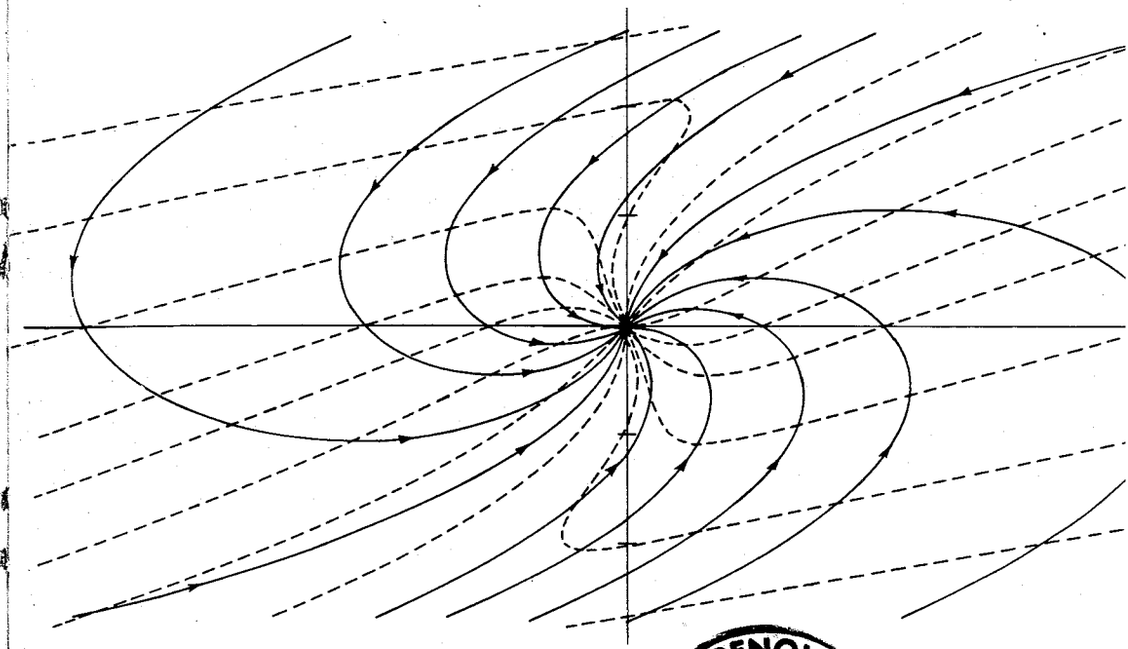
A Blaisdell Book in the Pure and Applied Sciences

Applied Optimal Control

OPTIMIZATION, ESTIMATION, AND CONTROL

Arthur E. Bryson, Jr.
Stanford University

Yu-Chi Ho
Harvard University



Blaisdell Publishing Company
A Division of Ginn and Company
Waltham, Massachusetts / Toronto / London

To Harvard University, its high standards, and its great tradition of seeking the truth, and to our teachers, colleagues, and students who have all taught and inspired us.

Library of Congress Catalog Card Number: 69–10425
Printed in the United States of America.

Preface

This text is intended for use at the senior-graduate level for university courses on the analysis and design of dynamic systems and for independent study by engineers and applied mathematicians. An elementary knowledge of mechanics and ordinary differential equations is presumed. Some acquaintance with matrix algebra and the properties of linear systems is desirable, but the required familiarity can be obtained by studying the two appendices. The book grew out of a set of lecture notes for a Harvard University summer school program on optimization of dynamic systems given in 1963. These notes were rewritten and extended for graduate courses given at Harvard from 1963 to 1968 and at M.I.T. in 1966.

The book is concerned with the analysis and design of complicated dynamic systems. Particular emphasis is placed on determining the "best" way to guide and/or control such systems. Over the past twenty-five years, a great body of knowledge has been built up on the subject of feedback control systems for linear, time-invariant dynamic systems. This knowledge plays an important role in our technology, and almost every engineering school recognizes this fact by teaching courses in this area. However, many dynamic systems—such as aerospace systems—are nonlinear and/or time-varying, and the techniques for analysis and design of linear, time-invariant systems are, in general, not applicable to these more complicated systems.

The appearance of practical, high-speed digital computers in the 1950's provided an essential tool for dealing with nonlinear and time-varying systems. Engineers were quick to take advantage of these remarkable computers to do extensive cut-and-try design work on paper instead of in the development laboratory. In many instances, particularly when designing guidance and control systems, it became clear that a more systematic approach was desirable. This led to a renewed interest in an old subject, the *calculus of variations*, and to

the discovery of an interesting extension of this subject, *dynamic programming*. The application of these techniques to deterministic, nonlinear and time-varying systems forms the basis of Chapters 1 through 9.

The first part of the book assumes precise knowledge of the structure and parameters of the dynamic systems investigated and precise measurements for feedback control. In practice, such precise knowledge is seldom available. Thus, it is important to be able to predict the sensitivity of controlled systems to *random fluctuations* in the environment and in the measurement system. Chapters 10 through 14 are concerned with this topic, starting with a review of the fundamentals of probability and random processes and proceeding to the design of the "best" control system, on the average, using noisy measurements and taking into account random perturbations of the system by the environment.

Our main objective has been to produce useful results which can be easily translated into digital computer programs. Several versions of the book in the form of lecture notes have been scrutinized by our competent and critical students and colleagues, and we therefore hope that most of the errors have been caught. However, we take responsibility for any that may still exist.

This book can be, and has been, used for both one-semester and two-semester courses in modern control theory. It can also be taught in either the deterministic–stochastic or the introductory–advanced sequence. The logical dependence of various chapters as well as the breakdown of chapters into semester–long parts are shown at the end of the table of contents.

The exercises form an integral part of the text. They either illustrate the subject matter covered or extend it and in some cases constitute a semi-research problem. Serious students should pursue them with diligence.

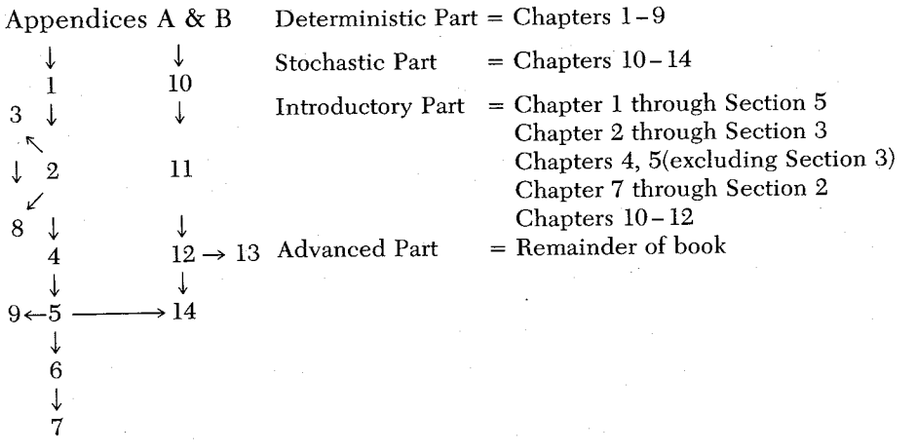
The authors are indebted to many persons for material. Special thanks go to John V. Breakwell, Henry J. Kelley, R. E. Kalman, George E. Leitmann, Placido Cicala, Colin C. Blaydon, Sheldon Baron, Ragasami L. Kashyap, Walter F. Denham, Stuart E. Dreyfus, Donald E. Johansen, Robert C. K. Lee, Stephen R. McReynolds, Jason L. Speyer, Kevin S. Tait, Laurie J. Henrikson, Raman Mehra, Mukund Desai, Robert Behn, and David Jacobson for papers and/or discussions.

We are also indebted to Marion Remillard, Sandra Nagy, and Skippi Torrance for typing the successive versions of the manuscript.

A. E. B.
Y.-C. H.

Contents

LOGICAL DEPENDENCE OF CHAPTERS



Numerical solution of optimal programming and control problems

7

7.1 Introduction

Unless the system equations, the performance index, and the constraints are quite simple, we must employ numerical methods to solve optimal programming and control problems. However, the amount of numerical computation required for even a relatively simple problem is forbidding if it must be done by hand. This is why the calculus of variations found very little use in engineering and applied science until quite recently. The development of economical, high-speed computers in the mid-1950's has dramatically changed this situation. It is now possible to solve complicated optimal programming and control problems in a reasonable length of time and at a reasonable cost.

High-speed computers solve *initial-value problems* for sets of differential equations very readily. However, as we have seen, optimal programming and control problems are at least *two-point boundary-value problems* and, in some cases, multipoint boundary-value problems (e.g., when there are interior point constraints or state variable inequality constraints). Finding solutions to these nonlinear two-point boundary-value problems is, in many cases, *not* a trivial extension of finding solutions to initial-value (one-point boundary-value) problems.

The nonlinear two-point boundary-value problem encountered in a large class of optimal programming problems was summarized in Section 2.8. The problem is to find

(a) the n state variables, $x(t)$,
(b) the n influence functions, $\lambda(t)$,
(c) the m control variables, $u(t)$,

to satisfy, simultaneously,

(i) the n system differential equations (involving x, u),
(ii) the n influence (adjoint, Euler-Lagrange) differential equations (involving λ, x, u),
(iii) the m optimality conditions (involving λ, x, u),
(iv) the initial and final boundary conditions (involving x, λ).

All numerical methods for the solution of such problems necessarily involve either flooding or iterative procedures.

Flooding (or dynamic programming), as applied to two-point boundary-value problems, can be described as a process of generating many solutions satisfying the specified boundary conditions at one end, using the unspecified boundary conditions as parameters. If the correct range of parameters is chosen, some of the solutions will pass through (or near) the desired boundary conditions at the other end.

At the present time, all the proposed iterative procedures use "successive linearization." A nominal solution is chosen that satisfies none, one, two, or three of the four conditions (i) through (iv) above; then this nominal solution is modified by successive linearization so that the remaining conditions are also satisfied. Only three of the fifteen possible approaches have been used extensively so far, namely, the three shown in Table 7.1.1.

Table 7.1.1 Iterative Procedures

	Nominal Solution Satisfies:			
	(i) System equations	(ii) Influence equations	(iii) Optimality conditions	(iv) Boundary conditions
Neighboring extremal methods	Yes	Yes	Yes	No
Gradient methods	Yes	Yes	No	No
Quasilinearization methods	No	No	Yes	Yes or No

When using neighboring extremal methods and quasilinearization methods, we must solve a succession of *linear* two-point boundary-value problems. Such problems can be solved by (a) finding the transition matrix between unspecified boundary conditions at one end and specified boundary conditions at the other end, or by (b) "sweeping" the boundary conditions from one end point to the other end point, which involves solving a matrix Riccati equation (see Sections 6.2 and 6.6).

For all three classes of iterative procedures, it is possible to handle

terminal constraints either by (a) gradient projection (linear penalty functions) or by (b) nonlinear (usually quadratic) penalty functions.

7.2 Extremal field methods; dynamic programming

One method of solving optimal programming problems is systematically to choose values for the unspecified initial (or terminal) conditions and compute the corresponding optimal solutions from the initial (or terminal) point. Computation continues until the region of the state space in and around the opposite point is so well covered with optimal solutions that it is possible to interpolate the desired optimal solution. Clearly, this is one way of solving the Hamilton-Jacobi-Bellman (HJB) equation in a certain domain of the state space (by the method of "characteristics")† and it would be useful for generating the optimal nonlinear feedback law for terminal control if all the optimal paths were computed *backward* from the terminal hypersurface. Relatively simple examples of nonlinear optimal feedback control schemes computed in this way were given in Sections 4.1 and 4.3.

Another possibility is to solve the HJB partial differential equation directly, starting at the terminal hypersurface. This is the procedure called dynamic programming, discussed in Chapter 4. For problems with more than two or three state variables, this procedure is not feasible even on the larger present-day computers. Even storing the answer (the whole extremal field) for a problem with three or more state variables usually requires an impractically large amount of space.

7.3 Neighboring extremal algorithms

Introduction

These methods are characterized by iterative algorithms for improving estimates of the unspecified initial (or terminal) conditions so as to satisfy the specified terminal (or initial) conditions.

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 that extremal solutions are often *very sensitive* to small changes in the unspecified boundary conditions. This extraordinary sensitivity is a direct result of the nature of the Euler-Lagrange equations, which, as we have seen

†R. Courant and D. Hilbert, *Methods of Mathematical Physics*, Vol. II, New York: Interscience, 1962, Chapter 2.

in Chapter 2, are influence function equations. They are, in fact, adjoint differential equations to the linear perturbation system equations, where the linearization is made about an extremal path. If the fundamental solutions to the linear perturbation system equations decrease in magnitude with increasing time, the fundamental solutions of the adjoint (Euler-Lagrange) equations increase in magnitude with increasing time. Thus, the solutions $x(t)$ and $\lambda(t)$ of the differential equations tend to become widely different in orders of magnitude as the integration proceeds in either direction. Since the number of significant digits that can be carried in a computer is limited regardless of whether fixed-point or floating-point arithmetic operations are used, the different growth of $x(t)$ and $\lambda(t)$ contributes greatly to the loss of accuracy.† One manifestation of this problem is that transition matrix solutions become ill-conditioned as the elements take on widely different ranges of value.‡ Since inversion of the transition matrix is necessarily part of the numerical method, the resultant accuracy is very poor. Another aspect of the same problem is the fact that small errors in guessing the influence functions at the initial time may produce enormous errors in the influence functions at the terminal time. This will be especially noticeable in highly dissipative systems, such as systems with friction or drag. 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!

In view of this starting difficulty, the direct integration method is usually practical only for finding neighboring extremal solutions after one extremal solution is obtained by some other method, such as a gradient method.

Some state variables specified at a fixed terminal time.

To set forth the basic ideas of this method, we consider first the relatively simple class of problems treated in Section 2.4: Find $\boldsymbol{u}(t)$ to minimize

$$J = \phi[x(t_f)] + \int_{t_o}^{t_f} L[x(t),u(t),t]\, dt\,, \tag{7.3.1}$$

where

$$\dot{x} = f(x,u,t)\,, \tag{7.3.2}$$

$$x(t_o) \quad \text{is specified,} \tag{7.3.3}$$

†See Kalman (1965).

‡Large values all appear to be equal and small values become zero.

$$\psi[x(t_f)] = \begin{bmatrix} x_1(t_f) - x_1^f \\ \cdot \\ \cdot \\ \cdot \\ x_q(t_f) - x_q^f \end{bmatrix} = 0\,, \qquad x_1^f, \ldots, x_q^f \text{ specified}, \tag{7.3.4}$$

$$t_o, t_f \text{ specified}. \tag{7.3.5}$$

First-order necessary conditions for an extremal solution are

$$\dot{\lambda} = -\left(\frac{\partial f}{\partial x}\right)^T \lambda - \left(\frac{\partial L}{\partial x}\right)^T, \tag{7.3.6}$$

$$\lambda_j(t_f) = \left(\frac{\partial \phi}{\partial x_j}\right)_{t=t_f}, \qquad j = q+1, \ldots, n\,, \tag{7.3.7}$$

$$0 = \left(\frac{\partial f}{\partial u}\right)^T \lambda + \left(\frac{\partial L}{\partial u}\right)^T. \tag{7.3.8}$$

The differential equations (7.3.2) and (7.3.6) are to be solved with the n initial conditions (7.3.3) and the n final conditions (7.3.4) plus (7.3.7), using (7.3.8) to determine $u(t)$. There are n unspecified initial conditions, $\lambda(t_o)$, and n unspecified terminal conditions, $[\lambda_1(t_f), \ldots, \lambda_q(t_f), x_{q+1}(t_f), \ldots, x_n(t_f)]$.

A Transition-Matrix Algorithm. This class of problems may be treated as follows:

STEP (a). Guess the n unspecified initial conditions, $\lambda(t_o)$. (Alternatively, guess the n unspecified final conditions with obvious modifications to the succeeding steps.)

STEP (b). Integrate (7.3.2) and (7.3.6) forward from t_o to t_f, using (7.3.8) to determine $u(t)$.

STEP (c). Record $x_1(t_f), \ldots, x_q(t_f), \lambda_{q+1}(t_f), \ldots, \lambda_n(t_f)$.

STEP (d). Determine the $(n \times n)$ transition matrix $[\partial\mu(t_f)/\partial\lambda(t_o)]$, where

$$\delta\mu(t_f) = \begin{bmatrix} \delta x_1(t_f) \\ \cdot \\ \cdot \\ \cdot \\ \delta x_q(t_f) \\ \delta\lambda_{q+1}(t_f) \\ \cdot \\ \cdot \\ \cdot \\ \delta\lambda_n(t_f) \end{bmatrix} = \frac{\partial\mu(t_f)}{\partial\lambda(t_o)}\,\delta\lambda(t_o)\,.$$

(See two methods for doing this below.)

STEP (e). Choose $\delta\mu(t_f)$ so as to bring the next solution *closer* to desired values of $\mu(t_f)$. For example, one might choose $\delta\mu(t_f) = -\epsilon[\mu(t_f) - \mu^f]$, $0 < \epsilon \leq 1$.

STEP (f). With chosen values of $\delta\mu(t_f)$ from Step (e), invert the transition matrix of Step (d) to find $\delta\lambda(t_o)$:

$$\delta\lambda(t_o) = \left[\frac{\partial\mu(t_f)}{\partial\lambda(t_o)}\right]^{-1} \delta\mu(t_f)\,.$$

STEP (g). Using $\lambda(t_o)_{\text{new}} = \lambda(t_o)_{\text{old}} + \delta\lambda(t_o)$, repeat Steps (a) through (f) until $\mu(t_f)$ has the specified values to the desired accuracy.

If the changes $\delta\mu(t_f)$ in Step (e) are too large, the iterative procedure may not converge. One way to check "size" is to compare $\mu(t_f)_{\text{new}} - \mu(t_f)_{\text{old}}$ with desired $\delta\mu(t_f)$; if they differ by more than, say, 10 to 20%, Step (e) should be repeated with smaller values of $\delta\mu(t_f)$.

The transition matrix in Step (d) above may be found in two different ways: (1) direct numerical differentiation, and (2) determination of unit solutions for the linear perturbation equations.

Direct numerical differentiation requires n additional integrations of the nonlinear system (7.3.2) and (7.3.6) using (7.3.8). On each of these integrations, one of components $\lambda_i(t_o)$ is changed by a small amount from the original guess in Step (a). The n quantities $\delta\mu(t_f)$ are recorded for each integration and divided by $\delta\lambda_i(t_o)$. In this way the transition matrix $[\partial\mu(t_f)/\partial\lambda(t_o)]$ is found. The difficulty with this straightforward approach is the following: If $\delta\lambda_i(t_o)$ is chosen too small, truncation error in integrating the nonlinear differential equations makes the determination of $\delta\mu(t_f)$ very inaccurate; if $\delta\lambda_i(t_o)$ is chosen too large, the linearity assumption will not be valid.

Determination of unit solutions requires n integrations of the $2n$th-order linear perturbation equations (6.1.21) and (6.1.22). On each of these integrations, one of the components of $\delta\lambda(t_o)$ is placed equal to unity, with all the other components zero and $\delta x(t_o) = 0$. This method is more accurate than direct numerical differentiation, but it requires additional computer programming. Furthermore, it may still produce an ill-conditioned transition matrix if unit solutions differ widely in numerical magnitude; the inversion required in Step (f) is then very inaccurate.†

A Backward-Sweep Algorithm. An effective way around the difficulty of an ill-conditioned transition matrix is offered by the following adaptation of the sweep method discussed in Section 6.2:

†See Problem 5, Section 5.2 for a way around the inversion of ill-conditioned matrices.

STEP (a). Estimate the q parameters $\nu^T = [\lambda_1(t_f), \ldots, \lambda_q(t_f)]$ and the $n - q$ unspecified terminal state variables $[x_{q+1}(t_f), \ldots, x_n(t_f)]$.

STEP (b). Integrate (7.3.2) and (7.3.6) *backward* from t_f to t_o using (7.3.8) to determine $u(t)$, with boundary conditions (7.3.4), (7.3.7), and the estimated quantities in Step (a).

STEP (c). Simultaneously with Step (b), integrate (6.2.11) through (6.2.13) backward with boundary conditions (6.2.3) through (6.2.5), which in this case are

$$S(t_f) = \phi_{xx}|_{t=t_f}, \quad Q(t_f) = 0, \quad \text{and}$$

$$R_{ij}(t_f) = \begin{cases} 1, & i = j, \\ 0, & i \neq j; \end{cases} \qquad i = 1, \ldots, n, \qquad j = 1, \ldots, q.$$

STEP (d). Record $x(t_o)$, $\lambda(t_o)$, and $(S - R^TQ^{-1}R)_{t=t_o}$. Choose $\delta x(t_o)$ to come closer to the specified values of $x(t_o)$. Then, from (6.2.15), we have $\delta\lambda(t_o) = (S - R^TQ^{-1}R)_{t=t_o}\delta x(t_o)$.

STEP (e). Integrate the perturbation equations (6.1.21) and (6.1.22) *forward* with boundary conditions $\delta x(t_o)$ and $\delta\lambda(t_o)$ from Step (d). Record $d\nu^T = [\delta\lambda_1(t_f), \ldots, \delta\lambda_q(t_f)]$ and $[\delta x_{q+1}(t_f), \ldots, \delta x_n(t_f)]$.

STEP (f). Using $\nu_{\text{new}} = \nu_{\text{old}} + d\nu$ and $(x_i(t_f))_{\text{new}} = [x_i(t_f)]_{\text{old}} + \delta x_i(t_f)$, $i = q+1, \ldots, n$, repeat Steps (a) through (f) until $x(t_o)$ has the specified values to the desired accuracy.

Functions of the state variables specified at an unspecified terminal time.

We now consider the more general problem of finding $u(t)$ to minimize

$$J = \phi[x(t_f),t_f] + \int_{t_o}^{t_f} L[x(t),u(t),t]\,dt \qquad \text{(performance index)}, \quad (7.3.9)$$

where

$$\dot{x} = f(x,u,t) \qquad (n \text{ equations}). \quad (7.3.10)$$

$$x(t_o) = x^o; \qquad t_o \text{ and } x^o \text{ specified} \qquad (n \text{ initial conditions}), \quad (7.3.11)$$

$$\psi[x(t_f),t_f] = 0 \qquad (q \text{ terminal conditions}). \quad (7.3.12)$$

The terminal time, t_f, is determined *implicitly* by the terminal conditions (7.3.12).

First-order necessary conditions for an extremal solution are

$$\dot{\lambda} = -\left(\frac{\partial H}{\partial x}\right)^T, \quad (7.3.13)$$

$$0 = \frac{\partial H}{\partial u}, \quad (7.3.14)$$

$$\lambda^T(t_f) = \left(\frac{\partial \Phi}{\partial x}\right)_{t=t_f}, \quad (7.3.15)$$

$$\Omega[x,u,\nu,t]_{t=t_f} \equiv \left(\frac{d\Phi}{dt} + L\right)_{t=t_f} = 0, \quad (7.3.16)$$

where

$$\Phi(x,\nu,t) = \phi(x,t) + \nu^T\psi(x,t), \qquad \frac{d\Phi}{dt} = \frac{\partial \Phi}{\partial t} + \frac{\partial \Phi}{\partial x}f.$$

The $2n$ differential equations (7.3.10) and (7.3.13) must be solved, and the $q + 1$ parameters ν and t_f must be determined to satisfy the n initial conditions (7.3.11) and the $q + n + 1$ terminal conditions (7.3.12), (7.3.15), and (7.3.16), using (7.3.14) to determine $u(t)$.

A Transition-Matrix Algorithm. The following procedure may be used.

STEP (a). Guess the n terminal conditions $x(t_f)$, the q parameters ν, and the terminal time t_f.

STEP (b). Determine $\psi[x(t_f),t_f]$, and $\lambda(t_f)$, $\Omega[x(t_f),u(t_f),\nu,t_f]$ from (7.3.15) and (7.3.16); $u(t_f)$ must be determined from (7.3.14) evaluated at $t = t_f$, using $\lambda(t_f)$ and $x(t_f)$.

STEP (c). Integrate (7.3.10) and (7.3.13) *backward* from t_f to t_o, using (7.3.14) to determine $u(t)$, and using the terminal conditions $x(t_f)$, $\lambda(t_f)$ from Steps (a) and (b).

STEP (d). Record $x(t_o)$.

STEP (e). Find the $[(n + q + 1) \times (n + q + 1)]$ transition matrix

$$\frac{\partial[x(t_o),\psi,\Omega]}{\partial[x(t_f),\nu,t_f]},$$

where

$$\begin{bmatrix} \delta x(t_o) \\ d\psi \\ d\Omega \end{bmatrix} = \frac{\partial[x(t_o),\psi,\Omega]}{\partial[x(t_f),\nu,t_f]} \begin{bmatrix} \delta x(t_f) \\ d\nu \\ dt_f \end{bmatrix}.$$

STEP (f). Choose values of $\delta x(t_o)$, $d\psi$, and $d\Omega$ so as to bring the next solution closer to the desired values of $x(t_o)$, $\psi = 0$, and $\Omega = 0$. For example, one might choose

$$\begin{bmatrix} \delta x(t_o) \\ d\psi \\ d\Omega \end{bmatrix} = -\epsilon \begin{bmatrix} x(t_o) - x^o \\ \psi[x(t_f),t_f] \\ \Omega[x(t_f),t_f] \end{bmatrix}, \qquad 0 < \epsilon \leq 1.$$

STEP (g). With chosen values of $\delta x(t_o)$, $d\psi$, and $d\Omega$ from Step (f), invert the transition matrix of Step (e) to find $\delta x(t_f)$, $d\nu$, and dt_f.

STEP (h). Using

$$\begin{bmatrix} x(t_f) \\ \nu \\ t_f \end{bmatrix}_{\text{new}} = \begin{bmatrix} x(t_f) \\ \nu \\ t_f \end{bmatrix}_{\text{old}} + \begin{bmatrix} dx(t_f) \\ d\nu \\ dt_f \end{bmatrix},$$

repeat Steps (a) through (h) until $x(t_0) = x^0$, $\psi[x(t_f),t_f] = 0$, and $\Omega[x(t_f),u(t_f),\nu,t_f] = 0$ to the desired accuracy. Note that $dx(t_f) = \delta x(t_f) + \dot{x}(t_f)\, dt_f$.

If the changes in Step (f) are too large, the iterative procedure will not converge. One way to avoid this is to compare the actual changes in $x(t_0)$, ψ, and Ω with desired changes; if they differ by more than, say, 10 to 20%, Step (h) should be repeated with smaller values of $\delta x(t_0)$, $d\psi$, and $d\Omega$.

The transition matrix in Step (e) may be found in two different ways: (1) direct numerical differentiation and (2) determination of unit solutions for the linear perturbation equations.

Direct numerical differentiation requires $n + q + 1$ additional backward integrations of the nonlinear system (7.3.10) and (7.3.13) using (7.3.24). On each of these integrations, one of the components of $x(t_f)$, ν, and t_f is changed by a small amount from the original guess in Step (a). The $n + q + 1$ quantities $\delta x(t_0)$, $d\psi$, and $d\Omega$ are recorded for each choice and divided by the change from the original guess. In this way the transition matrix in Step (e) can be found. The difficulty with this straightforward approach is the same as that mentioned in the previous section.

Determination of unit solutions requires $n + q + 1$ backward integrations of the $2n$th-order linear perturbation equations (6.1.21) and (6.1.22). On each of these integrations, one of the components of $[\delta x(t_f),d\nu,dt_f]$ is placed equal to unity with all the other components zero. The determination of $\delta\lambda(t_f)$, $d\psi$, and $d\Omega$ is made by considering the first-order perturbations of the terminal conditions (7.3.15), (7.3.12), and (7.3.16) which were given in (6.5.9) through (6.5.11). Equations (6.5.10) and (6.5.11) form part of the transition matrix in Step (e), whereas $\{\partial x(t_0)/\partial[x(t_f),\nu,t_f]\}$ must be found by integrating the perturbation equations backwards $n + q + 1$ times with $\delta\lambda(t_f)$ obtained from (6.5.9) for unit values of $\delta x(t_f)$, $d\nu$, and dt_f. While this method is more accurate than direct numerical differentiation, it obviously requires more computer programming and is prone to the same difficulties as described in the previous section.

Note that a necessary condition for a minimum is

$$\left(\frac{d\Omega}{dt}\right)_{t=t_f} \geq 0. \tag{7.3.17}$$

If the inequality holds in Equation (7.3.17), we may solve (6.5.11) for dt_f in terms of $\delta x(t_f)$ and $d\nu$ and substitute the result into (6.5.9) and (6.5.10); in this case, only $n + q$ unit solutions need be found to determine the transition matrix.

A Backward-Sweep Algorithm. One of the possible difficulties with the transition-matrix approach is that sufficient numerical accuracy may not be attainable even with the technique of finding unit solutions for the linear perturbation equations. This is particularly true for dissipative systems, for reasons mentioned earlier in this section. Usually, an effective way to avoid this difficulty is to use the following adaptation of the *backward-sweep approach*† presented in Section 6.6:

STEPS (a), (b), and (c). Same as in the Transition-Matrix Algorithm.

STEP (d). Simultaneously with Step (c), integrate (6.6.8) through (6.6.13) with boundary conditions given there.

STEP (e). Record x, λ, S, R, Q, m, n, α at $t = t_0$. Choose $\delta x(t_0)$, $d\psi$, and $d\Omega$ as in Step (f) of the Transition Matrix Algorithm. Then, from (6.6.14), (6.6.15), and (6.6.18), determine $d\nu$, dt_f, and $\delta\lambda(t_0)$. Record $d\nu$ and dt_f.

STEP (f). Integrate the perturbation equations (6.1.21) and (6.1.22) *forward* with boundary conditions $\delta x(t_0)$ and $\delta\lambda(t_0)$. Record $dx(t_f) = \delta x(t_f) + \dot{x}(t_f)\, dt_f$.

STEP (g). Using

$$\begin{bmatrix} x(t_f) \\ \nu \\ t_f \end{bmatrix}_{\text{new}} = \begin{bmatrix} x(t_f) \\ \nu \\ t_f \end{bmatrix}_{\text{old}} + \begin{bmatrix} dx(t_f) \\ d\nu \\ dt_f \end{bmatrix},$$

repeat Steps (a) through (g) until $x(t_0) = x^0$, $\psi[x(t_f),t_f] = 0$, and $\Omega[x(t_f),u(t_f),\nu,t_f] = 0$ to the desired accuracy.

7.4 First-order gradient algorithms

Introduction

Gradient methods were developed to surmount the "initial guess" difficulty associated with direct integration methods (see Introduction of Section 7.3). They are characterized by iterative algorithms for improving estimates of the control histories, $u(t)$, so as to come closer to satisfying the optimality conditions and the boundary conditions.

†The reason for this is that there is less chance of difference in growth in the elements of $S(t)$ than in $X(t)$ and $\Lambda(t)$, used in the transition matrix algorithm.