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 $k = 0,\ldots,n_a$, where the coefficients β_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 ℓ-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 θ_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 θ_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 θ_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 $\mathbf{h}_y \doteq (\partial h/\partial y_1, \ldots, \partial h/\partial y_n)$ 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 θ_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 $v = i(k)$. Then the value of the state needed in the residual calculation is

$$\begin{aligned} y_v(\theta_k) &= (1 - 3\delta^2 + 2\delta^3) y_{vj} + (3\delta^2 - 2\delta^3) y_{v,j+1} \\ &\quad + (h_j\delta - 2h_j\delta^2 + h_j\delta^3) f_{vj} + (-h_j\delta^2 + h_j\delta^3) f_{v,j+1}, \end{aligned} \tag{5.21}$$

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_{vj} is the value of state variable v at grid point j and f_{vj} 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

$$u_v(\theta_k) = (1 - \delta)(1 - 2\delta) u_{vj} + 4\delta(1 - \delta)\bar{u}_{v,j+1} - \delta(1 - 2\delta) u_{v,j+1}, \tag{5.22}$$

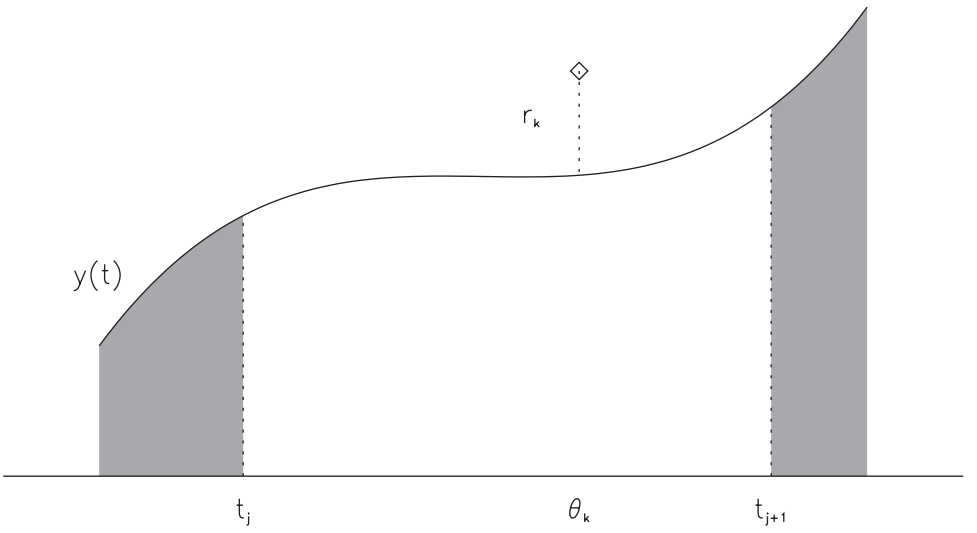

Figure 5.1. *Residual evaluation.*

where $\delta = (\theta_k - t_j)/h_j$. Similarly, when a trapezoidal discretization is used, linear interpolation between the grid points yields

$$u_v(\theta_k) = (1-\delta)u_{vj} + \delta u_{v,j+1}. \tag{5.23}$$

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 $\mathbf{f}$ at the grid points. In particular the quantities h_j are fixed by the mesh-refinement procedure, and the quantities δ 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 τ 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 $\mathbf{A}$ and $\mathbf{B}$ are matrices (constant during the NLP) and $\mathbf{q}$ 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 υ 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 $\mathbf{D}$ defined in Section 2.2.1. The columns of $\mathbf{D}$ 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 $\mathbf{D}$ 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 $\mathbf{D}$ and compute the finite difference index

sets. The first derivative information needed to solve the NLP can then be computed from

$$\begin{bmatrix} \mathbf{G} \\ \mathbf{R} \end{bmatrix} = \mathbf{A} + \mathbf{BD}, \tag{5.26}$$

where $\mathbf{G}$ is the Jacobian of the constraints and $\mathbf{R}$ 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 $\mathbf{R}$ and $\mathbf{V}$ 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 θ_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 $r_k = w_k \left[y_v(\theta_k) - \hat{y}_{vj} \right]$ 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

$$\frac{\partial r_k}{\partial y(t_r)} \neq 0 \quad \text{but} \quad \frac{\partial r_k}{\partial y(t_j)} = 0.$$

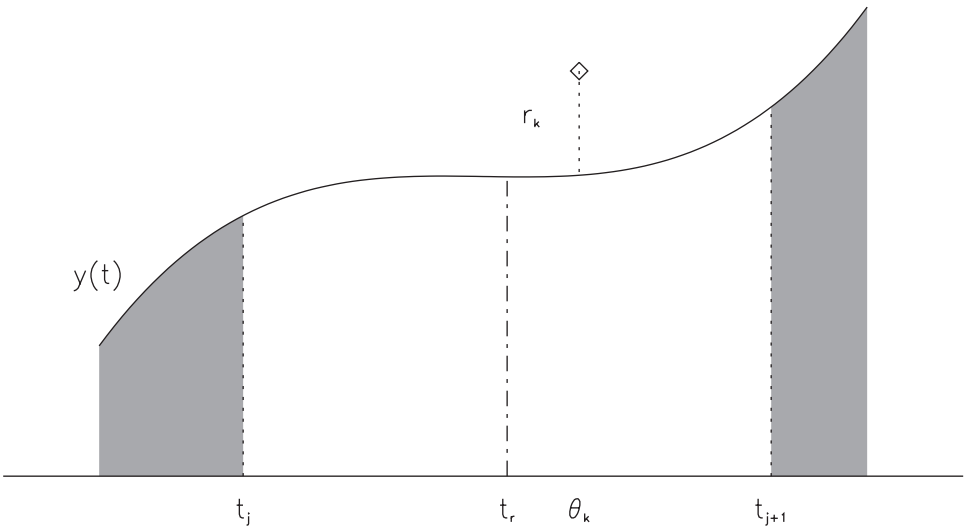

Figure 5.2. *Mesh refinement alters sparsity.*

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

$$\begin{aligned}\mathbf{r}_k(\mathbf{x}) &= \mathbf{A}_{k+m}\mathbf{x} + \mathbf{B}_{k+m}\mathbf{q}(\mathbf{x}) + \boldsymbol{\zeta}_{k+m} \\ &= \begin{bmatrix}\mathbf{0} & w_k(1-3\delta^2+2\delta^3) & \mathbf{0} & w_k(3\delta^2-2\delta^3) & \mathbf{0}\end{bmatrix}\begin{bmatrix}\cdot \\ y_{vj} \\ \cdot \\ y_{v,j+1} \\ \cdot\end{bmatrix} \\ &\quad + \begin{bmatrix}\mathbf{0} & w_k(h_j\delta - 2h_j\delta^2 + h_j\delta^3) & \mathbf{0} & w_k(-h_j\delta^2 + h_j\delta^3) & \mathbf{0}\end{bmatrix}\begin{bmatrix}\cdot \\ f_{vj} \\ \cdot \\ f_{v,j+1} \\ \cdot\end{bmatrix} \\ &\quad - w_k\hat{y}_{vj}. \end{aligned} \tag{5.27}$$

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

$$\begin{aligned}\mathbf{c}_k(\mathbf{x}) &= \mathbf{A}_k\mathbf{x} + \mathbf{B}_k\mathbf{q}(\mathbf{x}) + \boldsymbol{\zeta}_k \\ &= \begin{bmatrix}\mathbf{0} & \boldsymbol{\alpha}^{\mathsf{T}} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\cdot \\ \mathbf{y}_j \\ \mathbf{u}_j \\ \mathbf{p} \\ \cdot\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \beta_0 & \ldots, & \beta_{n_a} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\cdot \\ a_0(t_j) \\ \cdots \\ a_{n_a}(t_j) \\ \cdot\end{bmatrix}. \end{aligned} \tag{5.28}$$

Here, $\mathbf{A}_k = [\mathbf{0},\boldsymbol{\alpha}^{\mathsf{T}},\mathbf{0}]$, $\mathbf{B}_k = [\mathbf{0},\beta_0,\ldots,\beta_{n_a},\mathbf{0}]$, 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 $k = 0,\ldots,n_a$. 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 $k = 1, \ldots, M$, 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 ℓ

$$B_j(\tau_k) \geq 0, \qquad \frac{(\ell-1)}{N} \leq \tau_k \leq \frac{\ell}{N}, \tag{5.36a}$$

$$B_j(\tau_k) = 0 \qquad \text{otherwise.} \tag{5.36b}$$

Moreover for a C^1 cubic representation, there are at most four nonzero basis functions $B_j(\tau_k)$ at any grid point τ_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

$$\boldsymbol{\alpha}^{\mathsf{T}} = \left[\ldots, 1, \ldots, -B_\alpha(\tau_k), -B_{\alpha+1}(\tau_k), -B_{\alpha+2}(\tau_k), -B_{\alpha+3}(\tau_k), \ldots\right], \tag{5.37}$$

$$\mathbf{v}^{\mathsf{T}} = \left[\ldots, u_k, \ldots, b_\alpha, b_{\alpha+1}, b_{\alpha+2}, b_{\alpha+3}, \ldots\right]. \tag{5.38}$$

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 θ_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 ℓ 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 θ_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 θ_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
$\lvert p^* - \pi \rvert$	3.6×10^{-8}	4.7×10^{-8}	3.6×10^{-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.

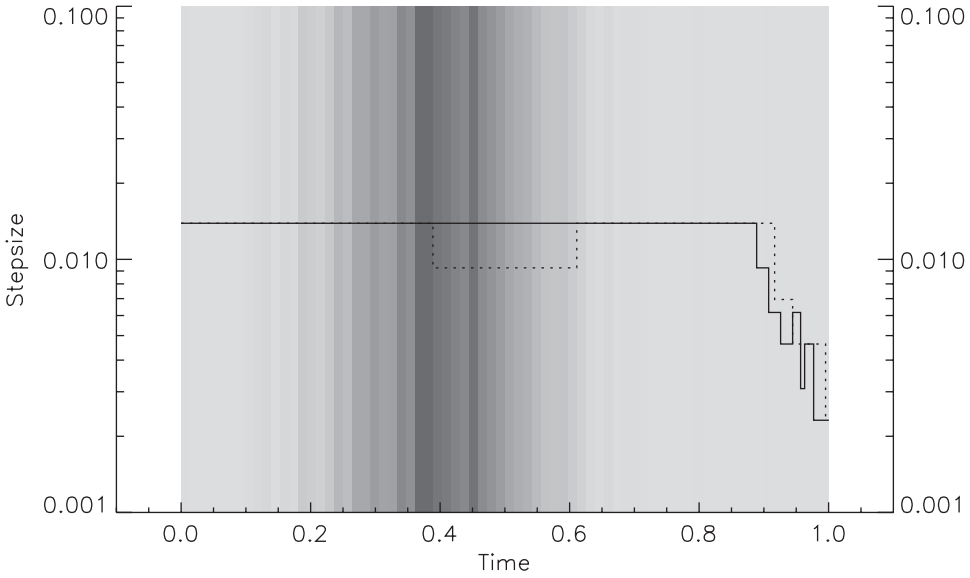

Figure 5.3. *Stepsize history (case* 1: *dashed; case* 3: *solid).*

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 $\mathbf{r}^{\mathrm{T}} = (x, y, z)$ is the earth centered inertial (ECI) position vector, $\mathbf{v}^{\mathrm{T}} = (\dot{x}, \dot{y}, \dot{z})$ 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

$$\mathbf{v}_r = \mathbf{v} - \boldsymbol{\omega} \times \mathbf{r}, \tag{5.45}$$

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 β 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 θ_k, and $\hat{\psi}_k$ is the corresponding radar measurement data, with standard deviation σ_1. Similarly, η_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 θ_s is the geocentric latitude of the radar site, $\varphi = \phi_s + \phi_0 + \omega t$ is the inertial longitude of the radar site, ϕ_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

$$\dot{s}(\mathbf{r}, \mathbf{v}, t) = \frac{\mathbf{s}^{\mathrm{T}}(\mathbf{v} - \dot{\mathbf{w}})}{\|\mathbf{s}\|}. \tag{5.57}$$

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 θ_d is the geodetic latitude of the radar site, and the geodetic elevation is given by

$$\eta(\mathbf{r}, \mathbf{v}, t) = \frac{\pi}{2} - \cos^{-1}\left(\frac{\mathbf{d}^{\mathrm{T}}\mathbf{s}}{\|\mathbf{s}\|}\right). \tag{5.60}$$

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 ϵ from 1.24×10^{-2} to 9.91×10^{-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 $\|\mathbf{q}_k^{\mathrm{T}}\mathbf{q}_k\|$ is plotted in Figure 5.5.

Table 5.2. *Mesh-refinement summary.*

k	M	NGC	NHC	NFE	NRHS	ϵ	Time (sec)
1	25	8	4	184	9016	1.24×10^{-2}	2.40
2	49	5	2	110	10670	8.27×10^{-4}	2.41
3	97	5	2	110	21230	4.02×10^{-5}	4.71
4	193	3	1	59	22715	2.40×10^{-6}	5.11
5	385	3	1	59	45371	1.53×10^{-7}	9.09
6	410	4	2	96	78624	9.91×10^{-8}	14.1
Total	410	28	12	618	187626		37.84

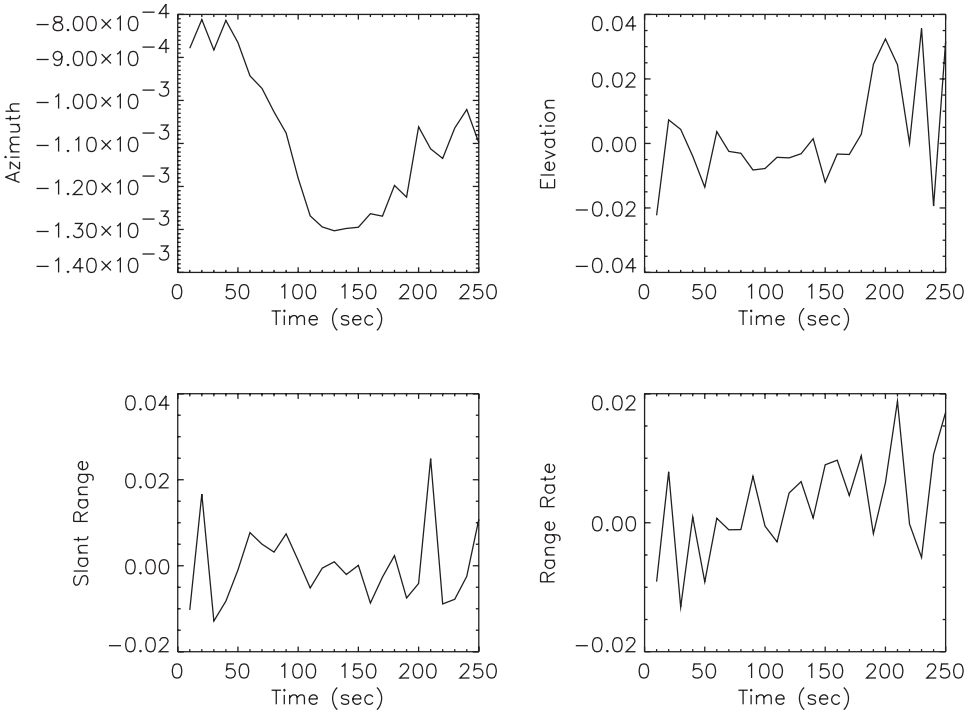

Figure 5.4. *Normalized residual errors.*

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.

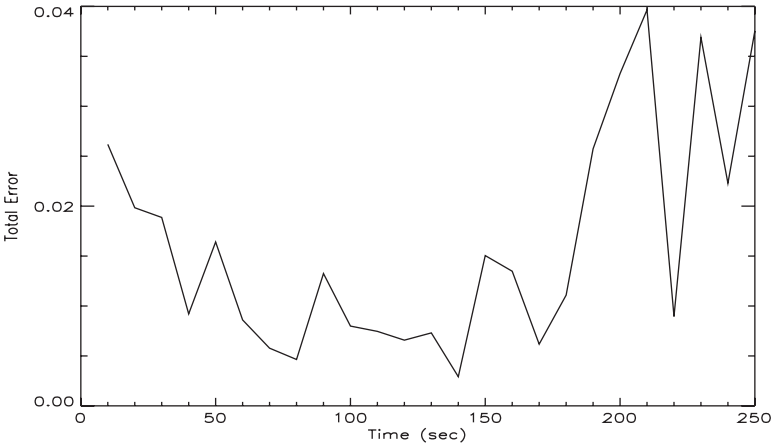

Figure 5.5. *Total normalized residual error.*

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 ϕ is the bank angle (rad), θ is the pitch angle (rad), ψ 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 σ_j. It should be clear that the residuals are of the form given by (5.11) with weights $w_k = 1/\sigma_v$. Note that for this example the symbol θ 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	ϵ	Time (sec)
1	200	10	1	105	39900	3.51×10^{-4}	13.0
2	380	3	1	38	28120	2.32×10^{-5}	6.19
3	740	3	1	38	55480	1.48×10^{-6}	10.9
4	1429	3	1	38	107844	9.39×10^{-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 $k = 1, \ldots, N_p$. 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

$$\mathbf{y}^{\mathsf{T}} = [\phi, \theta, \psi, p, q, r], \tag{5.78}$$

$$\mathbf{u}^{\mathsf{T}} = [a_p, a_q, a_r], \tag{5.79}$$

$$\mathbf{p}^{\mathsf{T}} = [b_{p1}, \ldots, b_{p42}, b_{q1}, \ldots, b_{q42}, b_{r1}, \ldots, b_{r42}]. \tag{5.80}$$

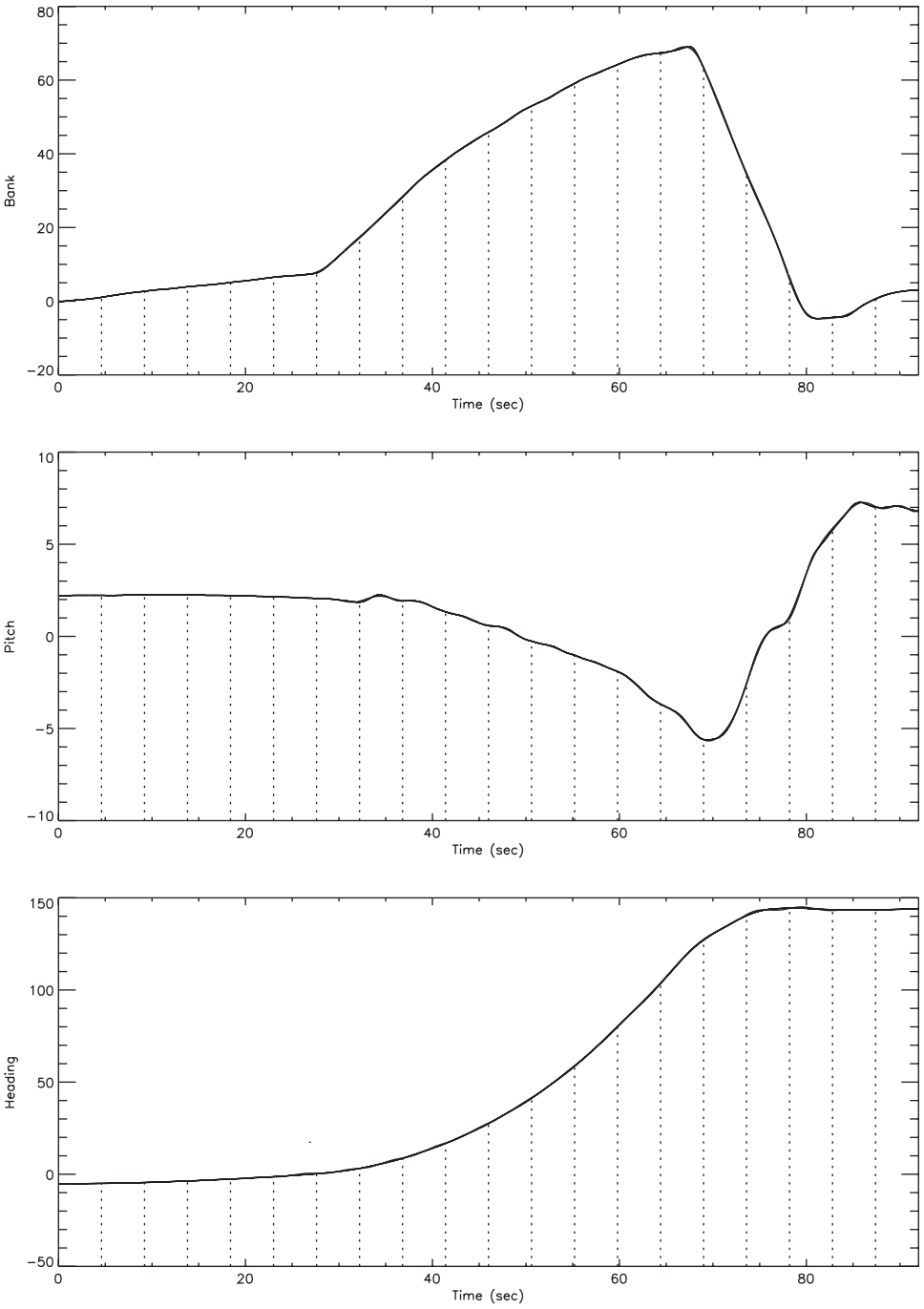

Figure 5.6. *Angular variable history.*

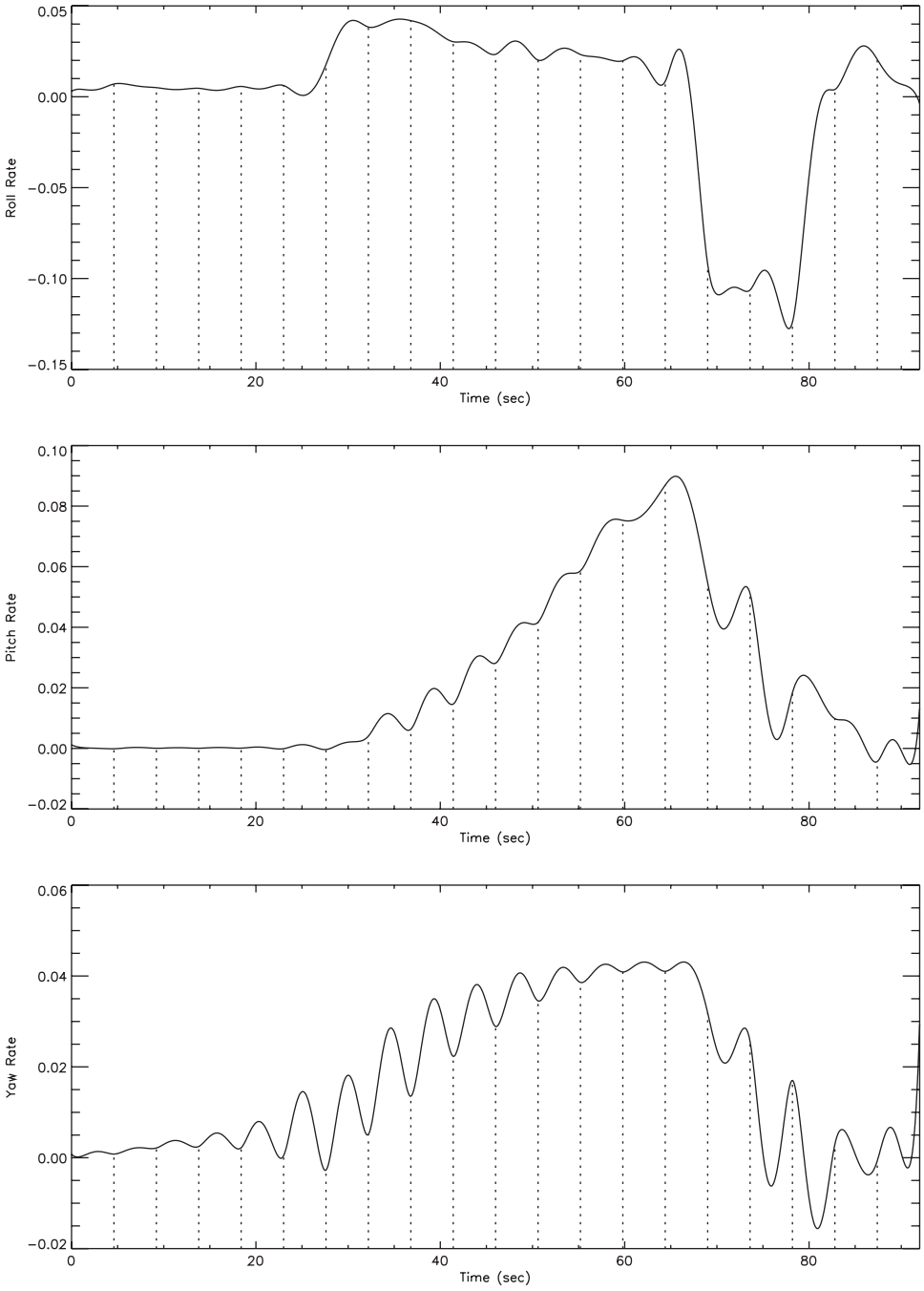

Figure 5.7. *Angular rate history.*

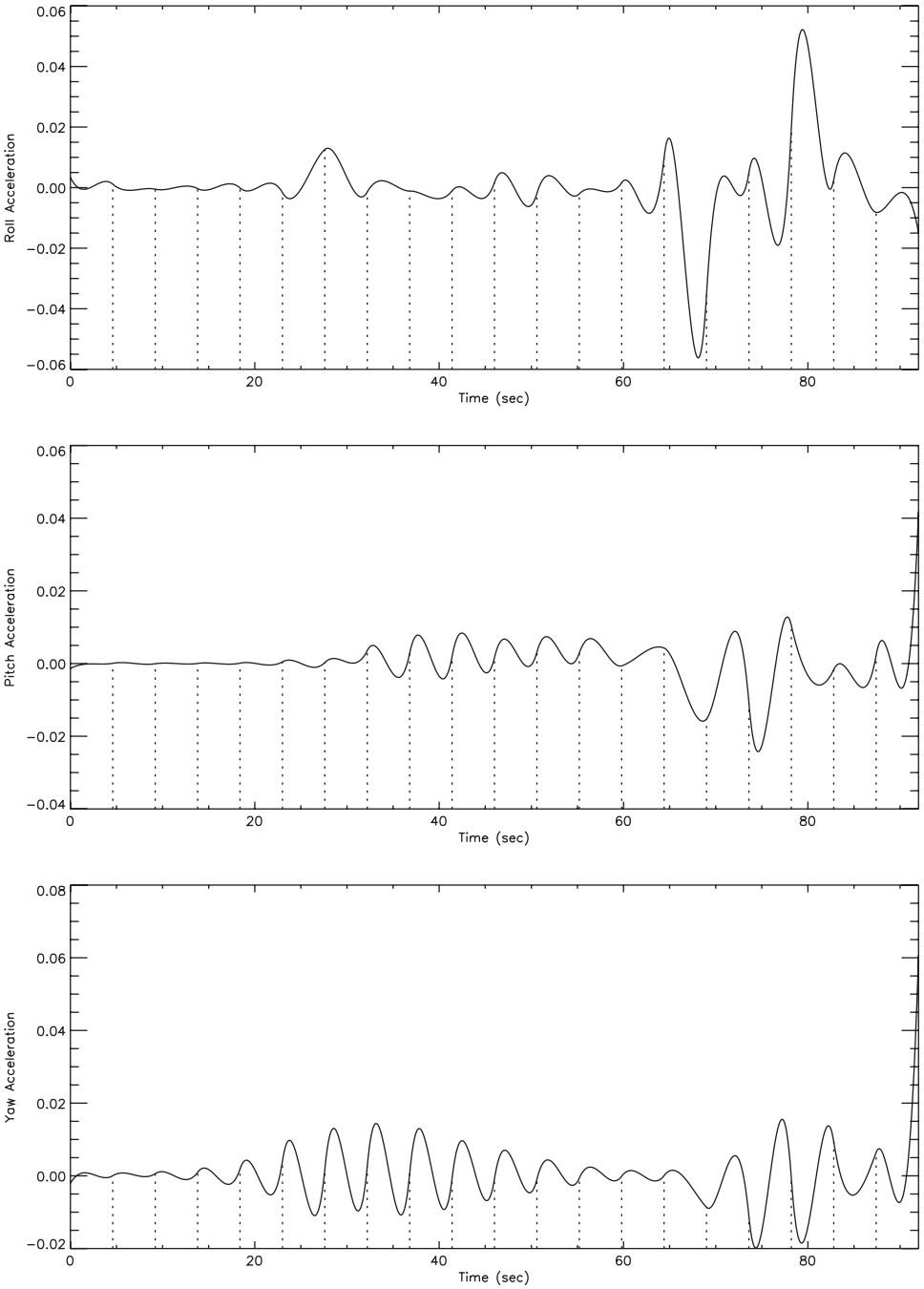

Figure 5.8. *Angular acceleration history.*

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	ϵ	n_d	Levenberg
1	10	7.07×10^{-4}	6	0
2	10	4.73×10^{-5}	6	0
3	19	3.03×10^{-6}	6	0
4	37	1.93×10^{-7}	6	0
5	73	1.20×10^{-8}	6	0

Formulation OCP

k	M	ϵ	n_d	Levenberg
1	10	2.6443×10^{-3}	14	6.0×10^{-6}
2	19	2.2177×10^{-3}	23	2.6×10^{-8}
3	19	1.3419×10^{-3}	41	3.7×10^{-8}
4	37	7.0506×10^{-4}	75	0
5	51	4.2101×10^{-4}	92	0
6	60	8.1262×10^{-5}	96	7.4×10^{-7}
7	77	4.1914×10^{-5}	117	7.4×10^{-7}
8	103	8.0736×10^{-6}	163	1.5×10^{-6}
9	142	2.3558×10^{-6}	230	1.5×10^{-6}
10	175	6.1396×10^{-7}	292	1.5×10^{-6}
11	203	1.1786×10^{-7}	347	1.5×10^{-6}
12	247	3.8995×10^{-8}	431	1.5×10^{-6}

But what if $\eta_j \neq 0$? To illustrate let us assume η_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

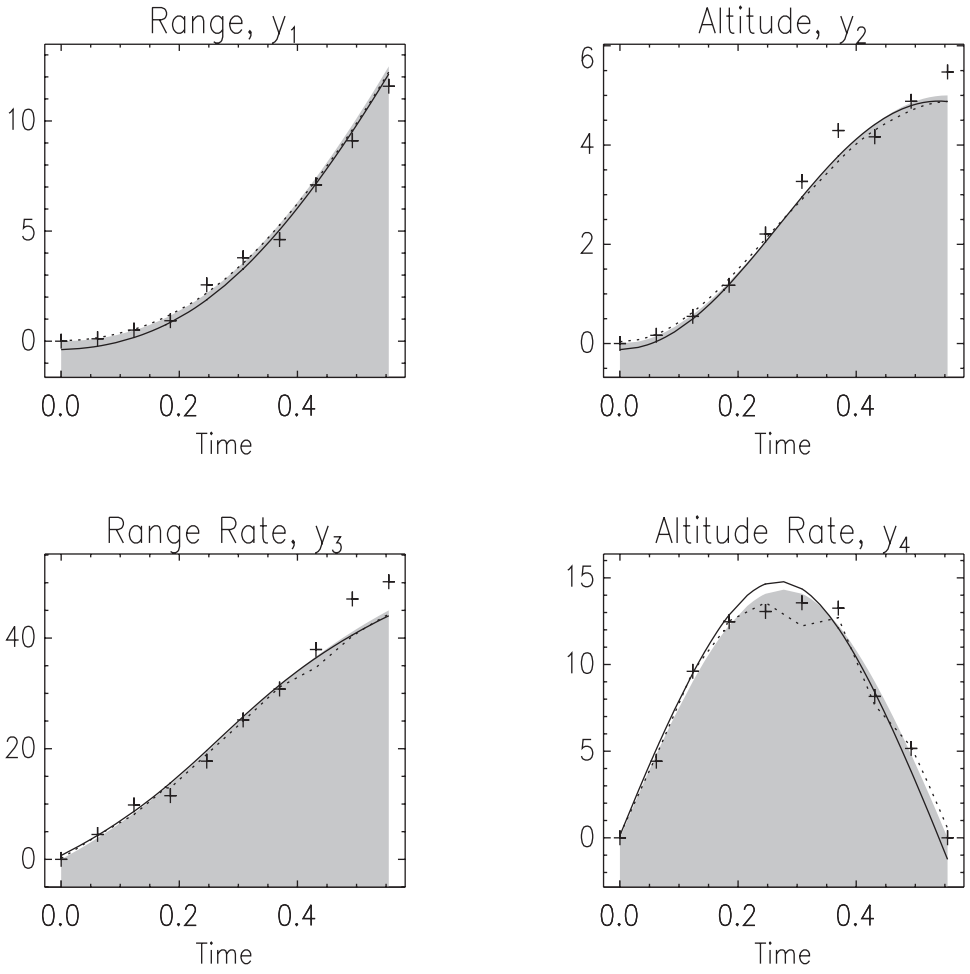

Figure 5.9. *Estimated state history.*

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

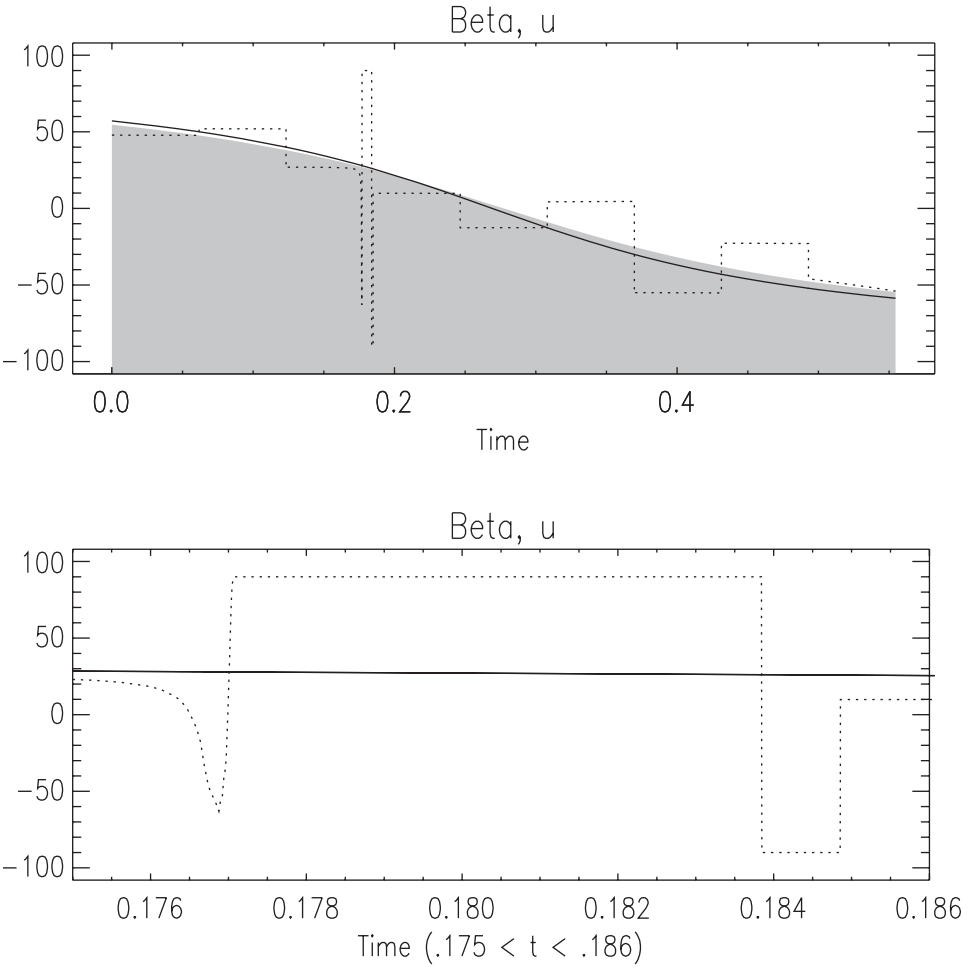

Figure 5.10. *Estimated control history.*

exceeds the number of residuals ℓ. 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.