\gamma_1 = 1$ and $\gamma_2 = 100$.

The solution of this problem requires at least two phases, which Leineweber refers to as "stages," because the path constraint (6.260) must be imposed during the first portion of the process. Since the DAE system is stiff it is also helpful to introduce an additional phase at the beginning of the process in order to effectively model the rapid transient behavior. The domain is subdivided as follows:

$$0 \leq t \leq t_\epsilon \qquad \text{Phase 1}, \tag{6.262}$$

$$t_\epsilon \leq t \leq (t_F/4) \qquad \text{Phase 2}, \tag{6.263}$$

$$(t_F/4) \leq t \leq t_F \qquad \text{Phase 3}, \tag{6.264}$$

where we choose $t_\epsilon = .01$. Unlike the phase boundary at $t_F/4$, which must be introduced in order to treat the path constraint (6.260), the first phase is introduced strictly for numerical reasons. This rather simple artifice serves two purposes. First, it decouples the nonlinear transient behavior at the beginning of the process during the early iterations in much the same way as a multiple shooting method does. Second, it affords a simple way to construct an initial guess with a sufficiently fine mesh in the transient region. A piecewise linear initial guess with 40 grid points in Phase 1 and 30 in Phases 2 and 3 was used for the computational results. The three-phase description is completed by ensuring continuity in the differential states and the control across the phase boundaries, i.e.,

$$y_k^{(j)} = y_k^{(j+1)}, \qquad j = 1,2, \quad k = 1,\ldots,6, \tag{6.265}$$

$$u_5^{(j)} = u_5^{(j+1)}, \qquad j = 1,2. \tag{6.266}$$

The algebraic states $(u_1, u_2, u_3, u_4)$ are implicitly linked across the phase boundaries by satisfying the algebraic path constraints (6.244)–(6.247). The differential and algebraic states are illustrated in Figures 6.48 and 6.49. The optimal control temperature is plotted in Figure 6.50. The piecewise linear guess for all dynamic variables is also plotted as a dashed line in the figures. Figure 6.51 illustrates the transient behavior during Phase 1 for some of the dynamic quantities. Table 6.26 summarizes the mesh-refinement history for this example.

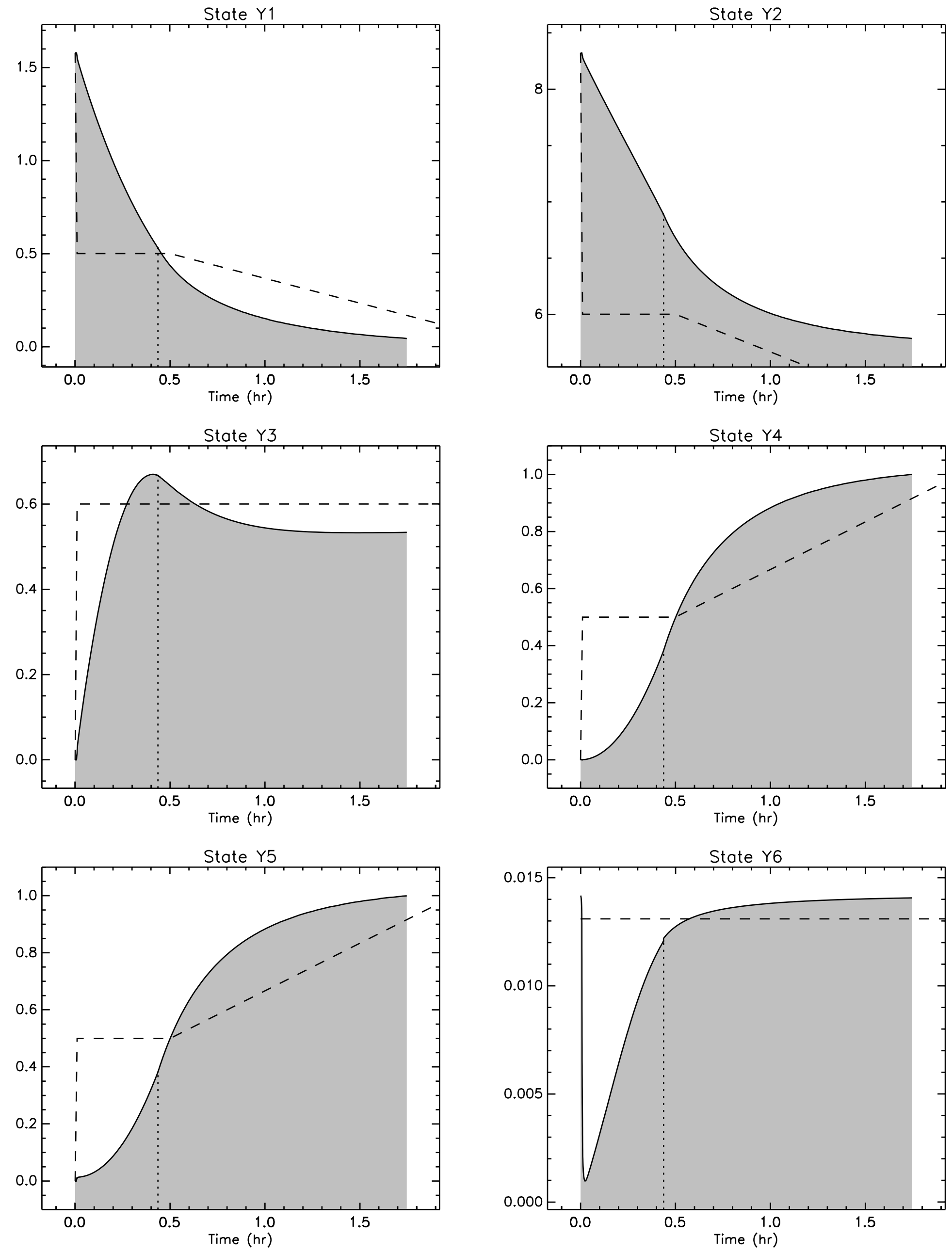


**Figure 6.48.** *Batch reactor differential states.*

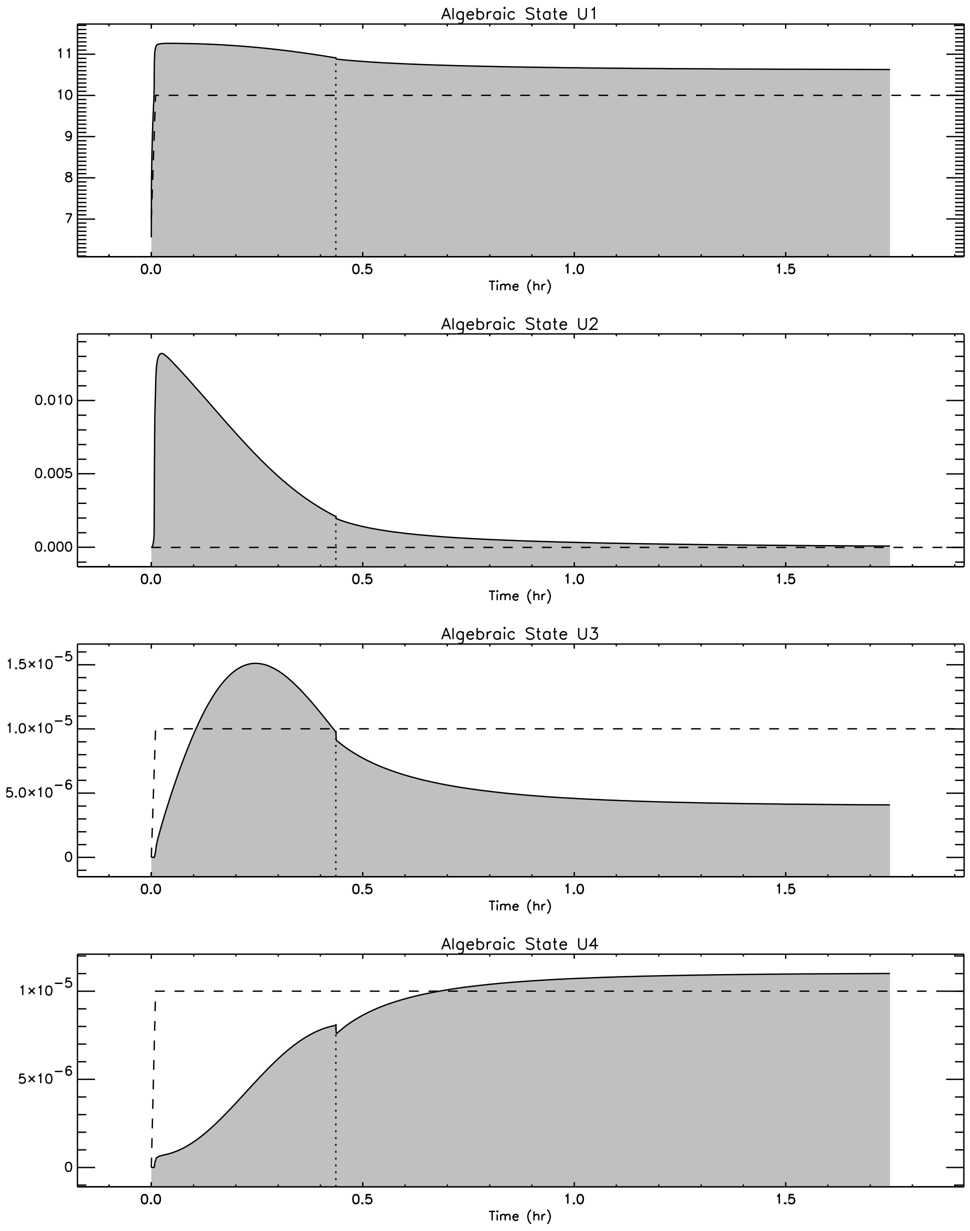


**Figure 6.49.** *Batch reactor algebraic states.*

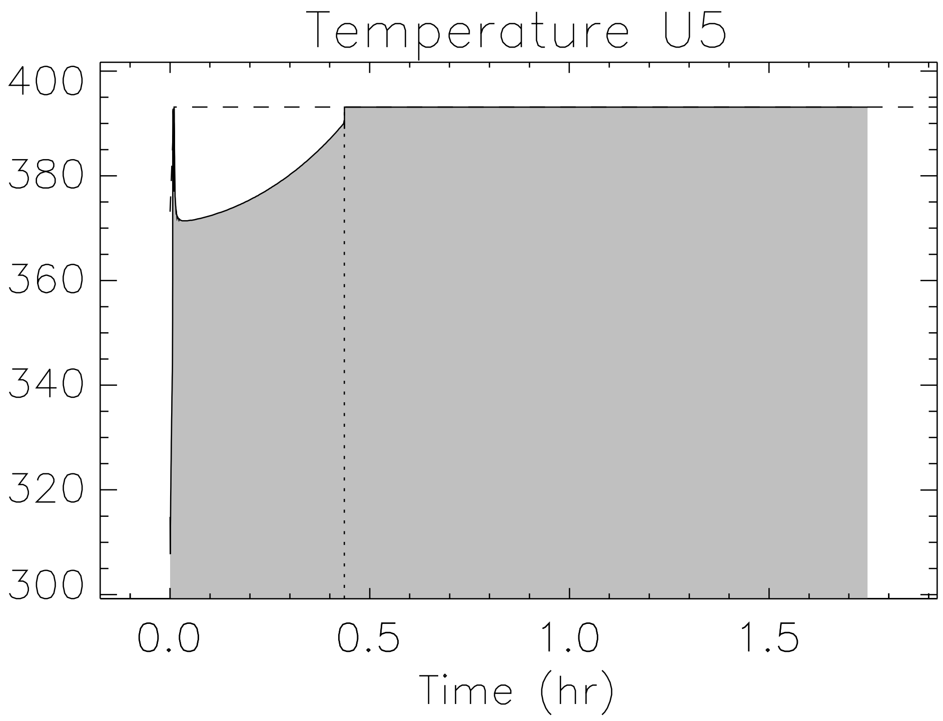


**Figure 6.50.** *Batch reactor control.*

## 6.15 Delta III Launch Vehicle

**Example 6.19** DELTA III LAUNCH VEHICLE. In his doctoral thesis, Benson [8] presents a simplified formulation of an ascent trajectory for the Delta III launch vehicle. This problem is presented as an example for the software GPOCS [147]. The dynamic model for the motion of a nonlifting point mass in flight over a spherical rotating earth expressed in Cartesian (ECI) coordinates is given by the differential-algebraic system

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{6.267}$$

$$\dot{\mathbf{v}} = -\frac{\mu}{\|\mathbf{r}\|^3}\mathbf{r} + \frac{T}{m}\mathbf{u} + \frac{1}{m}\mathbf{D}, \tag{6.268}$$

$$\dot{m} = -\xi, \tag{6.269}$$

$$1 = \|\mathbf{u}\|, \tag{6.270}$$

$$R_E \le \|\mathbf{r}\|. \tag{6.271}$$

In this formulation $\mathbf{r}$ is the position vector, $\mathbf{v}$ is the velocity vector, $T$ is the total vacuum thrust for all engines, and $m$ is the total mass. The total mass flow rate of all engines is denoted by $\xi$. The complete state vector is $\mathbf{y}^{\mathsf{T}} = (\mathbf{r}^{\mathsf{T}}, \mathbf{v}^{\mathsf{T}}, m)$ and the control vector $\mathbf{u}$ defines the inertial thrust direction. The path constraint (6.270) guarantees that $\mathbf{u}$ is a unit vector and (6.271) ensures the altitude is positive. The drag force is given by

$$\mathbf{D} = -\frac{1}{2}C_D S\rho\|\mathbf{v}_r\|\mathbf{v}_r, \tag{6.272}$$

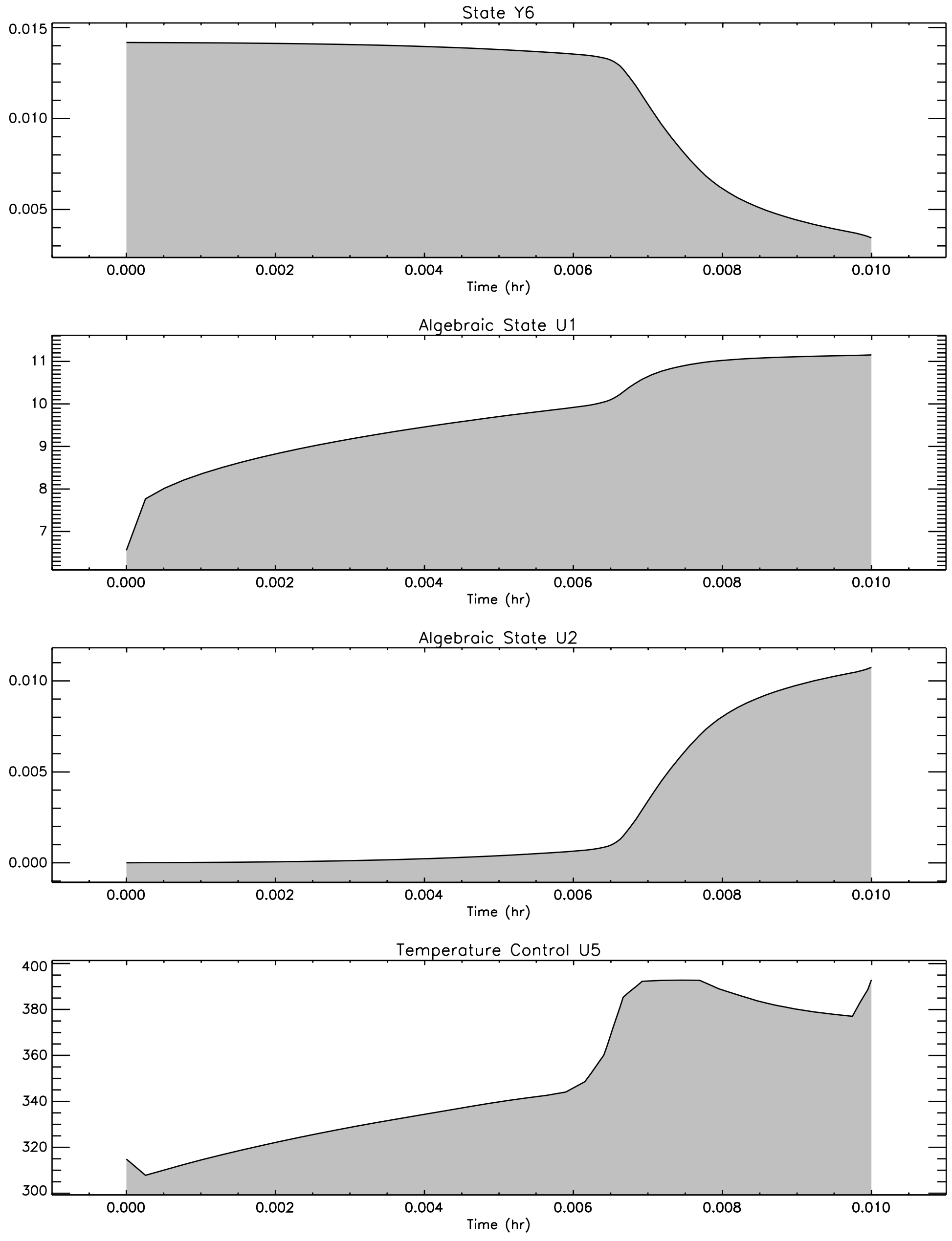


**Figure 6.51.** *Batch reactor transient behavior* $(0 \leq t \leq .01)$.

**Table 6.26.** *Batch reactor mesh-refinement summary.*

| $k$ | $M$ | NGC | NHC | NFE | NRHS | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|
| 1 | 100 | 226 | 213 | 11859 | 1210606 | $1.4918\times10^{-1}$ | $1.6920\times10^{+1}$ |
| 2 | 128 | 79 | 77 | 25099 | 3998429 | $4.5655\times10^{-2}$ | $1.7170\times10^{+1}$ |
| 3 | 157 | 8 | 6 | 2024 | 736266 | $2.0728\times10^{-2}$ | $7.5600\times10^{+0}$ |
| 4 | 236 | 7 | 5 | 1699 | 951477 | $1.5522\times10^{-3}$ | $1.3580\times10^{+1}$ |
| 5 | 249 | 12 | 10 | 3324 | 1807486 | $1.8736\times10^{-5}$ | $1.8260\times10^{+1}$ |
| 6 | 267 | 4 | 2 | 724 | 557582 | $3.1080\times10^{-7}$ | $9.8100\times10^{+0}$ |
| 7 | 411 | 3 | 1 | 399 | 583047 | $5.0851\times10^{-8}$ | $2.2290\times10^{+1}$ |
| Total | 411 | 339 | 314 | 45128 | 9844893 | | $1.0559\times10^{+2}$ |

where $C_D$ is the drag coefficient, and $S$ is the reference area. The aerodynamic force is defined in terms of the earth relative velocity vector

$$\mathbf{v}_r = \mathbf{v} - \boldsymbol{\omega} \times \mathbf{r}, \tag{6.273}$$

where $\boldsymbol{\omega}^{\mathsf{T}} = (0, 0, \omega_E)$ is the angular velocity of the earth relative to inertial space. The density $\rho$ is modeled using a simple exponential atmosphere

$$\rho = \rho_0 e^{(-h/h_0)}, \tag{6.274}$$

where $\rho_0$ is the atmospheric density at sea level, $h = \|\mathbf{r}\| - R_E$ is the altitude, $R_E$ is the spherical earth radius, and $h_0$ is the density scale height. Table 6.27 summarizes the parameters that define the dynamic model.

**Table 6.27.** *Dynamic model parameters.*

| | | |
|---|---|---|
| $\mu$ | $3.986012\times10^{14}$ | m$^3$/sec$^2$ |
| $R_E$ | 6378145 | m |
| $g_0$ | 9.80665 | m/sec$^2$ |
| $h_0$ | 7200 | m |
| $\rho_0$ | 1.225 | kg/m$^3$ |
| $\omega_E$ | $7.29211585\times10^{-5}$ | rad/sec |
| $C_D$ | .5 | (nd) |
| $S$ | $4\pi$ | m$^2$ |

The Delta III expendable launch vehicle had two stages augmented with nine strap-on solid rocket boosters (SRBs). Because of the vehicle configuration it is natural to model the trajectory using four distinct phases. The first phase begins with the vehicle at rest on the ground, when the main engine and six of the nine solid rocket booster SRBs ignite. When the solid rocket burn to depletion, the inert weight is ejected. The second phase begins when the remaining three SRBs are ignited and terminates when the solid propellant is depleted. After ejecting the inert weight for the solid rockets, the third phase, consisting of the main engine only, continues until the remaining liquid propellant in stage 1 is depleted.

After ejecting the inert weight from stage 1, the second stage is ignited and burns until the final orbit is achieved, at the time of second engine cutoff (TSECO). Table 6.28 summarizes the parameters that define the vehicle.

**Table 6.28.** *Vehicle parameters.*

| Description | Symbol | Solid Boosters | Stage 1 | Stage 2 |
|---|---|---|---|---|
| Total Mass, (kg) | $\varpi$ | 19290 | 104380 | 19300 |
| Propellant Mass, (kg) | $\varrho$ | 17010 | 95550 | 16820 |
| Structure Mass, (kg) | $\varphi$ | 2280 | 8830 | 2480 |
| Engine Thrust, (N) | $T$ | 628500 | 1083100 | 110094 |
| Specific Impulse, (sec) | $\mathcal{I}$ | 283.33364 | 301.68776 | 467.21311 |
| Burn Time, (sec) | $\tau$ | 75.2 | 261 | 700 |

The value for a particular component is denoted using the appropriate subscript; e.g., $\varrho_s$ is the propellant mass of a solid booster. Note that unlike [8], the specific impulse is given by $\mathcal{I} = T\tau/(g_0\varrho)$, to ensure consistency with the other vehicle parameters.[7] The mass of the payload is $\varpi_p = 4164$ (kg).

Within each phase of the ascent trajectory the thrust $T$ and mass flow $\xi$ used in the dynamics (6.267)–(6.270) are defined as follows:

**Phase 1** $0 = t_I^{(1)} \le t \le t_F^{(1)} = 75.2,$

$$T = 6T_s + T_1, \tag{6.275}$$

$$\xi = \frac{6T_s}{g_0\mathcal{I}_s} + \frac{T_1}{g_0\mathcal{I}_1}. \tag{6.276}$$

**Phase 2** $75.2 = t_I^{(2)} \le t \le t_F^{(2)} = 150.4,$

$$T = 3T_s + T_1, \tag{6.277}$$

$$\xi = \frac{3T_s}{g_0\mathcal{I}_s} + \frac{T_1}{g_0\mathcal{I}_1}. \tag{6.278}$$

**Phase 3** $150.4 = t_I^{(3)} \le t \le t_F^{(3)} = 261,$

$$T = T_1, \tag{6.279}$$

$$\xi = \frac{T_1}{g_0\mathcal{I}_1}. \tag{6.280}$$

**Phase 4** $261 = t_I^{(4)} \le t \le t_F^{(4)} \le 961,$

$$T = T_2, \tag{6.281}$$

$$\xi = \frac{T_2}{g_0\mathcal{I}_2}. \tag{6.282}$$

[7] Computational results presented here use $g_0 = 9.80665$ m/sec$^2$ and $\omega_E = 7.29211585 \times 10^{-5}$rad/sec. The results in [147] use $g_0 = 9.79827953736866$ and $\omega_E = 7.272205216643040 \times 10^{-5}$.

Furthermore, this vehicle definition determines a mass time line as follows:

$$m[t_I^{(1)}] = 9\varpi_s + \varpi_1 + \varpi_2 + \varpi_p, \tag{6.283}$$

$$m[t_F^{(1)}] = m[t_I^{(1)}] - 6\varrho_s - \frac{\tau_s}{\tau_1}\varrho_1, \tag{6.284}$$

$$m[t_I^{(2)}] = m[t_F^{(1)}] - 6\varphi_s, \tag{6.285}$$

$$m[t_F^{(2)}] = m[t_I^{(2)}] - 3\varrho_s - \frac{\tau_s}{\tau_1}\varrho_1, \tag{6.286}$$

$$m[t_I^{(3)}] = m[t_F^{(2)}] - 3\varphi_s, \tag{6.287}$$

$$m[t_F^{(3)}] = m[t_I^{(3)}] - \left(1 - 2\frac{\tau_s}{\tau_1}\right)\varrho_1, \tag{6.288}$$

$$m[t_I^{(4)}] = m[t_F^{(3)}] - \varphi_1, \tag{6.289}$$

where $m[t_I^{(k)}]$ is the mass at the beginning of phase $k$, and $m[t_F^{(k)}]$ is the mass at the end of the phase.

The launch vehicle begins at rest relative to the earth with the following initial state vector:

$$\mathbf{r}(0) = \mathbf{r}_0 \;=\; [R_E \cos\psi_L \,,\, 0 \,,\, R_E \sin\psi_L]^\mathsf{T}, \tag{6.290}$$

$$\mathbf{v}(0) = \mathbf{v}_0 \;=\; -\boldsymbol{\omega} \times \mathbf{r}_0, \tag{6.291}$$

$$m(0) = m_0 \;=\; 9\varpi_s + \varpi_1 + \varpi_2 + \varpi_p, \tag{6.292}$$

where $\psi_L = 28.5$ (deg) is the (geocentric) latitude of the launch site at Cape Canaveral, and (6.291) ensures the relative velocity given by (6.273) is zero. At the final time $t_f = t_F^{(4)}$, the launch vehicle must insert the payload into a geosynchronous transfer orbit (GTO). The classical elements

$$a_f = 24361140\,(\text{m}), \tag{6.293}$$

$$e_f = .7308, \tag{6.294}$$

$$i_f = 28.5\,(\text{deg}), \tag{6.295}$$

$$\Omega_f = 269.8\,(\text{deg}), \tag{6.296}$$

$$\omega_f = 130.5\,(\text{deg}), \tag{6.297}$$

referred to as semimajor axis, eccentricity, inclination, right ascension of the ascending node (RAAN), and argument of perigee, respectively, can all be computed from the terminal state vector $(\mathbf{r}_f, \mathbf{v}_f) = (\mathbf{r}[t_F^{(4)}], \mathbf{v}[t_F^{(4)}])$. Continuity in the position and velocity from phase to phase is enforced by imposing the conditions

$$\mathbf{r}[t_F^{(k)}] = \mathbf{r}[t_I^{(k+1)}], \tag{6.298}$$

$$\mathbf{v}[t_F^{(k)}] = \mathbf{v}[t_I^{(k+1)}] \tag{6.299}$$

for $k = 1,2,3$. Finally, the mass is linked by imposing (6.285), (6.287), and (6.289), which incorporates the requisite jettison of engine structure.

The objective is to choose the control vector $\mathbf{u}(t)$, and final time $t_f$, to maximize the final mass

$$F = m(t_f) \tag{6.300}$$

subject to the DAE constraints (6.267)–(6.271), with initial conditions (6.290)–(6.292), boundary conditions (6.293)–(6.297), and linkage conditions (6.298), (6.299), (6.285), (6.287), and (6.289).

A guess is required to initiate the iterative solution of the optimal control problem, and for the results presented here a simple linear interpolant of the dynamic variables was utilized. Specifically a guess for the state is

$$\mathbf{y}(t) = \mathbf{y}_0 + (\mathbf{y}_f - \mathbf{y}_0)\frac{t}{t_f}, \tag{6.301}$$

where $0 \le t \le t_f$. The initial state $\mathbf{y}_0$ can be defined immediately from (6.290)–(6.292). If we guess $t_f = 961$, the final mass is just $m_f = m[t_F^{(4)}] = m[t_I^{(4)}] - \varrho_2$, where (6.289) is used. A guess for the final values of the position and velocity vector $(\mathbf{r}_f, \mathbf{v}_f)$ can be constructed from the values of the classical elements (6.293)–(6.297) provided a guess for the true anomaly $\nu_f$ is also supplied. One can simply guess $\nu_f = 0$ or guess a value of $\nu_f$ such that the altitude has a specified value. For the results given, we guess $\nu_f$ such that $h_f = 200$ (km). The initial guess for the control angles is just $\mathbf{u}^\mathsf{T}(t) = (1,0,0)$.

The optimal steering commands $\mathbf{u}(t)$ are plotted in Figure 6.52. Figure 6.53 illustrates the optimal solution for the position $\mathbf{r}(t)$ and velocity state $\mathbf{v}(t)$. Figure 6.54 displays the history for the mass $m(t)$, the altitude $h(t) = \|\mathbf{r}(t)\| - R_E$, and the velocity $\|\mathbf{v}(t)\|$. The optimal value for the final mass is $F^* = m^*(t_f) = 7529.712412$ (kg), which occurs when $t_f = 924.139$ (sec).

Table 6.29 summarizes some of the important performance characteristics of the $\mathbb{SOCS}$ algorithm. Four mesh-refinement iterations were required to reduce the discretization error below the requested tolerance $\epsilon_{\max} = 10^{-7}$. The first refinement iteration used a trapezoidal discretization (denoted “TR”), with 10 grid points in each phase. The resulting sparse NLP had $n = 401$ variables, with 76 degrees of freedom (NDOF). This nonlinear program required solving 91 QP subproblems, which was completed in 2.00 seconds of CPU time, and yields a discretization error of $\epsilon_{\max} = 7.4 \times 10^{-4}$. To improve the solution accuracy, the mesh-refinement procedure did two things: in phase 1 it changed the discretization technique from trapezoidal to separated Hermite–Simpson (HS), and in the remaining phases increased the number of grid points from 10 to 19. This larger NLP with 761 variables was solved in .16 seconds and required six QP subproblems. Two additional mesh-refinement iterations were required, all using a compressed Hermite–Simpson (HC) discretization to achieve the requested accuracy, with the overall solution process taking 3.13 sec of CPU time. Observe that the number of QP iterations needed to solve the NLP problems does not grow with the number of degrees of freedom!

In comparison, the Gauss pseudospectral method implemented in the GPOCS software was also used to solve this problem. Using 20 nodes per phase, with no mesh refinement, the NLP subproblem was solved using the SNOPT [94] NLP algorithm. This formulation leads to a problem with 864 variables and 694 constraints. The GPOCS/SNOPT algorithm requires 1674 major iterations (QP subproblems) compared with 64 for the $\mathbb{SOCS}$

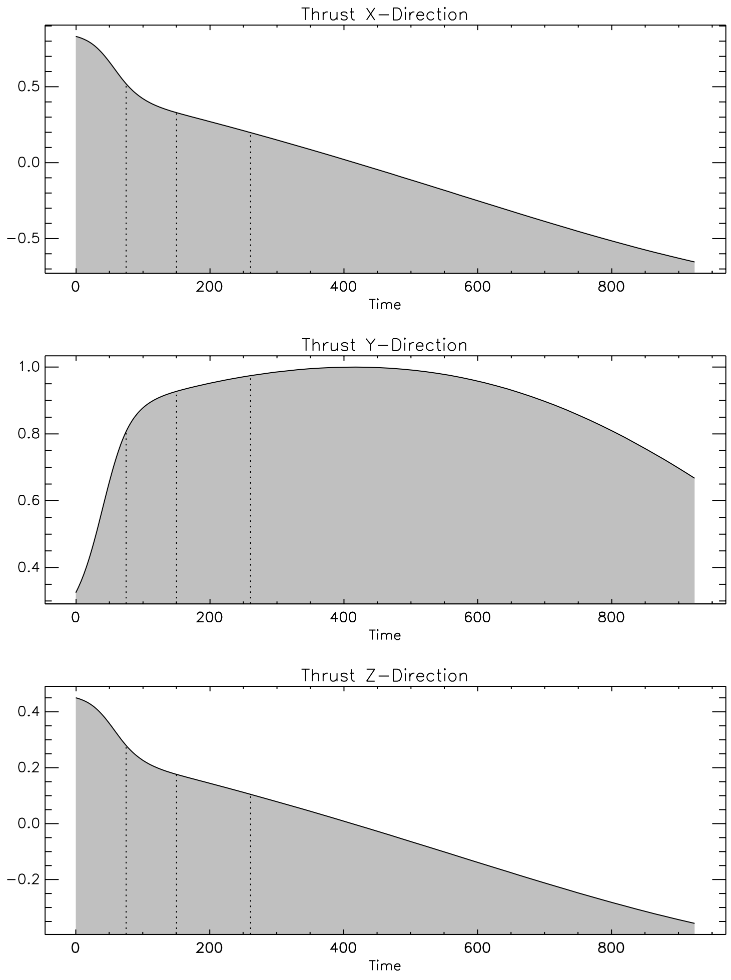


**Figure 6.52.** *Delta* III *ascent; controls* $\mathbf{u}(t)$.

results. The total solution time for the GPOCS/SNOPT algorithm was 2724.73 sec compared to 3.13 sec using $\mathbb{SOCS}$. The primary reason for this performance difference can be attributed to the underlying NLP algorithm. SNOPT constructs the Hessian matrix using a quasi-Newton method which requires 1674 iterations for this problem with 864 variables. The quasi-Newton approximation also does not exploit sparsity in the Hessian matrix. In contrast, $\mathbb{SOCS}$ is a Newton method and does exploit Hessian sparsity.

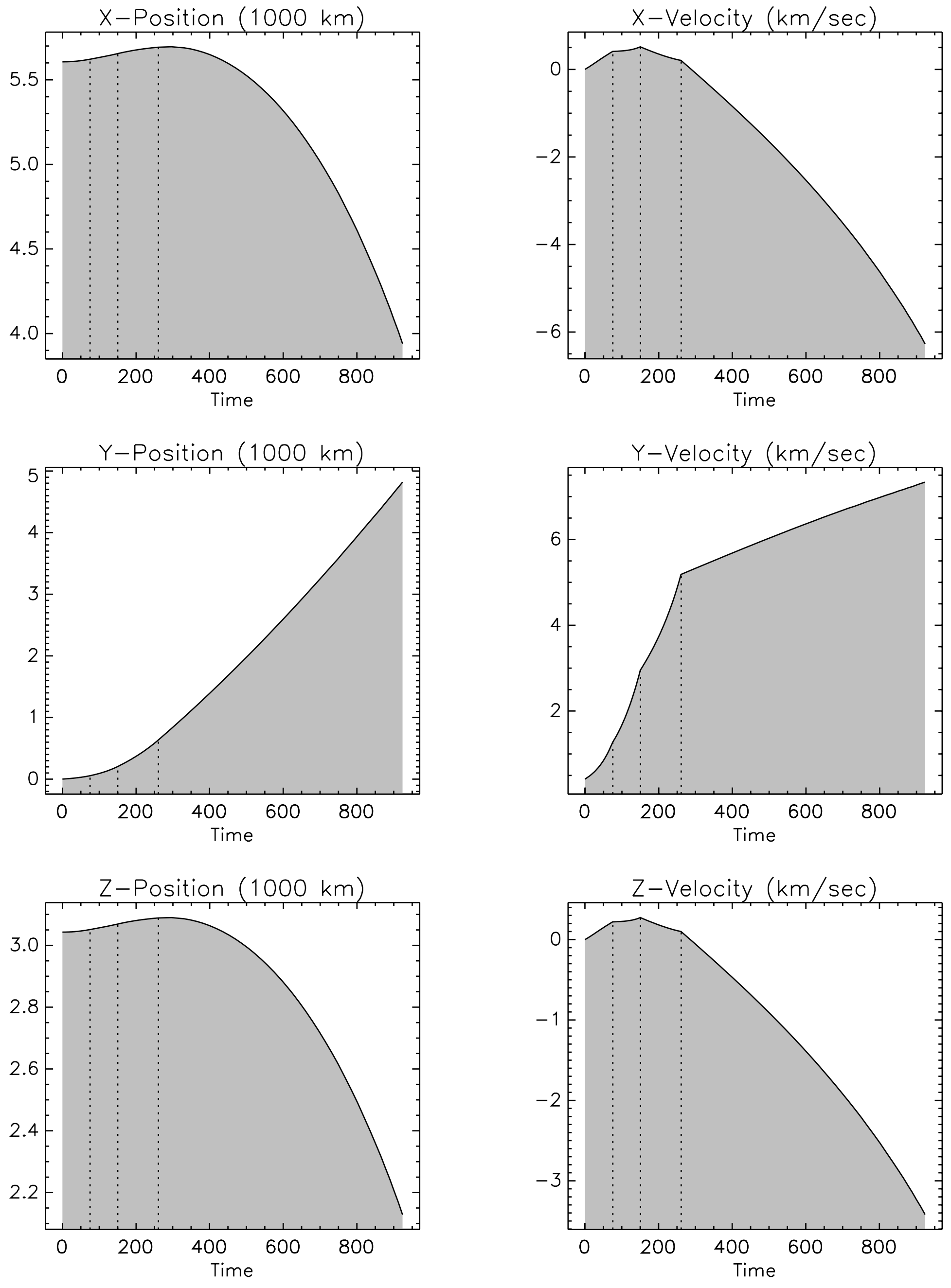


**Figure 6.53.** *Delta* III *ascent; states* $\mathbf{r}(t)$, $\mathbf{v}(t)$.

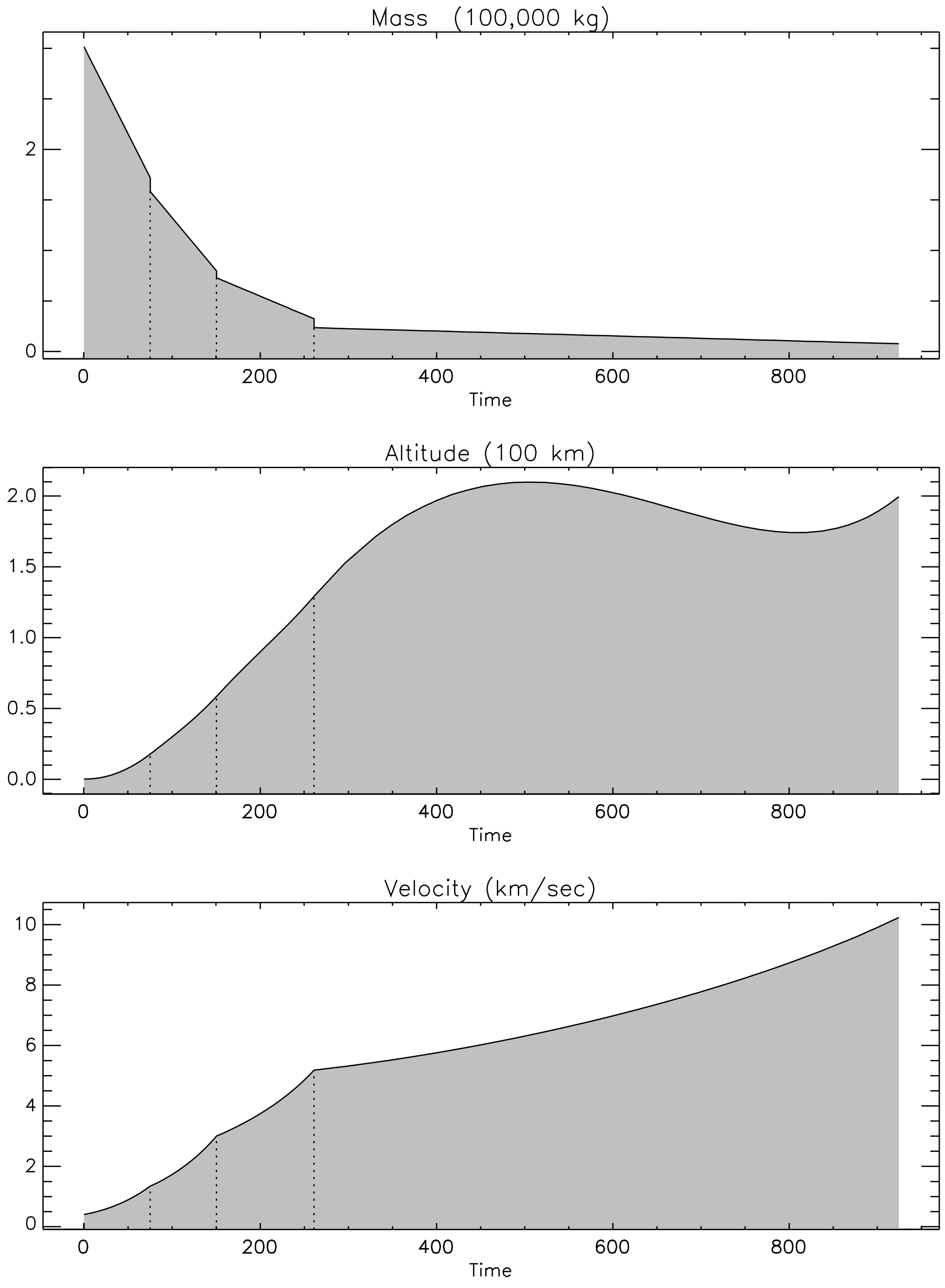


**Figure 6.54.** *Delta* III *ascent; mass, altitude, velocity.*

**Table 6.29.** *Delta* III *mesh-refinement summary.*

| $k$ | Disc. | $M$ | $n$ | NDOF | NQP | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|
| 1 | (TR,TR,TR,TR) | (10,10,10,10) | 401 | 76 | 91 | $7.4\times10^{-4}$ | 2.00 |
| 2 | (HS,TR,TR,TR) | (10,19,19,19) | 761 | 148 | 6 | $6.9\times10^{-5}$ | .16 |
| 3 | (HC,HC,HC,HC) | (19,19,19,19) | 977 | 292 | 5 | $2.5\times10^{-6}$ | .37 |
| 4 | (HC,HC,HC,HC) | (37,30,37,37) | 1822 | 552 | 3 | $9.9\times10^{-8}$ | .60 |
| Total | | 141 | | | 105 | | 3.13 |

## 6.16 A Two-Strain Tuberculosis Model

**Example 6.20** TWO-STRAIN TUBERCULOSIS MODEL. In their paper Jung, Lenhart, and Feng [119] describe a model for two-strain tuberculosis treatment as follows:

> In the absence of an effective vaccine, current control programs for TB have focused on chemotherapy. The antibiotic treatment for an active TB (with drug-sensitive strain) patient requires a much longer period of time and a higher cost than that for those who are infected with sensitive TB but have not developed the disease. Lack of compliance with drug treatments not only may lead to a relapse but to the development of antibiotic resistant TB—one of the most serious public health problems facing society today. A report released by the World Health Organization warns that if countries do not act quickly to strengthen their control of TB, the multi-drug resistant strains that have cost New York City and Russia hundreds of lives and more than \$1 billion each will continue to emerge in other parts of the world. The reduction in cases of drug sensitive TB can be achieved either by "case holding," which refers to activities and techniques used to ensure regularity of drug intake for a duration adequate to achieve a cure, or by "case finding," which refers to the identification (through screening, for example) of individuals latently infected with sensitive TB who are at high risk of developing the disease and who may benefit from preventive intervention. These preventive treatments will reduce the incidence (new cases per unit of time) of drug sensitive TB and hence indirectly reduce the incidence of drug resistant TB.

The dynamic model divides the host population into distinct epidemiological classes, in which the population of each class is treated as a state variable. Thus, the total population satisfies the relation

$$N = S + L_1 + I_1 + L_2 + I_2 + T, \tag{6.302}$$

where Table 6.30 describes the individual states as well as their respective initial conditions.

**Table 6.30.** *TB model state variables.*

| Description | State | Initial Condition |
|---|---|---|
| Susceptible | $S(t)$ | $76N/120$ |
| Treated Effectively | $T(t)$ | $1N/120$ |
| Latent, infected with typical TB, not infectious | $L_1(t)$ | $36N/120$ |
| Latent, infected with resistant TB, not infectious | $L_2(t)$ | $2N/120$ |
| Infectious, with typical TB | $I_1(t)$ | $4N/120$ |
| Infectious, with resistant TB | $I_2(t)$ | $1N/120$ |

The dynamic behavior of this system is described by the following system of ODEs:

$$\dot{S} = \Lambda - \beta_1 S \frac{I_1}{N} - \beta^* S \frac{I_2}{N} - \mu S, \tag{6.303}$$

$$\dot{T} = u_1 r_1 L_1 - \mu T + (1-(1-u_2)(p+q)) r_2 I_1 - \beta_2 T \frac{I_1}{N} - \beta^* T \frac{I_2}{N}, \tag{6.304}$$

$$\dot{L}_1 = \beta_1 S \frac{I_1}{N} - (\mu + k_1) L_1 - u_1 r_1 L_1 + (1-u_2) p r_2 I_1 + \beta_2 T \frac{I_1}{N} - \beta^* L_1 \frac{I_2}{N}, \tag{6.305}$$

$$\dot{L}_2 = (1-u_2) q r_2 I_1 - (\mu + k_2) L_2 + \beta^* (S + L_1 + T) \frac{I_2}{N}, \tag{6.306}$$

$$\dot{I}_1 = k_1 L_1 - (\mu + d_1) I_1 - r_2 I_1, \tag{6.307}$$

$$\dot{I}_2 = k_2 L_2 - (\mu + d_2) I_2. \tag{6.308}$$

The dynamic model incorporates two variables $u_1(t)$ and $u_2(t)$, referred to as "case finding" and "case holding" controls, respectively. The "case finding" control represents the fraction of typical TB latent individuals that are identified and put under treatment. The effort that prevents failure of the treatment of the typical TB infectious individuals appears as the coefficient $1 - u_2(t)$. Thus when the "case holding" control $u_2(t)$ is near 1, the implementation costs are high, but there is low treatment failure. Since the goal of the model is to reduce the latent and infectious groups with resistant-strain TB, while also keeping treatment costs low, they propose minimizing the composite objective

$$F = \int_0^{t_F} \left[ L_2 + I_2 + \frac{1}{2} B_1 u_1^2 + \frac{1}{2} B_2 u_2^2 \right] dt. \tag{6.309}$$

The control variables are bounded $.05 \le u_k(t) \le .95$ and the final time is fixed $t_F = 5$ (years). For the numerical results they also assume the total population $N$ is constant, so $\Lambda = \mu N$, and $d_1 = d_2 = 0$. Finally, the weight factors $B_1 = 50$ and $B_2 = 500$ emphasize the cost of holding patients in treatment in comparison to screening and finding them in the first place. Table 6.31 summarizes the various parameters used in the model.

**Table 6.31.** *TB model parameters.*

| | | |
|---|---|---|
| Infection rate (susceptible) | $\beta_1$ | 13 |
| Infection rate (treated) | $\beta_2$ | 13 |
| Per capita natural death rate | $\mu$ | .0143 |
| Per capita death rate induced by typical TB | $d_1$ | 0 |
| Per capita death rate induced by resistant TB | $d_2$ | 0 |
| Rate individual in $L_1$ becomes infectious | $k_1$ | .5 |
| Rate individual in $L_2$ becomes infectious | $k_2$ | 1 |
| Treatment rate for individual with latent, typical TB | $r_1$ | 2 |
| Treatment rate for individual with infectious, typical TB | $r_2$ | 1 |
| Fraction of $I_1$ not completing treatment | $p$ | .4 |
| Fraction of $I_2$ not completing treatment | $q$ | .1 |
| Total population $N = S + L_1 + I_1 + L_2 + I_2 + T$ | $N$ | 30000 |
| Infection rate (uninfected) | $\beta^*$ | .029 |
| Weight factor of "case finding" control $u_1$ | $B_1$ | 50 |
| Weight factor of "case holding" control $u_2$ | $B_2$ | 500 |
| Recruitment Rate | $\Lambda$ | $\mu N$ |

The optimal solution obtained using $\mathbb{SOCS}$ is illustrated in Figure 6.55 and yields an optimal objective function value of $F^* = 5.1520731 \times 10^3$. The total number of individuals infected with resistant TB at the final time is $L_2 + I_2 = 241.57031 + 881.34819 \approx 1123$.

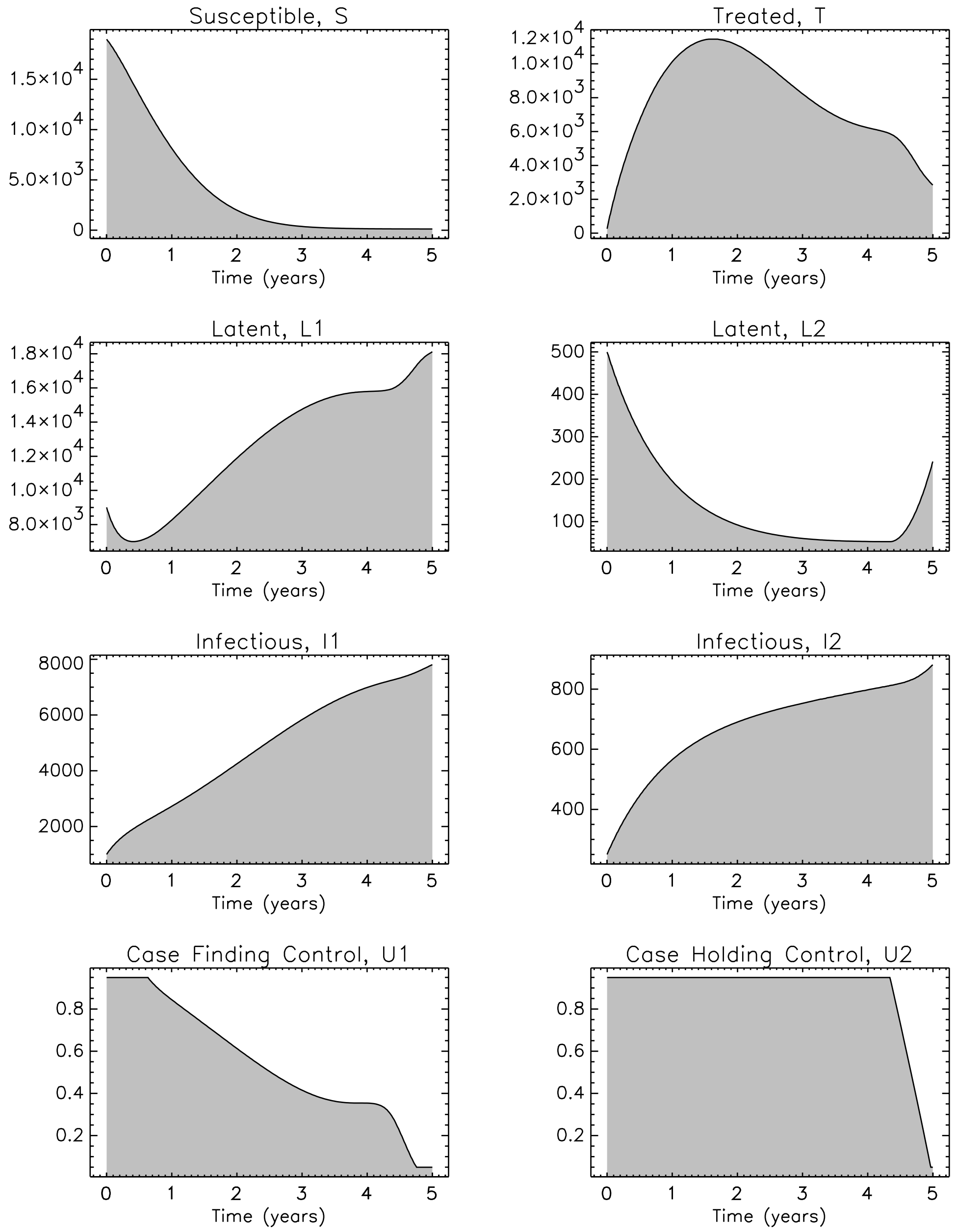


**Figure 6.55.** *Optimal state and control.*

## 6.17 Tumor Anti-angiogenesis

**Example 6.21** TUMOR ANTI-ANGIOGENESIS. In their paper Ledzewicz and Schättler [128] state

> ... the search for cancer treatment methods that would circumvent the problem of drug resistance is of tantamount importance. One such approach is tumour anti-angiogenesis.
>
> A growing tumour, after it reaches just a few millimetres in size, no longer can rely on blood vessels of the host for its supply of nutrients, but it needs to develop its own vascular system for blood supply. In this process, called *angiogenesis*, there is a bi-directional signaling between tumour cells and endothelial cells: tumour cells produce vascular endothelial growth factor (VEGF) to stimulate endothelial cell growth; endothelial cells in turn provide the lining for the newly forming blood vessels that supply nutrients to the tumour and thus sustain tumour growth. But endothelial cells also have receptors that make them sensitive to inhibitors of inducers of angiogenesis like, for example, endostatin, and pharmacological therapies typically target the growth factor VEGF trying to impede the development of new blood vessels and capillaries and thus block its growth. The tumour, deprived of necessary nutrition, regresses. Since the treatment targets normal cells, no occurrence of drug resistance has been reported in laboratory studies ... For this reason tumour anti-angiogenesis has been called a therapy resistant to resistance which provides a new hope in the treatment of tumour-type cancers.

In [128] the interaction between tumor cells and endothelial cells is described by a system with the primary tumor volume, $p$, and the carrying capacity of the vascular, $q$, as variables. The control which corresponds to the angiogenic dose rate is bounded $0 \le u(t) \le a$, and a constraint on the total anti-angiogenic treatment administered

$$\int_0^{t_F} u(t)dt \le A$$

is also imposed. An additional variable $y$ equal to the integral is introduced so that the constraint can be written as $y(t_F) \le A$. The resulting dynamics are thus given by

$$\dot{p} = -\xi p \ln\left(\frac{p}{q}\right), \tag{6.310}$$

$$\dot{q} = q\left[b - (\mu + dp^{\frac{2}{3}} + Gu)\right], \tag{6.311}$$

$$\dot{y} = u \tag{6.312}$$

for $0 \le t \le t_F$, with initial conditions $p(0) = p_0$, $q(0) = q_0$, and $y(0) = 0$. The constant $\xi$ denotes a tumor growth parameter, while $G$ is a constant that represents the anti-angiogenic killing parameter. The "birth" rate is modeled by the constant $b$, the "death" rate is given by $d$, and $\mu$ is a small parameter introduced to describe the loss of endothelial cells due to natural causes. The system has an asymptotically stable focus at $\bar{p} = \bar{q} = \left[(b-\mu)/d\right]^{3/2}$, and consequently the domain is restricted to $0 < p \le \bar{p}$ and $0 < q \le \bar{q}$. Ledzewicz and Schättler [128] demonstrate the problem is well-posed provided the initial conditions satisfy $p_0 \ge q_0$. For our illustration we choose $p_0 = \bar{p}/2$ and $q_0 = \bar{q}/4$ with the remaining parameters defined in Table 6.32. The final time $t_F$ is free and the goal is to minimize the size of the tumor at the final time, i.e., minimize $p(t_F)$.

**Table 6.32.** *Tumor model parameters.*

| Parameter | Value |
|---|---|
| $\xi$ | 0.084 per day |
| $b$ | 5.85 per day |
| $d$ | 0.00873 per mm$^2$ per day |
| $G$ | 0.15 kg per mg of dose per day |
| $\mu$ | 0.02 per day |
| $a$ | 75 |
| $A$ | 15 |

Because the control appears linearly in the differential equations the solution is "bang-bang," and a complete analysis of the necessary conditions is given in [128]. In particular, the adjoint equations are

$$\dot{\lambda}_1 = \xi\lambda_1\left[\ln\left(\frac{p}{q}\right)+1\right]+\frac{2}{3}\lambda_2 dqp^{-\frac{1}{3}}, \tag{6.313}$$

$$\dot{\lambda}_2 = -\xi\lambda_1\frac{p}{q}+\lambda_2\left[b-(\mu+dp^{\frac{2}{3}}+Gu)\right], \tag{6.314}$$

$$\dot{\lambda}_3 = 0 \tag{6.315}$$

with Hamiltonian function

$$H = -\lambda_1\xi p\ln\left(\frac{p}{q}\right)+\lambda_2 q\left[b-(\mu+dp^{\frac{2}{3}}+Gu)\right]+\lambda_3 u. \tag{6.316}$$

The Hamiltonian is minimized by choosing the optimal control given by

$$u^*(t) = \begin{cases} 0 & \text{if } \Phi(t) > 0, \\ a & \text{if } \Phi(t) < 0, \end{cases} \tag{6.317}$$

where the switching function is

$$\Phi(t) = \lambda_3(t) - \lambda_2(t)Gq(t). \tag{6.318}$$

For our illustration there is a single switch time $t_s$ that occurs when the switching function is zero, and as such it is natural to consider a formulation with two distinct phases. During phase 1, the control is at its upper bound $u(t) = a$ and during phase 2 at the lower bound $u(t) = 0$. The indirect formulation is complete when the remaining boundary conditions

$$\lambda_1(t_F) = 1, \qquad H(t_F) = 0, \tag{6.319}$$

$$\lambda_2(t_F) = 0, \qquad \Phi(t_s) = 0 \tag{6.320}$$

are satisfied.

The simplest way to compute numerical results for this problem is to apply the $\mathbb{SOCS}$ direct transcription method. This approach does not require knowledge of the switching structure, does not require implementation of the indirect necessary conditions, and does not require initial estimates for the adjoint variables. Table 6.33 briefly summarizes the mesh-refinement history and solution time needed to reach the requested accuracy of $\epsilon < 10^{-7}$.

A second alternative is to incorporate knowledge of the phase structure within the direct formulation. Specifically, one can formulate a problem with two phases. During

**Table 6.33.** *Mesh refinement—direct formulation.*

| $k$ | $M$ | $\epsilon$ | Time (sec) |
|---|---|---|---|
| 1 | 60 | $4\times10^{-5}$ | .11 |
| 2 | 119 | $2\times10^{-6}$ | 1.0 |
| 3 | 119 | $7\times10^{-7}$ | 1.2 |
| 4 | 124 | $5\times10^{-8}$ | .67 |
| Total | 124 | | 2.98 |

**Table 6.34.** *Method comparison.*

| Method | $p^*(t_F)$ | $t_F^*$ |
|---|---|---|
| One-Phase Direct | $7.5716831\times10^3$ | 1.1954773 |
| Two-Phase Direct | $7.5716700\times10^3$ | 1.1963497 |
| Two-Phase Indirect | $7.5716701\times10^3$ | 1.1963496 |

phase one the control is $u(t) = a$, and during phase two $u(t) = 0$. The switch time $t_s$ is treated as free. This approach incorporates knowledge about the discontinuous behavior in $u(t)$ without explicitly imposing the switching conditions. Using the solution from the single phase formulation provides an excellent initial guess for the two phase problem, and convergence is extremely rapid. Finally the third alternative is to derive the adjoint and transversality conditions and then solve the indirect formulation using either an indirect collocation or indirect shooting method. The discrete adjoint estimates from the second formulation provide an excellent guess for this approach, and again converged results are obtained easily. Table 6.34 presents a comparison of the results obtained by each method, all with the requested accuracy of $\epsilon < 10^{-7}$. Clearly the two-phase results are consistent to seven significant figures, whereas the one-phase direct result differs slightly in the seventh digit. This small inaccuracy can be attributed to the method used to represent the optimal bang-bang control. In particular when two phases are used, the step discontinuity in the control can be modeled accurately. In contrast, when the problem is treated using a single phase, the control history is continuous by construction, and the mesh must be refined in order to approximate the step discontinuity. Figure 6.56 illustrates the continuous control approximation (solid line) using the one-phase direct method, compared with the discon-

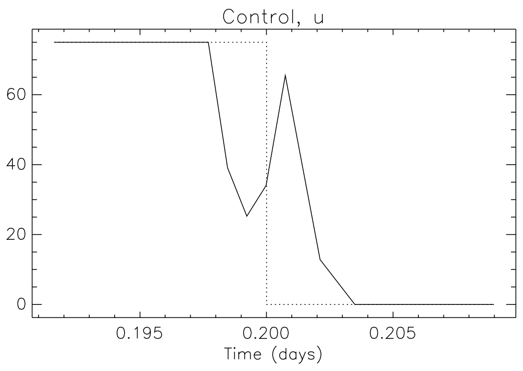


**Figure 6.56.** *Control.*

tinuous (dotted line) control from the two-phase method. Figure 6.57 illustrates the optimal states, adjoints, and switching function.

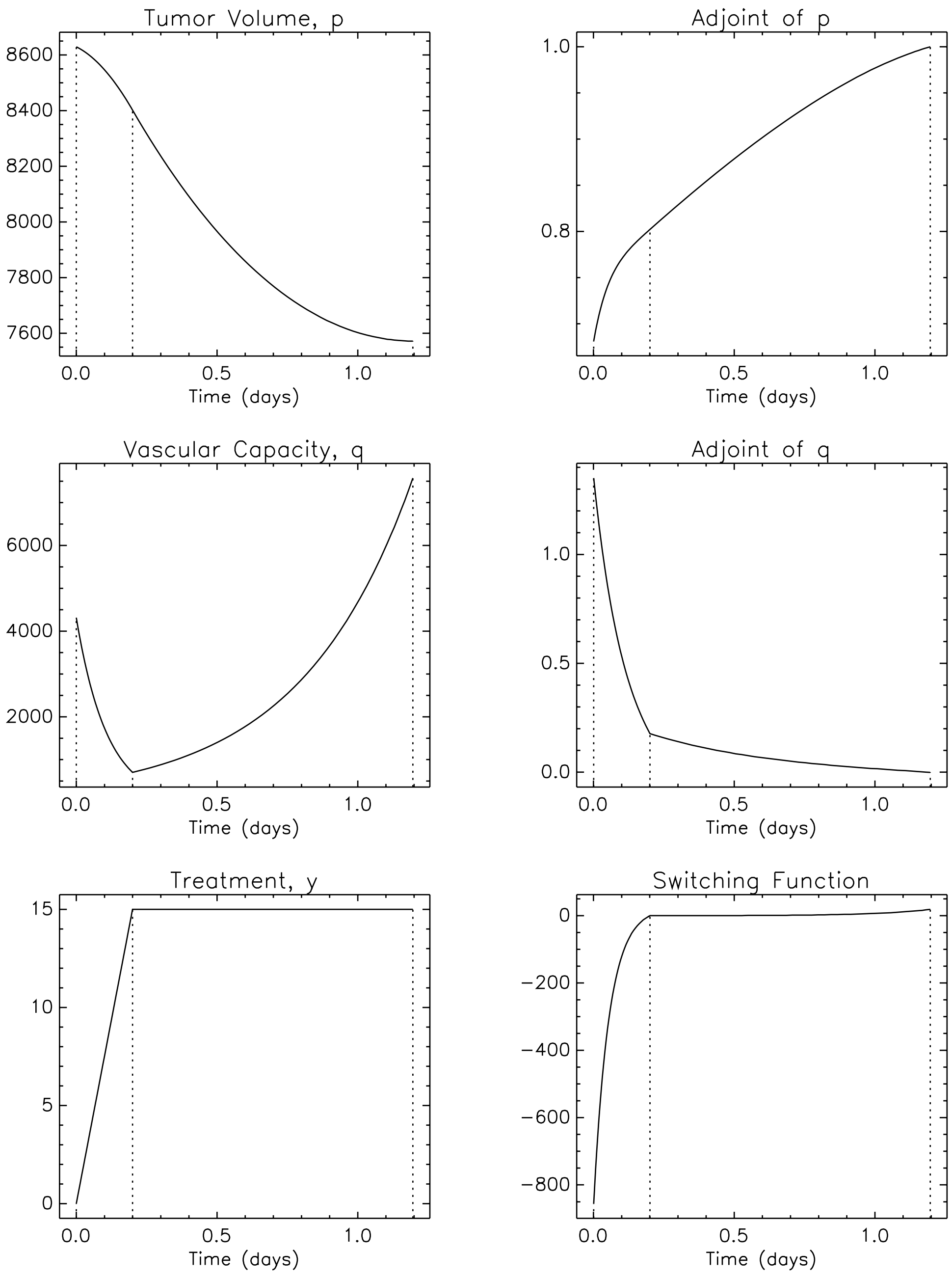


**Figure 6.57.** *States.*

This problem illustrates two important points. First, if the control (or state) is discontinuous at the solution, it is important to formulate the problem using multiple phases. The multiphase formulation permits an accurate representation of discontinuous behavior. Second, when mesh refinement is used with a direct method there is no loss of accuracy compared to an indirect method. In short, a direct and an indirect method have comparable accuracy. Thus, as a practical matter, there is no reason to derive the adjoint equations, unless they are needed for some other purpose.

Chapter 7

# Advanced Applications

Most effective numerical methods are based on the following premise:

*A hard problem can be broken into a sequence of easy subproblems.*

For example, to compute the root of a *nonlinear* constraint $c(x) = 0$ using Newton's method, one must solve the sequence of *linear* approximations $x_{k+1} = x_k - c(x_k)/c'(x_k)$. Similarly, when using a nonlinear programming (NLP) algorithm to solve a *nonlinear optimization* problem, one solves a sequence of (a) *quadratic programming* subproblems (an SQP method) or (b) *unconstrained* subproblems (a barrier method). In like fashion an optimal control solution can be obtained by solving a sequence of NLP subproblems. Of course there may be no clear definition of an "easy subproblem." For Newton's method is it better to compute $c'(x_k)$ or use a secant approximation? Is a quadratic program easier than an unconstrained subproblem? Ultimately the solution technique should consist of a sequence of subproblems that can be solved efficiently and reliably.

This chapter presents a number of advanced applications that require using a combination of the techniques presented in the book.

## 7.1 Optimal Lunar Swingby Trajectories

### 7.1.1 Background and Motivation

With the growth of the space age, it has become desirable to place spacecraft into a wide variety of mission orbits. Typically a launch vehicle is used to deploy a satellite into a low altitude park orbit. It then becomes necessary to transfer from the park orbit to the mission orbit. A major goal of mission design is to compute this *orbit transfer* to minimize some performance objective. The most common objective is to minimize the fuel consumed by the orbit transfer. For spacecraft using propulsion systems whose thrust is high compared to the mass of the vehicle, i.e., so-called *high thrust* systems, the duration of the burns is very short compared to the duration of the orbit transfer itself. Consequently it is common to model high thrust systems using an instantaneous or *impulsive* velocity change. In essence a minimum fuel orbit transfer is approximated using a so-called *minimum $\Delta v$ transfer.* In 1925, the German scientist Walter Hohmann demonstrated that a minimum $\Delta v$ transfer between two circular orbits in the same plane could be achieved using two impulsive velocity increments.

Although the original *Hohmann transfer* was strictly between coplanar orbits, the term is often loosely used to describe any two-impulse transfer. For example, one of the most common applications involves transferring between a circular orbit with an altitude of 150 nm and inclination of 28.5 deg (a standard shuttle park orbit) to a geosynchronous orbit which is circular at an altitude of 19323 nm and inclination of 0 deg. The Hohmann transfer for this application is illustrated in Figure 6.21. However, the park and mission orbits are not coplanar in this case, and some of the requisite inclination change must be accomplished by each of the $\Delta v$. For this transfer nearly all of the inclination change ($\approx$ 26 deg) is achieved by the second impulse. In fact for any orbit transfer it is most efficient to change the orbital plane at a high altitude when the velocity is smallest.

Because changing the orbit plane is the most expensive portion of the transfer one is led to consider a *three-burn transfer*. For this type of transfer, the second burn does all of the plane change at a very high altitude. In fact it was demonstrated in Betts [12] that a three-impulse solution is more efficient than two impulses for some transfers that require large plane changes. Figure 7.1 illustrates the situation. A critical property of a three-burn transfer is that the second burn be located a "long" distance from the primary body (earth). Thus one is led to consider locating the second impulse at or near the moon. In effect the lunar gravitational attraction can be used in lieu of the second burn, thereby producing a very efficient trajectory. The concept of using a lunar gravity assist to design a nominal mission trajectory has been considered by many authors (cf. [172]). Lee et al. [129] discuss using a lunar swingby; however, as with most analyses they do not minimize the impulsive velocity. A more detailed presentation is given in [22].

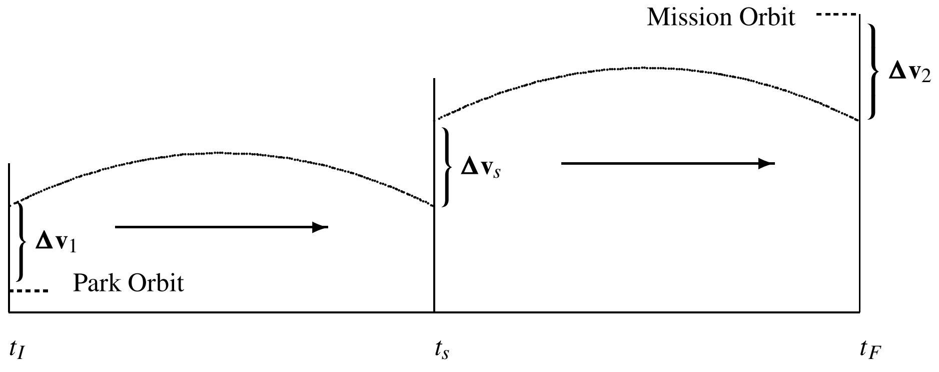


**Figure 7.1.** *Optimal three-impulse transfer orbit.*

This application deals with an approach for computing optimal lunar swingby trajectories between two earth orbits. First some representative optimal transfers are presented that illustrate the performance benefits. Then the problem formulation is explained and a number of challenging numerical issues are addressed. To facilitate using the examples as benchmark problems, the dynamics are modeled using three-body mechanics but do not require more sophisticated planetary ephemerides. The overall solution procedure requires solution of a dense nonlinear program, an optimal control problem, and an optimal estimation problem.

### 7.1.2 Optimal Lunar Transfer Examples

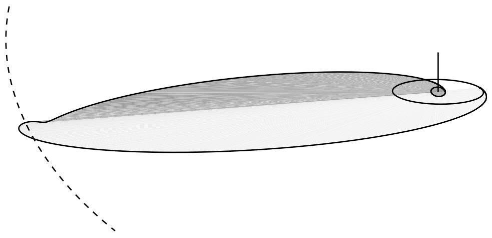
For the sake of illustration examples of lunar swingby transfers to four different mission orbits are presented. All of the transfers begin in a circular park orbit at an altitude of 150nm, with an inclination of 28 deg. The outbound transfer orbit is established by $\mathbf{\Delta v}_1$, and after passing around the moon the inbound transfer returns to the mission orbit which is established by applying $\mathbf{\Delta v}_2$. All of the transfers minimize the total $\Delta v$, that is,

$$F = \|\mathbf{\Delta v}_1\| + \|\mathbf{\Delta v}_2\|. \tag{7.1}$$

To obtain meaningful results it is also necessary to impose some limit on the total transfer time. An upper bound of 15 days is imposed on the total transfer time for the examples.

#### Synchronous Equatorial

Results are presented for a geosynchronous orbit which is circular at an altitude of 19323 nm and inclination of 0 deg. Figure 7.2 illustrates the optimal lunar swingby trajectory and for comparison summarizes the total $\Delta v$ of the swingby as well as a standard two-impulse Hohmann transfer. It is interesting to observe that even for this case, which requires only 28.5 degrees of plane change, the swingby solution saves 188.01 fps of total $\Delta v$.



| | $\|\Delta V_1\|$ | $\|\Delta V_2\|$ | Total (fps) |
|---|---|---|---|
| Hohmann | 8056.67 | 5851.44 | 13908.12 |
| Swingby | 10201.39 | 3518.72 | 13720.11 |

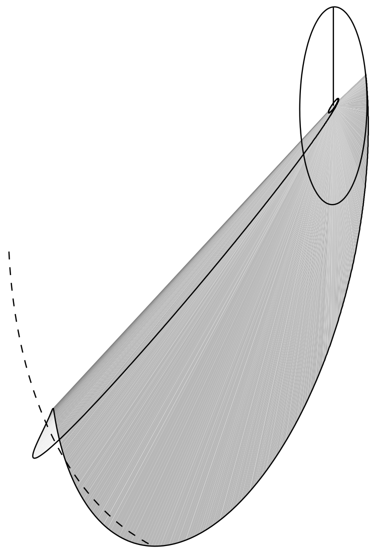
**Figure 7.2.** *Optimal swingby transfer to geosynchronous orbit.*

#### Polar, 24 hr (A)

Figure 7.3 illustrates the optimal lunar swingby trajectory to an orbit which is circular at an altitude of 19323 nm and inclination of 90 deg. This solution is characterized by an outbound transfer trajectory from the descending node of the park orbit and as such will be referred to as solution "A." The total plane change for this transfer is 61.5 deg, which is achieved using 3059.56 fps less than the corresponding Hohmann transfer.

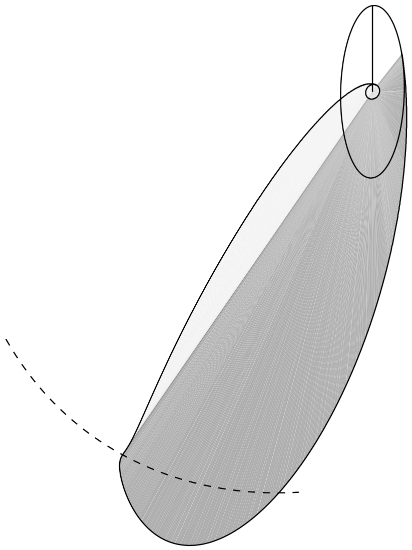
#### Polar, 24 hr (B)

Figure 7.4 illustrates the optimal lunar swingby trajectory to the same mission orbit as in section 7.1.2. In contrast to solution "A," this trajectory is characterized by an outbound



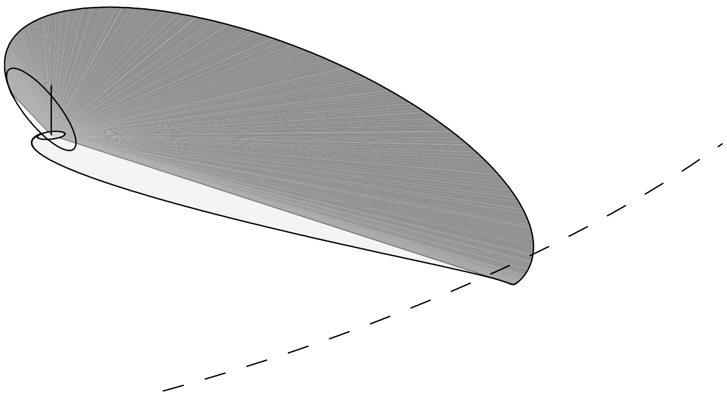
| | $\|\Delta V_1\|$ | $\|\Delta V_2\|$ | Total (fps) |
|---|---|---|---|
| Hohmann | 8113.34 | 8610.82 | 16724.17 |
| Swingby | 10225.16 | 3440.44 | 13665.61 |

**Figure 7.3.** *Optimal swingby transfer to polar,* 24 *hr orbit.*



| | $\|\Delta V_1\|$ | $\|\Delta V_2\|$ | Total (fps) |
|---|---|---|---|
| Hohmann | 8113.33 | 8610.77 | 16724.10 |
| Swingby | 10251.24 | 3459.66 | 13710.90 |

**Figure 7.4.** *Optimal swingby transfer to polar,* 24 *hr orbit.*



| | $\|\Delta V_1\|$ | $\|\Delta V_2\|$ | Total (fps) |
|---|---|---|---|
| Hohmann | 546.99 | 39682.76 | 40229.76 |
| Swingby | 10392.39 | 4826.32 | 15218.71 |

**Figure 7.5.** *Optimal swingby transfer to Molniya orbit.*

transfer from the ascending node of the park orbit and as such will be referred to as solution "B." Although the swingby solution "B" is slightly less efficient than solution "A" it still is 3013.20 fps less than the Hohmann transfer. It is also worth noting the very small difference in the Hohmann solutions between "A" and "B," which can be attributed to the small (yet different) lunar perturbations.

### Retrograde Molniya

The final example as illustrated in Figure 7.5 is a retrograde Molniya mission orbit. Specifically the orbit is elliptical with an (osculating) apogee altitude of 21450 nm, a perigee altitude of 350 nm, an argument of perigee of 270 deg, and a retrograde inclination of 116.6 deg. This particular orbit has a period of approximately 12 hr and requires an orbital plane change of 88.1 deg. For this example the lunar swingby saves 25011.05 fps of total $\Delta v$. In this extreme case the Hohmann transfer is approximately 264% more expensive.

## 7.1.3 Equations of Motion

The dynamic behavior of a spacecraft can be modeled using a simplified version of the N-body problem as described in Battin [6]. It is convenient to use an earth centered Cartesian coordinate system, with boldface notation used to distinguish a vector from a scalar. Let $\mathbf{r}$ and $\mathbf{v}$ denote the position and velocity of the spacecraft, and let $\mathbf{r}_L$ and $\mathbf{v}_L$ the position and velocity of the moon.[8] The motion is described by

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{7.2}$$

$$\dot{\mathbf{v}} = -\frac{\mu_e}{r^3}\mathbf{r} + \mathbf{g}_L, \tag{7.3}$$

[8]Subscript convention: $x_p$, park; $x_m$, mission; $x_L$, lunar; $x_\iota$, inbound; $x_o$, outbound; $x_s$, swingby.

$$\dot{\mathbf{r}}_L = \mathbf{v}_L, \tag{7.4}$$

$$\dot{\mathbf{v}}_L = -\frac{\mu_\circ}{r_L^3}\mathbf{r}_L, \tag{7.5}$$

where $\|\mathbf{r}\| = r$, $\|\mathbf{r}_L\| = r_L$, and $\mu_\circ = \mu_e + \mu_L = Gm_e + Gm_L$, where $G$ is the universal gravitation constant with $m_e$ and $m_L$ denoting the masses of the earth and moon, respectively. The disturbing acceleration caused by the gravitation of the moon is

$$\mathbf{g}_L = -\mu_L\left[\frac{1}{d^3}\mathbf{d} + \frac{1}{r_L^3}\mathbf{r}_L\right], \tag{7.6}$$

where

$$\mathbf{d} = \mathbf{r} - \mathbf{r}_L \tag{7.7}$$

is a vector from the moon to the vehicle with magnitude $\|\mathbf{d}\| = d$. To avoid losing precision when evaluating (7.6), Battin [6] suggests defining the function

$$F(q) = q\left[\frac{3+3q+q^2}{1+(\sqrt{1+q})^3}\right], \tag{7.8}$$

where

$$q = \frac{\mathbf{r}^{\mathsf{T}}(\mathbf{r} - 2\mathbf{r}_L)}{\mathbf{r}_L^{\mathsf{T}}\mathbf{r}_L}. \tag{7.9}$$

The perturbing acceleration (7.6) is then given by

$$\mathbf{g}_L = -\frac{\mu_L}{d^3}\left[\mathbf{r} + F(q)\mathbf{r}_L\right]. \tag{7.10}$$

## 7.1.4 Kepler Orbit Propagation

As written the differential equations (7.4)–(7.5) constitute a *two-body* approximation to the lunar motion. It is well known that this system has an analytic solution, and for convenience we briefly summarize the approach (cf. [6], [78]). In general, the technique is applicable to any two-body system using the appropriate definitions for the gravitational constant $\mu_\circ$ and coordinates $(\mathbf{r}, \mathbf{v})$, and consequently we omit the subscript for generality. Given a Cartesian state vector $\mathbf{r}_\circ, \mathbf{v}_\circ$ at time $t_\circ$ called the *reference epoch* and a specified change in eccentric anomaly $\Delta E = E - E_\circ$, a new state vector $(\mathbf{r}, \mathbf{v})$, with corresponding time change $\Delta t = t - t_\circ$, can be computed as follows:

$$r_\circ = \|\mathbf{r}_\circ\|, \tag{7.11}$$
$$\sigma_\circ = \frac{1}{\sqrt{\mu_\circ}}\mathbf{r}_\circ^{\mathsf{T}}\mathbf{v}_\circ, \tag{7.12}$$
$$v_\circ^2 = \mathbf{v}_\circ^{\mathsf{T}}\mathbf{v}_\circ, \tag{7.13}$$
$$\frac{1}{a} = \frac{2}{r_\circ} - \left[\frac{v_\circ^2}{\mu_\circ}\right], \tag{7.14}$$
$$\rho = 1 - \frac{r_\circ}{a}, \tag{7.15}$$
$$C = a(1-\cos\Delta E), \tag{7.16}$$
$$S = \sqrt{a}\sin\Delta E, \tag{7.17}$$
$$F = 1 - \frac{C}{r_\circ}, \tag{7.18}$$
$$G = \frac{1}{\sqrt{\mu_\circ}}(r_\circ S + \sigma_\circ C), \tag{7.19}$$
$$r = r_\circ + \rho C + \sigma_\circ S, \tag{7.20}$$
$$F_t = -\frac{\sqrt{\mu_\circ}}{r r_\circ} S, \tag{7.21}$$
$$G_t = 1 - \frac{C}{r}, \tag{7.22}$$
$$\mathbf{r} = F\mathbf{r}_\circ + G\mathbf{v}_\circ, \tag{7.23}$$
$$\mathbf{v} = F_t\mathbf{r}_\circ + G_t\mathbf{v}_\circ, \tag{7.24}$$
$$\Delta t = \sqrt{\frac{a^3}{\mu_\circ}}\left[\Delta E + \frac{\sigma_\circ C}{a\sqrt{a}} - \rho\frac{S}{\sqrt{a}}\right]. \tag{7.25}$$

Notice that the sequence of calculations (7.11)–(7.25) constitutes an *explicit* definition for the states, as well as the corresponding time increment, which can be expressed as

$$\mathbf{r} = \mathbf{h}_r(\Delta E), \tag{7.26}$$
$$\mathbf{v} = \mathbf{h}_v(\Delta E), \tag{7.27}$$
$$\Delta t = h_t(\Delta E). \tag{7.28}$$

The propagation is explicit with respect to the independent variable $\Delta E$ but is *implicit* with respect to the time $t$. To be more precise let us rewrite (7.28) as

$$t_E = t_\circ + h_t(\Delta E) \tag{7.29}$$

since $\Delta t = t_E - t_\circ$, where $t_E$ is the final time corresponding to the eccentric anomaly increment $\Delta E$.

## 7.1.5 Differential-Algebraic Formulation of Three-Body Dynamics

The original system of 12 differential equations (7.2)–(7.5) can be recast as a differential-algebraic (DAE) system involving the six states $\mathbf{r}(t)$ and $\mathbf{v}(t)$ and the single algebraic variable $\Delta E(t)$:

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{7.30}$$

$$\dot{\mathbf{v}} = -\frac{\mu_e}{r^3}\mathbf{r} + \mathbf{g}_L, \tag{7.31}$$

$$0 = t - t_\circ - h_t(\Delta E). \tag{7.32}$$

Observe that the implicit algebraic equation (7.32) is a modified form of Kepler's transcendental equation. The gravitational perturbation on the spacecraft $\mathbf{g}_L$ is defined by the position vectors $\mathbf{r}$ and $\mathbf{r}_L$ using (7.8)–(7.10). The lunar position vector is completely determined by (7.26) using the Kepler propagation procedure outlined in (7.11)–(7.25). Thus the complete functional dependency is given by

$$\mathbf{g}_L = \mathbf{g}_L(\mathbf{r}, \Delta E).$$

## 7.1.6 Boundary Conditions

Denote the state variables at a particular time (e.g., the beginning or end of the trajectory) by $\widetilde{\mathbf{r}}$ and $\widetilde{\mathbf{v}}$. Boundary conditions defining both the park and mission orbits are typically stated in terms of *osculating* orbit elements, i.e., nonlinear functions of $\widetilde{\mathbf{r}}$ and $\widetilde{\mathbf{v}}$. There are many equivalent ways to specify both the park and the mission orbits. A circular orbit with specified radius $\bar{r}$ can be achieved by imposing the following boundary constraints:

$$\bar{r} = \|\widetilde{\mathbf{r}}\|, \tag{7.33a}$$

$$\sqrt{\frac{\mu_e}{\bar{r}}} = \|\widetilde{\mathbf{v}}\|, \tag{7.33b}$$

$$0 = \frac{\widetilde{\mathbf{r}}^{\mathsf{T}}\widetilde{\mathbf{v}}}{\sqrt{\mu_e \bar{r}}}. \tag{7.33c}$$

Equation (7.33a) fixes the magnitude of the position vector, (7.33b) defines the circular velocity, and (7.33c) ensures the flight path angle (and eccentricity) is zero. Observe that when (7.33a) and (7.33b) are satisfied (7.33c) is just

$$\frac{\widetilde{\mathbf{r}}^{\mathsf{T}}\widetilde{\mathbf{v}}}{\sqrt{\mu_e \bar{r}}} = \frac{\widetilde{\mathbf{r}}^{\mathsf{T}}\widetilde{\mathbf{v}}}{\|\widetilde{\mathbf{r}}\|\,\|\widetilde{\mathbf{v}}\|},$$

which is equivalent to making the normalized position and velocity vectors orthogonal. However, as written the denominator in (7.33c) is a *constant*, which is the preferable, i.e., "more linear," way to pose the constraint.

To achieve a particular inclination $\bar{i}$ we must enforce the boundary condition

$$\cos\bar{i} = \widehat{\mathbf{h}}_3, \tag{7.34a}$$

where the angular momentum vector $\mathbf{h}$ and north vector $\mathbf{i}_z$ are

$$\mathbf{h} = \widetilde{\mathbf{r}} \times \widetilde{\mathbf{v}}, \tag{7.34b}$$

$$\widehat{\mathbf{h}} = \frac{\mathbf{h}}{\|\mathbf{h}\|}, \tag{7.34c}$$

$$\mathbf{i}_z = (0,0,1)^{\mathsf{T}}, \tag{7.34d}$$

$$\widehat{\mathbf{h}}_3 = \mathbf{i}_z^{\mathsf{T}}\widehat{\mathbf{h}}. \tag{7.34e}$$

Observe that (7.34a) is stated in terms of $\cos \bar{\imath}$ rather than the inclination $\bar{\imath}$ itself, thereby avoiding ambiguities when computing the inverse cosine.

If the desired orbit is elliptic, (7.33a)–(7.33c) are not applicable. Instead to achieve a specified semimajor axis $\bar{a}$, and eccentricity $\bar{e}$ ($> 0$), we must impose the boundary conditions

$$\bar{a} = \left( \frac{2}{\|\widetilde{\mathbf{r}}\|} - \frac{\|\widetilde{\mathbf{v}}\|^2}{\mu_e} \right)^{-1}, \tag{7.35a}$$

$$\bar{e} = \|\mathbf{e}\|, \tag{7.35b}$$

where the eccentricity vector is given by

$$\mathbf{e} = \frac{1}{\mu_e}(\widetilde{\mathbf{v}} \times \mathbf{h}) - \frac{\widetilde{\mathbf{r}}}{\|\widetilde{\mathbf{r}}\|}, \tag{7.35c}$$

and the angular momentum is given by (7.34b).

For elliptic orbits ($\|\mathbf{e}\| > 0$), the eccentricity vector is directed toward periapsis, thereby defining the orientation of the principal axis. In general, the argument of periapsis $\omega$ is an angle measured in the orbital plane from the ascending node to periapsis. The particular case $\bar{\omega} = 270°$ can be enforced by imposing the conditions

$$0 = \widehat{\mathbf{e}}^{\mathsf{T}}\mathbf{i}_n, \tag{7.36a}$$

$$0 > \widehat{\mathbf{e}}_3, \tag{7.36b}$$

where

$$\widehat{\mathbf{e}} = \frac{\mathbf{e}}{\bar{e}}, \tag{7.36c}$$

$$\widehat{\mathbf{e}}_3 = \mathbf{i}_z^{\mathsf{T}}\widehat{\mathbf{e}}, \tag{7.36d}$$

$$\mathbf{i}_n = \mathbf{i}_z \times \widehat{\mathbf{h}}, \tag{7.36e}$$

and $\mathbf{e}$ is given by (7.35c), with $\widehat{\mathbf{h}}$ defined by (7.34c), and $\mathbf{i}_z$ from (7.34d). Again notice that the denominator of (7.36c) involves the *constant* $\bar{e}$ instead of the quantity $\|\mathbf{e}\|$ appearing on the right-hand side of (7.35b).

## 7.1.7 A Four-Step Solution Technique

To compute a solution to an *optimal lunar swingby* let us first pose a sequence of "easier" subproblems. It is important to remark that the proposed technique is a heuristic—other equally valid methods can be postulated. However, the heuristic has proven to be both reliable and efficient. Section 7.1.8 will describe *how* each subproblem is solved.

The *four-step solution technique* can be summarized as follows:

**Step 1: Three-Impulse, Conic Solution**
Solve a small NLP with analytic propagation ignoring lunar gravity.

**Step 2: Three-Body Approximation to Conic Solution**
Solve an "inverse problem" to fit three-body dynamics to the conic solution.

**Step 3: Optimal Three-Body Solution with Fixed Swingby Time**
Use the solution from Step 2 to initialize.

**Step 4: Optimal Three-Body Solution**
Compute solution with free swingby time, using Step 3 as an initial guess.

### Step 1: Three-Impulse, Conic Solution

Since the optimal trajectory depends on the relative geometry of three bodies, namely the earth, moon, and spacecraft, it is critical that the solution process be initiated with a reasonable geometric configuration. This goal can be achieved by using a very simplified model of the dynamics. In particular for Step 1, it is assumed that the spacecraft dynamics are determined entirely by the gravitational attraction of the primary body, earth. Further, it is assumed that the lunar swingby can be approximated by a simple velocity increment. Because all of the dynamics are modeled using a two-body approximation, the analytic trajectory propagation approach outlined in Section 7.1.4 can be exploited. With these simplifying assumptions one can pose the following very simple NLP problem.

*Optimization Variables.* The problem can be formulated using 24 variables to define the trajectory as illustrated in Figure 7.6:

$$(\mathbf{r}_o, \mathbf{v}_o, \Delta\mathbf{v}_1, \Delta E_o) \quad \text{State at Park Orbit Departure,} \tag{7.37}$$

$$(\mathbf{r}_i, \mathbf{v}_i, \Delta\mathbf{v}_2, \Delta E_i) \quad \text{State at Mission Orbit Arrival,} \tag{7.38}$$

$$(\Delta\mathbf{v}_s, \Delta E_L) \quad \text{Swingby Velocity Increment and Lunar Angle to Intercept.} \tag{7.39}$$

The outbound (earth to moon) transfer orbit is defined by the variables in (7.37), namely the Cartesian state at the beginning of the outbound transfer $\mathbf{r}_o, \mathbf{v}_o$, the impulsive velocity $\Delta\mathbf{v}_1$, and the eccentric anomaly change for the outbound trajectory $\Delta E_o$. Equation (7.38) defines the corresponding variables for the inbound (moon to earth) transfer. The velocity increment provided by the moon at swingby is defined as $\Delta\mathbf{v}_s$, and the location of the moon relative to a reference epoch is defined by the angle $\Delta E_L$.

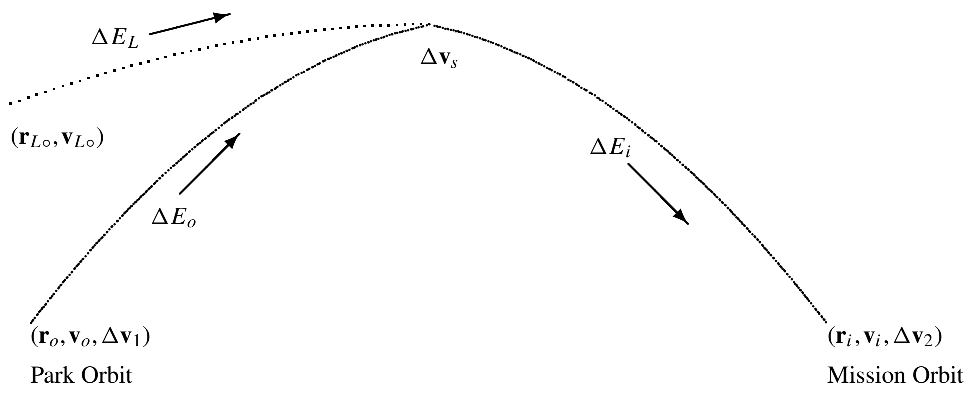

$\Delta E_L$
$\Delta \mathbf{v}_s$
$(\mathbf{r}_{L\circ}, \mathbf{v}_{L\circ})$
$\Delta E_i$
$\Delta E_o$
$(\mathbf{r}_o, \mathbf{v}_o, \Delta \mathbf{v}_1)$
Park Orbit
$(\mathbf{r}_i, \mathbf{v}_i, \Delta \mathbf{v}_2)$
Mission Orbit

**Figure 7.6.** *Three-impulse, conic solution.*

*Park Orbit Conditions.* In order to enforce the boundary conditions at the park orbit, the following NLP constraints must be imposed:

$$\mathbf{r}_p = \mathbf{r}_o \qquad \text{Position Continuity,} \tag{7.40}$$

$$\mathbf{v}_p = \mathbf{v}_o - \Delta \mathbf{v}_1 \qquad \text{Impulsive Velocity Change,} \tag{7.41}$$

$$\boldsymbol{\phi}_p \left( \mathbf{r}_p, \mathbf{v}_p \right) = \mathbf{0} \qquad \text{Park Orbit Constraints.} \tag{7.42}$$

Equation (7.40) ensures that the park orbit and outbound transfer orbit position is continuous. The impulsive velocity change at departure is enforced by (7.41). The park orbit state vector must also satisfy the nonlinear constraints denoted $\boldsymbol{\phi}_p$ in (7.42). For the circular park orbit illustrated here, the vector $\boldsymbol{\phi}_p$ is given by (7.33a), (7.33b), (7.33c), and (7.34a).

*Mission Orbit Conditions.* The boundary conditions imposed by the mission orbit lead to a set of NLP constraints similar to those enforced at departure, namely,

$$\mathbf{r}_m = \mathbf{r}_i \qquad \text{Position Continuity,} \tag{7.43}$$

$$\mathbf{v}_m = \mathbf{v}_i + \Delta \mathbf{v}_2 \qquad \text{Impulsive Velocity Change,} \tag{7.44}$$

$$\boldsymbol{\phi}_m \left( \mathbf{r}_m, \mathbf{v}_m \right) = \mathbf{0} \qquad \text{Mission Orbit Constraints.} \tag{7.45}$$

Constraints (7.43) and (7.44) are analogous to (7.40) and (7.41). The mission orbit constraints denoted by the vector $\boldsymbol{\phi}_m$ in (7.45) are computed using the appropriate expressions from Section 7.1.6.

*Lunar Conditions.* The conditions at the moon used to approximate the lunar swingby all involve expressions for the state vector computed using the Kepler propagation described by (7.26) and (7.27):

$$\mathbf{h}_r(\mathbf{r}_o, \mathbf{v}_o, \Delta E_o) = \mathbf{h}_r(\mathbf{r}_i, \mathbf{v}_i, \Delta E_i) \qquad \text{Outbound/Inbound Position,} \tag{7.46}$$

$$\mathbf{h}_r(\mathbf{r}_o, \mathbf{v}_o, \Delta E_o) = \mathbf{h}_r(\mathbf{r}_{L\circ}, \mathbf{v}_{L\circ}, \Delta E_L) \qquad \text{Outbound/Lunar Position,} \tag{7.47}$$

$$\begin{aligned} \mathbf{h}_v(\mathbf{r}_i, \mathbf{v}_i, \Delta E_i) = {} & \mathbf{h}_v(\mathbf{r}_o, \mathbf{v}_o, \Delta E_o) \\ & + \mathbf{h}_v(\mathbf{r}_{L\circ}, \mathbf{v}_{L\circ}, \Delta E_L) + \Delta \mathbf{v}_s \qquad \text{Velocity Change.} \end{aligned} \tag{7.48}$$

The constraints (7.46) force the outbound and inbound transfer trajectories to have the same position at the swingby. Observe that the outbound position $\mathbf{h}_r$ is completely determined by the departure state $\mathbf{r}_o, \mathbf{v}_o$, and the outbound eccentric anomaly change $\Delta E_o$, with similar comments applicable to the inbound leg. While (7.46) ensures continuity between the outbound and inbound trajectories, it is also necessary that the swingby occur at the moon's position. This is achieved using constraint (7.47). Of course this constraint will locate the swingby at the *center* of the moon, which can be achieved only by ignoring the lunar gravity. Finally, one must model the velocity change at the moon, and this is expressed by (7.48). Observe that (7.48) involves the Kepler velocities $\mathbf{h}_v$ and an impulsive change $\Delta \mathbf{v}_s$. This constraint requires the velocity *after* the swingby to equal the vector sum of (a) the velocity *before* the swingby, plus (b) the velocity of the moon, plus (c) the impulsive change.

*Objective.* The objective function for this simplified model problem is to *minimize* the total $\Delta v$, i.e.,

$$F = \|\Delta \mathbf{v}_1\| + \|\Delta \mathbf{v}_2\| + \|\Delta \mathbf{v}_s\|. \tag{7.49}$$

The solution to this problem should define a geometric configuration of the earth-moon system that is optimal when ignoring the lunar gravitational effects.

## Step 2: Three-Body Approximation

The solution to the simplified model problem computed in Step 1 is designed to construct an approximate solution to the overall problem. Unfortunately, by design, the simplified model ignores one of the primary dynamic aspects of the real problem. Specifically, the conic solution solves

$$\ddot{\mathbf{r}} = -\frac{\mu_e}{r^3}\mathbf{r}, \tag{7.50a}$$

$$\ddot{\mathbf{r}}_L = -\frac{\mu_\circ}{r_L^3}\mathbf{r}_L \tag{7.50b}$$

but *not* the three-body dynamics

$$\ddot{\mathbf{r}} = -\frac{\mu_e}{r^3}\mathbf{r} + \mathbf{g}_L, \tag{7.51a}$$

$$\ddot{\mathbf{r}}_L = -\frac{\mu_\circ}{r_L^3}\mathbf{r}_L. \tag{7.51b}$$

Let us denote the conic solution from (7.50a) by $\widetilde{\mathbf{r}}(t)$. If one assumes that the conic trajectory is approximately correct, then it is reasonable to find a "nearby" trajectory that does satisfy the correct dynamics given by (7.51a). This goal can be achieved by "fitting" the three-body trajectory to the conic, i.e., by minimizing

$$F = \sum_{k=1}^{N} \|\mathbf{r}_k - \widetilde{\mathbf{r}}_k\|^2 \tag{7.52}$$

subject to

$$\ddot{\mathbf{r}} = -\frac{\mu_e}{r^3}\mathbf{r} + \mathbf{g}_L, \tag{7.53}$$

$$\ddot{\mathbf{r}}_L = -\frac{\mu_\circ}{r_L^3}\mathbf{r}_L, \tag{7.54}$$

$$r_{Lmin} \le \|\mathbf{r} - \mathbf{r}_L\|, \tag{7.55}$$

where $\widetilde{\mathbf{r}}_k = \widetilde{\mathbf{r}}(t_k)$ is the spacecraft position on the conic trajectory and $\mathbf{r}_k = \mathbf{r}(t_k)$ is the state evaluated at the same time points $t_k$ on the three-body trajectory. Observe that in this *inverse problem* an algebraic inequality constraint (7.55) is included to ensure the trajectory is sufficiently above the lunar surface where $r_{Lmin}$ is a lower bound on the distance from the spacecraft to the moon. The conic solution serves as a good initial guess for this inverse problem, so it is reasonable to set $\mathbf{r}_k^{(0)} = \widetilde{\mathbf{r}}_k$, provided the $N$ data points *exclude* the single time point at the moon. Clearly the lunar acceleration $\mathbf{g}_L$ cannot be evaluated when $\|\mathbf{d}\| = \|\mathbf{r} - \mathbf{r}_L\| = \mathbf{0}$ in (7.6).

## Step 3: Fixed Swingby Time

The solution obtained from Step 2 provides an excellent initial guess for Step 3. In particular it supplies a time history for $\mathbf{r}(t), \mathbf{v}(t)$ that satisfies the full three-body dynamic equations (7.2)–(7.5). Furthermore, by construction the boundary conditions at the park and mission orbits will be approximately satisfied. Thus one can pose a problem with two distinct phases described by either the three-body dynamic equations (7.2)–(7.5) or the differential-algebraic system (7.30)–(7.32). Figure 7.7 illustrates the dynamic simulation.

*Phase* 1: *Outbound Transfer.* The outbound transfer begins at the free initial time $t_I$, which must satisfy the following park orbit conditions analogous to those used for Step 1:

$$\mathbf{r}_p = \mathbf{r}_o \quad \text{Position Continuity,} \tag{7.56}$$

$$\mathbf{v}_p = \mathbf{v}_o - \Delta\mathbf{v}_1 \quad \text{Impulsive Velocity Change,} \tag{7.57}$$

$$\boldsymbol{\phi}_p\left(\mathbf{r}_p, \mathbf{v}_p\right) = \mathbf{0} \quad \text{Park Orbit Constraints.} \tag{7.58}$$

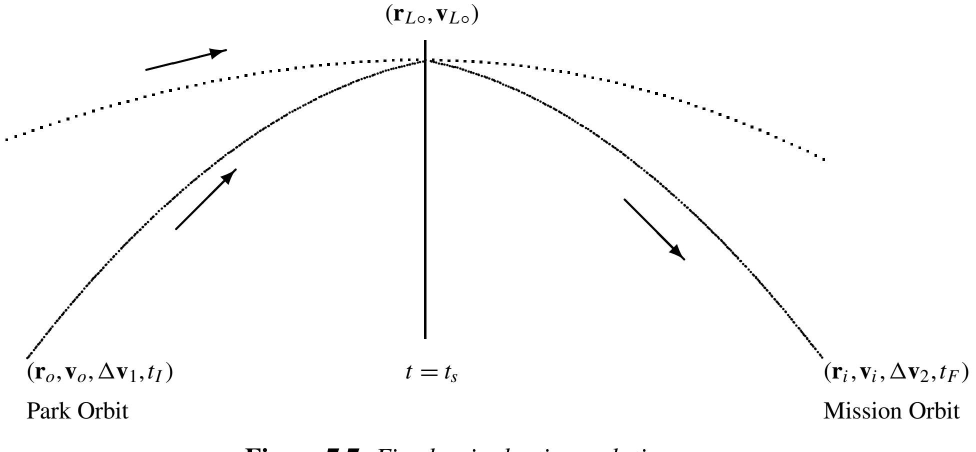

$(\mathbf{r}_{L\circ}, \mathbf{v}_{L\circ})$
$(\mathbf{r}_o, \mathbf{v}_o, \Delta\mathbf{v}_1, t_I)$
Park Orbit
$t = t_s$
$(\mathbf{r}_i, \mathbf{v}_i, \Delta\mathbf{v}_2, t_F)$
Mission Orbit

**Figure 7.7.** *Fixed swingby time solution.*

The phase terminates at the fixed time $t_s$, which must satisfy the lunar conditions

$$\|\mathbf{r}-\mathbf{r}_L\| \geq r_{Lmin} \quad \text{Closest Approach,} \tag{7.59}$$

$$(\mathbf{v}-\mathbf{v}_L)^\mathsf{T}(\mathbf{r}-\mathbf{r}_L) = 0 \quad \text{Lunar Flight Path Angle.} \tag{7.60}$$

Observe that by forcing the lunar flight path angle to be zero (7.60) ensures that the closest approach to the moon will occur at $t_s$.

*Phase* 2: *Inbound Transfer.* The inbound transfer begins at the fixed time $t_s$ and must satisfy the conditions

$$(\mathbf{r},\mathbf{v},\mathbf{r}_L,\mathbf{v}_L)^{(-)} = (\mathbf{r},\mathbf{v},\mathbf{r}_L,\mathbf{v}_L)^{(+)} \quad \text{State Continuity,} \tag{7.61}$$

$$(\mathbf{r}_L,\mathbf{v}_L) = (\bar{\mathbf{r}}_L,\bar{\mathbf{v}}_L) \quad \text{Lunar State.} \tag{7.62}$$

Equation (7.61) ensures continuity in all of the states across the phase boundary, and since the time $t_s$ is fixed (7.62) enforces consistency with the lunar reference epoch $(\bar{\mathbf{r}}_L,\bar{\mathbf{v}}_L)$.

Phase 2 terminates at the free time $t_F$ and at that point must satisfy the mission orbit conditions

$$\mathbf{r}_m = \mathbf{r}_i \quad \text{Position Continuity,} \tag{7.63}$$

$$\mathbf{v}_m = \mathbf{v}_i + \Delta\mathbf{v}_2 \quad \text{Impulsive Velocity Change,} \tag{7.64}$$

$$\boldsymbol{\phi}_m(\mathbf{r}_m,\mathbf{v}_m) = \mathbf{0} \quad \text{Mission Orbit Constraints,} \tag{7.65}$$

$$t_{max} \geq t_F - t_I \quad \text{Mission Duration.} \tag{7.66}$$

The objective for this dynamic optimization problem is to minimize

$$F = \|\Delta\mathbf{v}_1\| + \|\Delta\mathbf{v}_2\|. \tag{7.67}$$

### Step 4: Optimal Three-Body Solution

The completion of Step 3 will provide an excellent initial guess for the full optimal three-body lunar swingby. Indeed, only two modifications to the formulation in Step 3 are required. First, of course, the time of closest approach to the moon $t_s$ must be free. Second, one must ensure that the lunar state at the (free) initial time is consistent with the reference epoch for the moon. Thus at the beginning of Phase 1, the following additional boundary conditions must be satisfied:

$$\mathbf{r}_L = \mathbf{h}_r(\mathbf{r}_\circ,\mathbf{v}_\circ,\Delta E_\circ), \tag{7.68}$$

$$\mathbf{v}_L = \mathbf{h}_v(\mathbf{r}_\circ,\mathbf{v}_\circ,\Delta E_\circ). \tag{7.69}$$

Typically any convenient reference epoch for the moon can be chosen.

## 7.1.8 Solving the Subproblems

The preceding sections posed a set of subproblems that can be solved to obtain an optimal lunar swingby trajectory. But how does one efficiently solve the subproblems? In particular it is necessary to solve

- *an optimal control problem* in Steps 3 and 4, using the SNLP method presented in Chapter 4,
- *a parameter estimation (inverse) problem* in Step 2, using the algorithm described in Chapter 5, and
- *an NLP problem* in Step 1, using the methods in Chapter 1.

There are a number of efficiency issues that must be addressed within the context of an SNLP algorithm. The second step of the algorithm requires the solution of an NLP; however, among the many possible NLP algorithms it is desirable to choose the most efficient. The computational experience as discussed in Section 4.13 provides some insight to guide the choice. To reinforce the observations of Section 4.13, let us compare the performance of two different NLP algorithms when solving the small, dense NLP subproblems required by Step 1 (Section 7.1.7). Table 7.1 summarizes the performance of two different NLP algorithms, when used to solve three different NLP problems. In particular results are presented for the sequential quadratic programming (SQP) algorithm described in Chapter 1. Two different techniques were used to compute the Hessian matrix, namely a finite difference approximation denoted as "Newton" and a quasi-Newton BFGS (Broyden–Fletcher–Goldfarb–Shanno) recursive update. The second algorithm is the primal-dual interior-point or barrier algorithm described in Chapter 2. This algorithm was also tested using Newton and BFGS Hessian approximations.

**Table 7.1.** *NLP algorithm performance comparison.*

**Step 1–Small, Dense NLP Subproblems**

| Mission | Equatorial | Polar | Molniya |
|---|---|---|---|
| SQP-Newton | (10,4) | (16,10) | (135,44) |
| SQP-BFGS | (24,19) | (36,31) | (186,96) |
| Barrier-Newton | (24,22) | (57,55) | (70,68)† |
| Barrier-BFGS | (242,241)† | (58,56) | (286,284) |

Key: (Gradient Eval., Hessian Eval.) † No Solution

For each algorithm the table presents the number of gradient and Hessian evaluations needed to reach a solution. Although the number of problems in this test set is quite small, the basic findings are consistent with much more extensive testing as described in [33]. Generally the SQP algorithm was both more efficient and more robust than the barrier method. Although it is difficult to prove, one speculates that an SQP method is simply a better choice for solving very nonlinear optimization problems as discussed in Section 4.13. Conversely, our experience suggests a barrier algorithm may be preferable for problems with linear constraints, especially when there are many inequalities. The testing also suggests that a quasi-Newton Hessian approximation requires significantly more iterations to converge. While this is not surprising, it is extremely important when considering the very large, sparse NLP problems that arise when using discretization methods for optimal control.

A quasi-Newton Hessian approximation suffers from another deficiency first mentioned in Section 4.13 which is not demonstrated by this comparison. The results given utilize a quasi-Newton approximation to the *full* Hessian. Unfortunately this can become prohibitively expensive for large-scale optimization problems, especially when there are many degrees of freedom.

A second, more serious performance issue occurs when comparing NLP algorithms for use within the context of an SNLP. Specifically a *barrier algorithm cannot exploit a good guess* as discussed in Section 4.13. To illustrate this point consider the solution of the optimal control subproblem for the polar mission summarized in Tables 7.2 and 7.3. Both cases were initiated using the same information. Specifically the initial trajectory was constructed as the solution from a Step 2 inverse problem. Using this three-body solution trajectory a variable stepsize numerical integration algorithm was used to construct the initial grid points. Referring to Table 7.2, the first grid had 594 points, leading to an NLP problem with 7136 optimization variables and 7715 nonlinear constraints. The solution to this coarse grid problem requires 18 gradient evaluations (NGC), 10 Hessian evaluations (NHC), and 3794 function evaluations (NFE), including those needed for finite difference derivatives. The resulting solution had a relative discretization error of $\epsilon = 1 \times 10^{-4}$ and was computed in 30.1 sec. The grid was refined two times, with the final grid containing 1113 points. Table 7.3 presents exactly the same history, when a barrier algorithm is used to solve the NLP subproblems. For the SQP algorithm as the mesh is refined, the number of Hessian evaluations and iterations decreases—only 1 Hessian is needed on the second and third grids. Each coarse grid solution provides a very good guess, and the Newton method is within its region of quadratic convergence. For the barrier method this is not true! The second mesh required an additional 49 Hessian evaluations because the initial guess was perturbed. The overall penalty in computation time is catastrophic in this example. In addition, because the initial guesses were perturbed the barrier algorithm converged to a different local solution than did the SQP algorithm. All of the computational results were obtained using a Dell M60 laptop computer, with a Linux operating system.

**Table 7.2.** *Mesh refinement with an SQP algorithm.*

**SQP**

| $k$ | $M$ | $n$ | $m$ | NGC | NHC | NFE | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | 594 | 7136 | 7715 | 18 | 10 | 3794 | $1\times10^{-4}$ | 30.1 |
| 2 | 881 | 10580 | 11446 | 4 | 1 | 454 | $4\times10^{-7}$ | 7.8 |
| 3 | 1113 | 13364 | 14462 | 4 | 1 | 454 | $1\times10^{-8}$ | 10.6 |
| Total | | | | 26 | 12 | 4702 | | 48.6 |

**Table 7.3.** *Mesh refinement with a barrier algorithm.*

**Barrier†**

| $k$ | $M$ | $n$ | $m$ | NGC | NHC | NFE | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | 594 | 7720 | 7715 | 328 | 319 | 112475 | $1\times10^{-5}$ | 749.3 |
| 2 | 881 | 12985 | 12980 | 57 | 49 | 17637 | $2\times10^{-7}$ | 233.7 |
| 3 | 1113 | 14090 | 14085 | 6 | 2 | 881 | $1\times10^{-8}$ | 18.5 |
| Total | | | | 391 | 370 | 130993 | | 1001.6 |

† Different Local Solution than SQP

## Is Mesh Refinement Needed?

In light of the apparent conflict between mesh refinement and a barrier algorithm it is important to review why mesh refinement is needed. Let us consider the solution of the

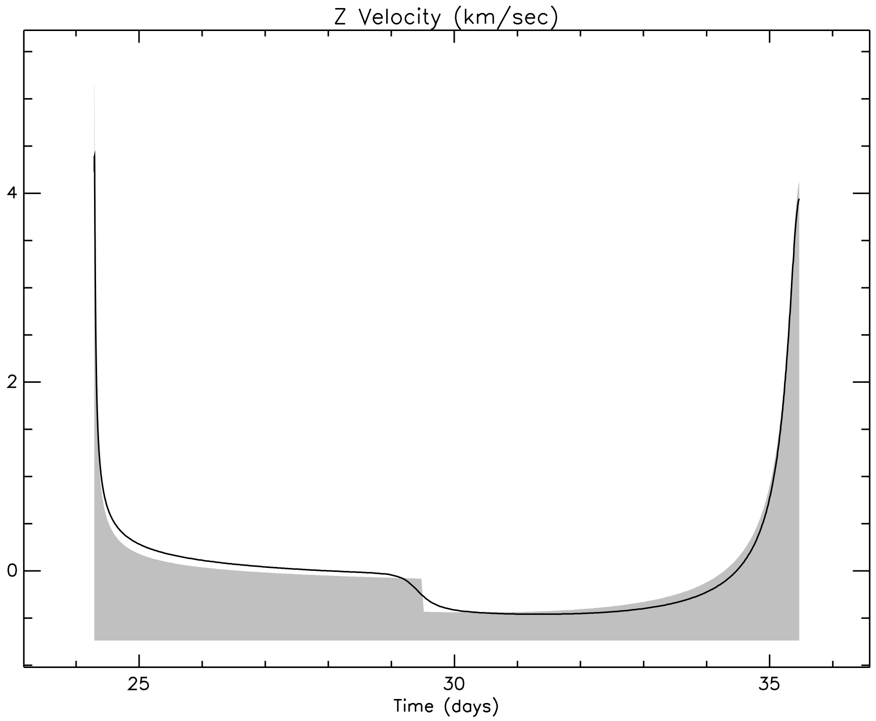


**Figure 7.8.** *Velocity discontinuity.*

Step 2 inverse problem solution for the polar mission. To review, the conic trajectory available from the solution of Step 1 can be used to construct an initial guess. In particular by sampling the conic at 600 equal $\Delta E$ increments, one obtains an NLS problem with 1800 residuals. To avoid a singularity, it is necessary to omit the point at the moon. The shaded region in Figure 7.8 illustrates the discontinuous behavior in one component of the velocity from the conic trajectory, and the solid line shows the smoothed approximation that results after "fitting" a three-body solution to the conic. Table 7.4 summarizes the behavior of the $\mathbb{SOCS}$ mesh-refinement procedure for this example, and Figure 7.9 illustrates what the procedure does to both the discretization error and the mesh distribution. Initially the discretization error is very large in the vicinity of the moon. Clearly, this error can be attributed to the approximate nature of the conic trajectory—i.e., the position goes through the center of the moon and the velocity change is impulsive. During the first few refinement iterations, grid points are added in the vicinity of the discontinuity, and this leads to a significant reduction in the discretization error as measured by the value of $\epsilon$ in Table 7.4. In fact after the first refinement iteration only six grid points were added, all in the neighborhood of the discontinuity, and this reduced the discretization error by nearly four orders of magnitude. It is also worth noting that solving the NLP subproblem after adding these grid points was significantly more expensive (taking 17 Hessian evaluations), because the entire solution had to be adjusted to account for this effect. Clearly, mesh refinement is needed in order to address the singularities in the vicinity of the moon.

**Table 7.4.** *Mesh refinement.*

| $k$ | $M$ | $n$ | $m$ | NGC | NHC | NFE | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | 599 | 7188 | 7775 | 12 | 2 | 144 | $6\times10^{-1}$ | 3.5 |
| 2 | 606 | 7272 | 7866 | 21 | 17 | 525 | $9\times10^{-5}$ | 1.1 |
| 3 | 606 | 7272 | 7866 | 4 | 2 | 724 | $1\times10^{-6}$ | 7.6 |
| 4 | 742 | 8904 | 9634 | 4 | 1 | 448 | $1\times10^{-8}$ | 5.6 |
| Total | | | | 41 | 22 | 1841 | | 27.7 |

| | | | | | |
|---|---|---|---|---|---|
| $k$ | Refinement No | $M$ | Grid Pts | $n$ | NLP vars |
| $m$ | NLP cons. | NGC | Grad Eval | NHC | Hess Eval |
| NFE | Func Eval | $\epsilon$ | Disc. Error | Time | CPU |

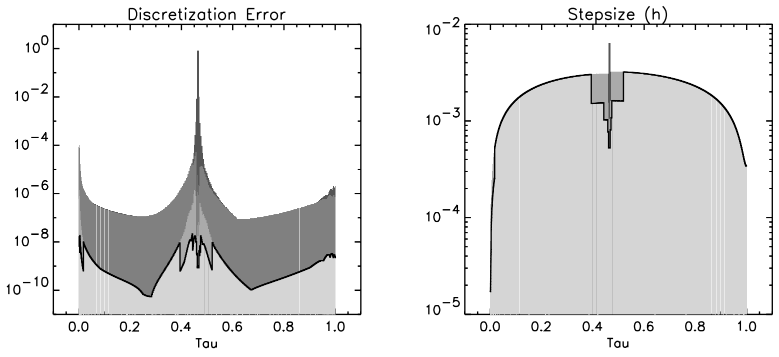


**Figure 7.9.** *Mesh refinement.*

## DAE or ODE Formulation?

The three-body dynamics of the system can be described by the ODEs summarized in Section 7.1.3. However, the differential-algebraic system presented in Section 7.1.5 can also be used when solving the subproblems in Steps 2, 3, and 4. Is one formulation preferable to the other in terms of either computational speed and/or solution accuracy? To address this question, Tables 7.5 and 7.6 compare the formulations. Specifically, the optimal control problems in Steps 3 and 4 for the Molniya mission were solved using the ODE and DAE formulations. Both were initialized with the same Step 2 solution trajectory.

The comparison reveals a number of issues. First, the DAE formulation required a final grid with 2056 points, whereas the ODE formulation achieved the same accuracy with 1190 points. There are 12 ODEs and the DAE system has only 7 equations. Since the number of grid points is related to the nonlinearity of the equations as well as the order of interpolation, this suggests the DAE system may be more nonlinear. However, the total solution time is dictated by the total number of NLP variables in the discretized subproblem. The ODE formulation requires 14291 variables for the final iteration. In contrast, the DAE formulation needs 16456 variables. Thus the size of the NLP problems is nearly the same, even though one formulation involves nearly twice as many dynamic equations. Overall, there was no clearcut difference between the ODE and DAE formulations in either speed or accuracy for the applications considered here.

**Table 7.5.** *Steps* 3 *and* 4*, optimal solution for Molniya mission.*

**ODE Formulation**

| $M$ | $n$ | NGC | NHC | Time (sec) |
|---|---|---|---|---|
| 630 | 7568 | 26 | 7 | 32.02 |
| 807 | 9692 | 7 | 2 | 12.30 |
| 940 | 11288 | 4 | 1 | 8.66 |
| 1190 | 14288 | 4 | 1 | 11.22 |
| 1190 | 14291 | 10 | 4 | 45.49 |

Total Time = 109.69 sec

**Table 7.6.** *Steps* 3 *and* 4*, optimal solution for Molniya mission.*

**DAE Formulation**

| $M$ | $n$ | NGC | NHC | Time (sec) |
|---|---|---|---|---|
| 1097 | 8782 | 30 | 14 | 45.27 |
| 1530 | 12246 | 4 | 1 | 7.98 |
| 2056 | 16454 | 4 | 1 | 11.82 |
| 2056 | 16456 | 7 | 2 | 31.02 |

Total Time = 96.09 sec

**Reference Epoch**

The reference epoch for all numerical results corresponds to a Julian date of 2453561.5 (July 10, 2005). All results utilize an ICRF (International Celestial Reference Frame). The lunar ephemeris was constructed using a least squares fit of two-body dynamics to the JPL DE405 lunar ephemeris [161] over a 60 day period beginning at the reference epoch. The resulting lunar state vector and corresponding equatorial radii and gravitational constants are given in Table 7.7.

**Table 7.7.** *Mission parameters.*

| | | |
|---|---|---|
| $\mu_e$ | 398600.436380820 | km$^3$/sec$^2$ |
| $R_e$ | 6378.14000000000 | km |
| $\mu_L$ | 4902.79881586123 | km$^3$/sec$^2$ |
| $R_L$ | 1737.40000000000 | km |
| $r_{ox}$ | $-.3398817123704749\times10^6$ | km |
| $r_{oy}$ | $.1956358580374241\times10^6$ | km |
| $r_{oz}$ | $.1139974125070158\times10^6$ | km |
| $v_{ox}$ | $-.5195857465292167\times10^0$ | km/sec |
| $v_{oy}$ | $-.7186784200912856\times10^0$ | km/sec |
| $v_{oz}$ | $-.3742849669631482\times10^0$ | km/sec |

## 7.2 Multiple-Pass Aero-Assisted Orbit Transfer

**Example 7.1** MULTIPLE-PASS AERO-ASSISTED ORBIT TRANSFER. A favorite summer pastime while at a seaside beach or lakefront is "stone skipping." As a flat stone hits the surface of the water a rapid change in direction takes place that alters the motion for the next "skip," and a "good throw" will result in many skips before the stone loses energy. An analogous situation occurs in orbit mechanics when a spacecraft reenters the atmosphere. When a spacecraft is outside of the sensible atmosphere, i.e., *exo-atmospheric*, there are no aerodynamic forces on it. However, as the vehicle gets closer to the earth, the *endo-atmospheric* motion is dominated by aerodynamic effects. This rapid change in the environment can dramatically change the path of the vehicle, much like a stone hitting the surface of a lake. Trajectories of this type present challenging optimization problems, primarily because the environmental state of the vehicle changes during the dynamic process. In fact, very similar computational issues arise when modeling chemical or thermodynamic systems that change state during the course of a process. The concept of a phase becomes critical when constructing a mathematical model of such dynamic systems.

Many authors have studied the use of atmospheric forces to achieve orbital plane change. An extensive study of the subject, as well as many pertinent references, can be found in the paper by Rao, Tang, and Hallman [149]. For the particular scenario considered here, the motion begins with the vehicle in orbit. After using a rocket to slow the spacecraft, it reenters the earth's atmosphere and then skips out again, much like a stone on a lake. After executing three additional passes through the atmosphere, the spacecraft propulsion system is used to add velocity, thereby inserting the vehicle into a different orbit. The basic goal is to change the orbit of the vehicle and minimize the fuel required to execute the transfer. To be specific let us focus on one of the cases described in [149]. In particular, the spacecraft begins in a geosynchronous orbit which is circular with zero inclination at an altitude of 19323 nm.[9] After four passes through the atmosphere, it is required that the vehicle be in a circular orbit at an altitude of 100 nm with an inclination of 89 deg. The goal is to minimize the fuel consumed to perform the transfer.

This orbit transfer can be modeled using nine distinct phases alternating between orbital (exo-atmospheric) and atmospheric (endo-atmospheric). During phases 1, 3, 5, 7, and 9 the dynamics do not include aerodynamic forces and are described in Section 7.2.1. The dynamics used in phases 2, 4, 6, and 8 are presented in Section 7.2.2.

### 7.2.1 Orbital Phases

As in [149] let us use a spherical nonrotating earth model, in which case the orbital motion can be defined by the simple two-body dynamics

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{7.70}$$

$$\dot{\mathbf{v}} = -\frac{\mu}{r^3}\mathbf{r}, \tag{7.71}$$

where $\mathbf{r}$ is the earth centered inertial (ECI) position vector and $\mathbf{v}$ is the ECI velocity vector. This system can be solved analytically as described in Section 7.1.4. Given a Cartesian state vector $\mathbf{r}_\circ, \mathbf{v}_\circ$ at time $t_\circ$ and a specified change in eccentric anomaly $\Delta E = E - E_\circ$, a

[9]1 nautical mile = 6076.1154855643 ft.

new state vector $(\mathbf{r},\mathbf{v})$ with corresponding time change $\Delta t = t - t_\circ$ can be computed; i.e., from (7.26)–(7.28) we can write

$$\mathbf{r} = \mathbf{h}_r(\mathbf{r}_\circ, \mathbf{v}_\circ, \Delta E), \tag{7.72}$$

$$\mathbf{v} = \mathbf{h}_v(\mathbf{r}_\circ, \mathbf{v}_\circ, \Delta E), \tag{7.73}$$

$$\Delta t = h_t(\mathbf{r}_\circ, \mathbf{v}_\circ, \Delta E). \tag{7.74}$$

Thus each orbital phase can be propagated analytically from the state at the beginning of the phase $(\mathbf{r}_\circ, \mathbf{v}_\circ)$ to the final state $(\mathbf{r},\mathbf{v})$. In fact, for this application it is not necessary to compute the time change $\Delta t$ (except for display), so it is convenient to simply set the initial eccentric anomaly $E_\circ = 0$ and treat $\Delta E$ as the independent variable during the orbital phases.

## 7.2.2 Atmospheric Phases

During the atmospheric phases (2, 4, 6, and 8), it is more convenient to express the dynamics using an intrinsic or flight path coordinate system as in (6.1)–(6.6). In general, atmospheric phase $k$ is defined in the domain $t_k^+ \le t \le t_{k+1}^-$. However, since the atmospheric dynamics are independent of time it is convenient to model the dynamics with respect to the beginning of the phase and simply set $t_k^+ = 0$ in a manner similar to the orbital phases where $E_k^+ = 0$. The state vector $(h, \phi, \theta, v, \gamma, \psi)$ is comprised of the altitude, longitude, geocentric latitude, velocity, flight path angle, and azimuth, respectively. The dynamics are described by the DAEs

$$\dot{h} = v \sin\gamma, \tag{7.75}$$

$$\dot{\phi} = \frac{v\cos\gamma \sin\psi}{r\cos\theta}, \tag{7.76}$$

$$\dot{\theta} = \frac{v\cos\gamma\cos\psi}{r}, \tag{7.77}$$

$$\dot{v} = -\frac{D}{m} - g\sin\gamma, \tag{7.78}$$

$$\dot{\gamma} = -\frac{1}{v}\left[\frac{qS}{m}u_2 + \left(g - \frac{v^2}{r}\right)\cos\gamma\right], \tag{7.79}$$

$$\dot{\psi} = \frac{1}{v}\left[\frac{-qS}{m\cos\gamma}u_1 + \frac{v^2}{r}\cos\gamma\sin\psi\tan\theta\right], \tag{7.80}$$

$$C_{LU} \ge C_L, \tag{7.81}$$

$$Q_U \ge Q, \tag{7.82}$$

where the stagnation point heating rate (BTU/(ft$^2$ sec)) is computed using the equation

$$Q = 17600\left(\frac{\rho}{\rho_E}\right)^{\frac{1}{2}}\left(\frac{v}{v_E}\right)^{3.15} \tag{7.83}$$

and the control variables $(u_1, u_2)$ are defined in terms of the lift coefficient $C_L$ and bank angle $\beta$ by

$$u_1 = -C_L \sin\beta, \tag{7.84}$$

$$u_2 = -C_L \cos\beta \tag{7.85}$$

with the inverse transformations given by

$$C_L = \sqrt{u_1^2 + u_2^2}, \tag{7.86}$$

$$\beta = \tan^{-1}(u_1/u_2). \tag{7.87}$$

The additional quantities

$$q = \frac{1}{2}\rho v^2, \tag{7.88}$$

$$D = qSC_D, \tag{7.89}$$

$$L = qSC_L, \tag{7.90}$$

$$C_D = C_{D0} + KC_L^2, \tag{7.91}$$

$$r = h + R_E, \tag{7.92}$$

$$g = \frac{\mu}{r^2}, \tag{7.93}$$

$$\alpha = \frac{C_L}{C_{L\alpha}} \tag{7.94}$$

complete the description of the dynamic model. The atmospheric density $\rho$ is computed using a smooth atmosphere model [46], and the sensible limit of the atmosphere is assumed to be 60 nm. It is important to note that the particular choice of control variables (7.84)–(7.85) suggested in [149] is preferable to lift coefficient and bank angle $(C_L, \beta)$, which is the more obvious engineering choice. The "winding" problem associated with angles as illustrated in Figure 6.15 is avoided. This is particularly important in order to achieve robust convergence for this application. The peak heating rate is limited by the path inequality constraint (7.82) to the value $Q_U = 400$ BTU/(ft$^2$ sec). The parametric values given in Table 7.8 complete the definition of the dynamic model.

To improve robustness it is also useful to limit the dynamic variables by imposing the following simple bounds:

$$0 \le h(t) \le 60\,\text{nm}, \tag{7.95a}$$

$$170\,\text{deg} \le \phi(t) \le 190\,\text{deg}, \tag{7.95b}$$

$$-20\,\text{deg} \le \theta(t) \le 80\,\text{deg}, \tag{7.95c}$$

$$25000\,\text{ft/sec} \le v(t) \le 35000\,\text{ft/sec}, \tag{7.95d}$$

$$-5\,\text{deg} \le \gamma(t) \le 5\,\text{deg}, \tag{7.95e}$$

$$0\,\text{deg} \le \psi(t) \le 40\,\text{deg}, \tag{7.95f}$$

$$-1.1C_{LU} \le u_1(t) \le 1.1C_{LU}, \tag{7.95g}$$

$$-1.1C_{LU} \le u_2(t) \le 1.1C_{LU}. \tag{7.95h}$$

**Table 7.8.** *Dynamic model parameters.*

| | | | |
|---|---|---|---|
| $g_0$ | 32.174 ft/sec$^2$ | $m_0$ | 519.5 slug |
| $I_{sp}$ | 310 sec | $R_E$ | 20926430 ft |
| $\mu$ | $1.40895\times10^{16}$ ft$^3$/sec$^2$ | $\rho_E$ | .0023769 slug/ft$^3$ |
| $S$ | 125.84 ft$^2$ | $C_{D0}$ | .032 |
| $K$ | 1.4 | $C_{L\alpha}$ | .5699 |
| $C_{LU}$ | 0.4 | $v_E$ | $\sqrt{\mu/R_E}$ ft/sec |

## 7.2.3 Boundary Conditions

The trajectory begins in a circular orbit at an altitude of 19323 nm, i.e., an ECI state vector

$$\mathbf{r}^{\mathsf{T}}(t_0) = (1.38335209528 \times 10^8, 0, 0), \qquad \mathbf{v}^{\mathsf{T}}(t_0) = (0, 1.00920971977 \times 10^4, 0). \tag{7.96}$$

The first de-orbit burn is approximated by an instantaneous velocity change as in examples (6.6) and (7.1). To be more precise, we use the *impulsive* $\Delta v$ approximation

$$\mathbf{v}(t_1) = \mathbf{v}(t_0) + \mathbf{\Delta v}_1, \tag{7.97}$$

where $\mathbf{v}(t_0)$ is the velocity before the burn, $\mathbf{v}(t_1)$ is the velocity after the burn, $\mathbf{\Delta v}$ is the velocity added by the burn, and $t_0 = t_1$. The velocity change is related to the mass by the boundary condition

$$m(t_0) = m(t_1)\exp\left(\frac{\|\mathbf{\Delta v}_1\|}{g_0 I_{sp}}\right), \tag{7.98}$$

where $m(t_0) = m_0$ is given in Table 7.8. The final value for $\Delta E_1^+$ must satisfy the boundary condition

$$h(t_2^-) = \|\mathbf{r}(\Delta E_1^+)\| - R_E = 60\,\text{nm}. \tag{7.99}$$

As with many orbital problems, this application has more than one local solution. To distinguish between transfers that differ only because of orbital symmetry, let us (arbitrarily) focus on solutions that begin with the initial de-orbit burn at a *descending node*. This can be achieved by imposing a bound on the $z$-component of the velocity change

$$\Delta v_{z1} \le 0, \tag{7.100}$$

which ensures that the $z$-component of the inertial velocity is negative.

Now, motion during orbital and atmospheric phases is described using two different coordinate systems; however, at the phase boundaries the coordinate systems must be consistent. Thus for each atmospheric phase $k$, where $k = 2,4,6,8$, the following consistency conditions must be satisfied:

$$60\,\text{nm} = h(t_k^+), \qquad h(t_{k+1}^-) = 60\,\text{nm}, \tag{7.101a}$$

$$\phi\left[\mathbf{r}(t_k^-), \mathbf{v}(t_k^-)\right] = \phi(t_k^+), \qquad \phi(t_{k+1}^-) = \phi\left[\mathbf{r}(t_{k+1}^+), \mathbf{v}(t_{k+1}^+)\right], \tag{7.101b}$$

$$\theta\left[\mathbf{r}(t_k^-), \mathbf{v}(t_k^-)\right] = \theta(t_k^+), \qquad \theta(t_{k+1}^-) = \theta\left[\mathbf{r}(t_{k+1}^+), \mathbf{v}(t_{k+1}^+)\right], \tag{7.101c}$$

$$v\left[\mathbf{r}(t_k^-), \mathbf{v}(t_k^-)\right] = v(t_k^+), \qquad v(t_{k+1}^-) = v\left[\mathbf{r}(t_{k+1}^+), \mathbf{v}(t_{k+1}^+)\right], \tag{7.101d}$$

$$\gamma\left[\mathbf{r}(t_k^-),\mathbf{v}(t_k^-)\right]=\gamma(t_k^+), \qquad \gamma(t_{k+1}^-)=\gamma\left[\mathbf{r}(t_{k+1}^+),\mathbf{v}(t_{k+1}^+)\right], \tag{7.101e}$$
$$\psi\left[\mathbf{r}(t_k^-),\mathbf{v}(t_k^-)\right]=\psi(t_k^+), \qquad \psi(t_{k+1}^-)=\psi\left[\mathbf{r}(t_{k+1}^+),\mathbf{v}(t_{k+1}^+)\right]. \tag{7.101f}$$

Observe that the conditions at the beginning of an atmospheric phase require computing flight path coordinates using the ECI state from the previous phase (e.g., $\phi\left[\mathbf{r}(t_k^-),\mathbf{v}(t_k^-)\right]$). Conversely, at the end of the atmospheric phase, these *linkage conditions* are computed using the ECI state vector from the next, orbital, phase. In general the transformation matrix $\mathbf{Q}_{LE}$ from local horizontal (LH) to earth centered inertial (ECI) coordinates at the position $\mathbf{r}$ is given by

$$\widehat{\mathbf{z}}=-r^{-1}\mathbf{r}, \tag{7.102}$$
$$\widehat{\mathbf{x}}=\|\mathbf{e}_3-\hat{z}_3\widehat{\mathbf{z}}\|^{-1}(\mathbf{e}_3-\hat{z}_3\widehat{\mathbf{z}}), \tag{7.103}$$
$$\widehat{\mathbf{y}}=\widehat{\mathbf{z}}\times\widehat{\mathbf{x}}, \tag{7.104}$$
$$\mathbf{Q}_{LE}(\mathbf{r})=\left[\widehat{\mathbf{x}}\quad\widehat{\mathbf{y}}\quad\widehat{\mathbf{z}}\right], \tag{7.105}$$

where $\mathbf{e}_3^{\mathsf{T}}=(0,0,1)$. Thus the intrinsic or flight path coordinates $(h,\phi,\theta,v,\gamma,\psi)$ can be computed from the ECI state $(\mathbf{r},\mathbf{v})$ as follows:

$$r=\|\mathbf{r}\|, \tag{7.106a}$$
$$\widetilde{\mathbf{v}}=\mathbf{Q}_{LE}^{\mathsf{T}}(\mathbf{r})\mathbf{v}, \tag{7.106b}$$
$$h=r-R_E, \tag{7.106c}$$
$$\phi=\tan^{-1}(r_2/r_1), \tag{7.106d}$$
$$\theta=\sin^{-1}(r_3/r), \tag{7.106e}$$
$$v=\|\mathbf{v}\|, \tag{7.106f}$$
$$\gamma=\sin^{-1}(-\tilde{v}_3/v), \tag{7.106g}$$
$$\psi=\tan^{-1}(\tilde{v}_2/\tilde{v}_1). \tag{7.106h}$$

The inverse of transformation (7.106a)–(7.106h) can be used to compute ECI coordinates given intrinsic quantities as follows:

$$r=h+R_E, \tag{7.107a}$$
$$r_1=r\cos\theta\cos\phi, \tag{7.107b}$$
$$r_2=r\cos\theta\sin\phi, \tag{7.107c}$$
$$r_3=r\sin\theta, \tag{7.107d}$$
$$\tilde{v}_1=v\cos\gamma\cos\psi, \tag{7.107e}$$
$$\tilde{v}_2=v\cos\gamma\sin\psi, \tag{7.107f}$$
$$\tilde{v}_3=-v\sin\gamma, \tag{7.107g}$$
$$\mathbf{v}=\mathbf{Q}_{LE}(\mathbf{r})\widetilde{\mathbf{v}}. \tag{7.107h}$$

To ensure that an atmospheric phase begins with decreasing altitude and terminates with altitude increasing, the flight path angle at the phase boundaries is restricted by the boundary conditions

$$0\ge\gamma(t_k^+), \qquad \gamma(t_{k+1}^-)\ge 0. \tag{7.108}$$

The vehicle mass during all atmospheric phases and the final phase must be consistent with the final mass after the first burn (in phase 1), so we must have

$$m(t_1) = m(t_2) = m(t_4) = m(t_6) = m(t_8) = m(t_9). \tag{7.109}$$

The intermediate orbital phases 3, 5, and 7 all begin and end at the atmospheric limit. Therefore we must impose conditions similar to (7.99) at the beginning and end of these phases:

$$\|\mathbf{r}(0)\| - R_E = \|\mathbf{r}(\Delta E_k^+)\| - R_E = 60\,\text{nm}. \tag{7.110}$$

The final outbound orbital phase is a mirror image of the first phase. The altitude at the beginning of the phase is constrained:

$$\|\mathbf{r}(0)\| - R_E = 60\,\text{nm}. \tag{7.111}$$

At the final time since there is no change in the position $\mathbf{r}(t_f) = \mathbf{r}(t_9^+)$. However, the final velocity is altered by the final burn, and so we have

$$\mathbf{v}(t_f) = \mathbf{v}(t_9^+) + \boldsymbol{\Delta}\mathbf{v}_2, \tag{7.112}$$

with a corresponding mass change defined by the condition

$$m(t_9) = m(t_f)\exp\left(\frac{\|\boldsymbol{\Delta}\mathbf{v}_2\|}{g_0 I_{sp}}\right). \tag{7.113}$$

Furthermore the final orbit conditions must be satisfied:

$$\|\mathbf{r}(t_f)\| - R_E = 100\,\text{nm}, \tag{7.114}$$

$$\|\mathbf{v}(t_f)\| = \sqrt{\frac{\mu}{r_f}}, \tag{7.115}$$

$$\mathbf{r}^{\mathsf{T}}(t_f)\mathbf{v}(t_9^+) = 0, \tag{7.116}$$

$$\mathbf{r}^{\mathsf{T}}(t_f)\boldsymbol{\Delta}\mathbf{v}_2 = 0, \tag{7.117}$$

$$i(t_f) = i_F, \tag{7.118}$$

where the inclination in (7.118) can be computed as in (7.34a)–(7.34e).

The objective is to maximize the final mass, that is,

$$F = m(t_f). \tag{7.119}$$

### 7.2.4 Initial Guess

The efficient solution for a nonlinear problem such as this can be dictated by how good the initial guess is. Of course it is also important that the initial guess be relatively easy to compute, at least compared to the actual problem. For this example a reasonably simple approximate solution can be obtained by first solving a small NLP problem. In particular, let us approximate the trajectory by *ignoring the atmosphere* and model the action of the atmosphere by impulsive velocity changes. The basic approach is illustrated schematically in Figure 7.10.

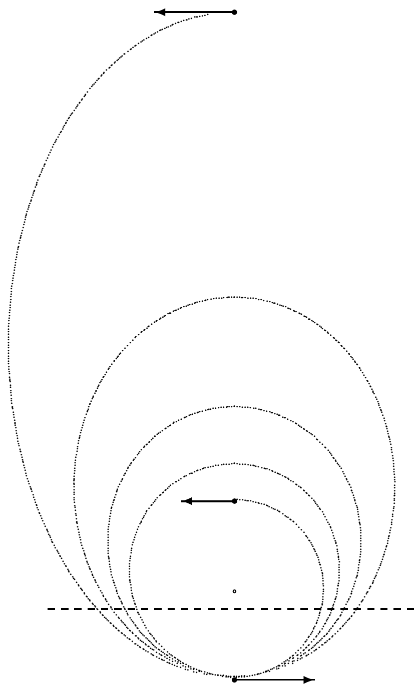


**Figure 7.10.** *Bi-elliptic transfer approximate solution.*

For this simple model we permit impulsive velocity changes to occur at the initial and final points and at each perigee location. Furthermore, we assume that there is no $\Delta\mathbf{v}$ in the $x$-direction and that all of the impulsive changes at perigee are identical. Thus we pose an NLP problem with $n = 2 \times (1 + 1 + 1) = 6$ variables. To fix the orbit geometry, analytic propagation is used with the inbound and outbound segments having $\Delta E = \pi$ and the intermediate arcs of length $\Delta E = 2\pi$. The desired initial and final conditions are imposed as constraints. Furthermore, we arbitrarily insist that each of the perigee burns occur at a fixed altitude of 40 nm, which is "inside" the atmosphere. This simple bi-elliptic orbit transfer problem can be solved easily to minimize the total $\Delta\mathbf{v}$ using the same NLP algorithm used by the complete $\mathbb{SOCS}$ algorithm. Table 7.9 summarizes the bi-elliptic

**Table 7.9.** *Bi-elliptic transfer solution.*

| Orbit | Perigee (nm) | Apogee (nm) | Inclination (deg) |
|---|---|---|---|
| Initial | 19323.0 | 19323.0 | 0.00000 |
| Inbound | 40.0000 | 19323.0 | 86.4340 |
| First Intermediate | 40.0000 | 8032.62 | 86.9445 |
| Second Intermediate | 40.0000 | 3733.28 | 87.5242 |
| Third Intermediate | 40.0000 | 1479.40 | 88.1873 |
| Outbound | 40.0000 | 100.000 | 88.9536 |
| Final | 100.000 | 100.000 | 89.0000 |

transfer solution. The bi-elliptic solution not only provides reasonable estimates for the orbit phases but also can be used to construct estimates for the states at the beginning and end of each atmosphere phase. Linear interpolation between the states at the beginning and end of each phase is used for all of the differential variables. Guessing constant values for $C_L = .25$ and $\beta = 0$ yields estimates for the control variables of $u_1(t) = 0$ and $u_2(t) = -.25$.

### 7.2.5 Numerical Results

The nine-phase problem described has been solved using $\mathbb{SOCS}$, and Table 7.10 presents a summary of the algorithm performance. The problem was initialized using the approach described in Section 7.2.4. A total of seven mesh-refinement iterations were performed as summarized in each row of Table 7.10. Column two gives the number of grid points. All of the orbital phases (1, 3, 5, 7, 9) utilized the analytic Kepler propagation scheme, and within $\mathbb{SOCS}$ this approach is implemented using two grid points—the initial and final phase boundaries. No mesh refinement is performed on analytic phases, and consequently the number of grid points remains unchanged. In contrast, the atmospheric phases (2, 4, 6, 8) are initiated using a trapezoidal discretization with 10 points equally spaced within the phase. The resulting sparse NLP has $n = 402$ variables and at the solution the number of degrees of freedom $n_d = n - \hat{m} = 77$. The total number of QP subproblems (NQP) required to solve the first coarse grid NLP was 173, and it took 3.4 CPU sec on a Dell M90 laptop. After two refinement iterations the discretization scheme is changed to an HSC method, which is used for all subsequent refinement iterations. The discretization error $\epsilon$ was reduced from $1.3\times10^{-1}$ on the first grid to $9.2\times10^{-8}$ on the last. The overall solution was computed in 16.95 CPU sec.

The optimal impulsive velocity increments are

$$\mathbf{\Delta v}_1^* = (-1.8392052 \times 10^2, -7.1911864 \times 10^3, -4.3121550 \times 10^3)^\mathsf{T}, \tag{7.120}$$

$$\mathbf{\Delta v}_2^* = (3.8614396 \times 10^0, -2.0351389 \times 10^1, -1.1615574 \times 10^2)^\mathsf{T} \tag{7.121}$$

and the total (minimum) $\mathbf{\Delta v}$ for this example is

$$\|\mathbf{\Delta v}_1^*\| = 8386.9940 \text{ ft/sec}, \tag{7.122}$$

$$\|\mathbf{\Delta v}_2^*\| = 117.98833 \text{ ft/sec}, \tag{7.123}$$

$$\|\mathbf{\Delta v}_1^*\| + \|\mathbf{\Delta v}_2^*\| = 8504.9823 \text{ ft/sec}, \tag{7.124}$$

**Table 7.10.** *Mesh-refinement summary.*

| $k$ | $M$ | $n$ | $n_d$ | NQP | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|
| 1 | (2,10,2,10,2,10,2,10,2) | 402 | 77 | 173 | $1.3\times10^{-1}$ | 3.4 |
| 2 | (2,19,2,19,2,19,2,19,2) | 690 | 144 | 21 | $7.5\times10^{-3}$ | 0.4 |
| 3 | (2,19,2,19,2,19,2,19,2) | 834 | 289 | 32 | $9.8\times10^{-4}$ | 1.7 |
| 4 | (2,37,2,37,2,37,2,37,2) | 1554 | 569 | 17 | $3.9\times10^{-5}$ | 1.4 |
| 5 | (2,73,2,73,2,73,2,73,2) | 2994 | 1138 | 17 | $1.5\times10^{-6}$ | 3.4 |
| 6 | (2,95,2,131,2,136,2,145,2) | 5144 | 1976 | 11 | $1.2\times10^{-7}$ | 4.6 |
| 7 | (2,95,2,131,2,136,2,150,2) | 5194 | 1996 | 5 | $9.2\times10^{-8}$ | 1.9 |
| Total | 522 | | | 4979 | | 16.95 |

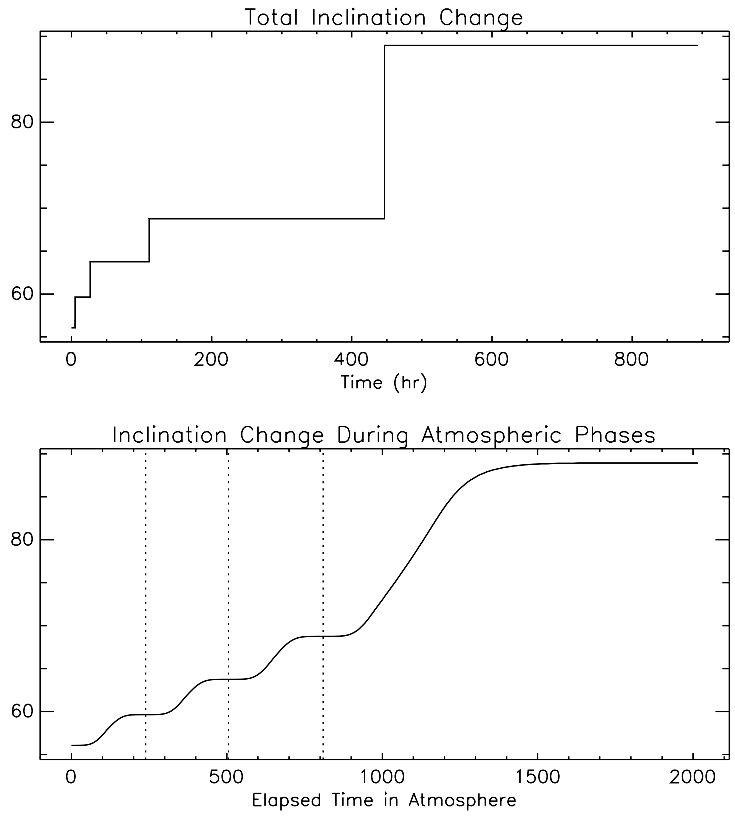


**Figure 7.11.** *Inclination change.*

which corresponds to masses given by

$$m^*(t_1) = 224.07393\,\text{slug}, \tag{7.125}$$

$$m^*(t_f) = 221.43883\,\text{slug}. \tag{7.126}$$

The dynamic histories of particular interest are illustrated in Figures 7.11, 7.12, and 7.13. Since the time scale for the atmospheric portions of the transfer are significantly shorter than the orbit transit times, it is convenient to display the results using elapsed time in the atmosphere as the independent variable. The phase boundaries are indicated using a vertical dotted line. In Figure 7.11 the inclination is plotted as a function of the total time, and also the elapsed time during the atmosphere passes. This figure clearly illustrates that the inclination change is accomplished primarily in the last atmospheric pass. It is also clear that the inclination changes appear to occur almost instantaneously because the duration of the atmospheric passes is much shorter than the entire transfer time. This also suggests that approximating the aerodynamic pass by an impulsive velocity as was done in the bi-elliptic initialization procedure is a reasonable approximation. Figure 7.12 plots the state variables, and it is clear from the altitude plot that each atmospheric pass begins at the atmospheric limit of 60 nm. The velocity decreases monotonically because of

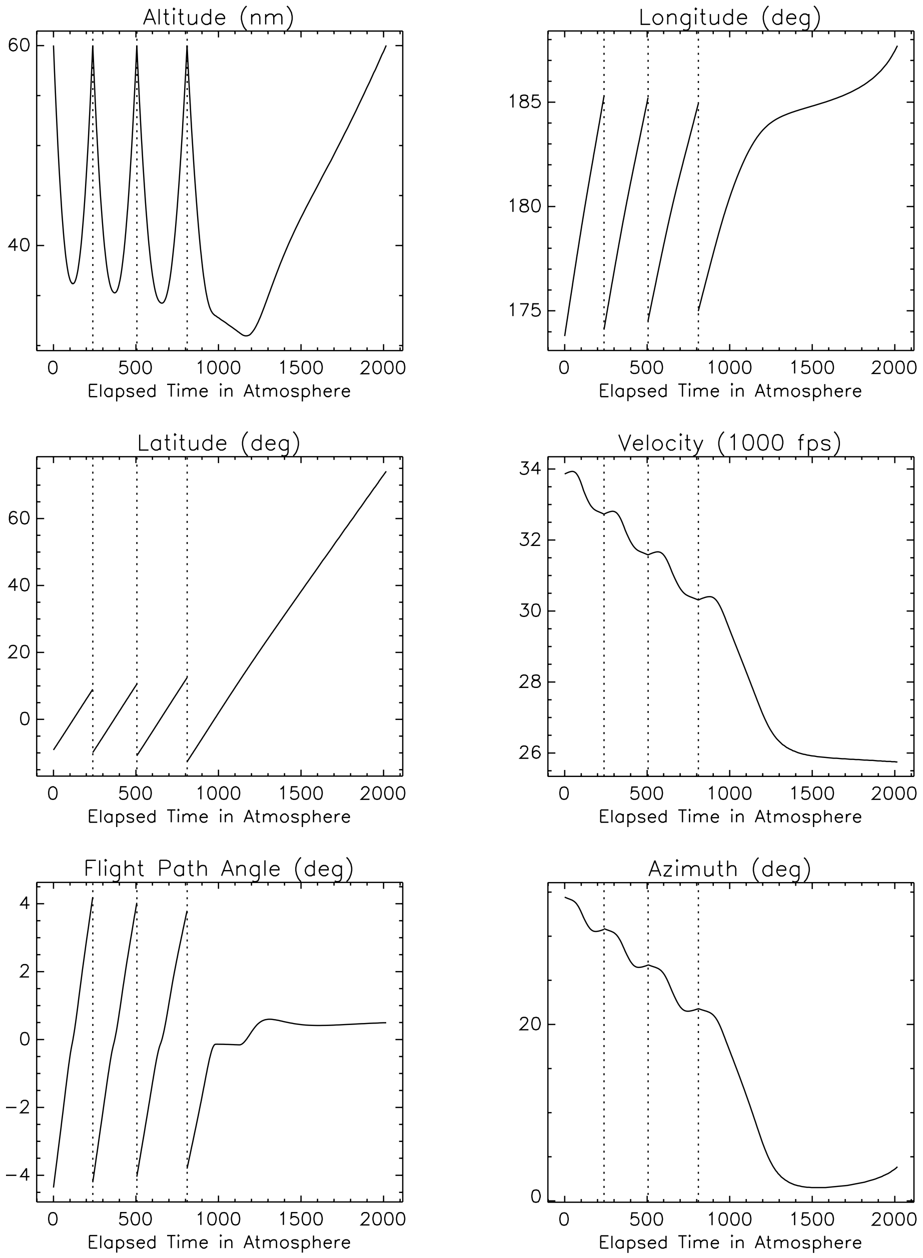


**Figure 7.12.** *States during four atmospheric passes.*

atmospheric drag, and for each phase the flight path angle begins negative and ends positive as required. Figure 7.13 presents the actual controls $u_1(t)$ and $u_2(t)$, as well as the derived values for angle of attack and bank angle. Also included are the time histories for the

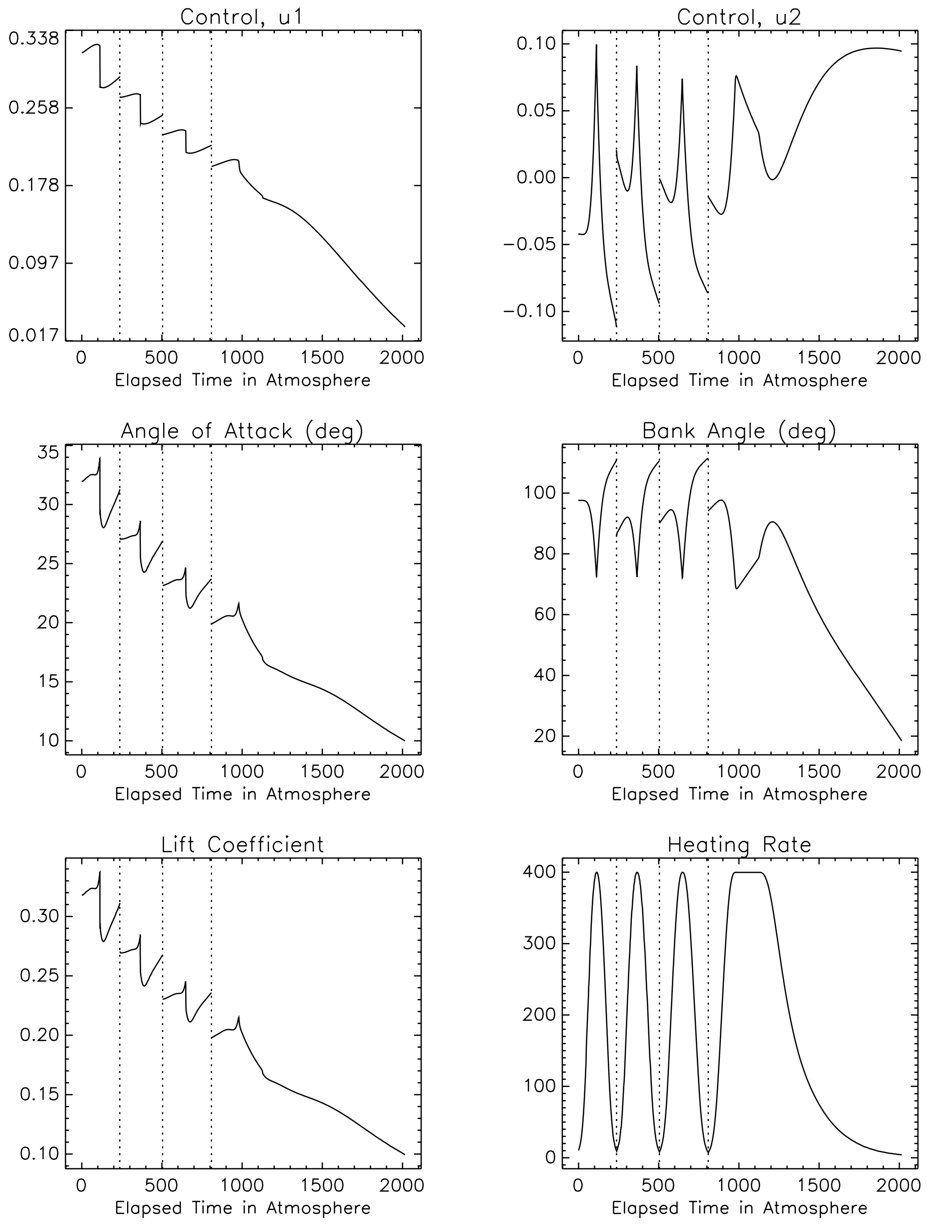


**Figure 7.13.** *Controls during four atmospheric passes.*

path constraint functions $C_L$ and $Q$ that appear in (7.81) and (7.82). For comparison, the same problem was solved using a single atmospheric pass, and the corresponding dynamic histories are illustrated in Figures 7.14 and 7.15. The final mass using a single pass was $m^*(t_f) = 212.16080$ slug. Finally, Figure 7.16 illustrates the entire multipass aero-assisted

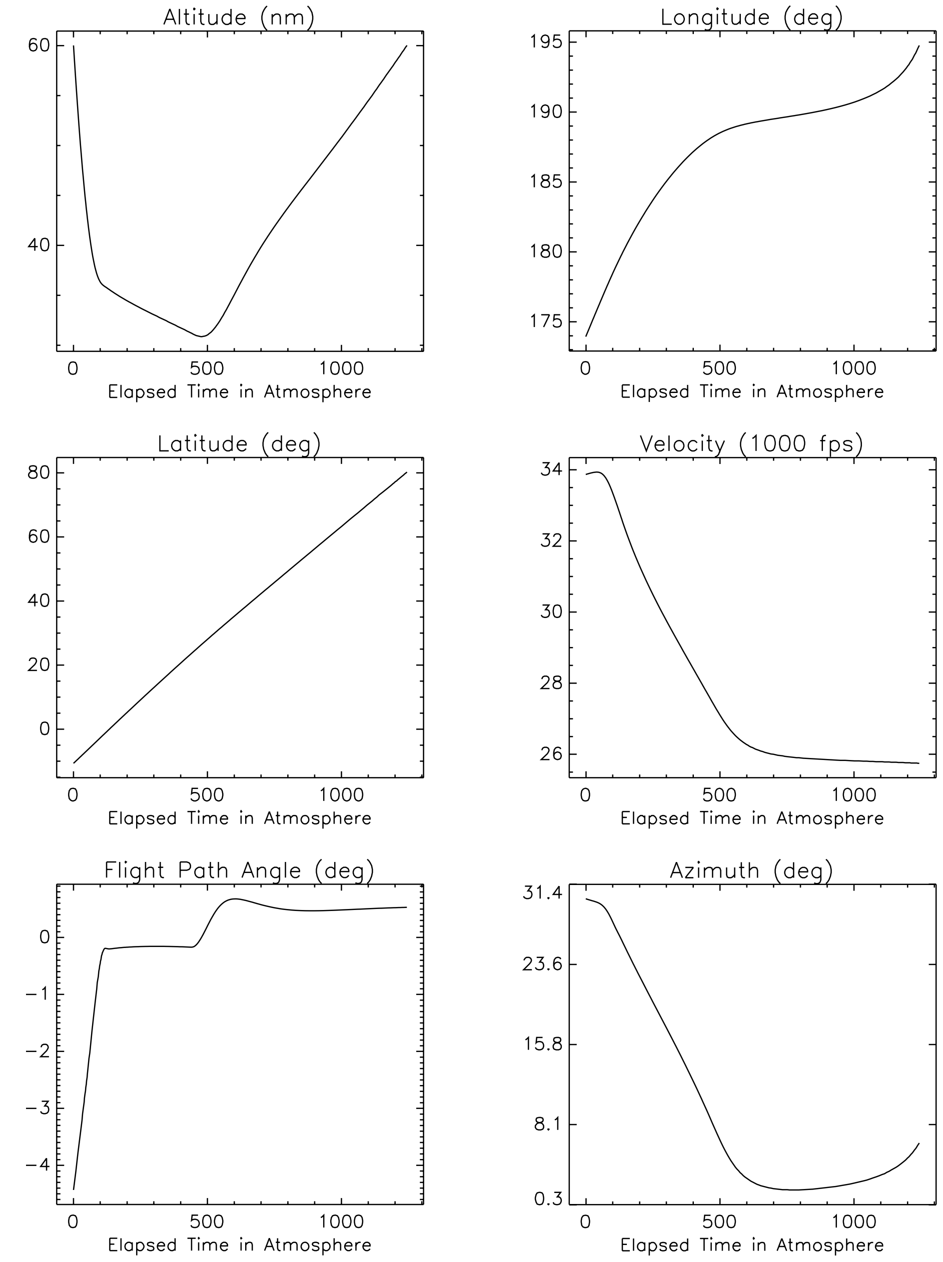


**Figure 7.14.** *States during a single atmospheric pass.*

transfer trajectory and Figure 7.17 illustrates the single pass solution. The trajectory plane shading is defined by the instantaneous inclination, which illustrates the transition from $i \approx 56$ (deg) shown in light gray to $i = 89$ (deg) in dark.

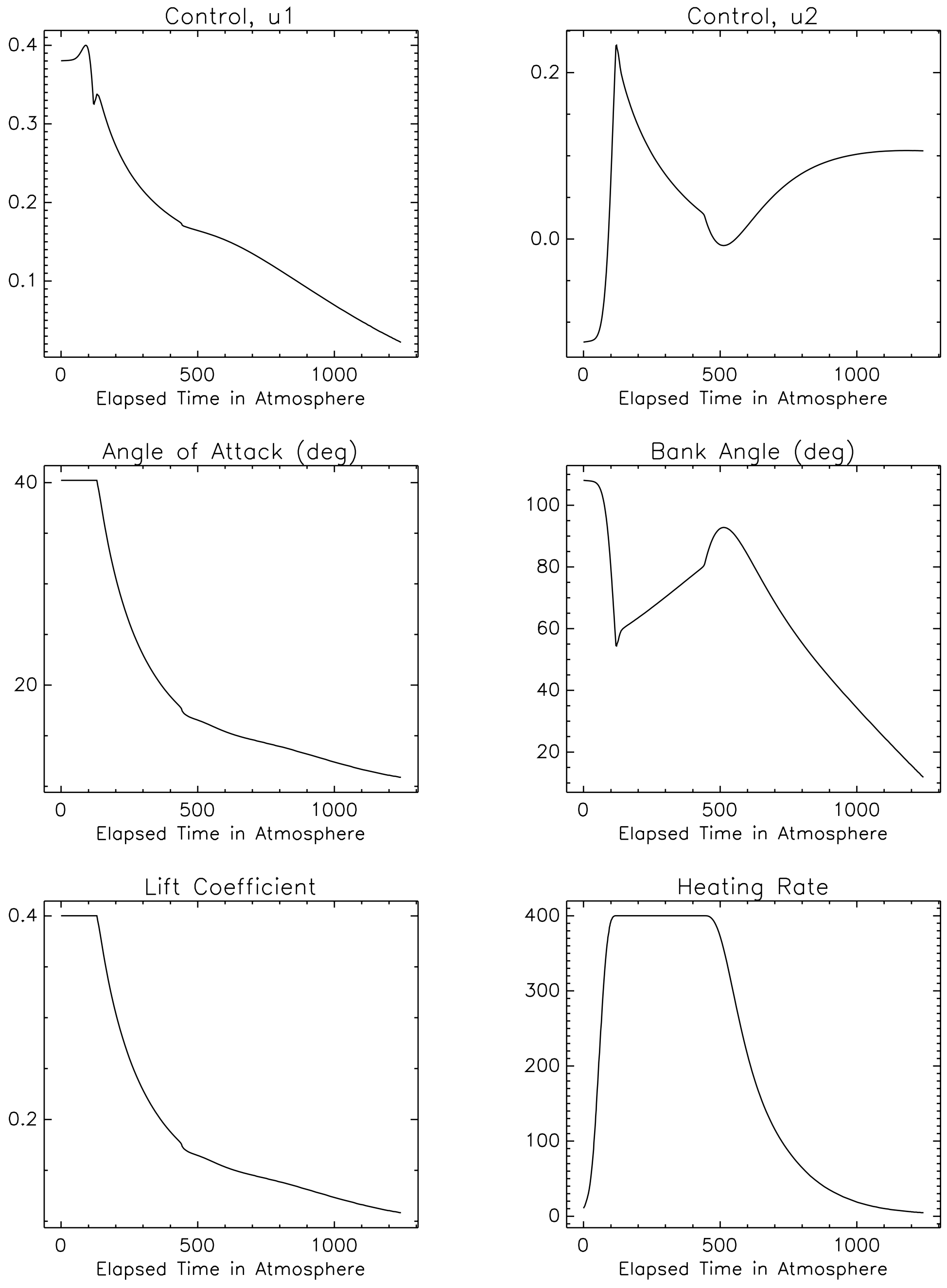


**Figure 7.15.** *Controls during a single atmospheric pass.*

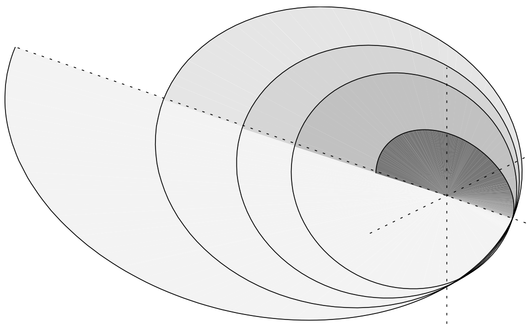


**Figure 7.16.** *Optimal four-pass aero-assisted transfer.*

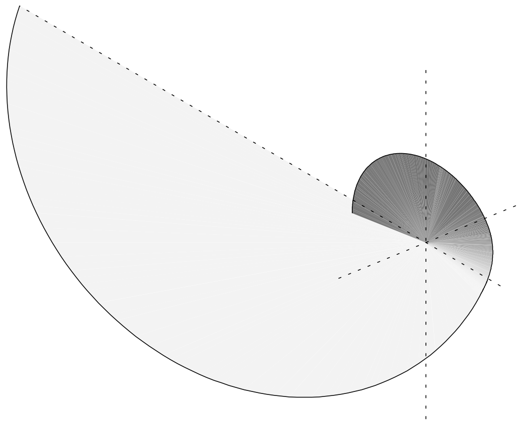


**Figure 7.17.** *Optimal single-pass aero-assisted transfer.*

## 7.3 Delay Differential Equations

For many physical processes the mathematical description involves "time lags" or "retarded arguments." Thus instead of the usual ODE model

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

the problem description entails a delay differential equation (DDE) model such as

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{y}(t-\tau), t, t-\tau]. \tag{7.127}$$

Observe that the dynamics are expressed in terms of the state $\mathbf{y}(t-\tau)$ evaluated at the delay time $t-\tau$, where $\tau > 0$. This *fixed delay* model is one of the simplest types of DDEs. Clearly more complicated descriptions can be envisioned if, for example, one considers multiple delay times $\tau_k$ and/or nonlinear delay arguments such as $\mathbf{y}(\boldsymbol{\phi}[t-\tau])$. Furthermore, one can consider delay-differential-algebraic equation (DDAE) problem formulations [3]. To fully appreciate the computational challenges caused by stiffness and propagation of discontinuities it is instructive to review the paper by Shampine and Gahinet [157].

Although a complete discussion of the subject is beyond the scope of this book, it is instructive to illustrate some of the issues that arise. To do so we will focus on examples having fixed delay that can be converted to BVPs with ODEs using an approach called *method of steps*. It should be emphasized that this is not the only (or perhaps the best) approach for these problems.

**Example 7.2** ENZYME KINETICS. Let us focus on a particular example originally published by Okamoto and Hayashi [137] and cited by Hairer, Norsett, and Wanner [106, pp. 348–349] that describes enzyme kinetics. The enzyme concentrations on the time interval $0 \le x \le 160$ are described by the DDE system

$$s_1' = I - z s_1(x), \tag{7.128}$$

$$s_2' = z s_1(x) - c_2 s_2(x), \tag{7.129}$$

$$s_3' = c_2 s_2(x) - c_3 s_3(x), \tag{7.130}$$

$$s_4' = c_3 s_3(x) - c_4 s_4(x), \tag{7.131}$$

where the process is inhibited by

$$z = \frac{c_1}{1 + \alpha [s_4(x-4)]^3} \tag{7.132}$$

with $I = 10.5$, $c_1 = c_2 = c_3 = 1$, $c_4 = 0.5$, and $\alpha = 0.0005$. As $x \to \infty$, the system approaches an equilibrium state $\mathbf{s}(x) \to \tilde{\mathbf{s}}$ with $\tilde{s}_1 = I(1 + .004I^3)$, $\tilde{s}_2 = \tilde{s}_3 = I$, and $\tilde{s}_4 = 2I$. In this example the number of delay equations $L = 4$, and the number of delay intervals $N = 40$, with a delay time $\tau = 4$.

For systems such as this with constant delay, it is possible to reformulate the problem as a BVP involving just ODEs, using a technique called the *method of steps*. Specifically let us consider a fixed time domain $0 \le t \le \tau$ and then transform the original time into multiples of the delay time, that is,

$$x = t + k\tau \tag{7.133}$$

for $k = 0, \ldots, N-1$, where $N$ is the total number of delay steps. Observe that when $k = 0$, $0 \le x \le \tau$, and when $k = 1$, $\tau \le x \le 2\tau$, and so forth. Using this transformation, each delay interval has been mapped onto the fixed interval $0 \le t \le \tau$. Using this mapping, the original time domain can be "folded" onto a single interval, by defining new dynamic variables according to

$$\begin{aligned} y_1(t) &= s_1(t), \\ &\vdots \\ y_5(t) &= s_1(t+\tau), \\ &\vdots \\ y_{j+kL}(t) &= s_j[t + k\tau] = s_j(x) \end{aligned} \tag{7.134}$$

for $j = 1, \ldots, L$, and $k = 0, \ldots, N-1$. Now if we differentiate (7.134), we obtain

$$\dot{y}_{j+kL} = \frac{dy_{j+kL}}{dt} = \frac{ds_j}{dt} = \frac{ds_j}{dx}\frac{dx}{dt} = s_j'. \tag{7.135}$$

Using the new dynamic variables, the original system of delay equations (7.128)–(7.132) is replaced by the system of ODEs

$$\dot{y}_{1+kL} = I - z y_{1+kL}, \tag{7.136}$$

$$\dot{y}_{2+kL} = z y_{1+kL} - c_2 y_{2+kL}, \tag{7.137}$$

$$\dot{y}_{3+kL} = c_2 y_{2+kL} - c_3 y_{3+kL}, \tag{7.138}$$

$$\dot{y}_{4+kL} = c_3 y_{3+kL} - c_4 y_{4+kL}, \tag{7.139}$$

where the delay term becomes

$$z = \frac{c_1}{1 + \alpha [y_{4+(k-1)L}]^3}. \tag{7.140}$$

The delay term given by (7.140) follows from (7.132) by noting that

$$s_4(x-4) = s_4[(t+k\tau) - \tau] = s_4[t + (k-1)\tau] = y_{4+(k-1)L}. \tag{7.141}$$

Of course the problem statement is not complete without the appropriate boundary conditions. By construction the dynamic variables $y$ describe behavior on a single delay interval. Furthermore at the boundary of neighboring delay intervals since $\tau + k\tau = (0) + (k+1)\tau$ we must have

$$y_{j+kL}(\tau) = s_j[\tau + k\tau] = s_j[(0) + (k+1)\tau] = y_{j+(k+1)L}(0) \tag{7.142}$$

for $j = 1, \ldots, L$, and $k = 0, \ldots, N-1$.

DDEs pose one additional complication not shared by ODEs, namely how to get started. For an ODE, it is sufficient to specify the initial values $\mathbf{y}(0)$. In contrast for a DDE, one must specify function histories over the entire startup region $-\tau \leq x \leq 0$. For the particular example illustrated here, we define the startup functions by

$$y_{1-L}(t) = s_1(x) = 60, \tag{7.143}$$

$$y_{2-L}(t) = s_2(x) = 10, \tag{7.144}$$

$$y_{3-L}(t) = s_3(x) = 10, \tag{7.145}$$

$$y_{4-L}(t) = s_4(x) = 20 \tag{7.146}$$

for $-\tau \leq t \leq 0$.

To summarize, the original problem was posed in terms of a system of DDEs for the dynamic variables $\mathbf{s}(x)$ defined on the time domain $0 \leq x \leq 160 = 40 \times 4$. The problem can be recast as a BVP for a system of ODEs in the dynamic variables $\mathbf{y}(t)$ which is defined on the domain $0 \leq t \leq \tau = 4$. The approach is attractive since it provides a mechanism to extend methods for optimal control and estimation of ODEs to DDEs. In spite of its attractive simplicity the method of steps does present a number of potentially problematic issues. First, a DDE IVP is replaced by an ODE BVP. Second, the number of ODEs $(LN)$ is much larger than the number of DDEs $(L)$. Finally, the technique used to control ODE integration (discretization) error may not be appropriate or efficient for a DDE system.

To illustrate the process, let us take a two step approach. First, let us treat the constants $c_1 = c_2 = c_3 = 1$, $c_4 = 0.5$ as known and solve the DDE system. Then as a second step, let us construct an inverse problem and attempt to estimate the parameters $\mathbf{p} = (c_1, c_2, c_3, c_4)$. Treating the solution to the first step as a truth model $\widetilde{\mathbf{y}}$, data for the second step can be constructed by adding noise to the truth model, i.e.,

$$\widehat{\mathbf{y}}(\theta_i) = \widetilde{\mathbf{y}}(\theta_i) + \boldsymbol{\nu}, \tag{7.147}$$

where $\boldsymbol{\nu} \sim \mathcal{N}(0, \sigma)$ is a vector of independent identically distributed Gaussian random variables with mean zero and variance $\sigma = 0.01$. For this example 10 evaluation times $\theta_i$ are equally spaced over the time interval. The objective is to minimize

$$F = \frac{1}{2} \sum_i [\mathbf{y}_i - \widehat{\mathbf{y}}_i]^{\mathsf{T}} [\mathbf{y}_i - \widehat{\mathbf{y}}_i] \tag{7.148}$$

by choosing the parameters $\mathbf{p}$ and dynamic variables $\mathbf{y}(t)$ while satisfying the differential equations (7.136)–(7.140) and boundary conditions (7.142). As initial conditions we impose

$$y_1(0) = 60, \tag{7.149}$$
$$y_2(0) = 10, \tag{7.150}$$
$$y_3(0) = 10, \tag{7.151}$$
$$y_4(0) = 20 \tag{7.152}$$

with the startup functions given by (7.143)–(7.146). The resulting problem has 160 state variables $\mathbf{y}$ and 156 boundary conditions (7.142). The objective function has 1600 total residuals.

A piecewise constant initial guess is used to initiate the iterative process. Thus with two grid points the initial guess is just $y_j^{(0)}(0) = y_j^{(0)}(\tau) = 0$ for $j = 5, \ldots, 160$. On the first delay interval when $j = 1, 2, 3, 4$ and the initial conditions (7.149)–(7.152) provide the guess $y_j^{(0)}(0) = y_j^{(0)}(\tau) = y_j(0)$. Eight mesh-refinement iterations are required to achieve the desired accuracy of $\epsilon \le \delta = 1 \times 10^{-7}$. Table 7.11 presents a summary of the algorithm performance using $\mathbb{SOCS}$ to solve Step 1. Table 7.12 presents similar information for the parameter estimation step. It is worth noting that by exploiting the right-hand-side sparsity, it is possible to compute finite difference gradient information quite efficiently for this application as indicated by the values in Table 7.13. Figure 7.18 illustrates the solution,

**Table 7.11.** *Mesh-refinement summary (nominal).*

| $k$ | $M$ | $n$ | NGC | NFE | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|
| 1 | 2 | 320 | 16 | 87 | $1.04\times10^{0}$ | .13 |
| 2 | 3 | 480 | 12 | 74 | $3.07\times10^{-1}$ | .09 |
| 3 | 3 | 480 | 15 | 196 | $5.51\times10^{-2}$ | .14 |
| 4 | 5 | 800 | 3 | 29 | $4.88\times10^{-3}$ | .08 |
| 5 | 9 | 1440 | 2 | 20 | $2.99\times10^{-4}$ | .12 |
| 6 | 17 | 2720 | 1 | 11 | $1.96\times10^{-5}$ | .20 |
| 7 | 33 | 5280 | 1 | 11 | $1.23\times10^{-6}$ | .42 |
| 8 | 65 | 10400 | 1 | 11 | $7.68\times10^{-8}$ | 1.34 |
| Total | 65 |  | 51 | 439 |  | 2.52 |

**Table 7.12.** *Mesh-refinement summary (parameter estimation).*

| $k$ | $M$ | $n$ | NGC | NFE | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|
| 1 | 2 | 324 | 20 | 260 | $2.91\times10^{-1}$ | .48 |
| 2 | 3 | 484 | 23 | 314 | $4.39\times10^{-1}$ | 1.42 |
| 3 | 3 | 484 | 19 | 733 | $5.07\times10^{-2}$ | 2.31 |
| 4 | 5 | 804 | 6 | 402 | $3.88\times10^{-3}$ | .63 |
| 5 | 9 | 1444 | 4 | 154 | $2.26\times10^{-4}$ | .42 |
| 6 | 17 | 2724 | 4 | 154 | $1.53\times10^{-5}$ | .92 |
| 7 | 33 | 5284 | 3 | 129 | $9.68\times10^{-7}$ | 1.84 |
| 8 | 65 | 10404 | 3 | 129 | $6.06\times10^{-8}$ | 5.59 |
| Total | 65 |  | 82 | 2275 |  | 13.61 |

**Table 7.13.** *Number of index sets $\gamma$.*

| Discretization | Trapezoidal | Hermite–Simpson |
|---|---|---|
| Step 1 | 3 | 8 |
| Step 2 | 5 | 12 |

with the delay intervals “unfolded” to correspond with the original problem statement. The estimated parameter values for this case are

$$\mathbf{p} = (1.0000097 \times 10^0, 9.9999697 \times 10^{-1}, 9.9999450 \times 10^{-1}, 5.0000131 \times 10^{-1})^{\mathsf{T}}, \tag{7.153}$$

which corresponds to the minimum value $F^* = 8.8747308 \times 10^{-2}$.

**Example 7.3** IMMUNOLOGY EXAMPLE. A second example originally published in Russian by G. I. Marchuk is also cited by Hairer, Norsett, and Wanner [106, pp. 349–351]. The dynamics are described by the DDE system

$$\frac{dV}{dx} = (h_1 - h_2 F)V, \tag{7.154}$$

$$\frac{dC}{dx} = \xi(m) h_3 F(x-\tau) V(x-\tau) - h_5(C-1), \tag{7.155}$$

$$\frac{dF}{dx} = h_4(C-F) - h_8 FV, \tag{7.156}$$

$$\frac{dm}{dx} = h_6 V - h_7 m. \tag{7.157}$$

The dynamics model the struggle of viruses $V(t)$, antibodies $F(t)$, and plasma cells $C(t)$ in a person infected with a viral disease. The relative characteristic damage is represented by $m(t)$, where the first term in (7.157) accounts for damaging and the second term for recuperation. The fact that plasma cell creation slows down when the organism is damaged by the viral infection is modeled by the term

$$\xi(m) = \begin{cases} 1 & \text{if } m \le 0.1, \\ (1-m)\frac{10}{9} & \text{if } 0.1 \le m \le 1. \end{cases} \tag{7.158}$$

Equation (7.154) is referred to as a predator-prey equation, and (7.155) models the creation of new plasma cells with a time lag due to infection. Notice that an equilibrium occurs

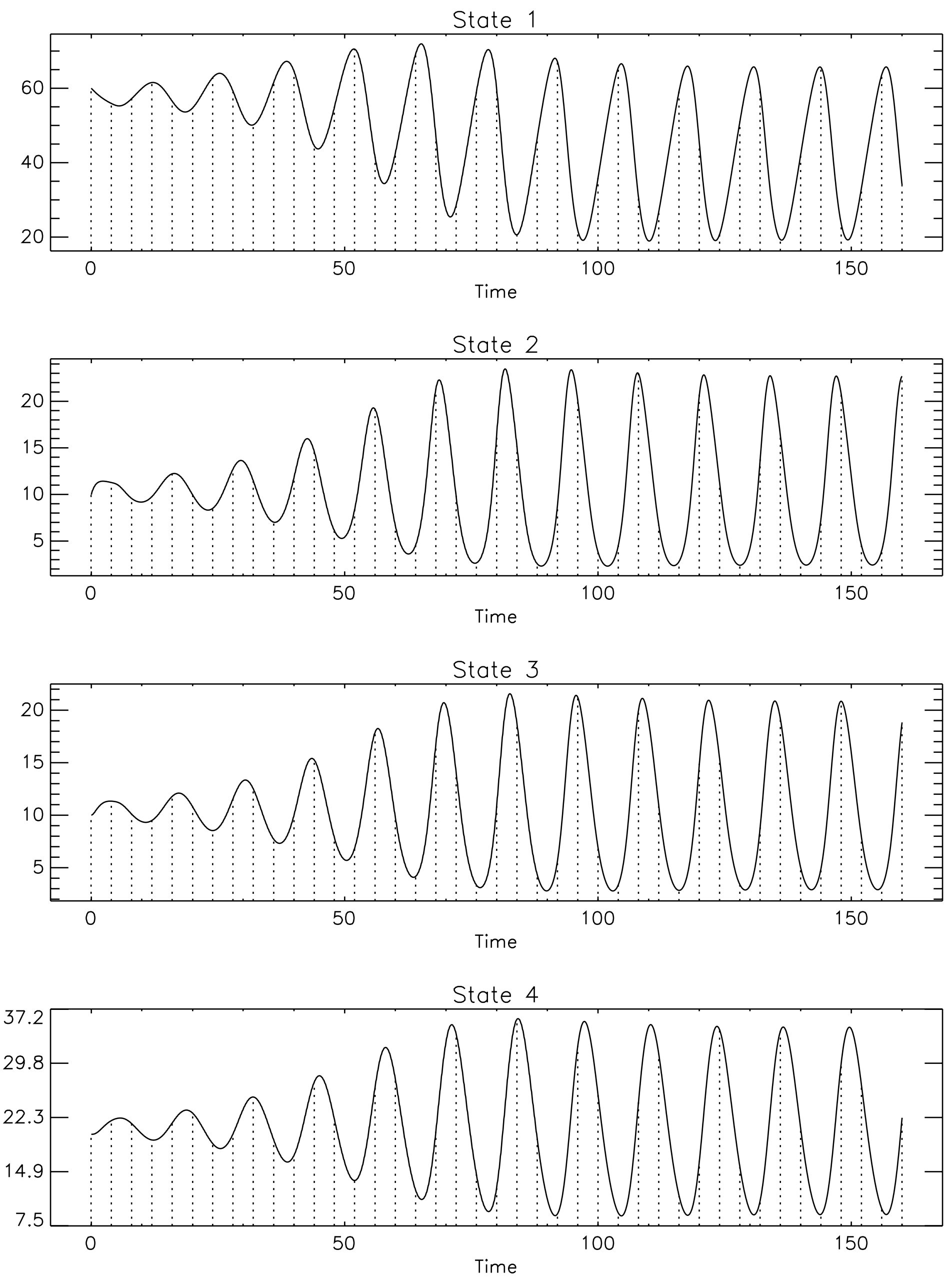


**Figure 7.18.** *Enzyme kinetic delay equation solution.*

with $C = 1$ if the first term in (7.155) is omitted. Equation (7.156) models three effects, namely creation of antibodies from plasma cells ($h_4C$), decrease in plasma cells due to aging ($-h_4F$), and binding with antigens ($-h_8FV$).

The DDE system (7.154)–(7.157) can be converted to a system of ODEs using the same technique introduced for the previous example. In particular the number of delay equations $L = 4$, and if we identify the variables

$$(y_{1+kL}, y_{2+kL}, y_{3+kL}, y_{4+kL}) = (V, C, F, m)$$

on delay intervals given by (7.133), one obtains the following equations:

$$\dot{y}_{1+kL} = \left[h_1 - h_2 y_{3+kL}\right] y_{1+kL}, \tag{7.159}$$

$$\dot{y}_{2+kL} = \xi(y_{4+kL}) h_3 y_{3+(k-1)L} y_{1+(k-1)L} - h_5\left[y_{2+kL} - 1\right], \tag{7.160}$$

$$\dot{y}_{3+kL} = h_4\left[y_{2+kL} - y_{3+kL}\right] - h_8 y_{3+kL} y_{1+kL}, \tag{7.161}$$

$$\dot{y}_{4+kL} = h_6 y_{1+kL} - h_7 y_{4+kL} \tag{7.162}$$

for $k = 0, \ldots, N-1$, where $N = 120$ is the total number of delay steps. For the specific case of interest, $\tau = 0.5$, $h_1 = 2$, $h_2 = 0.8$, $h_3 = 10^4$, $h_4 = 0.17$, $h_5 = 0.5$, $h_6 = 300$, $h_7 = 0.12$, and $h_8 = 8$. As startup conditions we use

$$y_{1-L}(t) = \max\left(0, 10^{-6} + t\right), \tag{7.163}$$

$$y_{2-L}(t) = 1, \tag{7.164}$$

$$y_{3-L}(t) = 1, \tag{7.165}$$

$$y_{4-L}(t) = 0 \tag{7.166}$$

for $-\tau \le t \le 0$. As before, we must also impose the "wrapping" boundary conditions (7.142). In summary, the complete solution to this DDE requires solving a nonlinear BVP with 480 ODEs and 476 nonlinear boundary conditions. After the continuous problem is discretized the resulting sparse NLP has no degrees of freedom, i.e., no objective function. Nevertheless, the constraints are very nonlinear, and as such initiating the iteration with a "good guess" is very important.

The serial nature of the problem suggests a very natural way to proceed. As a first step let us solve a problem with only one delay interval. Thus first solve

$$\dot{y}_1 = \left[h_1 - h_2 y_3\right] y_1, \tag{7.167a}$$

$$\dot{y}_2 = \xi(y_4) h_3 y_{3-L} y_{1-L} - h_5\left[y_2 - 1\right], \tag{7.167b}$$

$$\dot{y}_3 = h_4\left[y_2 - y_3\right] - h_8 y_3 y_1, \tag{7.167c}$$

$$\dot{y}_4 = h_6 y_1 - h_7 y_4. \tag{7.167d}$$

This problem involving four differential equations can be solved using a coarse discretization, e.g., a trapezoidal method with two grid points. Since the goal is to construct an initial guess for the "real problem," mesh refinement is not needed at this stage. To initiate this first step it is reasonable to guess constant values for the states. Denote the approximate solution obtained by this first step

$$\left[\tilde{y}_1(t), \tilde{y}_2(t), \tilde{y}_3(t), \tilde{y}_4(t)\right] \qquad \text{for } 0 \le t \le \tau. \tag{7.168}$$

Now let us solve a problem with two delay intervals, i.e., the system

$$\dot{y}_1 = \left[h_1 - h_2 y_3\right] y_1, \tag{7.169a}$$
$$\dot{y}_2 = \xi(y_4) h_3 y_{3-L} y_{1-L} - h_5\left[y_2 - 1\right], \tag{7.169b}$$
$$\dot{y}_3 = h_4\left[y_2 - y_3\right] - h_8 y_3 y_1, \tag{7.169c}$$
$$\dot{y}_4 = h_6 y_1 - h_7 y_4, \tag{7.169d}$$
$$\dot{y}_5 = \left[h_1 - h_2 y_7\right] y_5, \tag{7.169e}$$
$$\dot{y}_6 = \xi(y_8) h_3 y_{7-L} y_{5-L} - h_5\left[y_6 - 1\right], \tag{7.169f}$$
$$\dot{y}_7 = h_4\left[y_6 - y_7\right] - h_8 y_7 y_5, \tag{7.169g}$$
$$\dot{y}_8 = h_6 y_5 - h_7 y_8 \tag{7.169h}$$

subject to the boundary conditions

$$y_j(\tau) = y_{j+L}(0) \tag{7.170}$$

for $j = 1, \dots, L$. It is now necessary to supply an initial guess for this two-delay problem. Clearly the solution (7.168) satisfies (7.169a)–(7.169d). Also from (7.170) we can construct a guess for the remaining variables

$$\begin{aligned}\left[\tilde{y}_1(\tau), \tilde{y}_2(\tau), \tilde{y}_3(\tau), \tilde{y}_4(\tau)\right] &= \left[y_5^{(0)}(0), y_6^{(0)}(0), y_7^{(0)}(0), y_8^{(0)}(0)\right] \\ &= \left[y_5^{(0)}(\tau), y_6^{(0)}(\tau), y_7^{(0)}(\tau), y_8^{(0)}(\tau)\right].\end{aligned} \tag{7.171}$$

In effect this is a constant "prediction" for the next delay interval.

Obviously, this stepwise procedure can be continued. The procedure requires solving a sequence of BVPs. The number of differential equations grows from step to step. The solution to the BVP computed at step $k$ provides an excellent initial guess for the BVP that must be solved at step $(k+1)$. In essence the technique can be viewed as an extension of the predictor-corrector method for solving ODEs (cf. (3.42)). Even though the number of ODEs increases from step to step, each solution can be computed quite efficiently. The desired accuracy is achieved using mesh refinement only on the final step of the process. Table 7.14 summarizes the final solution refinement behavior. Figure 7.19 illustrates the solution, with the delay intervals "unfolded" to correspond with the original problem statement.

**Table 7.14.** *Mesh-refinement summary.*

| $k$ | $M$ | $n$ | NGC | NFE | $\epsilon$ | Time (sec) |
|---|---|---|---|---|---|---|
| 1 | 2 | 324 | 9 | 47 | $2.36\times10^{-2}$ | .57 |
| 2 | 3 | 484 | 26 | 159 | $3.40\times10^{-3}$ | .97 |
| 3 | 3 | 484 | 36 | 520 | $2.09\times10^{-4}$ | 1.57 |
| 4 | 5 | 804 | 4 | 54 | $2.69\times10^{-5}$ | .53 |
| 5 | 9 | 1444 | 5 | 67 | $1.03\times10^{-5}$ | 1.33 |
| 6 | 15 | 2724 | 1 | 15 | $1.30\times10^{-6}$ | .51 |
| 7 | 26 | 5284 | 1 | 15 | $1.82\times10^{-7}$ | 1.63 |
| 8 | 42 | 10404 | 1 | 15 | $7.17\times10^{-8}$ | 2.20 |
| Total | 42 | | 83 | 892 | | 9.31 |

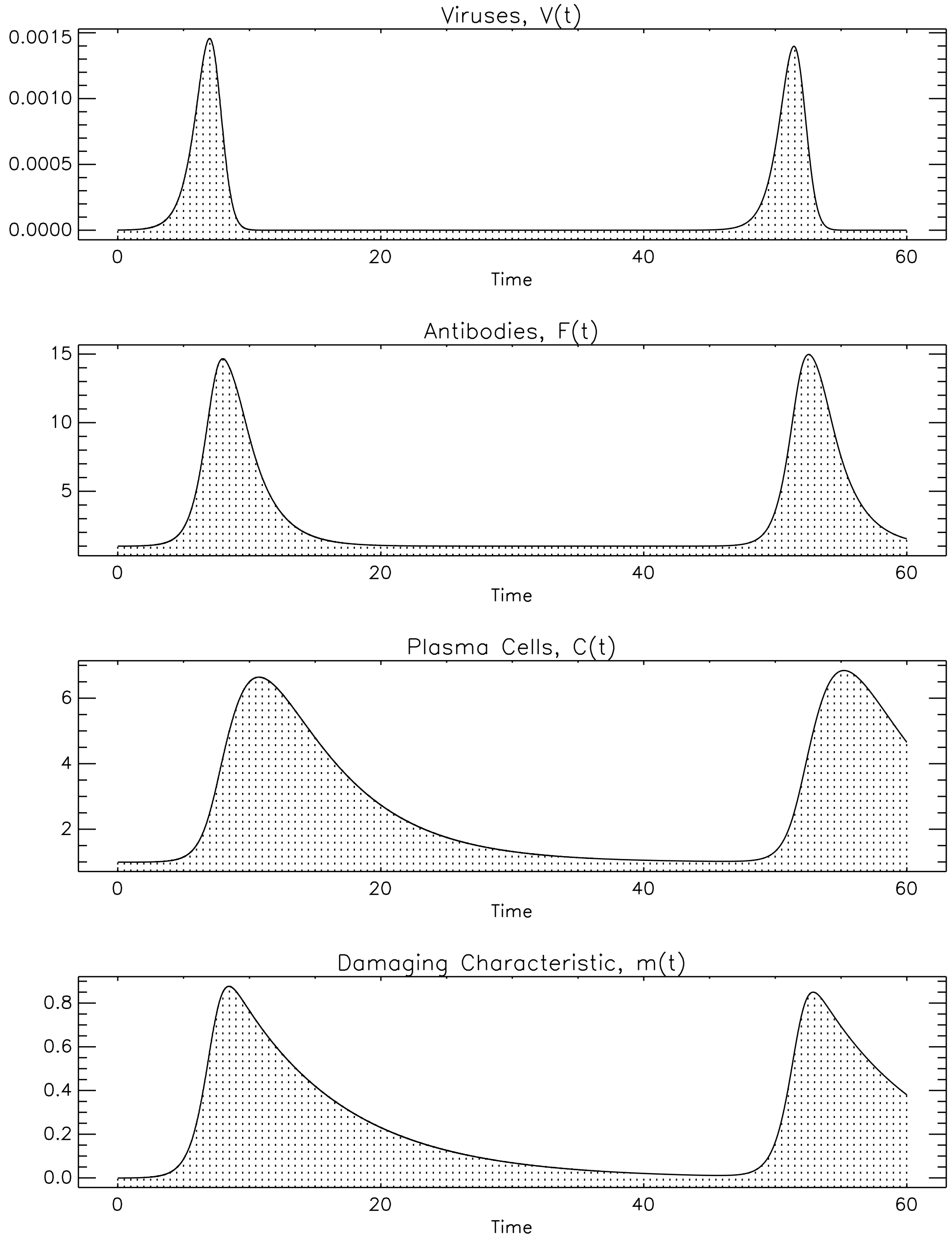


**Figure 7.19.** *Marchuk delay equation solution.*

**Example 7.4** FINITE HORIZON OPTIMAL CONTROL. Deshmukh, Ma, and Butcher [72] present an example they describe as follows:

> The mathematical models of certain engineering processes and systems are represented by delay differential equations with time periodic coefficients. Such processes and systems include the machine tool dynamics in metal cutting operations such as milling and turning with periodically varying cutting speed or impedance and parametric control of robots, etc. Delay differential equations have been used to model nonlinear systems where finite delay in feedback control can have adverse effects on closed loop stability.

They propose a linear time-periodic delay differential system

$$\mathbf{x}'(\alpha) = \mathbf{A}_1(\alpha)\mathbf{x}(\alpha) + \mathbf{A}_2(\alpha)\mathbf{x}(\alpha - \tau) + \mathbf{B}(\alpha)\mathbf{v}(\alpha), \tag{7.172}$$

$$\mathbf{x}(\alpha) = \boldsymbol{\phi}(\alpha) \qquad \text{for } -\tau \le \alpha \le 0, \tag{7.173}$$

where $\mathbf{x}(\alpha)$ is the $n$-dimensional state vector, and $\mathbf{v}(\alpha)$ is the $m$-dimensional control vector. Differentiation with respect to time $\alpha$ is denoted by $\mathbf{x}' = d\mathbf{x}/d\alpha$. The matrices $\mathbf{A}_1(\alpha) = \mathbf{A}_1(\alpha - T)$ and $\mathbf{A}_2(\alpha) = \mathbf{A}_2(\alpha - T)$ are $n \times n$ periodic matrices with period $T$, and the startup vector function $\boldsymbol{\phi}(\alpha)$ is defined on the interval $[-\tau, 0]$. The $n \times m$ matrix $\mathbf{B}(\alpha) = \mathbf{B}(\alpha - T)$ is also periodic. In [72] the authors also assume a single fixed delay equal to the natural period of the parametric excitations $\tau = T > 0$. The objective is to minimize the quadratic

$$J = \frac{1}{2}\mathbf{x}^{\mathsf{T}}(\alpha_F)\mathbf{S}\mathbf{x}(\alpha_F) + \frac{1}{2}\int_0^{\alpha_F} \left[\mathbf{x}^{\mathsf{T}}(\alpha)\mathbf{Q}(\alpha)\mathbf{x}(\alpha) + \mathbf{v}^{\mathsf{T}}(\alpha)\mathbf{R}(\alpha)\mathbf{v}(\alpha)\right] d\alpha, \tag{7.174}$$

over the *finite horizon* $0 \le \alpha \le \alpha_F$. $\mathbf{S}$ and $\mathbf{Q}(\alpha)$ are $n \times n$ symmetric positive semidefinite matrices, $\mathbf{R}(\alpha)$ is $m \times m$ symmetric positive definite, and both $\mathbf{Q}(\alpha)$ and $\mathbf{R}(\alpha)$ are periodic with period $T$.

To illustrate their approach the authors in [72] address an example with two delay intervals that describes a controlled delay Mathieu equation

$$\begin{aligned}\begin{bmatrix} x_1'(\alpha) \\ x_2'(\alpha) \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -4\pi^2(a + c\cos 2\pi\alpha) & 0 \end{bmatrix} \begin{bmatrix} x_1(\alpha) \\ x_2(\alpha) \end{bmatrix} \\ &\quad + \begin{bmatrix} 0 & 0 \\ 4\pi^2 b\cos 2\pi\alpha & 0 \end{bmatrix} \begin{bmatrix} x_1(\alpha - 1) \\ x_2(\alpha - 1) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v(\alpha)\end{aligned} \tag{7.175}$$

with startup function

$$\begin{bmatrix} x_1(\alpha) \\ x_2(\alpha) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \text{for } -1 \le \alpha \le 0 \tag{7.176}$$

for $k = 1, \ldots, N$. The system parameters are given by the values $a = 0.2$, $b = 0.5$, and $c = 0.2$ and the uncontrolled system is unstable. The delay interval and period are both one, $\tau = T = 1$. The goal is to drive the final state to zero, which is reflected by the objective

$$J = \frac{10^4}{2}\mathbf{x}^{\mathsf{T}}(\alpha_F)\mathbf{x}(\alpha_F) + \int_0^{\alpha_F} \left[\mathbf{x}^{\mathsf{T}}(\alpha)\mathbf{x}(\alpha) + v^2(\alpha)\right] d\alpha, \tag{7.177}$$

where the final time is $\alpha_F = N\tau = N$. Two cases will be considered, one with two delay intervals $N = 2$ for comparison with the original authors, and a second, more realistic case with $N = 50$ intervals.

To apply the method of steps we introduce the expression $\alpha = t + (k-1)\tau$ that relates the original time $\alpha$ to the time on a delay interval $t$ with $0 \le t \le \tau$. Then let us define the expanded state vector

$$\mathbf{y}(t) = \begin{bmatrix} \mathbf{y}_1(t) \\ \mathbf{y}_2(t) \\ \vdots \\ \mathbf{y}_N(t) \end{bmatrix} \tag{7.178}$$

with a corresponding definition for the expanded control $\mathbf{u}(t)$. Departing from our standard notational convention the components of $\mathbf{y}$ and $\mathbf{u}$ are defined by

$$\mathbf{y}_k(t) = \mathbf{x}(\alpha) \qquad \text{for } (k-1)\tau \le \alpha \le k\tau, \tag{7.179a}$$

$$\mathbf{u}_k(t) = \mathbf{v}(\alpha) \qquad \text{for } (k-1)\tau \le \alpha \le k\tau, \tag{7.179b}$$

where the subscript $k = 1, \dots, N$ corresponds to a particular delay interval. Applying this transformation to (7.172) leads to the system of $nN$ ODEs

$$\dot{\mathbf{y}}_k = \mathbf{A}_1(\alpha)\mathbf{y}_k + \mathbf{A}_2(\alpha)\mathbf{y}_{k-1} + \mathbf{B}(\alpha)\mathbf{u}_k \tag{7.180}$$

for $k = 1, \dots, N$ defined on the domain $0 \le t \le \tau$, with $\alpha = t + (k-1)\tau$. To ensure consistency with the original DDE we impose the additional boundary conditions

$$\mathbf{y}_k(0) = \mathbf{y}_{k-1}(\tau), \tag{7.181a}$$

$$\mathbf{u}_k(0) = \mathbf{u}_{k-1}(\tau). \tag{7.181b}$$

The startup conditions are given by

$$\mathbf{y}_0(t) = \boldsymbol{\phi}(t) \qquad \text{for } -\tau \le t \le 0 \tag{7.182}$$

and the objective function (7.174) is recast as

$$J = \frac{1}{2}\mathbf{y}_N^{\mathsf{T}}(\tau)\mathbf{S}\mathbf{y}_N(\tau) + \frac{1}{2}\int_0^{\tau} \sum_{k=1}^{N} \left[\mathbf{y}_k^{\mathsf{T}}\mathbf{Q}(\alpha)\mathbf{y}_k + \mathbf{u}_k^{\mathsf{T}}\mathbf{R}(\alpha)\mathbf{u}_k\right] dt. \tag{7.183}$$

Thus the original DDE control problem has be transformed to a standard optimal control problem defined by the ODE system (7.180), with boundary conditions (7.181a), (7.181b), and (7.182), and objective (7.183). As such, the $\mathbb{SOCS}$ algorithm is directly applicable. Furthermore, observe that when the linear ODE system (7.180) is discretized, the resulting finite-dimensional NLP constraints are *linear* functions of the NLP variables. Also, after discretization the objective function is a *quadratic* function of the NLP variables. Since the boundary conditions are also linear functions of the NLP variables the resulting finite-dimensional NLP problem is just a QP problem (cf. Section 1.10). As such, the QP subproblem can be solved in one step, regardless of the initial guess. To exploit this computational efficiency we set the NLP algorithm option to M (cf. Section 2.6.2), thereby omitting an unnecessary step to locate a feasible point. Second, the gradient, Jacobian, and Hessian information needed to define the QP subproblem is normally computed using sparse central differences. However, since the truncation error for all linear and quadratic functions is zero, we have $|f'''(x)| = 0$ in (1.165). Thus to minimize the finite difference error given by (1.165) a "large" perturbation size should be used. For this case we choose $\delta = 1$. Furthermore, with limited experimentation one can simply pick a fixed discretization mesh size, thereby avoiding mesh refinement and the underlying solution of many NLP subproblems. In summary we choose to discretize the problem with 100 equally spaced grid points, using the HSC method. As an initial guess for all of the NLP variables we simply use zero, with the exception of the boundary values (7.176). Figure 7.20 displays the solutions obtained for both cases. The solution with two delay intervals is shown in the top plot, and the other three plots contain the time history for the state and control over 50 delay intervals. For the latter case the optimal objective value is $F^* = 4.5677520 \times 10^1$ with discretization error $\epsilon = 6.61 \times 10^{-8}$.

## 7.4 In-Flight Dynamic Optimization of Wing Trailing Edge Surface Positions

**Example 7.5** TRAILING EDGE VARIABLE CAMBER. When designing a commercial aircraft, one major consideration is the aerodynamic efficiency. Ideally, the shape of the aircraft is designed to optimize some measure of aerodynamic performance such as the ratio of lift to drag ($L/D$). However, aerodynamic forces depend on many factors, so for example an airfoil designed to maximize $L/D$ when flying at 32,000 ft and Mach .84 is *not optimal* when the aircraft flies at any other condition (e.g., 35,000 ft and $M = .83$). This deficiency can be addressed by changing the shape of the airfoil to give the best performance at the actual operating condition. One technique that can be implemented on modern commercial aircraft is to vary the camber of the wing using minor changes in some of the control surfaces. An approach that was tested for an aircraft using a single control surface is described by Gilyard [101] and Gilyard, Georgie, and Barnicki [102]. Similar ideas are presented by Martins and Catalano [132].

In spite of its obvious appeal, practical implementation of this *trailing edge variable camber* (TEVC) approach must address a number of nontrivial technical challenges. First, the expected change in drag caused by changing the camber of the wing is very small—typically less than 1% of the base drag. Second, on Boeing commercial aircraft as many as four different control surfaces can be used to modify the camber. Third, the displacement of the control surfaces is small—typically less than 2 deg—and because of mechanical limitations the flap positions can be specified only to within 1/4 degree. Finally, the *base drag* is affected by many random factors, including wind and atmospheric affects, engine performance, vehicle weight, etc.

This example describes a rather straightforward camber estimation process developed in collaboration with Paul H. Carpenter.[10] After reaching the operating flight condition (typically cruise altitude and velocity), the camber control surfaces are "swept" through a predetermined time profile. During this period, which typically lasts from 3 to 5 min, dynamic behavior is observed and measurement data is recorded at a high frequency (e.g. 20 samples per second.) Using a high-fidelity model of the dynamics, the trajectory is reconstructed. The solution to this *inverse problem* yields an estimate for the time history of the aerodynamic coefficients ($C_D$, $C_L$, and $C_y$). From the time history of the aerodynamic coefficients, we then construct a parametric representation for the drag coefficient

[10]Aero Performance Tools and Methods, The Boeing Company, P.O. Box 3707, MS 67-FE, Seattle, WA 98124-2207.

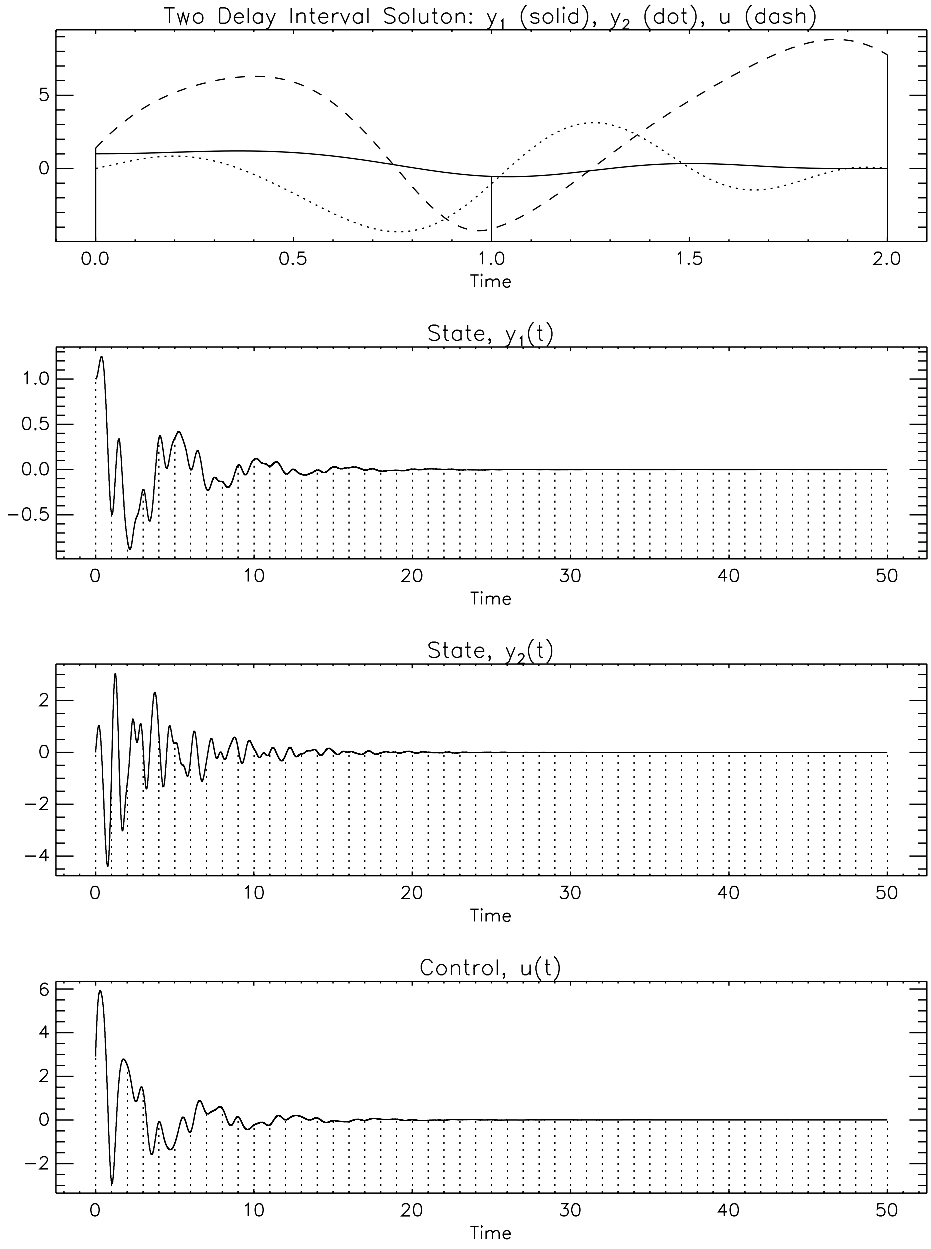


**Figure 7.20.** *Finite horizon optimal control delay equation solutions.*

as a function of the camber control variables. This drag model is constructed by solving a scattered data least squares approximation using a constrained multivariate B-spline representation. As the final step, we compute the camber control variables that minimize the drag model.

## 7.4.1 Aircraft Dynamics for Drag Estimation

In general, the dynamic behavior of an aircraft can be described by a set of differential equations. Using a six degree of freedom model as a starting point a number of changes can be introduced in order to construct a dynamic model that is appropriate for the estimation process. The dynamics can be formulated as the following *differential-algebraic system*:

$$\dot{z} = -u\sin\theta + v\sin\phi\cos\theta + w\cos\theta\cos\phi, \tag{7.184}$$

$$\dot{u} = vr - wq - g\sin\theta + \frac{F_x}{m}, \tag{7.185}$$

$$\dot{v} = wp - ur + g\sin\phi\cos\theta + \frac{F_y}{m}, \tag{7.186}$$

$$\dot{w} = uq - vp + g\cos\phi\cos\theta + \frac{F_z}{m}, \tag{7.187}$$

$$\dot{\phi} = p + (q\sin\phi + r\cos\phi)\tan\theta, \tag{7.188}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi, \tag{7.189}$$

$$\dot{\psi} = (q\sin\phi + r\cos\phi)\sec\theta, \tag{7.190}$$

$$\dot{W} = -\mu(h, M, T_g), \tag{7.191}$$

$$0 = n_x - \left[\frac{F_x}{m} + \Delta n_x\right], \tag{7.192}$$

$$0 = n_y - \left[\frac{F_y}{m} + \Delta n_y\right], \tag{7.193}$$

$$0 = n_z - \left[\frac{F_z}{m} + \Delta n_z\right], \tag{7.194}$$

$$0 = V_g - \sqrt{u^2 + v^2 + w^2 - \dot{z}^2}, \tag{7.195}$$

$$0 = \delta - \arctan\left(\frac{v\cos\phi - w\sin\phi}{u\cos\theta + v\sin\phi\sin\theta + w\cos\phi\sin\theta}\right). \tag{7.196}$$

Observe that the time varying roll, pitch, and yaw rates $p(t)$, $q(t)$, and $r(t)$ are treated as algebraic variables rather than differential variables; i.e., they are *algebraic states*. The approximation is reasonable since the rates are very small at a cruise condition, and this eliminates the need to estimate the moments $L(t)$, $M(t)$, and $N(t)$.

The total forces that appear in (7.185)–(7.187) are expressed in the body axes as the sum of aerodynamic and thrust terms, i.e.,

$$F_x = (-C_D\cos\alpha + C_L\sin\alpha)\bar{q}S + \frac{1}{\kappa}T_g - D_r\cos\beta\cos\alpha, \tag{7.197}$$

$$F_y = C_Y\bar{q}S - D_r\sin\beta, \tag{7.198}$$

$$F_z = (-C_L\cos\alpha - C_D\sin\alpha)\bar{q}S - \frac{1}{\kappa}\tan\eta_{xz}T_g - D_r\cos\beta\sin\alpha. \tag{7.199}$$

Assuming the left and right engines have equal gross thrust $T_g$ and net thrust $T_n$, and $\eta_{xy}$ and $\eta_{xz}$ are constants that define the wing incidence angles for a particular engine, the ram drag, lift, and drag are given by

$$D_r = T_g - T_n, \qquad \kappa = \sqrt{\tan^2 \eta_{xz} + \tan^2 \eta_{xy} + 1,}$$
$$L = C_L \bar{q} S, \qquad D = C_D \bar{q} S.$$

The lift and drag forces are defined in terms of the drag coefficient $C_D$, the lift coefficient $C_L$, the reference area $S$, and the dynamic pressure $\bar{q}$. The vehicle weight $W$ is related to the mass $m$ by the expression $W = mg$.

As the aircraft moves the aerodynamic forces change as a function of time, in particular the total pressure and static pressure. Using values for $P_{total}(t)$, and $P_{static}(t)$ it is possible to construct auxiliary quantities. Specifically the following equations can be evaluated in sequence:

$$M = \sqrt{5}\left[\left(\frac{P_{total}}{P_{static}}\right)^{\frac{1}{3.5}} - 1\right]^{\frac{1}{2}}, \tag{7.200}$$

$$V_e = (14.3791496702540) M \sqrt{P_{static}}, \tag{7.201}$$

$$\bar{q} = \frac{V_e^2}{295.3714}. \tag{7.202}$$

These expressions define the Mach number $M(t)$, the equivalent airspeed $V_e(t)$ (knots), and the dynamic pressure $\bar{q}(t)$ (psf). The *ground speed* is given by

$$V_g = \sqrt{u^2 + v^2 + w^2 - \dot{z}^2} \tag{7.203}$$

and the *drift angle* is given by

$$\delta = \arctan\left(\frac{v\cos\phi - w\sin\phi}{u\cos\theta + v\sin\phi\sin\theta + w\cos\phi\sin\theta}\right). \tag{7.204}$$

These two consistency conditions appear as algebraic equations (7.195) and (7.196). A similar ambiguity in the decomposition of the total forces (7.197)–(7.199) is resolved by using information about the total sensed accelerations

$$n_x = \left[\frac{F_x}{m} + \Delta n_x\right], \tag{7.205}$$

$$n_y = \left[\frac{F_y}{m} + \Delta n_y\right], \tag{7.206}$$

$$n_z = \left[\frac{F_z}{m} + \Delta n_z\right], \tag{7.207}$$

where the constants $\Delta n_x$, $\Delta n_y$, and $\Delta n_z$ represent observation instrumentation biases. These expressions also appear as the algebraic equations (7.192)–(7.194). A summary of the nomenclature is presented in Table 7.15.

**Table 7.15.** *Nomenclature.*

| | | | |
|---|---|---|---|
| $\alpha$ | Angle of Attack (deg) | $P_{total}$ | Total Pressure (psf) |
| $\beta$ | Sideslip (Yaw) Angle (deg) | $P_{static}$ | Static Pressure (psf) |
| $\delta$ | Drift Angle (deg) | $q$ | Pitch Rate $Y$-Stability Axis (rad/sec) |
| $\mu$ | Total Fuel Flow (lb/sec) | $r$ | Yaw Rate $Z$-Stability Axis (rad/sec) |
| $\phi$ | Euler Roll (Bank) Angle (deg) | $T_g$ | Total Gross Thrust (lb) |
| $\psi$ | Euler Yaw Angle (deg) | $T_n$ | Total Net Thrust (lb) |
| $\theta$ | Euler Pitch Angle (deg) | $u$ | $X$-Velocity (fps) |
| $C_D$ | Reference Drag Coefficient | $u_1$ | Aileron Deflection (deg) |
| $C_L$ | Reference Lift Coefficient | $u_2$ | Flaperon Deflection (deg) |
| $C_y$ | Reference Side Force Coefficient | $u_3$ | Inboard Flap Deflection (deg) |
| $n_x$ | Acceleration $X$-Direction (g) | $v$ | $Y$-Velocity (fps) |
| $n_y$ | Acceleration $Y$-Direction (g) | $V_g$ | Ground Speed (knots) |
| $n_z$ | Acceleration $Z$-Direction (g) | $w$ | $Z$-Velocity (fps) |
| $p$ | Roll Rate $X$-Stability Axis (rad/sec) | $z$ | $Z$-Position (ft) |

The motion is described by 8 differential variables $(z,u,v,w,\phi,\theta,\psi,W)$ and 18 algebraic variables

$$(T_g,T_n,\mu,C_L,C_D,C_y,p,q,r,n_x,n_y,n_z,\alpha,\beta,V_g,\delta,P_{total},P_{static}).$$

In addition, the three observation biases $\Delta n_x$, $\Delta n_y$, and $\Delta n_z$ must be determined. The total set of 26 dynamic variables and 3 parameters must satisfy the 8 differential equations (7.184)–(7.191) and the 5 algebraic equations (7.192)–(7.196).

## 7.4.2 Step 1: Reference Trajectory Estimation

The goal of the first step is to compute time histories for the complete set of dynamic variables such that the trajectory dynamics defined by the DAEs (7.184)–(7.196) are satisfied. Denote the dynamic variables by the vector $\mathbf{z}(t)$ with corresponding observations by the vector $\widehat{\mathbf{z}}(t)$. These quantities must be chosen to minimize the error at the observation data points, i.e.,

$$F = \sum_{k=1}^{N} \sum_{\substack{j \\ j \subset \mathcal{O}}} \left[ \frac{(\mathbf{z}_j(t_k) - \widehat{\mathbf{z}}_j(t_k))}{\sigma_j} \right]^2. \tag{7.208}$$

For a typical flight test of 300 sec duration, with 20 data points per second, the total number of points $N \sim 6000$. Statistical information about the observation data is included using specified values for the standard deviations $\sigma_j$. Note the summation includes all dynamic variables for which data are available (the set of observations $\mathcal{O}$) but *excludes* the quantities $u(t)$, $v(t)$, $W(t)$, $C_L(t)$, $C_D(t)$, and $C_y(t)$. Thus observation data are available for only 20 of the 26 total dynamic variables. Note that for this step information about the camber control surfaces, i.e., the camber variable data $\widehat{\mathbf{u}}_k$, is *not* used. The solution to this least squares problem is the best estimate for the time histories, which are denoted as $C_L^*(t)$, $C_D^*(t)$, and $C_y^*(t)$. This large-scale parameter estimation problem can be solved using the method described in Chapter 5.

## 7.4.3 Step 2: Aerodynamic Drag Model Approximation

The first step provides estimates for the aerodynamic coefficients as a function of time; however, it is desirable to construct a parametric model for the drag coefficient. The aerodynamic properties of the aircraft can be altered by control surfaces on the wing. For the Boeing 777 aircraft, two control surfaces, namely the aileron and flaperon, can be utilized. For the 787 aircraft, the inboard flap can also be used in addition to aileron and flaperon. Thus we propose constructing the tensor product B-spline model

$$\tilde{C}_D(u_1,u_2) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} a_{ij} B_i(u_1) B_j(u_2) \tag{7.209a}$$

for aircraft with two control surfaces and

$$\tilde{C}_D(u_1,u_2,u_3) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\sum_{k=1}^{n_3} a_{ijk} B_i(u_1) B_j(u_2) B_k(u_3) \tag{7.209b}$$

for aircraft with three control surfaces. To ensure that the model $\tilde{C}_D$ has a minimum it must satisfy the constraints

$$\frac{\partial^2 \tilde{C}_D}{\partial u_j^2} \geq 0 \tag{7.210}$$

with respect to the camber control surfaces $u_j$. The number of coefficients $a$ in model (7.209a) or (7.209b) is dictated by two factors: the order of the B-spline representation, and the number of internal knots.

In order to compute the coefficients $a_{ij}$ and thereby define the aerodynamic model (7.209a) or (7.209b) we utilize the results of the reference trajectory estimation, i.e., Step 1. Specifically we choose the coefficients $a_{ij}$ to minimize

$$F = \sum_{k=1}^{N} \left\{ C_D^*(t_k) - \tilde{C}_D\left[\hat{u}_1(t_k), \hat{u}_2(t_k)\right] \right\}^2 \tag{7.211a}$$

for aircraft with two control surfaces and

$$F = \sum_{k=1}^{N} \left\{ C_D^*(t_k) - \tilde{C}_D\left[\hat{u}_1(t_k), \hat{u}_2(t_k), \hat{u}_3(t_k)\right] \right\}^2 \tag{7.211b}$$

for aircraft with three control surfaces and satisfy the curvature constraints (7.210). Note that the objective function (7.211a) or (7.211b) involves the quantity $C_D^*(t_k)$, which is available from the reference trajectory reconstruction in Step 1. Since the variation in drag is attributed to the changes in the camber variables, this information is incorporated into the model using the data $\hat{u}_j(t_k)$. This constrained scattered data fit problem can be solved using the sparse NLP techniques introduced in Chapter 2.

## 7.4.4 Step 3: Optimal Camber Prediction

The final step in the process is to determine the best performance over an extended period of time at cruise using a constant setting for the camber variables. This can be achieved if we determine the camber to minimize drag. By construction the camber variables appear *only* in the quantity $\tilde{C}_D(\mathbf{u})$, so the optimal values with respect to $\mathbf{u}$ can be computed just by minimizing

$$F = \tilde{C}_D(u_1, u_2) \tag{7.212a}$$

for aircraft with two control surfaces and

$$F = \tilde{C}_D(u_1, u_2, u_3) \tag{7.212b}$$

for aircraft with three control surfaces, at fixed values of $\bar{C}_L$ and $\bar{M}$. As a practical matter we impose simple bounds on this two or three variable optimization problem. Let us denote the minimum value as

$$C_D^{\circledast} = \tilde{C}_D(u_1^*, u_2^*). \tag{7.213}$$

## 7.4.5 Numerical Results

### 777-200ER Flight Test

The approach described has been implemented using flight test data for a particular Boeing 777-200ER airplane. To illustrate the method a series of representative results are plotted in Figures 7.21–7.24. Figures 7.21–7.23 share a common format. The left column of each figure displays the time history for a dynamic variable, as well as the measured (observation) for the same quantity. The right column presents the error between the data and the reconstructed variable. So, for example, the first row in Figure 7.21 illustrates the variation in the horizontal velocity during the test. The right column plots the error between the measured velocity and the reconstructed value. For this example the error in velocity varies between $\pm.1$ (fps). The average error in the reconstructed velocity over the entire test .00020674815 is displayed in the title. Figure 7.23 displays the quantities $C_L^*(t) - C_L^{\circledast}$ and $C_D^*(t) - C_D^{\circledast}$, which are the estimated time histories for the aerodynamic coefficients relative to their respective optimal values. Using the reconstructed aerodynamic coefficient data, the tensor product spline model (7.209a) is computed. Figure 7.24 displays the percent change relative to the minimum, i.e., the quantity

$$\Delta C_D = 100 \times \frac{(\tilde{C}_D(u_1, u_2) - C_D^{\circledast})}{C_D^{\circledast}}.$$

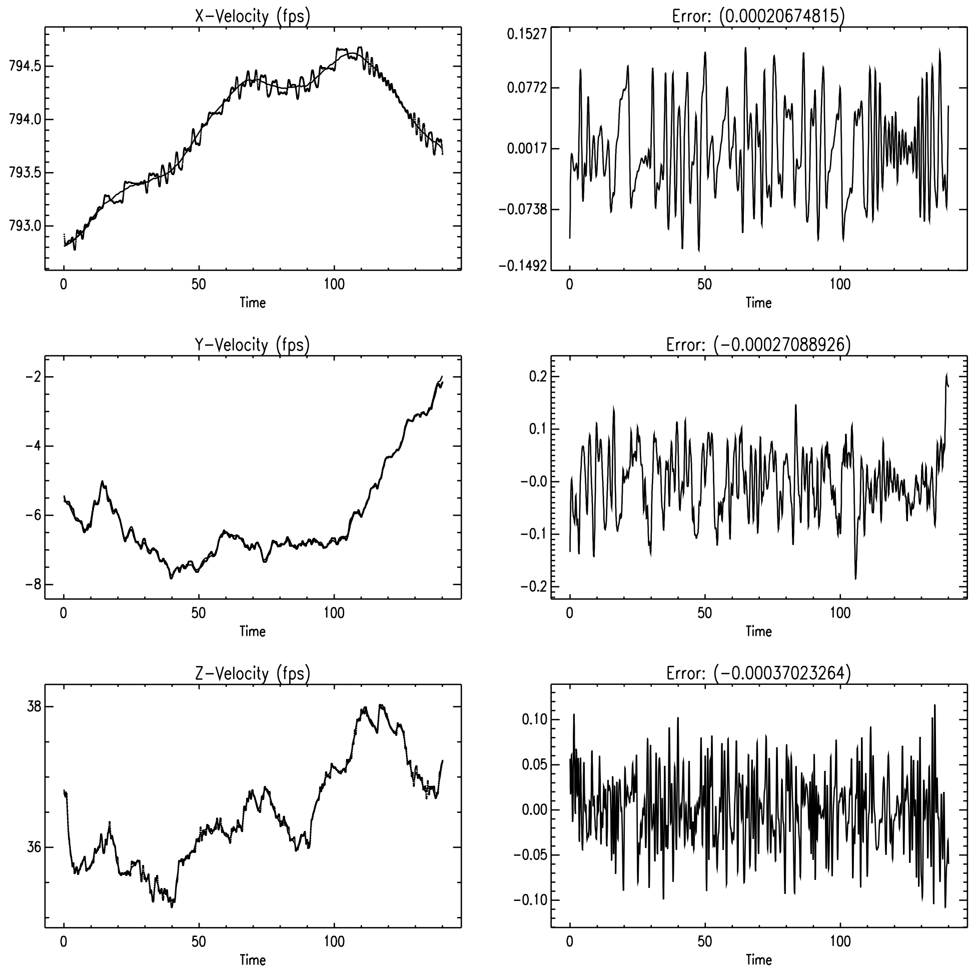


**Figure 7.21.**

## Performance Comparison

Step 1 requires the solution of a large-scale parameter estimation problem, and Figure 7.25 illustrates how the algorithm proceeds. The overall solution required 10 iterations. The first three iterations locate a feasible point; that is, the defect constraints are solved. An initial guess for the optimization variables $\mathbf{x}$ (4.105) is constructed by linearly interpolating the observation data. In Figure 7.25 we see that the initial constraint error $\|\mathbf{c}(\mathbf{x}^{(1)})\|_2 = .137$, and after two iterations it is reduced to $\|\mathbf{c}(\mathbf{x}^{(3)})\|_2 = 8.5 \times 10^{-9}$. The next six iterations are directed toward minimizing the objective function while also satisfying the constraints. At the beginning of this portion of the algorithm, the constraints are satisfied $\|\mathbf{c}(\mathbf{x}^{(4)})\|_\infty = 2.3 \times 10^{-9}$, and they are also satisfied at the solution $\|\mathbf{c}(\mathbf{x}^{(10)})\|_\infty = 7.1 \times 10^{-15}$. During the optimization process the objective function is reduced from $F(\mathbf{x}^{(4)}) = 2.92 \times 10^{-5}$ to $F(\mathbf{x}^{(10)}) = 8.1 \times 10^{-7}$, and the error in the projected gradient is reduced from $9.9 \times 10^{-5}$

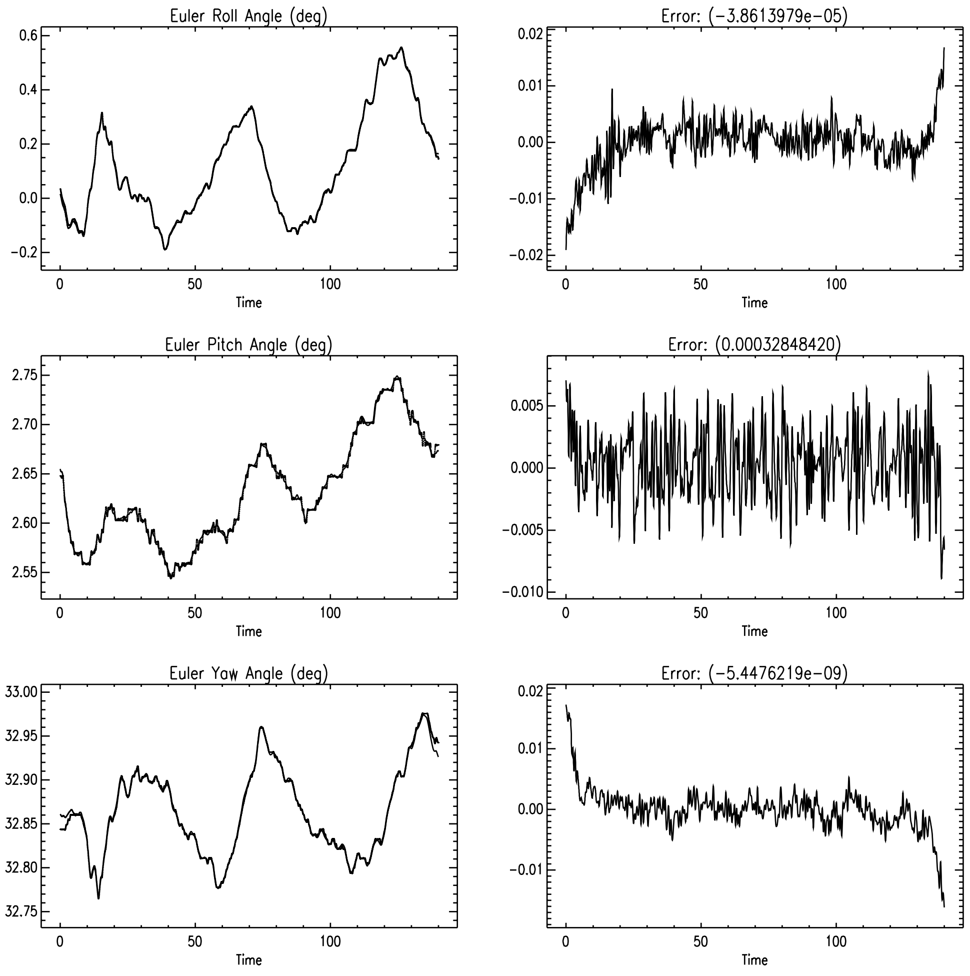


**Figure 7.22.**

to $5.1 \times 10^{-16}$. However, note that constraint feasibility is not maintained during the optimization steps, and at the seventh iteration $\|\mathbf{c}(\mathbf{x}^{(7)})\|_\infty = 5.9 \times 10^{-5}$. Although the values quoted have all been scaled to improve numerical conditioning, it is important to note that the optimal value of the objective function is *nonzero*. In fact, at the solution the five largest (unscaled) residuals are, respectively, $5.29, -4.32, 4.25, -4.21, 3.91$. Because the objective is nonzero and the residuals are nonlinear, a traditional Gauss–Newton method would exhibit linear convergence. In contrast, the $\mathbb{SOPE}$ algorithm demonstrates quadratic convergence, which is achieved because the full Hessian matrix (2.56) is utilized. This behavior is illustrated in Figure 7.25 by the dramatic reduction in the projected gradient and constraint error during the final iterations.

In order to fully appreciate the algorithmic behavior it is worthwhile to look a little deeper into the problem characteristics which are summarized in Table 7.16. At each iteration the optimization algorithm must solve a large system of linear equations, the KKT

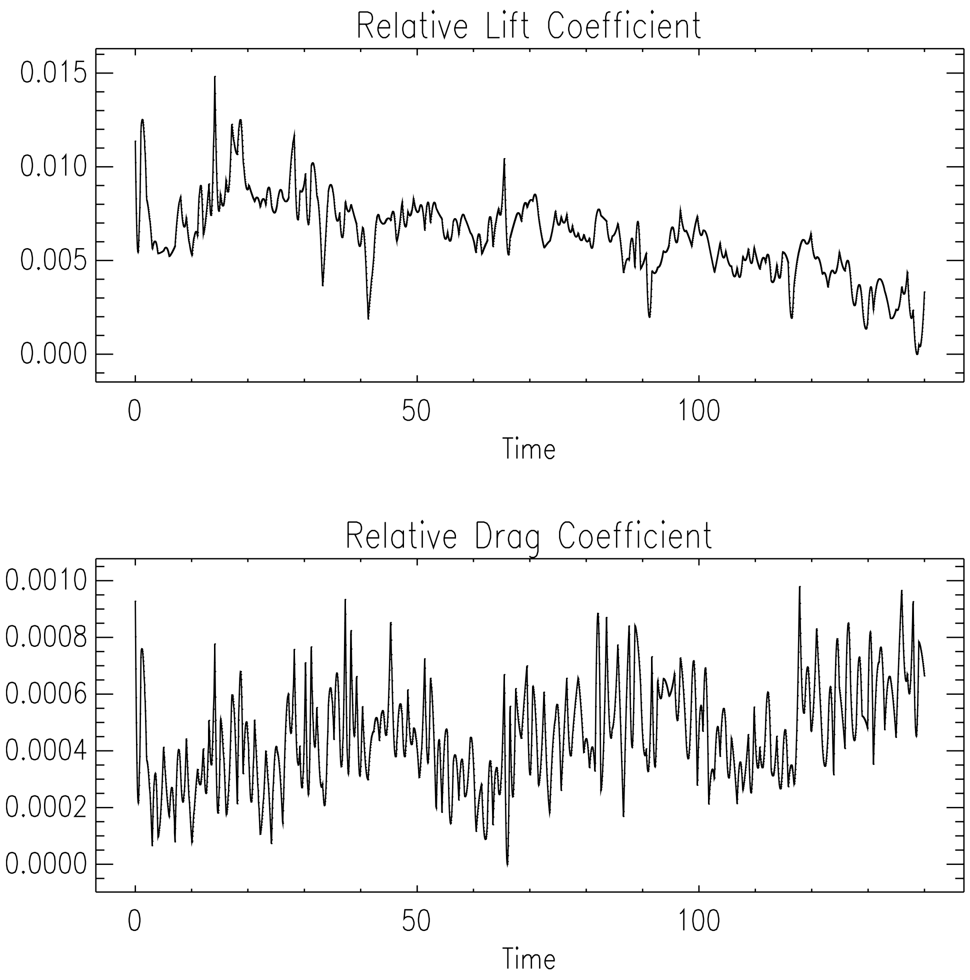


**Figure 7.23.**

system (7.215). For this example there are 176050 equations. Because there are 18735 variables, the matrix **V** is 18735 $\times$ 18735. Since there are 147290 residuals the matrix **R** is 147290 $\times$ 18735. The total number of defect constraints is 10025, and consequently the matrix **G** is 10025 $\times$ 18735. Observe that of the 18735 variables, 10025 are determined by constraints, and one is fixed by the initial condition on airplane weight $W(0) = w_0$, leaving 8709 degrees of freedom available to minimize the objective function (7.208). Even though the KKT system is large, it is also *sparse*. It is imperative that sparsity is exploited in order to efficiently solve the problem. This is achieved in two ways.

First, we must compute the matrices **V**, **R**, and **G**, and this is done using finite difference approximations. Now a standard finite difference technique would require at least 18735 perturbations just to compute a forward approximation, which is clearly prohibitive. However, by exploiting the sparse difference technique described in Section 5.4 it is possible to compute all first and second derivative information with just 170 perturbations!

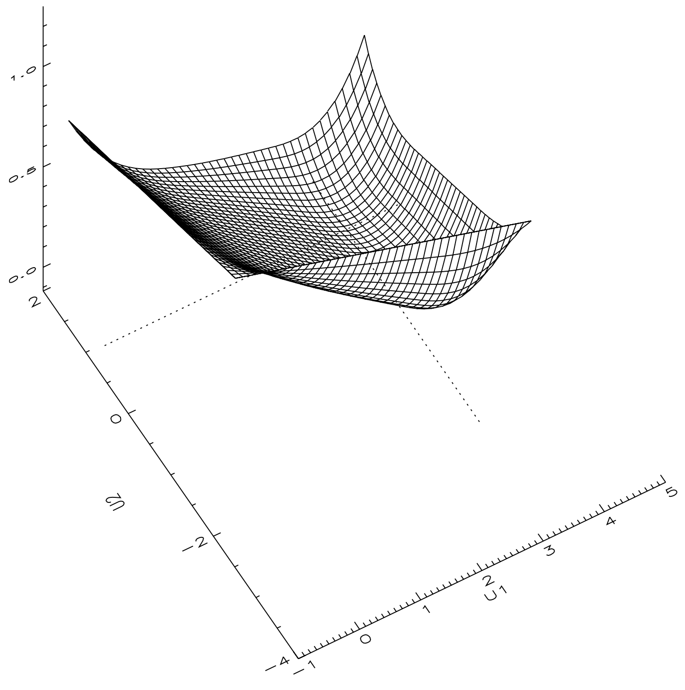


**Figure 7.24.** *Aerodynamic model surface change.*

This is possible because the total number of index sets for this problem is just 17 and is determined by the right-hand-side sparsity template given by

$$
\mathcal{T} = \text{struct}\left[\begin{array}{c|c|c} \dfrac{\partial \mathbf{f}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{u}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{p}} \\ \dfrac{\partial \mathbf{g}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{u}} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{p}} \\ \dfrac{\partial \mathbf{w}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{w}}{\partial \mathbf{u}} & \dfrac{\partial \mathbf{w}}{\partial \mathbf{p}} \end{array}\right] = \left[\begin{array}{cccccccccccccccccccccccccccc}
\times&\times&\times&\times&\times&&&&&&&&&&&&&&&&&&&&&&& \\
&\times&\times&&\times&&\times&\times&\times&&\times&\times&&&\times&\times&&&&\times&\times&&&\times&\times&&& \\
\times&&\times&\times&\times&&\times&\times&\times&&&&\times&\times&&\times&&&&&\times&&&\times&\times&&& \\
\times&\times&&\times&\times&&\times&\times&\times&&\times&\times&&\times&\times&&&&&\times&\times&&&\times&\times&&& \\
&&&\times&\times&&&&&&&&&\times&\times&\times&&&&&&&&&&&& \\
&&&\times&&&&&&&&&&&\times&\times&&&&&&&&&&&& \\
&&&\times&\times&&&&&&&&&&\times&\times&&&&&&&&&&&& \\
&&&&&&&&&\times&&&&&&&&&&&&&&&&&& \\
&&&&&&\times&\times&\times&&\times&\times&&&&&\times&&&\times&\times&&&\times&\times&\times&& \\
&&&&&&\times&\times&\times&&&&\times&&&&&\times&&&\times&&&\times&\times&&\times& \\
&&&&&&\times&\times&\times&&\times&\times&&&&&&&\times&\times&\times&&&\times&\times&&&\times \\
\times&\times&\times&\times&\times&&&&&&&&&&&&&&&&&\times&&&&&& \\
\times&\times&\times&\times&\times&&&&&&&&&&&&&&&&&&\times&&&&&
\end{array}\right]. \tag{7.214}
$$

The combined benefit of sparse finite differences in conjunction with quadratic convergence means that the entire problem is solved using only 707 evaluations, or 612901 evaluations of the right-hand-side dynamic functions given in (7.184)–(7.196).

The second computational benefit accrues from the matrix sparsity itself. It is well known that the computational complexity for solving a system of *n dense* linear equations is $\mathcal{O}(n^3)$. In contrast for a sparse linear system the cost is $\mathcal{O}(\kappa n)$, where $\kappa$ is a factor related to sparsity. By using a symmetric indefinite decomposition of the underlying KKT matrix, as described in Section 2.10 the time required to solve the linear equations on all 10

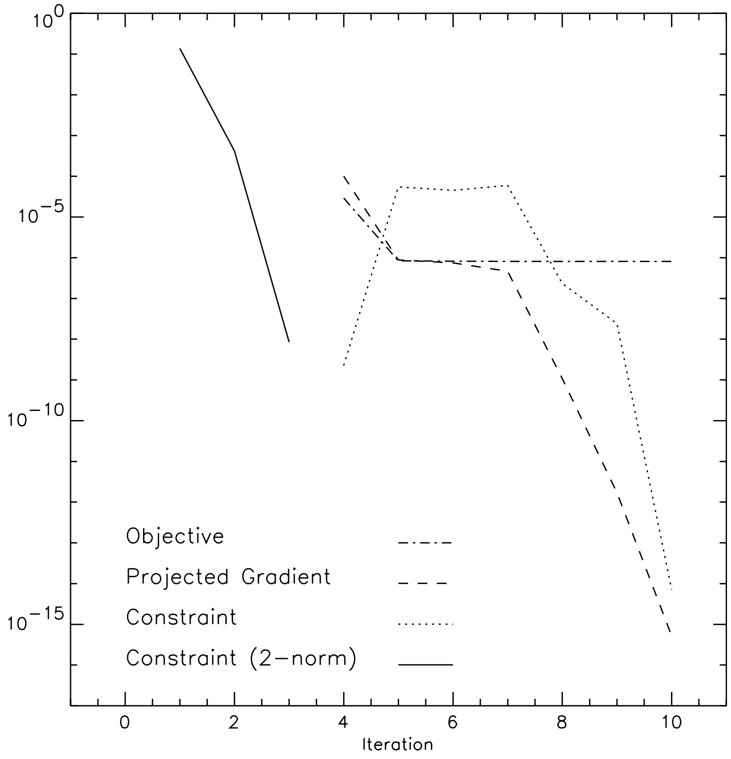


**Figure 7.25.** *Iteration history.*

**Table 7.16.** *Step* 1 *computational performance characteristics.*

| | |
|---|---|
| Number of KKT Equations | 176050 |
| Number of Residuals | 147290 |
| Number of Variables | 18735 |
| Number of Constraints | 10025 |
| Number of Degrees of Freedom | 8709 |
| Number of Index Sets | 17 |
| Number of Function Evaluations per Jacobian | 34 |
| Number of Function Evaluations per Jacobian/Hessian | 170 |
| Total Number of Function Evaluations | 707 |
| Total Number of Right-Hand-Side Evaluations | 612901 |
| Number of Iterations | 10 |
| Number of Hessian Evaluations | 3 |
| Total Time Inside NLP (sec) | 28.37 |
| Total CPU Time (sec) | 30.11 |

iterations was only 28.37 sec, and the entire solution was obtained in just 30.11 sec. This would not be possible without exploiting matrix sparsity.

In order to solve the large-scale least squares problems encountered in Step 1 of the process one must face the computational issues associated with the normal matrix $\mathbf{R}^T\mathbf{R}$. It is well known that simply forming the matrix is ill-conditioned, and accurate solution of linear systems involving the normal matrix can be problematic. This ill-conditioning is avoided by using a symmetric indefinite decomposition of the underlying KKT matrix, as described in Section 2.10. Thus a new estimate for the variables (4.105) is given by $\bar{\mathbf{x}} = \mathbf{x} + \mathbf{p}$, which is the solution to the QP subproblem (2.61)–(2.62). This approach requires solving the linear system

$$\begin{bmatrix} \mathbf{V} & \mathbf{R}^\mathsf{T} & \mathbf{G}^\mathsf{T} \\ \mathbf{R} & -\mathbf{I} & \mathbf{0} \\ \mathbf{G} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ -\mathbf{w} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{R}^\mathsf{T}\mathbf{r} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \tag{7.215}$$

where $\mathbf{G}$ is the Jacobian matrix of the defect constraints. In contrast when the normal matrix is formed explicitly, the search direction can be computed by solving the linear system

$$\begin{bmatrix} \mathbf{H}_L & \mathbf{G}^\mathsf{T} \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{R}^\mathsf{T}\mathbf{r} \\ \mathbf{0} \end{bmatrix}. \tag{7.216}$$

The optimization search direction $\mathbf{p}$ can be computed by solving the KKT system written in either the *sparse tableau* format (7.215) or in the *normal matrix* format (7.216). The sparse tableau format is well-conditioned, whereas the normal matrix format is ill-conditioned. On the other hand the sparse tableau format is much larger than the normal matrix format. To assess the relative merits of each approach for this example a series of cases were solved using various options. Three different discretization stepsizes were used, namely $h = 1, 2, 4$ seconds. Two different (fourth order) discretization techniques, specifically HSC and HSS, were tested. In each case both the sparse tableau and the normal matrix formats were tried. The results are summarized in Table 7.17. Three of the

**Table 7.17.** *Algorithm parameter options.*

| Case | $h$[a] | Disc | KKT Option | $N$[b] | CPU[c] |
|---|---|---|---|---|---|
| 1 | 1 | C[d] | N | 47154 | 334.780 |
| 2 | 1 | C | T | 351434 | F |
| 3 | 1 | S[e] | N[f] | 59314 | 594.830 |
| 4 | 1 | S | T[g] | 363594 | F |
| 5 | 2 | C | N | 23532 | 287.780 |
| 6 | 2 | C | T | 327812 | 102.540 |
| 7 | 2 | S | N | 29596 | 277.700 |
| 8 | 2 | S | T | 314139 | 70.5900 |
| 9 | 4 | C | N | 11752 | F |
| 10 | 4 | C | T | 316032 | 85.4500 |
| 11 | 4 | S | N | 14776 | 436.830 |
| 12 | 4 | S | T | 319056 | 62.9000 |

[a]Discretization stepsize
[b]Dimension of KKT matrix
[c]CPU seconds
[d]HSC discretization
[e]HSS discretization
[f]Normal matrix KKT option (7.216)
[g]Sparse tableau KKT option (7.215)

cases failed to converge either because of storage or iteration limits. Cases 6, 8, 10, and 12 suggest that the sparse tableau format is more efficient provided sufficient storage is available. When storage is limited, the normal matrix approach can be utilized; however, there is a significant CPU time penalty which can be attributed to the ill-conditioning of this alternative. The default settings for production usage employ the sparse tableau format with the separated Hermite–Simpson discretization.

Constructing the parametric representation for the drag coefficient as described in Section 7.4.3 also entails solving a sparse constrained optimization problem just as in Step 1. Specifically we must compute the coefficients $a_{ij}$ that define the spline model (7.209a). In this case it is possible to compute the necessary derivatives analytically; however, the same sparse optimization algorithm is used.

Chapter 8

# Epilogue

Solving a problem in optimal control or estimation is not easy. Pieces of the puzzle are found scattered throughout many different disciplines. At the very least, one needs an "optimization" method and a "differential equation" method. A rudimentary understanding of modern control theory is helpful, to say nothing of expertise in the domain of application. In my experience, the most challenging practical problems originate with experts in a particular domain. An aerodynamicist feels comfortable discussing "lift" and "drag" but is less familiar with index-two DAEs. Chemical engineers can readily describe a batch feed process but become uneasy when discussing the KKT conditions. Often this domain expertise has been painstakingly developed over many years and involves complex computer simulations and/or expensive experimental studies.

Thus, when faced with an optimal control or estimation problem it is tempting for the domain expert (i.e., the structural engineer, biochemist, etc.) 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. Furthermore design and development of the "simulation" or experimental trial should be done with the intent to optimize. Optimization is not an afterthought. Indeed, perhaps the most important issue needed to successfully solve a problem is the proper formulation. And so I close with the following:

THEERUM

If there is a flaw in the problem formulation, the optimization algorithm will find it.[11]

□

[11]The proof is left to the student.

# Appendix

# Software

All of the algorithms described in the book have been implemented and tested on realistic test problems. In addition, all of the examples presented have been solved using this software, and FORTRAN implementations of the examples can be obtained by contacting the author. The $\mathbb{SOCS}$ [38] library, which contains all of the software, has a great deal of functionality and it is not intended that this appendix should be viewed as a user's manual. On the other hand, it is worthwhile to outline the basic capability of the tool to assist the reader in the formulation and solution of practical problems. To that end, the following sections present a brief overview of the library. Additional information can be obtained by contacting the author at `john.betts@comcast.net`.

## A.1 Simplified Usage Dense NLP

The subroutine HDNLPD provides the most basic functionality for solving small, dense NLP problems. The primary information that must be supplied by the user (outlined with a double box in Figure A.1) is a subroutine called FUNBOX that evaluates the objective and constraint functions. This is the *function generator* described in Section 3.8. All algorithm parameters are given default values, which are appropriate for a simple problem. If desired, the default values can be redefined using a utility routine HHSNLP that is common to all software in the $\mathbb{SOCS}$ library. The simplified usage software uses a *forward-communication* format and, as such, the optimization algorithm HDNLPD calls the user-supplied subroutine, which has a fixed calling argument list. For more sophisticated applications, the *reverse-communication* subroutine HDNLPR can be used. HDNLPR is appropriate for small, dense applications when the user can supply gradient and Hessian information. It is also appropriate for use with complicated simulation programs and when parallel processing is used.

## A.2 Sparse NLP with Sparse Finite Differences

Large, sparse NLP problems necessarily demand more information from the user because of the need to specify matrix sparsity and provide corresponding derivative information.

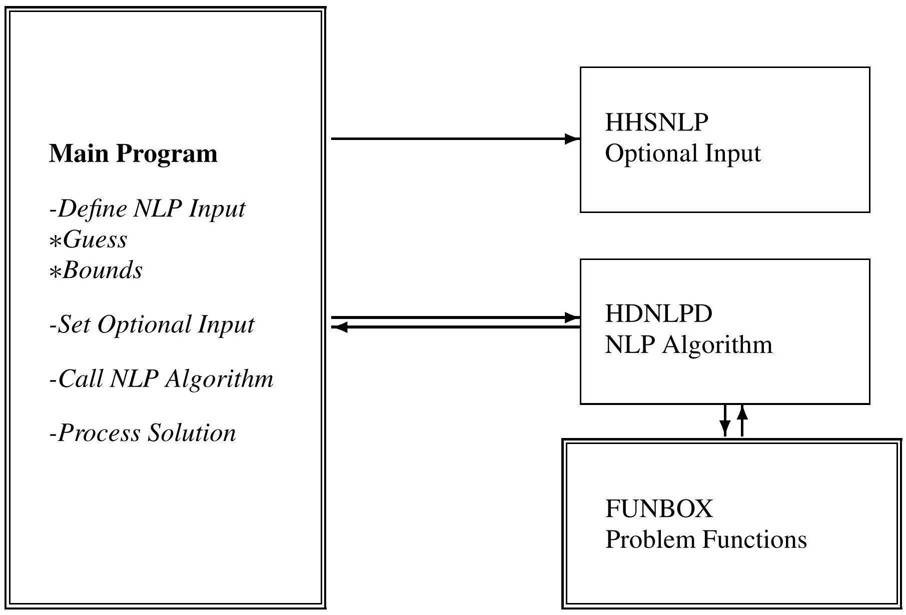


**Figure A.1.** *Dense NLP software.*

Figure A.2 illustrates the usage of the sparse NLP algorithm HDSNLP, which employs a reverse-communication format. Again, the user-supplied software is shown inside a double box. In addition, software for constructing sparse finite difference first and second derivatives HDSFDJ and HDSFDH is also available. A utility procedure (HJSFDI) can be used for constructing sparse difference index sets based on the user-supplied matrix sparsity patterns. Optional input can be set using the standard $\mathbb{SOCS}$ utility HHSNLP. Similar functionality is available when solving large, sparse nonlinear least squares problems using subroutine HDSLSQ rather than HDSNLP.

## A.3 Optimal Control Using Sparse NLP

Implementing the solution of an optimal control problem using the direct transcription method in $\mathbb{SOCS}$ requires some software supplied by the user. Figure A.3 illustrates the software organization of an application. As before, user-supplied procedures are shown with double boxes. However, many of the user-supplied routines are optional, as indicated by an asterisk in the illustration. The user must call the $\mathbb{SOCS}$ algorithm HDSOCS. The user must define the problem via the subroutine ODEINP. All other information is optional and can be supplied either by the user or by using dummy routines from the $\mathbb{SOCS}$ library instead. Typically, the user will specify the right-hand sides of the DAEs and quadrature functions using subroutine ODERHS. For applications with nonlinear boundary conditions, the point function routine ODEPTF must be supplied. If special output (e.g., graphics files) is desired, the user can supply a special-purpose ODEPRT routine. The default initial guess

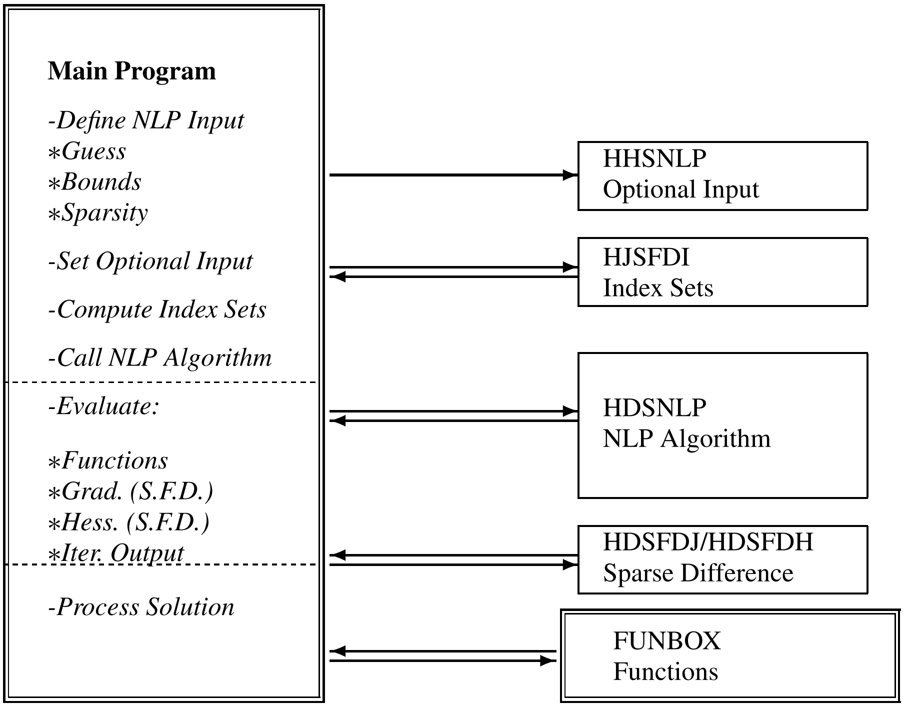


**Figure A.2.** *Sparse NLP software.*

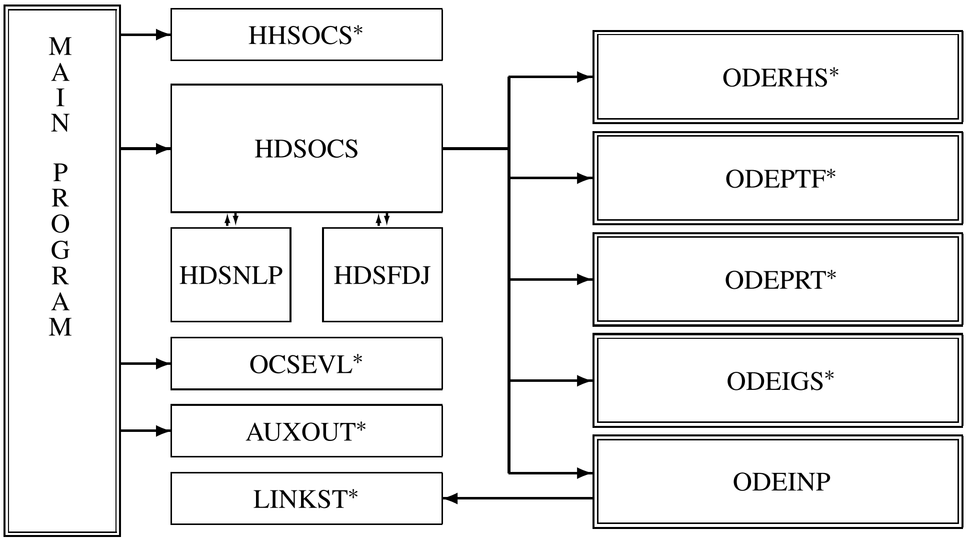


**Figure A.3.** *Sparse optimal control software.*

for the state and control variables is a linear function between the phase boundaries. When other initialization procedures are used, subroutine ODEIGS provides this functionality. When solving an optimal control problem, *it is not necessary* to define the matrix sparsity, to compute derivatives, or to call the sparse NLP algorithm. However, nonstandard NLP options can be set using the HHSNLP utility. Nonstandard optimal control algorithm options can be set using the HHSOCS utility. For example, the discretization accuracy and number of mesh-refinement iterations can be set by HHSOCS. A feasible (but suboptimal) trajectory can be computed by using the feasibility (F) option as set by HHSNLP. The solution computed by $\mathbb{SOCS}$ is represented using B-spline function(s). The solution can be evaluated at arbitrary points with the utility OCSEVL and/or the auxiliary output procedure AUXOUT. Multiphase formulations may also incorporate the LINKST utility when linking phases together. The overall $\mathbb{SOCS}$ library has been designed with flexibility and functionality in mind and is especially suited for use with complex simulation systems.

# Bibliography

[1] S. M. ALESSANDRINI, *A Motivational Example for the Numerical Solution of Two-Point Boundary-Value Problems*, SIAM Review, 37 (1995), pp. 423–427.

[2] U. M. ASCHER, R. M. M. MATTHEIJ, AND R. D. RUSSELL, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1988.

[3] U. M. ASCHER AND L. R. PETZOLD, *The Numerical Solution of Delay-Differential-Algebraic Equations of Retarded and Neutral Type*, SIAM Journal on Numerical Analysis, 32 (1995), pp. 1635–1657.

[4] C. ASHCRAFT, R. G. GRIMES, AND J. G. LEWIS, *Accurate Symmetric Indefinite Linear Equation Solvers*, SIAM Journal on Matrix Analysis and Applications, 20 (1998), pp. 513–561.

[5] F. BASHFORTH AND J. C. ADAMS, *Theories of Capillary Action*, Cambridge University Press, New York, 1883.

[6] R. H. BATTIN, *An Introduction to the Mathematics and Methods of Astrodynamics*, AIAA Education Series, American Institute of Aeronautics and Astronautics, Inc., 1633 Broadway, New York, NY 10019, 1987.

[7] B. BAUMRUCKER, J. RENFRO, AND L. T. BIEGLER, *MPEC Problem Formulations and Solution Strategies with Chemical Engineering Applications*, Computers & Chemical Engineering, 32 (2008), pp. 2903–2913.

[8] D. A. BENSON, *A Gauss Pseudospectral Transcription for Optimal Control*, PhD thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 2004.

[9] P. BERKMANN AND H. J. PESCH, *Abort Landing in Windshear: Optimal Control Problem with Third-Order State Constraint and Varied Switching Structure*, Journal of Optimization Theory and Applications, 85 (1995), pp. 21–57.

[10] J. T. BETTS, *An Improved Penalty Function Method for Solving Constrained Parameter Optimization Problems*, Journal of Optimization Theory and Applications, 16 (1975), pp. 1–24.

[11] ——, *Solving the Nonlinear Least Square Problem: Application of a General Method*, Journal of Optimization Theory and Applications, 18 (1976), pp. 469–483.

[12] ——, *Optimal Three-Burn Orbit Transfer*, AIAA Journal, 15 (1977), pp. 861–864.

[13] ——, *Using Sparse Nonlinear Programming to Compute Low Thrust Orbit Transfers*, The Journal of the Astronautical Sciences, 41 (1993), pp. 349–371.

[14] ——, *The Application of Sparse Least Squares in Aerospace Design Problems*, in Optimal Design and Control, Proceedings of the Workshop on Optimal Design and Control, J. Borggaard, J. Burkardt, M. Gunzburger, and J. Peterson, eds., vol. 19 of Progress in Systems and Control Theory, Birkhäuser, Basel, 1994, pp. 81–96.

[15] ——, *Optimal Interplanetary Orbit Transfers by Direct Transcription*, The Journal of the Astronautical Sciences, 42 (1994), pp. 247–268.

[16] ——, *The Application of Optimization Techniques to Aerospace Systems*, in Fourth International Conference on Foundations of Computer-Aided Process Design, Proceedings of the Conference at Snowmass, Colorado, July 10–14, 1994, L. T. Biegler and M. F. Doherty, eds., CACHE, American Institute of Chemical Engineers, New York, 1995.

[17] ——, *Experience with a Sparse Nonlinear Programming Algorithm*, in Large-Scale Optimization with Applications, L. T. Biegler, T. F. Coleman, A. R. Conn, and F. N. Santosa, eds., vol. 93 of The IMA Volumes in Mathematics and its Applications, Springer-Verlag, New York, 1997, pp. 53–72.

[18] ——, *Survey of Numerical Methods for Trajectory Optimization*, AIAA Journal of Guidance, Control, and Dynamics, 21 (1998), pp. 193–207.

[19] ——, *Very Low Thrust Trajectory Optimization*, in High Performance Scientific and Engineering Computing, Proceedings of the International FORTWIHR Conference on HPSEC, Munich, March 16–18, 1998, H.-J. Bungartz, F. Durst, and C. Zenger, eds., Springer-Verlag, Berlin, Heidelberg, 1999.

[20] ——, *Very Low Thrust Trajectory Optimization Using a Direct SQP Method*, Journal of Computational and Applied Mathematics, 120 (2000), pp. 27–40.

[21] ——, *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, Philadelphia, PA, 2001.

[22] ——, *Optimal Lunar Swingby Trajectories*, The Journal of the Astronautical Sciences, 55 (2007), pp. 349–371.

[23] J. T. Betts, N. Biehn, and S. L. Campbell, *Convergence of Nonconvergent IRK Discretizations of Optimal Control Problems with State Inequality Constraints*, SIAM Journal on Scientific Computing, 23 (2002), pp. 1981–2007.

[24] J. T. Betts, N. Biehn, S. L. Campbell, and W. P. Huffman, *Exploiting Order Variation in Mesh Refinement for Direct Transcription Methods*, Tech. Report M&CT-TECH-99-022, Mathematics and Computing Technology, The Boeing Company, P.O. Box 3707, Seattle, WA 98124-2207, Oct. 1999.

[25] ——, *Compensating for Order Variation in Mesh Refinement for Direct Transcription Methods*, Journal of Computational and Applied Mathematics, 125 (2000), pp. 147–158.

[26] J. T. BETTS AND S. L. CAMPBELL, *Discretize then Optimize*, in Mathematics for Industry: Challenges and Frontiers, D. R. Ferguson and T. J. Peters, eds., SIAM Proceedings Series, SIAM, Philadelphia, PA, 2005, pp. 140–157.

[27] J. T. BETTS, S. L. CAMPBELL, AND A. ENGELSONE, *Direct Transcription Solution of Inequality Constrained Optimal Control Problems*, in Proceedings, 2004 American Control Conference, Boston, MA, June 2004, pp. 1622–1626.

[28] ——, *Direct Transcription Solution of Optimal Control Problems with Higher Order State Constraints: Theory vs Practice*, Optimization and Engineering, 8 (2007), pp. 1–19.

[29] J. T. BETTS, M. J. CARTER, AND W. P. HUFFMAN, *Software for Nonlinear Optimization*, Mathematics and Engineering Analysis Library Report MEA-LR-83 R1, Boeing Information and Support Services, The Boeing Company, P.O. Box 3707, Seattle, WA 98124-2207, June 1997.

[30] J. T. BETTS, S. K. ELDERSVELD, AND W. P. HUFFMAN, *A Performance Comparison of Nonlinear Programming Algorithms for Large Sparse Problems*, in Proceedings of the AIAA Guidance, Navigation, and Control Conference, AIAA-93-3751-CP, Monterey, CA, Aug. 1993, pp. 443–455.

[31] J. T. BETTS AND S. O. ERB, *Optimal Low Thrust Trajectories to the Moon*, SIAM Journal on Applied Dynamical Systems, 2 (2003), pp. 144–170. `http://www.siam.org/journals/siads/2-2/40908.html`.

[32] J. T. BETTS AND P. D. FRANK, *A Sparse Nonlinear Optimization Algorithm*, Journal of Optimization Theory and Applications, 82 (1994), pp. 519–541.

[33] J. T. BETTS AND J. M. GABLONSKY, *A Comparison of Interior Point and SQP Methods on Optimal Control Problems*, Technical Document Series MCT-TECH-02-004, Mathematics and Engineering Analysis, The Boeing Company, P.O. Box 3707, Seattle, WA 98124-2207, Mar. 2002. `http://www.boeing.com/phantom/socs/benchmark/benchmark.pdf`.

[34] J. T. BETTS AND W. P. HUFFMAN, *Trajectory Optimization on a Parallel Processor*, AIAA Journal of Guidance, Control, and Dynamics, 14 (1991), pp. 431–439.

[35] ——, *Application of Sparse Nonlinear Programming to Trajectory Optimization*, AIAA Journal of Guidance, Control, and Dynamics, 15 (1992), pp. 198–206.

[36] ——, *Path Constrained Trajectory Optimization Using Sparse Sequential Quadratic Programming*, AIAA Journal of Guidance, Control, and Dynamics, 16 (1993), pp. 59–68.

[37] ——, *Sparse Nonlinear Programming Test Problems (Release* 1.0*)*, BCS Technology Technical Report BCSTECH-93-047, Boeing Computer Services, The Boeing Company, P.O. Box 3707, Seattle, WA 98124-2207, 1993.

[38] ——, *Sparse Optimal Control Software* $\mathbb{SOCS}$, Mathematics and Engineering Analysis Technical Document MEA-LR-085, Boeing Information and Support Services, The Boeing Company, P.O. Box 3707, Seattle, WA 98124-2207, July 1997.

[39] ——, *Mesh Refinement in Direct Transcription Methods for Optimal Control*, Optimal Control Applications and Methods, 19 (1998), pp. 1–21.

[40] ——, *Exploiting Sparsity in the Direct Transcription Method for Optimal Control*, Computational Optimization and Applications, 14 (1999), pp. 179–201.

[41] S. A. BHATT, *Optimal Reorientation of Spacecraft Using Only Control Moment Gyroscopes*, Master's thesis, Rice University, Houston, TX, May 2007.

[42] G. A. BLISS, *Lectures on the Calculus of Variations*, University of Chicago Press, Chicago, IL, 1946.

[43] H. G. BOCK, *Numerical Treatment of Inverse Problems in Chemical Reaction Kinetics*, in Modelling of Chemical Reaction Systems, K. H. Ebert, Peter Deuflhard, and W. Jäger, eds., vol. 18 of Springer Series in Chemical Physics, Springer-Verlag, New York, 1981, pp. 102–125.

[44] ——, *Recent Advances in Parameter Identification Techniques for O.D.E.*, in Numerical Treatment of Inverse Problems in Differential and Integral Equations, P. Deuflhard and Ernst Hairer, eds., Birkhäuser Verlag, Heidelberg, 1982, pp. 95–121.

[45] I. BONGARTZ, A. R. CONN, N. I. M. GOULD, AND P. L. TOINT, *CUTE: Constrained and Unconstrained Testing Environment*, ACM Transactions on Mathematical Software, 21 (1995), pp. 123–160.

[46] K. E. BRENAN, *A Smooth Approximation to the GTS* 1962 *Standard Atmosphere Model*, Aerospace Technical Memorandum ATM 82-(2468-04)-7, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, May 1982.

[47] ——, *Engineering Methods Report: The Design and Development of a Consistent Integrator/Interpolator For Optimization Problems*, Aerospace Technical Memorandum ATM 88(9975)-52, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, 1988.

[48] K. E. BRENAN, S. L. CAMPBELL, AND L. R. PETZOLD, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, vol. 14 of Classics in Applied Mathematics, SIAM, Philadelphia, PA, 1995.

[49] R. A. BROUCKE AND P. J. CEFOLA, *On Equinoctial Orbit Elements*, Celestial Mechanics, 5 (1972), pp. 303–310.

[50] C. G. BROYDEN, *A Class of Methods for Solving Nonlinear Simultaneous Equations*, Mathematics of Computation, 19 (1965), pp. 577–593.

[51] ——, *The Convergence of a Class of Double-Rank Minimization Algorithms*, Journal of the Institute of Mathematics and Applications, 6 (1970), pp. 76–90.

[52] A. E. BRYSON, JR., M. N. DENHAM, AND S. E. DREYFUS, *Optimal Programming Problems with Inequality Constraints* I: *Necessary Conditions for Extremal Solutions*, AIAA Journal, 1 (1963), pp. 2544–2550.

[53] A. E. BRYSON, JR., M. N. DESAI, AND W. C. HOFFMAN, *Energy-State Approximation in Performance Optimization of Supersonic Aircraft*, Journal of Aircraft, 6 (1969), pp. 481–488.

[54] A. E. BRYSON, JR. AND Y.-C. HO, *Applied Optimal Control*, John Wiley and Sons, New York, 1975.

[55] R. BULIRSCH, *Die Mehrzielmethode zur numerischen Lösung von nichtlinearen Randwertproblemen und Aufgaben der optimalen Steuerung*, Report of the Carl-Cranz Gesellschaft, Carl-Cranz Gesellschaft, Oberpfaffenhofen, Germany, 1971.

[56] R. BULIRSCH, F. MONTRONE, AND H. J. PESCH, *Abort Landing in the Presence of Windshear as a Minimax Optimal Control Problem, Part* 1: *Necessary Conditions*, Journal of Optimization Theory and Applications, 70 (1991), pp. 1–23.

[57] ——, *Abort Landing in the Presence of Windshear as a Minimax Optimal Control Problem, Part* 2: *Multiple Shooting and Homotopy*, Journal of Optimization Theory and Applications, 70 (1991), pp. 223–253.

[58] R. BULIRSCH, E. NERZ, H. J. PESCH, AND O. VON STRYK, *Combining Direct and Indirect Methods in Optimal Control: Range Maximization of a Hang Glider*, in Optimal Control, R. Bulirsch, A. Miele, J. Stoer, and K. H. Well, eds., vol. 111 of International Series of Numerical Mathematics, Birkhäuser Verlag, Basel, 1993, pp. 273–288.

[59] R. H. BYRD, M. E. HRIBAR, AND J. NOCEDAL, *An Interior Point Algorithm for Large-Scale Nonlinear Programming*, SIAM Journal on Optimization, 9 (1999), pp. 877–900.

[60] S. L. CAMPBELL, N. BIEHN, L. JAY, AND T. WESTBROOK, *Some Comments on DAE Theory for IRK Methods and Trajectory Optimization*, Journal of Computational and Applied Mathematics, 120 (2000), pp. 109–131.

[61] M. D. CANON, C. D. CULLUM, AND E. POLAK, *Theory of Optimal Control and Mathematical Programming*, McGraw–Hill, New York, 1970.

[62] M. CARACOTSIOS AND W. E. STEWART, *Sensitivity Analysis of Initial Value Problems with Mixed ODE's and Algebraic Equations*, Computers & Chemical Engineering, 9 (1985), pp. 359–365.

[63] T. F. COLEMAN, B. S. GARBOW, AND J. J. MORÉ, *Software for Estimating Sparse Hessian Matrices*, ACM Transactions on Mathematical Software, 11 (1985), pp. 363–377.

[64] T. F. COLEMAN AND J. J. MORÉ, *Estimation of Sparse Jacobian Matrices and Graph Coloring Problems*, SIAM Journal on Numerical Analysis, 20 (1983), pp. 187–209.

[65] A. R. CURTIS, M. J. D. POWELL, AND J. K. REID, *On the Estimation of Sparse Jacobian Matrices*, Journal of the Institute of Mathematics and Applications, 13 (1974), pp. 117–120.

[66] G. DAHLQUIST AND AKE BJÖRK, *Numerical Methods*, Prentice–Hall, Englewood Cliffs, NJ, 1974.

[67] G. B. DANTZIG, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[68] W. C. DAVIDON, *Variable Metric Methods For Minimization*, A. E. C. Res. and Develop. Report ANL-5900, Argonne National Laboratory, Argonne, IL, 1959.

[69] C. DE BOOR, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[70] J. E. DENNIS, JR., D. M. GAY, AND R. E. WELSCH, *An Adaptive Nonlinear Least-Squares Algorithm*, ACM Transactions on Mathematical Software, 7 (1981), pp. 348–368.

[71] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1983.

[72] V. DESHMUKH, H. MA, AND E. BUTCHER, *Optimal Control of Parametrically Excited Linear Delay Differential Systems via Chebyshev Polynomials*, in Proceedings of the American Control Conference, Denver, CO, June 2003.

[73] E. D. DICKMANNS, *Maximum Range Three-Dimensional Lifting Planetary Entry*, Tech. Report TR R-387, National Aeronautics and Space Administration, Washington, D.C., 1972.

[74] T. N. EDELBAUM, L. L. SACKETT, AND H. L. MALCHOW, *Optimal Low Thrust Geocentric Transfer*, in AIAA 10th Electric Propulsion Conference, AIAA 73-1074, Lake Tahoe, NV, Oct.–Nov. 1973.

[75] A. ENGELSONE AND S. L. CAMPBELL, *Adjoint Estimation Using Direct Transcription Multipliers: Compressed Trapezoidal Method*, Optimization and Engineering, 9 (2008), pp. 291–305.

[76] P. J. ENRIGHT AND B. A. CONWAY, *Optimal Finite-Thrust Spacecraft Trajectories Using Collocation and Nonlinear Programming*, AIAA Journal of Guidance, Control, and Dynamics, 14 (1991), pp. 981–985.

[77] ——, *Discrete Approximations to Optimal Trajectories Using Direct Transcription and Nonlinear Programming*, AIAA Journal of Guidance, Control, and Dynamics, 15 (1992), pp. 994–1002.

[78] P. R. ESCOBAL, *Methods of Orbit Determination*, Second Edition, Robert E. Krieger Publishing Company, Malabar, FL, 1985.

[79] A. V. FIACCO AND G. P. MCCORMICK, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons, New York, 1968.

[80] A. FLEMING, P. SEKHAVAT, AND I. M. ROSS, *Minimum-Time Reorientation of an Asymmetric Rigid Body*, in AIAA Guidance, Navigation and Control Conference, AIAA 2008-7012, Honolulu, HI, Aug. 2008.

[81] R. FLETCHER, *A New Approach to Variable Metric Algorithms*, Computer Journal, 13 (1970), pp. 317–322.

[82] ———, *Practical Methods of Optimization, Vol.* 2, *Constrained Optimization*, John Wiley and Sons, New York, 1985.

[83] R. FLETCHER, N. I. M. GOULD, S. LEYFFER, AND P. L. TOINT, *Global Convergence of Trust-Region SQP-Filter Algorithms for General Nonlinear Programming*, Tech. Report RAL-TR-1999-041, Rutherford Appleton Laboratory, White Horse, South Oxfordshire, England, 1999.

[84] R. FLETCHER AND S. LEYFFER, *Nonlinear Programming without a Penalty Function*, University of Dundee Numerical Analysis Report NA/171, University of Dundee, Department of Mathematics, Dundee, Scotland, UK, 1997.

[85] R. FLETCHER AND M. J. D. POWELL, *A Rapidly Convergent Descent Method for Minimization*, Computer Journal, 6 (1963), pp. 163–168.

[86] A. FORSGREN, *On Warm Starts for Interior Methods*, in System Modeling and Optimization, IFIP International Federation for Information Processing, F. Ceragioli, A. Dontchev, H. Furuta, K. Marti, and L. Pandolfi, eds., vol. 199, Springer-Verlag, Boston, 2006, pp. 51–66.

[87] A. FORSGREN AND P. E. GILL, *Primal-Dual Interior Methods for Nonconvex Nonlinear Programming*, SIAM Journal on Optimization, 8 (1998), pp. 1132–1152.

[88] D. M. GAY, M. L. OVERTON, AND M. H. WRIGHT, *A Primal-Dual Interior Method for Nonconvex Nonlinear Programming*, in Advances in Nonlinear Programming, Y. Yuan, ed., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998, pp. 31–56.

[89] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1971.

[90] ———, *The Simultaneous Numerical Solution of Differential-Algebraic Equations*, IEEE Transactions on Circuit Theory, CT-18 (1971), pp. 89–95.

[91] C. W. GEAR AND T. V. VU, *Smooth Numerical Solutions of Ordinary Differential Equations*, in Proceedings of the Workshop on Numerical Treatment of Inverse Problems for Differential and Integral Equations, Heidelberg, 1982.

[92] M. GERDTS, *Direct Shooting Method for the Numerical Solution of Higher-Index DAE Optimal Control Problems*, Journal of Optimization Theory and Applications, 117 (2003), pp. 267–294.

[93] P. E. GILL, W. MURRAY, AND M. A. SAUNDERS, *SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization*, Tech. Report SOL 97-3, Department of Operations Research, Stanford University, Stanford, CA, 1997.

[94] ——, *SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization*, SIAM Review, 47 (2005), pp. 99–131.

[95] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *Some Theoretical Properties of an Augmented Lagrangian Merit Function*, Tech. Report SOL 86-6, Department of Operations Research, Stanford University, Stanford, CA, Apr. 1986.

[96] ——, *User's Guide for NPSOL (Version* 4.0*): A Fortran Package for Nonlinear Programming*, Tech. Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, CA, Jan. 1986.

[97] ——, *A Schur-Complement Method for Sparse Quadratic Programming*, Tech. Report SOL 87-12, Department of Operations Research, Stanford University, Stanford, CA, Oct. 1987.

[98] ——, *A Schur-Complement Method for Sparse Quadratic Programming*, in Reliable Numerical Computation, M. G. Cox and S. Hammarling, eds., Oxford University Press, Oxford, UK, 1990, pp. 113–138.

[99] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London, 1981.

[100] ——, *Numerical Linear Algebra and Optimization*, Addison–Wesley Publishing Company, Redwood City, CA, 1991.

[101] G. B. Gilyard, *Development of a Real-Time Transport Performance Optimization Methodology*, Tech. Report NASA TM-4730, NASA Dryden Flight Research Center, Edwards, CA, Jan. 1996.

[102] G. B. Gilyard, J. Georgie, and J. S. Barnicki, *Flight Test of an Adaptive Configuration Optimization System for Transport Aircraft*, Tech. Report NASA/TM-1999-206569, NASA Dryden Flight Research Center, Edwards, CA, Jan. 1999.

[103] D. Goldfarb, *A Family of Variable Metric Methods Derived by Variational Means*, Mathematics of Computation, 24 (1970), pp. 23–26.

[104] N. I. M. Gould, *On Practical Conditions for the Existence and Uniqueness of Solutions to the General Equality Quadratic Programming Problem*, Mathematical Programming, 32 (1985), pp. 90–99.

[105] W. W. Hager, *Rates of Convergence for Discrete Approximations to Unconstrained Optimal Control Problems*, SIAM Journal on Numerical Analysis, 13 (1976), pp. 449–472.

[106] E. Hairer, S. P. Norsett, and G. Wanner, *Solving Ordinary Differential Equations.* I. *Nonstiff Problems*, Springer-Verlag, New York, 1993.

[107] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations.* II. *Stiff and Differential-Algebraic Problems*, Springer-Verlag, New York, 1996.

[108] S. M. HAMMES, *Optimization Test Problems*, Aerospace Technical Memorandum ATM 89(4464-06)-12, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, 1989.

[109] C. R. HARGRAVES AND S. W. PARIS, *Direct Trajectory Optimization Using Nonlinear Programming and Collocation*, AIAA Journal of Guidance, Control, and Dynamics, 10 (1987), pp. 338–342.

[110] M. T. HEATH, *Numerical Methods for Large Sparse Linear Least Squares Problems*, SIAM Journal on Scientific and Statistical Computing, 5 (1984), pp. 497–513.

[111] M. HEINKENSCHLOSS, *Projected Sequential Quadratic Programming Methods*, SIAM Journal on Optimization, 6 (1996), pp. 373–417.

[112] A. L. HERMAN AND B. A. CONWAY, *Direct Optimization Using Collocation Based on High-Order Gauss-Lobatto Quadrature Rules*, AIAA Journal of Guidance, Control, and Dynamics, 19 (1996), pp. 592–599.

[113] A. C. HINDMARSH, *ODEPACK, A Systematized Collection of ODE Solvers*, in Scientific Computing, R. S. Stepleman et al., eds., North–Holland, Amsterdam, 1983, pp. 55–64.

[114] W. HOCK AND K. SCHITTKOWSKI, *Test Examples for Nonlinear Programming Codes*, Springer-Verlag, New York, 1981.

[115] W. P. HUFFMAN, *An Analytic Propagation Method for Low Earth Orbits*, Internal Document, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, Nov. 1981.

[116] P. C. HUGHES, *Spacecraft Attitude Dynamics*, Dover Publications, Inc., Mineola, NY, 2004.

[117] D. H. JACOBSEN, M. M. LELE, AND J. L. SPEYER, *New Necessary Conditions of Optimality for Control Problems with State Variable Inequality Constraints*, Journal of Mathematical Analysis and Applications, 35 (1971), pp. 255–284.

[118] L. JAY, *Convergence of a Class of Runge-Kutta Methods for Differential-Algebraic Systems of Index* 2, BIT, 33 (1993), pp. 137–150.

[119] E. JUNG, S. LENHART, AND Z. FENG, *Optimal Control of Treatments in a Two-Strain Tuberculosis Model*, Discrete and Continuous Dynamical Systems—Series B, 2 (2002), pp. 479–482.

[120] S. K. KAMESWARAN AND L. T. BIEGLER, *A Further Analysis of the Betts and Campbell Heat Problem*, Tech. Report, Chemical Engineering Department, Carnegie Mellon University, Pittsburgh, PA, 2004.

[121] J. A. KECHICHIAN, *Equinoctial Orbit Elements: Application to Optimal Transfer Problems*, in AIAA/AAS Astrodynamics Specialist Conference, AIAA 90-2976, Portland, OR, Aug. 1990.

[122] ——, *Trajectory Optimization with a Modified Set of Equinoctial Orbit Elements*, in AIAA/AAS Astrodynamics Specialist Conference, AAS 91-524, Durango, CO, Aug. 1991.

[123] H. B. KELLER, *Numerical Methods for Two-Point Boundary Value Problems*, Waltham: Blaisdell, London, 1968.

[124] C. T. KELLEY AND E. W. SACHS, *A Pointwise Quasi-Newton Method for Unconstrained Optimal Control Problems*, Numerische Mathematik, 55 (1989), pp. 159–176.

[125] ——, *Pointwise Broyden Methods*, SIAM Journal on Optimization, 3 (1993), pp. 423–441.

[126] H. W. KUHN AND A. W. TUCKER, *Non-linear Programming*, in Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, University of California Press, Berkeley, CA, 1951, pp. 481–493.

[127] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice–Hall, Englewood Cliffs, NJ, 1974.

[128] U. LEDZEWICZ AND H. SCHÄTTLER, *Analysis of Optimal Controls for a Mathematical Model of Tumour Anti-angiogenesis*, Optimal Control Applications and Methods, 29 (2008), pp. 41–57.

[129] T. LEE, D. DUNHAM, S. HSU, AND C. ROBERTS, *Large Inclination Change Using Lunar-Swingby Techniques*, in AIAA/AAS Astrodynamics Conference, AIAA 1988-4290, Minneapolis, MN, Aug. 1988.

[130] D. B. LEINEWEBER, *Efficient Reduced SQP Methods for the Optimization of Chemical Processes Described by Large Sparse DAE Models*, PhD thesis, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Universität Heidelberg, 1998.

[131] K. LEVENBERG, *A Method for the Solution of Certain Problems in Least Squares*, Quarterly of Applied Mathematics, 2 (1944), pp. 164–168.

[132] A. L. MARTINS AND F. M. CATALANO, *Drag Optimization for Transport Aircraft Mission Adaptive Wing*, in 38th Aerospace Sciences Meeting and Exhibit, AIAA-2000-0648, Reno, NV, Jan. 2000.

[133] H. MAURER AND D. AUGUSTIN, *Sensitivity Analysis and Real-Time Control of Parametric Optimal Control Problems Using Boundary Value Methods*, in Online Optimization of Large Scale Systems, M. Grötschel, S. O. Krumke, and J. Rambau, eds., Springer-Verlag, Berlin, 2001, pp. 17–55.

[134] A. MIELE, T. WANG, AND W. W. MELVIN, *Optimal Abort Landing Trajectories in the Presence of Windshear*, Journal of Optimization Theory and Applications, 55 (1987), pp. 165–202.

[135] F. R. MOULTON, *New Methods in Exterior Ballistics*, University of Chicago Press, Chicago, IL, 1926.

[136] W. Murray and M. H. Wright, *Line Search Procedures for the Logarithmic Barrier Function*, SIAM Journal on Optimization, 4 (1994), pp. 229–246.

[137] M. Okamoto and K. Hayashi, *Frequency Conversion Mechanism in Enzymatic Feedback Systems*, Journal of Theoretical Biology, 108 (1984), pp. 529–537.

[138] M. Otter and S. Türk, *The DFVLR Models* 1 *and* 2 *of the Manutec r*3 *Robot*, DFVLR-Mitt. 88-3, Institut für Dynamik der Flugsysteme, Oberpfaffenhoffen, Germany, 1988.

[139] H. J. Pesch, *A Practical Guide to the Solution of Real-Life Optimal Control Problems*, Control and Cybernetics, 23 (1994), pp. 7–60.

[140] L. Petzold, *Differential/Algebraic Equations Are Not ODEs*, SIAM Journal on Scientific and Statistical Computing, 3 (1982), pp. 367–384.

[141] ——, *A Description of DASSL:, A Differential/Algebraic System Solver*, in Scientific Computing, R. S. Stepleman et al., eds., North–Holland, Amsterdam, 1983, pp. 65–68.

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

[143] E. Polak, *Computational Methods in Optimization*, Academic Press, New York, 1971.

[144] ——, *Optimization: Algorithms and Consistent Approximations*, Springer-Verlag, New York, 1997.

[145] L. S. Pontryagin, *The Mathematical Theory of Optimal Processes*, Wiley-Interscience, New York, 1962.

[146] M. J. D. Powell, *Log Barrier Methods for Semi-infinite Programming Calculations*, Tech. Report DAMTP 1992/NA11, University of Cambridge, Department of Applied Mathematics and Theoretical Physics, Silver Street, Cambridge CB3 9EW, England, Dec. 1992.

[147] A. V. Rao, *User's Manual for GPOCS Version* 1.1*, A MATLAB Implementation of the Gauss Pseudospectral Method for Solving Multiple-Phase Optimal Control Problems*, Tech. Report, University of Florida, Gainesville, FL, Aug. 2007.

[148] A. V. Rao and K. D. Mease, *Eigenvector Approximate Dichotomic Basis Method for Solving Hyper-Sensitive Optimal Control Problems*, Optimal Control Applications and Methods, 20 (1999), pp. 59–77.

[149] A. V. Rao, S. Tang, and W. P. Hallman, *Numerical Optimization Study of Multiple-Pass Aeroassisted Orbital Transfer*, Optimal Control Applications and Methods, 23 (2002), pp. 215–238.

[150] D. Redding and J. V. Breakwell, *Optimal Low-Thrust Transfers to Synchronous Orbit*, AIAA Journal of Guidance, Control, and Dynamics, 7 (1984), pp. 148–155.

[151] R. T. ROCKAFELLAR, *The Multiplier Method of Hestenes and Powell Applied to Convex Programming*, Journal of Optimization Theory and Applications, 12 (1973), pp. 555–562.

[152] G. SACHS AND K. LESCH, *Periodic Maximum Range Cruise with Singular Control*, AIAA Journal of Guidance, Control, and Dynamics, 16 (1993), pp. 790–793.

[153] Y. SAKAWA, *Trajectory Planning of a Free-Flying Robot by Using the Optimal Control*, Optimal Control Applications and Methods, 20 (1999), pp. 235–248.

[154] W. SCHIEHLEN, *Multibody Systems Handbook*, Springer-Verlag, Berlin, Heidelberg, 1990.

[155] K. SCHITTKOWSKI, *Parameter Estimation in Dynamic Systems*, in Progress in Optimization, X. Yang, ed., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000, pp. 183–204.

[156] H. SEYWALD AND E. M. CLIFF, *On the Existence of Touch Points for First-Order State Inequality Constraints*, Optimal Control Applications and Methods, 17 (1996), pp. 357–366.

[157] L. F. SHAMPINE AND P. GAHINET, *Delay-Differential-Algebraic Equations in Control Theory*, Applied Numerical Analysis, 56 (2005), pp. 574–588.

[158] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman and Co., San Francisco, CA, 1975.

[159] D. F. SHANNO, *Conditioning of Quasi-Newton Methods for Function Minimization*, Mathematics of Computation, 24 (1970), pp. 647–657.

[160] J. L. SPEYER, *Periodic Optimal Flight*, AIAA Journal of Guidance, Control, and Dynamics, 19 (1996), pp. 745–755.

[161] E. M. STANDISH, *JPL Planetary and Lunar Ephemerides, DE*405/*LE*405, Interoffice Memorandum IOM 312.F - 98 -048, Jet Propulsion Laboratory, Pasadena, CA, Aug. 1998.

[162] D. STEWART AND M. ANITESCU, *Optimal Control of Systems with Discontinuous Differential Equations*, Tech. Report ANL/MCS-P1258-0605, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, June 2005.

[163] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, New York, Berlin, Heidelberg, 1980.

[164] D. TABAK AND B. C. KUO, *Optimal Control by Mathematical Programming*, Prentice–Hall, Englewood Cliffs, NJ, 1971.

[165] H. S. TSIEN AND R. C. EVANS, *Optimum Thrust Programming for a Sounding Rocket*, Journal of the American Rocket Society, 21 (1951), pp. 99–107.

[166] R. J. VANDERBEI AND D. F. SHANNO, *An Interior-Point Method for Nonconvex Nonlinear Programming*, Computational Optimization and Applications, 13 (1999), pp. 231–252.

[167] O. VON STRYK, *Numerical Solution of Optimal Control Problems by Direct Collocation*, in Optimal Control, R. Bulirsch, A. Miele, J. Stoer, and K. H. Well, eds., vol. 111 of International Series of Numerical Mathematics, 1993, Birkhäuser Verlag, Basel, pp. 129–143.

[168] O. VON STRYK AND R. BULIRSCH, *Direct and Indirect Methods for Trajectory Optimization*, Annals of Operations Research, 37 (1992), pp. 357–373.

[169] O. VON STRYK AND M. SCHLEMMER, *Optimal Control of the Industrial Robot Manutec r*3, in Computational Optimal Control, R. Bulirsch and D. Kraft, eds., vol. 115 of International Series of Numerical Mathematics, Birkhäuser Verlag, Basel, 1994, pp. 367–382.

[170] T. V. VU, *Numerical Methods for Smooth Solutions of Ordinary Differential Equations*, Tech. Report UIUCDCS-R-83-1130, Department of Computer Science, University of Illinois, Urbana, IL, May 1983.

[171] M. J. H. WALKER, B. IRELAND, AND J. OWENS, *A Set of Modified Equinoctial Orbit Elements*, Celestial Mechanics, 36 (1985), pp. 409–419.

[172] R. S. WILSON AND K. C. HOWELL, *Trajectory Design in the Sun-Earth-Moon System Using Lunar Gravity Assists*, Journal of Spacecraft and Rockets, 35 (1998), pp. 191–198.

[173] S. J. WRIGHT, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, PA, 1997.

[174] T. YEE AND J. A. KECHICHIAN, *On the Dynamic Modeling in Optimal Low-Thrust Orbit Transfer*, in AAS/AIAA Spaceflight Mechanics Meeting, AAS 92-177, Colorado Springs, CO, Feb. 1992.

[175] K. P. ZONDERVAN, T. P. BAUER, J. T. BETTS, AND W. P. HUFFMAN, *Solving the Optimal Control Problem Using a Nonlinear Programming Technique Part* 3: *Optimal Shuttle Reentry Trajectories*, in Proceedings of the AIAA/AAS Astrodynamics Conference, AIAA-84-2039, Seattle, WA, Aug. 1984.

[176] K. P. ZONDERVAN, L. J. WOOD, AND T. K. CAUGHY, *Optimal Low-Thrust, Three-Burn Transfers with Large Plane Changes*, The Journal of the Astronautical Sciences, 32 (1984), pp. 407–427.

# Index

Entries set in *italic* type indicate an example.