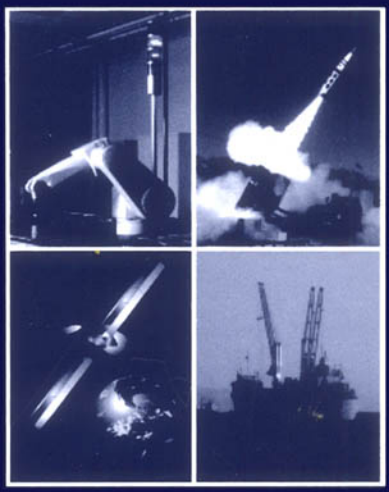
Applied Dynamic Programming for Optimization of Dynamical Systems



Rush D. Robinett III
David G. Wilson
G. Richard Eisler
John E. Hurtado

Advances in Design and Control
siam

Applied Dynamic Programming for Optimization of Dynamical Systems

Advances in Design and Control

SIAM's Advances in Design and Control series consists of texts and monographs dealing with all areas of design and control and their applications. Topics of interest include shape optimization, multidisciplinary design, trajectory optimization, feedback, and optimal control. The series focuses on the mathematical and computational aspects of engineering design and control that are usable in a wide variety of scientific and engineering disciplines.

Series Volumes

Robinett III, Rush D., Wilson, David G., Eisler, G. Richard, and Hurtado, John E., *Applied Dynamic Programming for Optimization of Dynamical Systems*

Huang, J., *Nonlinear Output Regulation: Theory and Applications*

Haslinger, J. and Mäkinen, R. A. E., *Introduction to Shape Optimization: Theory, Approximation, and Computation*

Antoulas, Athanasios C., *Approximation of Large-Scale Dynamical Systems*

Gunzburger, Max D., *Perspectives in Flow Control and Optimization*

Delfour, M. C. and Zolésio, J.-P., *Shapes and Geometries: Analysis, Differential Calculus, and Optimization*

Betts, John T., *Practical Methods for Optimal Control Using Nonlinear Programming*

El Ghaoui, Laurent and Niculescu, Silviu-Iulian, eds., *Advances in Linear Matrix Inequality Methods in Control*

Helton, J. William and James, Matthew R., *Extending H^∞ Control to Nonlinear Systems: Control of Nonlinear Systems to Achieve Performance Objectives*

Applied Dynamic Programming for Optimization of Dynamical Systems

Rush D. Robinett III
Sandia National Laboratories
Albuquerque, New Mexico

David G. Wilson
Sandia National Laboratories
Albuquerque, New Mexico

G. Richard Eisler
Sandia National Laboratories
Albuquerque, New Mexico

John E. Hurtado
Texas A&M University
College Station, Texas

Society for Industrial and Applied Mathematics
Philadelphia

10 9 8 7 6 5 4 3 2 1

This research was supported by funding from the Director of Central Intelligence Postdoctoral Research Fellowship Program.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC0494AL-8500.

Library of Congress Cataloging-in-Publication Data

Applied dynamic programming for optimization of dynamical systems / Rush D. Robinett III ... [et al.].
p. cm. – (Advances in design and control)
Includes bibliographical references and index.
ISBN 0-89871-586-5
1. Dynamic programming. I. Robinett, Rush D. II. Series.

T57.83.A67 2005
519.7'03–dc22 2005045058

Figures 3.9, 3.10, and 3.11 in Chapter 3 © IEEE. Reprinted with permission from C. R. Dohrmann and R. D. Robinett III, "Input Shaping for Three-Dimensional Slew Maneuvers of a Precision Pointing Flexible Spacecraft," *Proceedings of the American Control Conference*, June 1994, Baltimore, Maryland, pp. 2543–2547.

Figures 4.2 and 4.3 in Chapter 4 are from C. R. Dohrmann, G. R. Eisler, and R. D. Robinett III, "Dynamic Programming Approach for Burnout-to-Apogee Guidance of Precision Munitions," *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 2, 1996, pp. 340–346; reprinted by permission of the American Institute of Aeronautics and Astronautics, Inc.

Figures 4.4, 4.5, and 4.7 in Chapter 4 © Kluwer Academic/Plenum Publishers. Reprinted with permission from C. R. Dohrmann and R. D. Robinett III, "Efficient Sequential Quadratic Programming Implementations for Equality-Constrained Discrete-Time Optimal Control," *Journal of Optimization Theory and Applications*, Vol. 95, No. 2, 1997, pp. 323–346.

Figures 4.8, 4.9, and 4.10 in Chapter 4 © Kluwer Academic/Plenum Publishers. Reprinted with permission from C. R. Dohrmann and R. D. Robinett III, "Dynamic Programming Method for Constrained Discrete-Time Optimal Control," *Journal of Optimization Theory and Applications*, Vol. 101, No. 2, 1999, pp. 259–283.

Figures 5.17, 5.18, 5.19, and 5.20 in Chapter 5 © IEEE. Reprinted with permission from D. G. Wilson, R. D. Robinett III, and G. R. Eisler, "Discrete Dynamic Programming for Optimized Path Planning of Flexible Robots," *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept. 28–Oct. 2, 2004, Sendai, Japan, pp. 2918–2923.

About the cover photos:

Robot: A Stäubli model RX90 robotic manipulator is shown as part of an Agile Manufacturing Prototyping System (AMPS) at the Intelligent Systems and Robotics Center (ISRC) used to demonstrate reconfigurable manufacturing production processes. (Photograph courtesy of AMPS project ISRC at Sandia National Laboratories.)

Satellite: Spacecraft concept for geosynchronous orbit high-power, Boeing™ model 702 Satellite designed to provide state-of-the-art digital audio radio programming directly to cars, homes and portable radios coast-to-coast. (Photograph courtesy of Boeing Satellite Systems, Inc. and XM Satellite Radio, Inc. ©Boeing Management Company. Used under license.)

Missile Launch: The Multiple Launch Rocket System (MLRS) is a high mobility rocket artillery system manufactured by Lockheed Martin Missiles and Fire Control. MLRS fires surface-to-surface rockets and the long-range Lockheed Martin Army Tactical Missile Systems (ATACMS) guided missiles. (Photograph courtesy of the U.S. Army.)

Crane Ship: SS Flickertail State Auxiliary Crane Ship used to support ship-to-ship cargo transfer is in ready reserve for the U.S. Navy under the Military Sealift Command. (Photograph courtesy of the Department of Transportation's Maritime Administration (MARAD) and the Naval Surface Warfare Center (NSWC).)

siam is a registered trademark.

Contents

List of Figures

List of Tables

This page intentionally left blank

Preface

This book presents a broad cross-section of dynamic programming (DP) techniques applied to the optimization of dynamical systems based on the results of over 10 years of research and development at Sandia National Laboratories by the authors. The main goal of the research effort was to develop a robust path planning/trajectory optimization tool that didn't require an initial guess. The goal was partially met with a combination of DP and homotopy algorithms. The theoretical development of the DP algorithms is included with a primary emphasis on their application to practical engineering problems. Further, the DP algorithms were successfully applied to a variety of engineering problems including rigid, flexible, and redundant robots; rigid and flexible spacecraft; autonomous rocket guidance and control; and rotary crane systems.

In general, the DP algorithms may potentially address a wide class of applications composed of many different physical systems described by dynamical equations of motion that require optimized trajectories for effective maneuverability. The DP algorithms determine control inputs and corresponding state histories of dynamic systems for a specified time while minimizing a performance index. Constraints may be applied to the final states of the dynamic system or to the states and control inputs during the transient portion of the maneuver.

Research scientists, practicing engineers, and engineering students with a background in and basic understanding of dynamics and controls will be able to develop and apply the DP algorithms to their particular problems. Considerable emphasis is placed on the design steps applied in a case study format to representative applications in robotics and aerospace systems. A fundamental development in classical constrained optimization is also provided that helps set the stage for later developments and analogies.

The organization of the book makes it possible for readers to actually use the DP algorithms even before thoroughly comprehending the full theoretical development. This should appeal to practicing engineers and professionals. The book is also suitable in engineering academia, as either a reference or a supplemental textbook for a graduate course in optimization of dynamic and control systems. Most certainly, an important outcome would be to aid in the continued development of individuals who strive for a systems perspective with a broadened understanding of the modeling, analysis, design, and testing of engineering systems.

In addition to conventional constrained optimization, three distinctive features are developed in this book:

- A general architecture is introduced for the DP algorithm that emphasizes the solution to nonlinear problems. Several key topics are included that employ and intertwine nominal trajectories, quadratic convergence, and techniques that help systematically lead to the final solution.
- The DP algorithm development is introduced gradually with illustrative examples that surround linear systems applications. Incrementally more complex examples are introduced until full nonlinear systems are addressed. Initially, constraints are included in the performance index. Subsequent developments highlight equality- and inequality-type constraints applied to the full nonlinear problem.
- Many examples and explicit design steps applied to case studies illustrate the ideas and principles behind the DP algorithm. The dynamic programming interior point (DPIP) software driver functions are coupled with the popular MATLAB® modeling and simulation tool, which provides an effective teaching atmosphere. The MATLAB environment allows the user to explore other relevant concepts and variations in an efficient manner. This relieves the user from the intricate details of a formal programming language.

These three distinctive features are covered in five chapters with corresponding appendices.

Chapter 1 introduces a general control system viewpoint. The specific role that the DP algorithms play in feedforward or open-loop control is identified. Several illustrative examples help to rapidly demonstrate the benefits and usefulness of a DP approach.

Chapter 2 presents an overview of general constrained optimization concepts and reviews the main components that support recursive quadratic programming (RQP). These include specific constraint categories and conditions, various search methods, and convergence criteria. This chapter finishes with three diversified examples that apply RQP constrained optimization.

Chapters 3 and 4 are the theoretical development chapters of the DP algorithm. However, specific examples and practical problems are reviewed throughout to help demonstrate the DP algorithm's applicability.

Chapter 3 introduces the discrete DP algorithm for feedforward command and optimization of linear systems, with several descriptive examples. Initially, the DP algorithm is applied to a data smoothing problem and then to discrete-time optimal control problems. This chapter concludes with a nonlinear optimal control problem with constraints of a spacecraft with flexible members.

Chapter 4 provides advanced concepts such as the implementation of both equality and inequality constraints to the DP optimization problem. Before introducing the DP algorithms, this chapter begins with an advanced application that highlights real-time rocket guidance and control. The advanced DP theory is then introduced incrementally. Equality constraints are enforced by employing a Lagrange multiplier approach. Solutions are obtained by efficiently solving a series of constrained quadratic programming subproblems. The inequality constraints are accommodated by adding a primal-dual interior point method to help iteratively solve the subproblems. For the examples investigated, the combined DPIP algorithms displayed quadratic convergence of the iterations. Several practical examples are included to illustrate the benefits of the DPIP algorithms. Both specified and free final time problems were accommodated.

Chapter 5 elaborates on a series of case studies that demonstrate the procedures for applying the DPIP algorithm developed in earlier chapters. These case studies are formulated in the following manner:

1. Introduction
2. Model development
3. Optimization design
4. Implementation issues
5. Numerical simulation results
6. Discussion (summary/conclusions).

These case studies draw on popular themes and applications in the robotics and aerospace disciplines. Where applicable, the DPIP algorithm is compared to more conventional numerical optimization techniques such as the SQP/RQP algorithms. The details of each case study DPIP MATLAB implementation are included as an appendix.

DP is the best of both worlds. It solves the direct problem and is linear in the optimization parameters. This enables one to solve for smooth input/control histories. This is in contrast to SQP/RQP, where the number of operations is quadratic or cubic. However, the cost of the DP linear behavior is as order n^2 or n^3 in the state variables. Therefore, for large-scale or a large number of DOFs (degrees of freedom) in the state variables, DP is not practical and one should consider alternative methods such as SQP/RQP algorithms.

Although the theoretical development is included, the focus throughout the book is on the practical implementation of the DPIP algorithm. The goal is to present applications that may be adapted easily to the reader's own interests and problems.

Nomenclature

The nomenclature used throughout the book is summarized as follows:

- Scalars are represented as nonbold letters, e.g., A, a, α, and Γ.
- Vectors will normally be represented as bold lowercase letters, e.g., $\mathbf{x}$, $\mathbf{y}$, and $\boldsymbol{\beta}$. Symbols with subscripts are nonbold, e.g., x_i and y_i normally represent elements in row i of the vectors $\mathbf{x}$ and $\mathbf{y}$, respectively. Bold symbols with subscripts or superscripts normally represent a vector at a particular iteration step (e.g., $\mathbf{x}^j$).
- Matrices are two-dimensional arrays of scalars and will normally be represented as bold uppercase letters, e.g., $\mathbf{A}$, $\mathbf{W}$, and $\boldsymbol{\Gamma}$. Nonbold symbols with subscripts are used, e.g., A_{ij} refers to the element in row i and column j of the matrix $\mathbf{A}$.
- The notational convention in Sections 4.3 and 4.4 is to use superscripts for vector and matrix components (e.g., b^α, $a^{\alpha\beta}$). In addition, Einstein's summation convention is employed as a notational convenience. With this convention, the repeated appearance of a superscript index implies summation over all possible values of the index.

On a more conventional note, a capital superscript T defines the transpose of a vector or matrix, e.g., $\mathbf{A}^T$, and a superscript -1 defines the inverse of a matrix, e.g., $\mathbf{A}^{-1}$. A dot ($\dot{}$) or dots above a variable define differentiation with respect to time, e.g., $\dot{\omega}$ and $\ddot{\omega}$. A prime or primes on a variable represent differentiation with respect to a spatial variable, e.g., $y'(x) = d(y(x))/dx$. A hat or tilde above a variable, e.g., $\hat{x}$, $\tilde{y}$, defines a variable estimation or approximation. However, a hat above a variable could also represent a unit vector, e.g., $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$. A bar above a variable represents a mean or average, e.g., $\bar{x}$. All other special variables and symbols are defined as needed. Necessary exceptions are explained.

Acknowledgments

The field of control clearly does play a major role in America's effort to combat terrorism [1] and provide homeland security. Continued breakthroughs in this area will be required

> "... for command and control of unmanned vehicles, to robust networks linking businesses, transportation systems, and energy infrastructure, to improved techniques for sensing and detection of biological and chemical agents ..." [1]

and it will be seen that

> "... the techniques and insights from control will enable new methods for protecting human life and safeguarding our society" [1].

The authors would like to thank the many sponsors who contributed to the sustained effort of this work.

Sandia National Laboratories provided a necessary, all-important research outlet for the advancement of control research and development of needed control tool sets through their continued commitment and support.

Most recently, this research was supported by funding from the Director of Central Intelligence Postdoctoral Research Fellowship Program.

The authors wish to acknowledge those who have been helpful in the preparation of this book and our many collaborators and coauthors over the course of this decade.

Special thanks to Dr. Clark R. Dohrmann of Sandia National Laboratories for his unique contribution and the major role he has played in the initial development, implementation, and testing of the DPIP algorithm and results that are described in Chapters 3 and 4.

Many thanks to Dr. David Martinez of Sandia National Laboratories for his continued support during the early years and his encouragement of the genesis of the DP techniques.

Our appreciation to Dr. Ann Campbell of Sandia National Laboratories for her guidance and assistance during the postdoctoral program and process.

Many thanks to Prof. Gordon G. Parker of Michigan Technological University for his early review and numerous helpful and constructive suggestions that ultimately led to a much improved final book manuscript.

Finally, many thanks to Mrs. Marna E. Wilson and Mrs. Lucy M. Jydstrup for their constructive suggestions and editing related to the manuscript preface and support materials.

Rush D. Robinett III
David G. Wilson
G. Richard Eisler
John E. Hurtado

Chapter 1
Introduction

1.1 Key Role of Dynamic Programming in Control Design

Today's engineering systems sustain desirable performance by using well-designed control systems based on fundamental principles and mathematics. Many engineering breakthroughs and improvements in sensing and computation have helped to advance the field. Control systems currently play critical roles in many areas, including manufacturing, electronics, communications, transportation, computers, and networks, as well as many commercial and military systems [1].

New developments in this increasingly information-rich world will require a significant expansion of the basic tool sets of the control field [1]. Over the next decade, extensions and improvements through control research will help enable future applications in the following systems: aerospace, automotive, and naval transportation; communication; robotics, computer-controlled machines, and intelligent systems; automated manufacturing processes; and industrial process control.

Control systems play a vital part in the development of technologies for the aforementioned applications. Feedback is an enabling technology in a variety of application areas and has been reinvented numerous times [1]. Traditionally, feedback control adjusts the dynamic behavior of a system. The operation of the system is observed through measurement and the feedback response is adjusted by comparison to an ideal reference input. Even if the dynamic behavior is not well known or if external disturbances are present, with feedback control, the dynamic behavior can still respond properly. Therefore, it is important that engineering systems operate reliably and efficiently under a multitude of conditions [1]. It is specifically this aspect of control that ensures robustness to uncertainty. In the modern technological world, feedback control systems are essential.

Modern control system architectures are composed of several key components, as shown in Fig. 1.1. This particular architecture consists of a *trajectory generator/optimizer* and a *feedback control system*. The trajectory generator provides a feasible feedforward or open-loop control reference trajectory. It is desirable that the reference trajectory be *optimized* with respect to a performance objective that reaches for the ideal dynamic behavior of the system, keeping in mind the presence of system and actuation constraints. Theoretically,

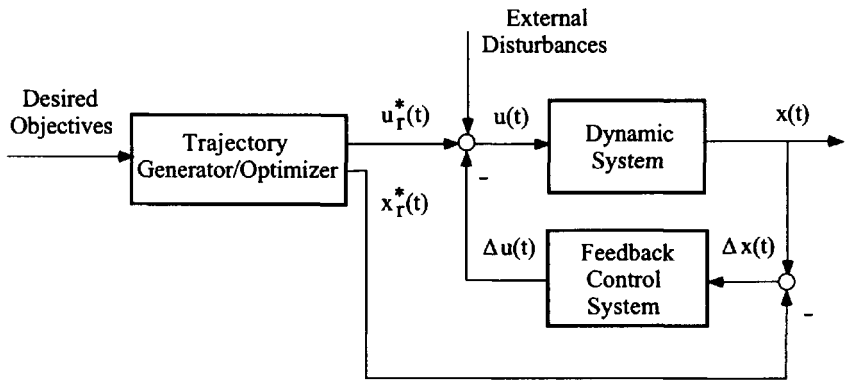


Figure 1.1. *Architecture consists of a trajectory generator/optimizer and feedback control system.*

if $\Delta x = 0$, then $\Delta u = 0$ (see Fig. 1.1). Then the *dynamic system* would result, ideally, in optimized performance $(u_r^*(t), x_r^*(t))$. However, in real-world applications, there will always be inaccuracies. The feedback control system provides stability and robustness around the reference trajectory, subject to inherent modeling uncertainty and external disturbance [2].

This approach has the advantage that the system is tracking a feasible *optimized* trajectory while maintaining stable system response. The *desired objectives* can be commanded by the operator or preprogrammed autonomously. Typically, this higher level of input may consist of set points or regulation, pass through of several way-points, or tracking of specific targets.

The key role that dynamic programming (DP) plays in the control system design is to provide an ideal trajectory optimizer or path planner [3] that can generate arbitrarily smooth input profiles with no initial guess (i.e., zero values) for the input parameters. The primary goal of this book is to apply DP to the optimization of dynamical systems. Three illustrative applications in the field of robotics are given next that quickly demonstrate the ability of DP to meet these needs.

1.2 Illustrative Applications

As an illustration, conventional rigid robot trajectory planning approaches are typically based on inverse kinematics techniques [3, 4]. These techniques are appropriate in many situations; however, in some important instances they are not. Such instances arise whenever robot flexibility or dynamics is of importance. For example, inverse kinematics schemes cannot accommodate structural flexibility in high payload-to-robot weight ratios [5]. Another feature of inverse kinematics solutions is that trajectories are not unique for redundant robots. The purpose of this section is to present an alternative approach for trajectory planning, based on the DP method, that is applicable to rigid [3], flexible [3, 5], and redundant [3] robots.

The optimal trajectory planning problem for rigid robots is posed as a linear time-invariant system for both nonredundant and redundant configurations. Examples 1.1 and 1.2

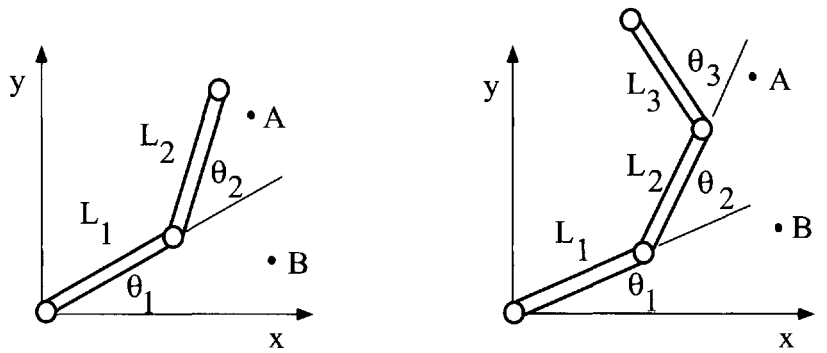


Figure 1.2. *Sketch of planar robots considered in Examples* 1.1 *and* 1.2.

demonstrate this approach by generating a pair of trajectories for end-effector tracking of a straight line with rest-to-rest motions of planar two-link and three-link rigid robots. The generality of the DP method is presented in Example 1.3. A single-link manipulator with a flexible payload is optimally slewed through a set of rest-to-rest maneuvers. The problem is posed as a nonlinear system with state variable equality constraints and input inequality constraints. The simulation results of the single-link manipulator display interesting symmetries characteristic of such motions.

Examples 1.1 and 1.2 deal with straight line tracking, rest-to-rest maneuvers of rigid two-link (see Fig. 1.2, *left schematic*) and three-link (see Fig. 1.2, *right schematic*) robots moving in a plane. The goal is to move the tip of the terminal link from point A to point B while tracking a straight line connecting the two points and bringing the robot to rest at time T.

Joint trajectories for such maneuvers are not unique; that is, there is an infinite number of joint time histories that meet the above-stated goal. The optimal trajectory is defined as the one that minimizes the function J, that is,

$$J = \sum_{k=1}^{N} \sum_{j=1}^{N_j} \left(u_k^j\right)^2,$$

where N is the number of time steps, N_j is the number of joints, $u_k^J = \ddot{\theta}_j$, and $\tau_k = (k-1)t/T$. Note that differentiation is with respect to nondimensional $\tau = t/T$. The function J can be thought of as a measure of the smoothness of the trajectory, and the nondimensional tip position is defined as $\bar{x} = x/L_1$ and $\bar{y} = y/L_1$.

Example 1.1. *Consider a two-link robot with* $\theta_{1A} = \pi/4$, $\theta_{2A} = \pi/12$, $\bar{x}_B = 1.8$, $\bar{y}_B = -0.2$, $L_2/L_1 = 1$, *and* $N = 201$. *The states for this problem are the joint angles and the joint velocities, and the inputs are the joint accelerations.*

Solution

The state vector is $\mathbf{x}_k = [\theta_1(t_k), \theta_2(t_k), \dot{\theta}_1(t_k), \dot{\theta}_2(t_k)]^T$ for a massless two-link robot. The dynamic model is linear since one is integrating the input vector $\mathbf{u}_k = [\ddot{\theta}_1(t_k), \ddot{\theta}_2(t_k)]^T$.

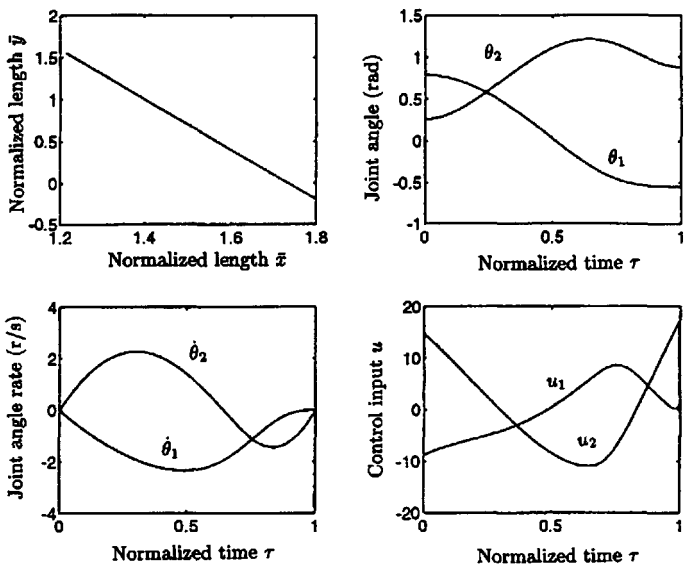


Figure 1.3. *Straight line tracking rest-to-rest motion of the two-link robot considered in Example* 1.1.

The constraints are defined via the forward kinematics, which gives the tip motion that is supposed to track a straight line. Plots of the states and inputs as functions of τ are shown in Fig. 1.3. Also shown in Fig. 1.3 is the path followed by the tip of the second link. Note that the straight line path is tracked and the robot is at rest at the conclusion of the maneuver. It is worth mentioning again that the trajectory shown in Fig. 1.3 is one of many that can accomplish the goal of the stated maneuver.

Example 1.2. *Consider a three-link robot with* $\theta_{1A} = \pi/4$, $\theta_{3A} = \pi/6$, $\bar{x}_B = 2.6$, $\bar{y}_B = -0.4$, $L_2/L_1 = 1$, *and* $N = 201$.

Solution

Plots of the states, inputs, and tip path are shown in Fig. 1.4. Note again that the straight line path is tracked and the robot comes to rest at the endpoint. It is worth mentioning that there is not a one-to-one correspondence between the joint angles for straight line tracking in the case of the three-link robot. This is in contrast to the situation for the two-link robot, in which there is only a single value of θ_2 for each value of θ_1. The redundant DOF introduced by the additional link in this example poses no computational difficulties since the minimization of J leads to a unique solution.

Example 1.3. *Consider a single-link robot with a flexible payload.*

This example is concerned with the rest-to-rest maneuvers of a single-link robot with a flexible payload. A sketch of the model showing the relevant parameters is given in Fig. 1.5.

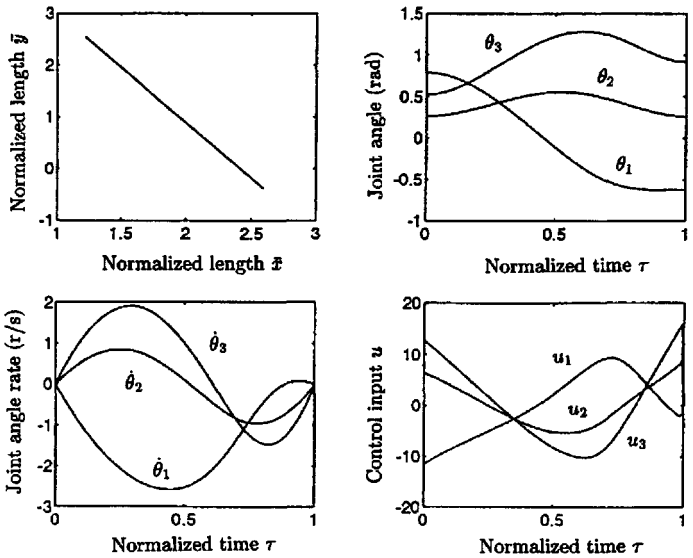


Figure 1.4. *Straight line tracking rest-to-rest motion of the three-link robot considered in Example* 1.2.

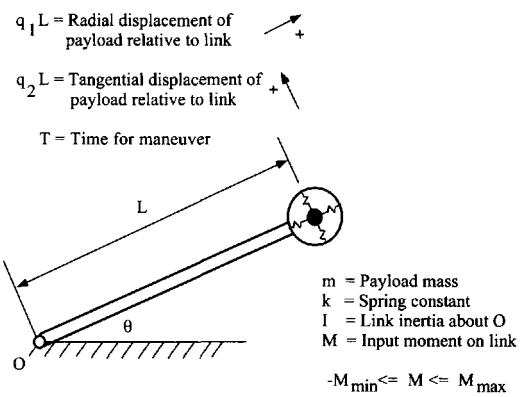


Figure 1.5. *Sketch of the single-link manipulator with flexible payload considered in Example* 1.3.

The equations of motion in dimensionless form are given by

$$\begin{bmatrix} c_1 + (1+q_1)^2 + q_2^2 & -q_2 & 1+q_1 \\ -q_2 & 1 & 0 \\ 1+q_1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} c_2 u - 2[(1+q_1)\dot{q}_1 + q_2\dot{q}_2]\dot{\theta} \\ 2\dot{q}_2\dot{\theta} + (1+q_1)(\dot{\theta})^2 - c_3 q_1 \\ -2\dot{q}_1\dot{\theta} + q_2(\dot{\theta})^2 - c_3 q_2 \end{bmatrix}, \tag{1.1}$$

where $c_1 = I(mL^2)$, $c_2 = AT^2/(mL^2)$, $c_3 = (k/m)T^2$, and $u = M/A$ is the input subject to the constraints $-1 \leq u \leq 1$. For this example, the optimal rest-to-rest trajectory is defined as the one that minimizes the function J defined as

$$J = \sum_{k=1}^{N} u_k^2,$$

where $u_k = u(\tau_k)$.

Solution

Since closed-form solutions of (1.1) are generally not available, the DP algorithm made use of a numerical integration scheme, given in (3.16) in Chapter 3, to help determine the g_k's and their first two derivatives. In this example, a fixed-step, fourth-order Runge–Kutta (RK) numerical integration scheme with $N = 201$ was used.

The results presented in Fig. 1.6 show the states and inputs as functions of τ for a maneuver with $\theta(0) = 0, \theta(1) = 1/2$ rad, $c_1 = 1, c_2 = 5$, and $c_3 = (8\pi)^2$. The symmetries of the inputs and states about $\tau = 1/2$ are evident in Fig. 1.6. Similar symmetries have been observed with the control of a flexible rod [6] and for planar, time-optimal, rest-to-rest maneuvers of undamped flexible satellites [7].

The effect of varying the payload stiffness on the optimal torque profile is also shown in Fig. 1.6. Results are presented for the two values of c_3 indicated in Fig. 1.6. As c_3 increases, the torque profile approaches the one for a rigid payload (i.e., single-link rigid robot). As c_3 decreases, a critical value is reached, below which the rest-to-rest maneuver

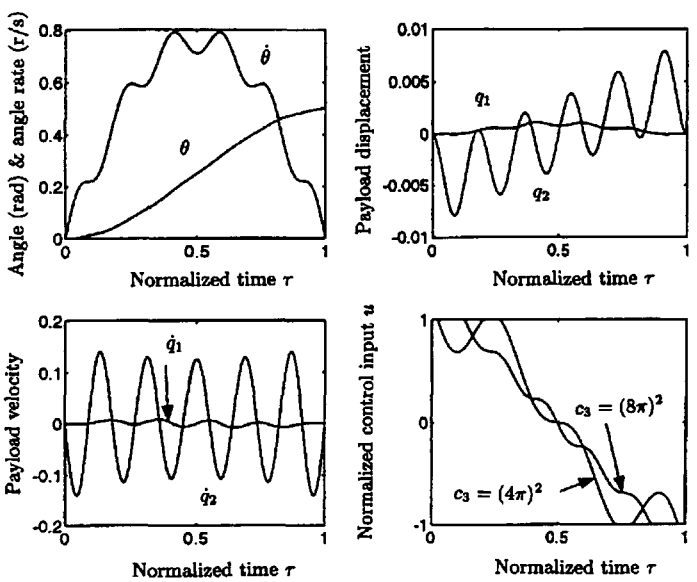


Figure 1.6. *Slewing results for the single-link robot with flexible payload in Example* 1.3.

is not possible. This lower bound is an important constraint to consider for the control of flexible payloads where the final angle and final time are specified (i.e., may not be attainable).

1.3 Summary

This chapter introduced the key role that DP plays in control system design. Several descriptive examples applied in the robotics field were reviewed. In the next chapter, constrained optimization concepts are introduced that allow for a baseline methodology to be established for the remainder of the book. Several diversified examples are presented that employ recursive quadratic programming techniques. Chapter 3 introduces the basics of discrete dynamic programming and demonstrates its ability to find smooth input shaping profiles and optimal trajectories from zero initial guesses. In Chapter 4, advanced DP techniques are developed for full nonlinear dynamic systems. These include the direct enforcement of equality and inequality constraints on the states and controls. These techniques are implemented in Chapter 5 as applied case studies. Each case study application follows a systematic set of design steps.

The main emphasis of this book is DP applied to the optimization of dynamical systems that deal with robotic and aerospace applications. However, the systematic incremental presentation of the theoretical development is included in Chapters 3 and 4.

This page intentionally left blank

Chapter 2
Constrained Optimization

2.1 Introduction

Dynamic programming (DP) is an example of a method for constrained optimization. Therefore the stage must be set for its development with a comparison to other constrained optimization algorithms to enable one to see how DP fits into this area. Because one simply cannot do justice to this topic in one chapter, a substitute is offered by developing and assembling the concepts necessary to implement one of the more time-tested methods—that of recursive quadratic programming (RQP) [8]. This method combines the intuitiveness of a minimizing surface approximation with a robust method for the inclusion of general nonlinear constraints via linearization.

2.1.1 Nomenclature for Iterative Optimization Methods

Constrained optimization is synonymous with nonlinear optimization, an iterative process, and as such will require repeated computation of the following components:

1. A scalar cost function value of general nonlinear form, $f(\mathbf{x})$, which is the optimization criteria or figure of merit and is configured to be a function of an n-length vector of decision variables, $\mathbf{x}$.

2. An n-length vector of first-order derivatives of the cost with respect to the decision variables or cost gradient:

$$\nabla \mathbf{f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}.$$

3. An approximation, $\mathbf{B}(\mathbf{x})$ (see (2.22)), to the set of second-order derivatives of the cost function, which is the $n \times n$ Hessian matrix:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}.$$

4. An l-length vector of nonlinear equality constraints

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_l(\mathbf{x}) \end{bmatrix} = [\mathbf{0}]$$

and an m-length vector of nonlinear inequality constraints, $\mathbf{h}(\mathbf{x}) \leq [\mathbf{0}]$, where $n > l + m$ and $[\mathbf{0}]$ is a vector of zeros. The constraints $\mathbf{g}(\mathbf{x})$ and those components of $\mathbf{h}(\mathbf{x})$ satisfying $h_i(\mathbf{x}) = 0$ are labeled as "active."

5. The $l \times n$ and $m \times n$ matrices, respectively, of first-order derivatives of the constraint functions with respect to the decision variables, or constraint gradient matrices, $\nabla \mathbf{g}(\mathbf{x})$ and $\nabla \mathbf{h}(\mathbf{x})$. These are of the form

$$\nabla \mathbf{g}(\mathbf{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_2} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_l}{\partial x_1} & \frac{\partial g_l}{\partial x_2} & \cdots & \frac{\partial g_l}{\partial x_n} \end{bmatrix},$$

where individual constraint equations are $g_i(\mathbf{x})$, $i = 1, \ldots, l$, and individual decision variables are x_k, $k = 1, \ldots, n$.

It will be assumed that the functions $\mathbf{f}(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$, and $\mathbf{h}(\mathbf{x})$ are smooth and differentiable by analytic or approximate (i.e., finite difference) means. The constrained optimal search will ultimately provide a local minimum decision vector, $\mathbf{x}^*$, which generates the minimum value for cost that still allows the constraints to be satisfied.

2.2 Unconstrained Optimization—The Descent Property

The goal is to generate an iterative scheme to find a minimum that will update a current decision vector, $\mathbf{x}^j$, via an n-length search direction vector, $\mathbf{p}^j$, such that the next iterate is expressed as

$$\mathbf{x}^{j+1} = \mathbf{x}^j + \alpha \mathbf{p}^j \tag{2.1}$$

for some positive step size coefficient, α, to be determined, where j is the iterate index. Derived next are the required concepts, starting from the minimization of a nonlinear, unconstrained, cost function. The updated decision vector, $\mathbf{x}^{j+1}$, can be expanded about a current decision vector, $\mathbf{x}^j$, to first order using the Taylor series approximation

$$f(\mathbf{x}^{j+1}) = f(\mathbf{x}^j) + \nabla \mathbf{f}(\mathbf{x}^j)^T(\mathbf{x}^{j+1} - \mathbf{x}^j) + \text{H.O.T.}, \tag{2.2}$$

where H.O.T. is higher-order terms and $\nabla \mathbf{f}(\mathbf{x}^j)$ is the gradient vector of the cost function evaluated about the current decision vector:

$$\left[\left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\mathbf{x}^j} \right].$$

The assumption on the expansion in (2.2) is that the H.O.T. are less dominant in size. This can be restated with the use of the search direction definition (2.1) to be the change in the cost function from current to updated decision vectors given by

$$\nabla f^{j+1} = f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j) = \alpha \nabla \mathbf{f}(\mathbf{x}^j)^T \mathbf{p}^j + \text{H.O.T.} \tag{2.3}$$

If the goal is to minimize, then it is desired that the iterate-to-iterate change Δf^{j+1} be not less than zero, which implies that

$$\nabla \mathbf{f}(\mathbf{x}^j)^T \mathbf{p}^j \leq 0$$

for positive α. This is commonly known as the descent property. The domain of search vectors that satisfies (2.4) is the "usable" region for cost.

A common choice for descent to satisfy (2.4) is to choose the search direction, $\mathbf{p}^j = -\nabla \mathbf{f}(\mathbf{x}^j)$, that gives rise to the gradient or steepest descent update algorithm for unconstrained functions:

$$\mathbf{x}^{j+1} = \mathbf{x}^j + \alpha \mathbf{p}^j = \mathbf{x}^j - \alpha \nabla \mathbf{f}(\mathbf{x}^j). \tag{2.4}$$

Steepest descent search directions at two different points, $\mathbf{x}^j$, are shown in Fig. 2.1 for a cost that is a function of a single decision variable. The direction of motion depends on the sign of $\nabla \mathbf{f}(\mathbf{x}^j)$ for a single decision variable and forces the update toward the local minimum, $\mathbf{x}^*$.

The step size parameter, α, may be set as a constant based on trial and error to attempt to maintain a smooth descent, or it may be adaptive. Typically it is set as the result of a "one-dimensional search" along the $\nabla \mathbf{f}(\mathbf{x}^j)$ search direction until a minimum in $f(\mathbf{x}^{j+1})$ in this direction has been reached. This could be done by finding three points, $\mathbf{x}'$, which have an "up-down-up" behavior in the cost, $f(\mathbf{x}')$, and then fitting them with a parabola to find an iterate minimum, as shown in Fig. 2.2.

The cost function alone has been used here for determining α, the extent of movement along the search direction for the unconstrained cost. For constrained cases, the cost or "merit" function will be augmented with penalty terms, which include the active constraint residuals, to form a composite function used to deduce the extent of the step in the $\mathbf{p}^j$ direction.

Repeated use of the algorithm in (2.4) will continue local descent in the cost function (to first order) as long as $\nabla \mathbf{f}(\mathbf{x}^j)^T \mathbf{p}^j < 0$. When $\nabla \mathbf{f}(\mathbf{x}^j)^T \mathbf{p}^j = 0$, change stops and an

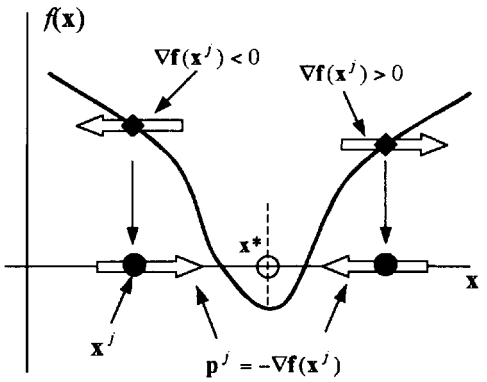


Figure 2.1. *Depiction of steepest descent for single decision variable.*

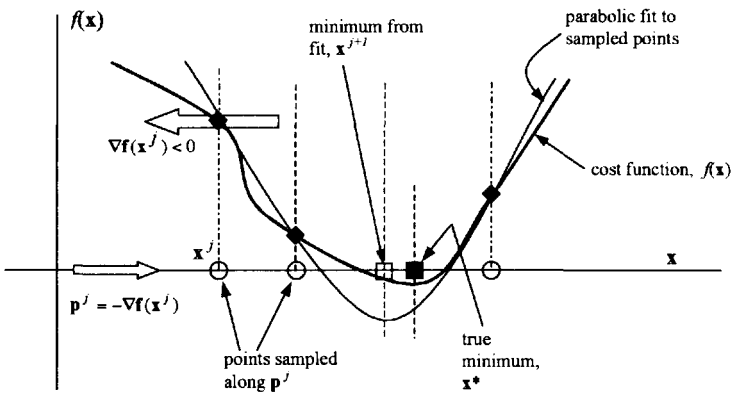


Figure 2.2. *Minimization along the search direction using cost as a merit function.*

observation must be made about the two components. If a point has been reached where there is no further change in a given direction, then the two possibilities are (1) either $\nabla \mathbf{f}(\mathbf{x}^j) = 0$ or $\mathbf{p}^j = 0$ and (2) $\nabla \mathbf{f}(\mathbf{x}^j)$ and $\mathbf{p}^j$ are orthogonal. Since $\mathbf{p}^j = -\nabla \mathbf{f}(\mathbf{x}^j)$, the two have the same magnitude, and the only way to maintain this relation is that both must go to zero at this "extremum" point, $\mathbf{x}^*$.

The fact that $\nabla \mathbf{f}(\mathbf{x}^*) = [\mathbf{0}]$ at an extremum, or point of no further local change in a given direction, is a statement of the justly famous "necessary" condition. Both it and the fact that $\mathbf{p}^j = [\mathbf{0}]$ at the local minimum are used as measures of convergence to the solution, $\mathbf{x}^*$, where convergence can be defined as

$$\left|\left|\nabla \mathbf{f}(\mathbf{x}^j)\right|\right| \leq \varepsilon_1, \quad \left|\left|\mathbf{p}^j\right|\right| \leq \varepsilon_2 \quad \text{as} \quad \mathbf{x}^j \to \mathbf{x}^*, \tag{2.5}$$

where $||\cdot||$ indicates the l_2-norm of the vectors and ε_1, ε_2 are suitably small numbers.

Since the choice of $\mathbf{p}^j = -\nabla \mathbf{f}(\mathbf{x}^j)$ was predicated as being one of descent, this supposition can be bolstered by expanding the unconstrained cost function to second order

and evaluating it about the extremum condition $\nabla \mathbf{f}(\mathbf{x}^*) = [\mathbf{0}]$:

$$f(\mathbf{x}^{j+1}) = f(\mathbf{x}^*) + [\mathbf{0}] + \frac{1}{2}(\mathbf{x}^{j+1} - \mathbf{x}^*)^T \nabla^2 \mathbf{f}(\mathbf{x}^*)(\mathbf{x}^{j+1} - \mathbf{x}^*) + \text{H.O.T.}, \tag{2.6}$$

$$\Delta f^{j+1} = \frac{1}{2}\alpha^2 \mathbf{p}^{jT} \nabla^2 \mathbf{f}(\mathbf{x}^*)\mathbf{p}^j + \text{H.O.T.}, \tag{2.7}$$

where $\nabla^2 \mathbf{f}(\mathbf{x}^*)$ is the matrix of second partial derivatives with respect to the decision variables, evaluated at the extremum, $\mathbf{x}^*$. If the local minimum has been attained, then a small step in any direction, $\mathbf{p}^j$, should result in an increase in the cost, $\Delta f^{j+1} > 0$. If it is now assumed that the second-order effects are dominant at a local extremum, then for any arbitrarily small change at the extremum point, $\mathbf{x}^*$, one must conclude that $\nabla^2 \mathbf{f}(\mathbf{x}^*) \geq \mathbf{0}$ (i.e., it is positive definite) at a local minimum, which is the equally famous "sufficiency" condition.

2.3 Including the Constraints

2.3.1 Equality Constraints

Now that some important criteria for minima have been set forth for an unconstrained problem, how does one involve constraints? First, one must look at the inclusion of equality constraints, $\mathbf{g}(\mathbf{x}^j) = [\mathbf{0}]$, which will be accomplished through the use of an l-length vector of constant Lagrange multipliers, $\boldsymbol{\lambda}^j$, to produce an augmented cost function

$$L(\mathbf{x}^j, \boldsymbol{\lambda}^j) = f(\mathbf{x}^j) + \boldsymbol{\lambda}^{jT} \mathbf{g}(\mathbf{x}^j). \tag{2.8}$$

Using the Taylor series mode of expansion, the updated value of L can be stated as

$$\begin{aligned}
L(\mathbf{x}^{j+1}, \boldsymbol{\lambda}^{j+1}) &= L(\mathbf{x}^j, \boldsymbol{\lambda}^j) + \left.\frac{\partial L}{\partial \mathbf{x}}\right|_{\mathbf{x}^j} \left(\mathbf{x}^{j+1} - \mathbf{x}^j\right) + \left.\frac{\partial L}{\partial \boldsymbol{\lambda}^j}\right|_{\boldsymbol{\lambda}^j} \left(\boldsymbol{\lambda}^{j+1} - \boldsymbol{\lambda}^j\right) + \text{H.O.T.}, \\
L(\mathbf{x}^{j+1}, \boldsymbol{\lambda}^{j+1}) &= L(\mathbf{x}^j, \boldsymbol{\lambda}^j) + (\nabla \mathbf{f}(\mathbf{x}^j) + \boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^j))^T \left(\mathbf{x}^{j+1} - \mathbf{x}^j\right) \\
&\quad + \mathbf{g}(\mathbf{x}^j) \left(\boldsymbol{\lambda}^{j+1} - \boldsymbol{\lambda}^j\right) + \text{H.O.T.}, \\
\Delta L(\mathbf{x}^{j+1}, \boldsymbol{\lambda}^{j+1}) &= \alpha(\nabla \mathbf{f}(\mathbf{x}^j) + \boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^j))^T \mathbf{p}^j + \text{H.O.T.},
\end{aligned} \tag{2.9}$$

since $\mathbf{x}^{j+1} - \mathbf{x}^j = \mathbf{p}^j$ and $(\partial L/\partial \boldsymbol{\lambda})|_{\boldsymbol{\lambda}^j} = \mathbf{g}(\mathbf{x}^j) = [\mathbf{0}]$ if the constraint is satisfied. (Again, constraints of the form $\mathbf{g}(\mathbf{x}^j) = [\mathbf{0}]$ are known as active constraints.)

Initiating descent in the augmented cost, L, to first order implies that

$$\left.\frac{\partial L}{\partial \mathbf{x}}\right|_{\mathbf{x}^j} (\mathbf{x}^{j+1} - \mathbf{x}^j) = \alpha(\nabla \mathbf{f}(\mathbf{x}^j) + \boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^j))^T \mathbf{p}^j \leq 0 \tag{2.10}$$

for positive α. Descent can be established by choosing $\mathbf{p}^j = -(\nabla \mathbf{f}(\mathbf{x}^j) + \boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^j))$. This states that the linear combination of the cost gradient and the constraint gradients modified by their Lagrange multipliers is opposite and collinear to the search vector. This is commonly known as the descent property [9]. This was graphically depicted by Vanderplaats [10] and is shown in Fig. 2.3. (This figure shows contours of constant cost, $f_i(\mathbf{x}) = c_i$, as functions of a two-variable decision vector. The cost increases from c_1 to c_4 over the contours. Superimposed on these cost contours is a single equality constraint, $g(\mathbf{x})$.)

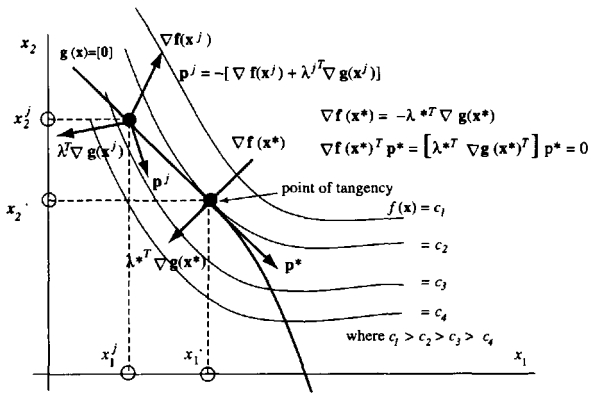


Figure 2.3. *Evolution of the constrained optimal solution.*

Descent in the augmented cost function continues in Fig. 2.3 until $(\nabla \mathbf{f}(\mathbf{x}^j) + \boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^*))^T \mathbf{p}^* = 0$, which, as in the unconstrained case, implies that the gradient of the augmented cost, L, with respect to the decision variables is

$$\left.\frac{\partial L}{\partial \mathbf{x}}\right|_{\mathbf{x}^j} = \nabla \mathbf{f}(\mathbf{x}^*) + \boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^*) = \mathbf{p}^* = [\mathbf{0}]. \tag{2.11}$$

However, now the geometry is somewhat more involved. In order for

$$\nabla \mathbf{f}(\mathbf{x}^*) + \boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^*) = [\mathbf{0}] \tag{2.12a}$$

to be true,

$$\nabla \mathbf{f}(\mathbf{x}^*) = -\boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^*) \tag{2.12b}$$

must be true, which means that the cost gradient is the opposite of the linear combination of the equality constraint gradients modified by their Lagrange multipliers. (In Fig. 2.3, there is only one constraint, $g(\mathbf{x}) = 0$, and therefore the inclusion of λ is unnecessary and $\nabla \mathbf{f}(\mathbf{x}^*)$ can be depicted as opposite to $\nabla \mathbf{g}(\mathbf{x}^*)$.)

$\mathbf{p}^j$ was initially opposed to the linear combination, $\nabla \mathbf{f}(\mathbf{x}^j) + \boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^j)$. As they approached the extremum, $\mathbf{x}^*$, the two additive components ended up opposing each other. The only way that all of these conditions can be maintained is for $\mathbf{p}^*$ to be orthogonal to both $\nabla \mathbf{f}(\mathbf{x}^*)$ and $\boldsymbol{\lambda}^{jT} \nabla \mathbf{g}(\mathbf{x}^*)$, as shown in Fig. 2.3. For the case of a single constraint, the fact that the cost and constraint gradients (i.e., the directions of maximum change in the cost, $\mathbf{f}(\mathbf{x})$, and constraint vector, $\mathbf{g}(\mathbf{x})$), with respect to the decision variables, oppose each other at $\mathbf{x}^*$, and that $\mathbf{p}^*$ is orthogonal to both at this point, implies that the curves $\mathbf{g}(\mathbf{x}) = [\mathbf{0}]$ and $\mathbf{f}(\mathbf{x}) = \mathbf{c}_2$ are tangent at $\mathbf{x}^*$ for the minimum constrained cost, $\mathbf{c}_2$!

2.3.2 Inequality Constraints and the Karush–Kuhn–Tucker Conditions

Now that equality constraints have been considered, how about inequality ones, $\mathbf{h}(\mathbf{x}^j) \leq 0$? Again an augmented cost function form will be used, but with a different set of constraints

in the form of constant Lagrange multipliers, $\boldsymbol{\nu}$. However, since Lagrange multipliers are a creative way of attaching "zeros," what does one do with that inequality sign?

One uses the concept of slack variables to convert the inequality to an equality constraint. Therefore, $\mathbf{h}(\mathbf{x}) \leq 0$ becomes $\mathbf{h}(\mathbf{x}) + \mathbf{s}^2 = [\mathbf{0}]$ for an m-length vector of slack variables, $\mathbf{s}$. If the current decision vector, $\mathbf{x}^j$, makes one or more ($i \leq 1, \ldots, m$) of the constraints active, then the components $(s_i^j)^2 = 0$; otherwise, $(s_i^j)^2 > 0$. If $h_i(\mathbf{x}^j) < 0$, these constraints are satisfied and *inactive* as far as having an effect on the computed value of $\mathbf{x}_j$ and are removed from consideration. (Slack variables are used to expedite this discussion and are not used in the implementation of the RQP algorithm.)

The augmented cost function is

$$L(\mathbf{x}^j, \boldsymbol{\nu}^j, \mathbf{s}^j) = f(\mathbf{x}^j) + \boldsymbol{\nu}^{jT}\left(\mathbf{h}(\mathbf{x}^j) + (\mathbf{s}^j)^2\right). \tag{2.13}$$

The previous expansion technique gives

$$\begin{aligned}
L(\mathbf{x}^{j+1}, \boldsymbol{\nu}^{j+1}, \mathbf{s}^{j+1}) &= L(\mathbf{x}^j, \boldsymbol{\nu}^j, \mathbf{s}^j) + \left.\frac{\partial L}{\partial \mathbf{x}}\right|_{\mathbf{x}^j}\left(\mathbf{x}^{j+1} - \mathbf{x}^j\right) \\
&\quad + \left.\frac{\partial L}{\partial \boldsymbol{\nu}}\right|_{\boldsymbol{\nu}^j}\left(\boldsymbol{\nu}^{j+1} - \boldsymbol{\nu}^j\right) + \left.\frac{\partial L}{\partial \mathbf{s}}\right|_{\mathbf{s}^j}\left(\mathbf{s}^{j+1} - \mathbf{s}^j\right) + \text{H.O.T.}, \\
L(\mathbf{x}^{j+1}, \boldsymbol{\nu}^{j+1}, \mathbf{s}^{j+1}) &= L(\mathbf{x}^j, \boldsymbol{\nu}^j, \mathbf{s}^j) + \left(\nabla\mathbf{f}(\mathbf{x}^j) + \boldsymbol{\nu}^{jT}\nabla\mathbf{h}(\mathbf{x}^j)\right)^T\left(\mathbf{x}^{j+1} - \mathbf{x}^j\right) \\
&\quad + \left(\mathbf{h}(\mathbf{x}^j) + (\mathbf{s}^j)^2\right)^T\left(\boldsymbol{\nu}^{j+1} - \boldsymbol{\nu}^j\right) + 2\boldsymbol{\nu}^{jT}\mathbf{S}^j\left(\mathbf{s}^{j+1} - \mathbf{s}^j\right) + \text{H.O.T.}, \\
\Delta L(\mathbf{x}^{j+1}, \boldsymbol{\nu}^{j+1}, \mathbf{s}^{j+1}) &= \alpha\left(\nabla\mathbf{f}(\mathbf{x}^j) + \boldsymbol{\nu}^{jT}\nabla\mathbf{h}(\mathbf{x}^j)\right)^T\mathbf{p}^j \\
&\quad + \left(\mathbf{h}(\mathbf{x}^j) + (\mathbf{s}^j)^2\right)^T\left(\boldsymbol{\nu}^{j+1} - \boldsymbol{\nu}^j\right) + 2\boldsymbol{\nu}^{jT}\mathbf{S}^j\left(\mathbf{s}^{j+1} - \mathbf{s}^j\right) + \text{H.O.T.},
\end{aligned} \tag{2.14}$$

where

$$\mathbf{S}^j = \begin{bmatrix} s_1^j & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_m^j \end{bmatrix}$$

is a diagonal matrix. The second term in the expansion disappears since the first variation, $\partial(\Delta L)/\partial\boldsymbol{\nu} = h(\mathbf{x}^j) + (\mathbf{s}^j)^2$, is equal to $[\mathbf{0}]$ in the "feasible region," leaving the following:

$$\Delta L(\mathbf{x}^{j+1}, \boldsymbol{\nu}^{j+1}, \mathbf{s}^{j+1}) = \alpha\left[\nabla\mathbf{f}(\mathbf{x}^j) + \boldsymbol{\nu}^{jT}\nabla\mathbf{h}(\mathbf{x}^j)\right]^T\mathbf{p}^j + 2\boldsymbol{\nu}^{jT}\mathbf{S}^j\left(\mathbf{s}^{j+1} - \mathbf{s}^j\right) + \text{H.O.T.} \tag{2.15}$$

As before, it is desired to maintain the descent property by choosing

$$\mathbf{p}^j = -\left(\nabla f(\mathbf{x}^j) + \boldsymbol{\nu}^{jT}\nabla\mathbf{h}(\mathbf{x}^j)\right). \tag{2.16}$$

However, it is necessary to deal with the final first-order term to establish conditions for descent applicable to the multipliers, $\boldsymbol{\nu}^j$. This can be done by choosing $\boldsymbol{\nu}^j$ such that the term $2\boldsymbol{\nu}^{jT}\mathbf{S}^j(\mathbf{s}^{j+1} - \mathbf{s}^j) \leq 0$.

If the approximation

$$2\boldsymbol{\nu}^{jT}\mathbf{S}^j\left(\mathbf{s}^{j+1}-\mathbf{s}^j\right)=\boldsymbol{\nu}^{jT}\left(2\mathbf{S}^j(\mathbf{s}^{j+1}-\mathbf{s}^j)\right)\approx\boldsymbol{\nu}^{jT}(2\mathbf{S}^j d(\mathbf{s}^j))\approx\boldsymbol{\nu}^{jT}d((\mathbf{s}^j)^2) \tag{2.17}$$

is made, where the $d()$ notation indicates a differential, $d() = (d()/ds)ds$, then as a move is made from off constraint, $h(\mathbf{x}^j) < 0$, to on constraint, $h(\mathbf{x}^j) = 0$, $(\mathbf{s}^j)^2$ transitions from greater than zero to zero. Therefore, $d((\mathbf{s}^j)^2) \leq 0$ and descent is maintained (i.e., $2\boldsymbol{\nu}^{jT}\mathbf{S}^j(\mathbf{s}^{j+1}-\mathbf{s}^j)\leq 0$) in the vicinity $\mathbf{h}(\mathbf{x}^j)=[\mathbf{0}]$ if $\boldsymbol{\nu}^j \geq [\mathbf{0}]$.

Next, combine the results from the equality and inequality constraint treatments to yield the following first-order necessary conditions for optimality that must occur at a local constrained extremum, $\mathbf{x}^*$:

1. $$\frac{\partial(\Delta L^{j+1}(\mathbf{x}^*))}{\partial\mathbf{p}}=\nabla\mathbf{f}(\mathbf{x}^*)+\left(\nabla\mathbf{g}(\mathbf{x}^*)\right)^T\boldsymbol{\lambda}+\left(\nabla\mathbf{h}(\mathbf{x}^*)\right)^T\boldsymbol{\nu}=[\mathbf{0}] \quad (n \text{ equations}), \tag{2.18a}$$

2. $$\frac{\partial(\Delta L^{j+1}(\mathbf{x}^*))}{\partial\boldsymbol{\lambda}}=\mathbf{g}(\mathbf{x}^*)=[\mathbf{0}] \quad (l \text{ equations}), \tag{2.18b}$$

3. $$\frac{\partial(\Delta L^{j+1}(\mathbf{x}^*))}{\partial\boldsymbol{\nu}}=\mathbf{h}(\mathbf{x}^*)\leq[\mathbf{0}], \qquad \boldsymbol{\nu}^*\geq[\mathbf{0}] \quad (2m \text{ equations}). \tag{2.18c}$$

These results are commonly referred to as the Kuhn–Tucker [11, 12] conditions or, more contemporarily, the Karush–Kuhn–Tucker (KKT) conditions after the respective independent developments. Though enough conditions exist to solve for the unknowns, $\mathbf{x}^*$, $\boldsymbol{\lambda}^*$, and $\boldsymbol{\nu}^*$, a problem form is not yet available that can be used to produce an iterative algorithm. To provide the additional pieces, let us digress a bit.

2.4 Improving the Search with Second-Order Methods

2.4.1 Newton's Method

The major drawback to first-order optimization methods is just that—only gradient information (or *slope* in simplest terms) is considered. Execution proceeds without knowing how fast the minimum is approached. In other words, there is no mechanism available to take advantage of the function's *curvature*. If one were to use a first-order method with a constant step size, α, to find the local minimum, progress would tend to be fraught with "chattering" about the solution as the minimum is approached. This manifests itself as the algorithm overstepping the minimum enough to defeat the convergence criteria in (2.5). By overstepping, the gradient, $\nabla\mathbf{f}(\mathbf{x}^j)$, reverses itself and the same phenomenon happens in the opposite direction and repeats until the gradient and the search vector, $\mathbf{p}^j$, slowly decrease in magnitude enough to satisfy the convergence norm. A one-dimensional search on α expedites this process, but still with more iterations than needed. Second-order methods provide dramatic improvement.

Returning to the unconstrained problem, a search direction that includes second-order derivatives (i.e., curvature) will be generated. If one were to expand the *gradient*, $\nabla\mathbf{f}(\mathbf{x})$,

about some known iterate value, j, the result would be

$$\nabla \mathbf{f}(\mathbf{x}^{j+1}) = \nabla \mathbf{f}(\mathbf{x}^j) + \nabla^2 \mathbf{f}(\mathbf{x}^j)\left(\mathbf{x}^{j+1} - \mathbf{x}^j\right) + \text{H.O.T.} \tag{2.19}$$

If the H.O.T. are eliminated and a substitution for the update vector, $(\mathbf{x}^{j+1} - \mathbf{x}^j)$, is made in terms of the search vector, $\alpha \mathbf{p}^j$, the following is produced:

$$\nabla \mathbf{f}(\mathbf{x}^{j+1}) = \nabla \mathbf{f}(\mathbf{x}^j) + \alpha \nabla^2 \mathbf{f}(\mathbf{x}^j)\mathbf{p}^j. \tag{2.20}$$

At the extremum point, $\mathbf{x}^*$, one has the gradient $\nabla \mathbf{f}(\mathbf{x}^{j+1}) = \nabla \mathbf{f}(\mathbf{x}^*) = [\mathbf{0}]$, which if substituted above provides a nonlinear equation that can be solved by Newton's method as follows:

$$\nabla \mathbf{f}(\mathbf{x}^j) + \alpha \nabla^2 \mathbf{f}(\mathbf{x}^j)\mathbf{p}^j = \mathbf{0} \quad \rightarrow \quad \mathbf{p}^j = -\left[\nabla^2 \mathbf{f}(\mathbf{x}^j)\right]^{-1} \nabla \mathbf{f}(\mathbf{x}^j), \tag{2.21}$$

where α has been temporarily dropped. This search direction solution is aptly known as the Newton update form.

This is certainly a concise form, which, given a reasonably simple cost function, would be easily implemented. Life being what it is, however, most useful problems involve a cost formulation that is complex enough to require approximate methods to compute just the vector of gradients, $\nabla \mathbf{f}(\mathbf{x}^j)$, let alone the Hessian, $\mathbf{H}(\mathbf{x}^j) = \nabla^2 \mathbf{f}(\mathbf{x}^j)$.

One could take a cue from the approximation of gradient vectors via finite differences and do the same to approximate the Hessian. If one can evaluate the cost n times about $\mathbf{x}^j$ to compute finite differences for $\nabla \mathbf{f}(\mathbf{x}^j)$, then the cost could be evaluated a second set of n times about some $\mathbf{x}^j + \Delta \mathbf{x}$ to produce $\nabla \mathbf{f}(\mathbf{x}^j + \Delta \mathbf{x})$. These two vectors could be differenced and divided by the various combinations of $\Delta x_i \Delta x_k, k = 1, \ldots, n$, to approximate the symmetric, positive definite matrix $\mathbf{H}(\mathbf{x}^j)$.

The problems with this are as follows: (1) $2n$ cost evaluations are needed at each iterate, which could be quite costly; (2) the computational error may be compounded by having two levels of approximation (one for the gradients and the second for the Hessian); and (3) there is no guarantee of maintaining the positive definiteness of $\mathbf{H}(\mathbf{x}^j)$.

2.4.2 Updating the Hessian for Curvature Input

A more successful scheme categorized under the heading of Broyden methods was devised to approximate $\mathbf{H}(\mathbf{x}^j)$ iteratively by updating a current approximate value, $\mathbf{B}^j$, with the latest decision variables, cost gradients, and search directions, which together are assumed to carry curvature information. The starting point for developing this update is to assume an expansion of the gradient of the form

$$\begin{aligned} \nabla \mathbf{f}(\mathbf{x}^{j+1}) &= \nabla \mathbf{f}(\mathbf{x}^j) + \mathbf{H}(\mathbf{x}^{j+1})\left(\mathbf{x}^{j+1} - \mathbf{x}^j\right) \\ \Rightarrow \nabla \mathbf{f}(\mathbf{x}^{j+1}) - \nabla \mathbf{f}(\mathbf{x}^j) &= \mathbf{H}(\mathbf{x}^{j+1})\left(\mathbf{x}^{j+1} - \mathbf{x}^j\right) \Rightarrow \mathbf{q}^j = \mathbf{B}^{j+1}\mathbf{p}^j, \end{aligned} \tag{2.22}$$

where $\mathbf{q}^j$ is the gradient difference, $\mathbf{B}^j$ is the current approximation to the Hessian, $\mathbf{p}^j$ is the current search direction, and α has been temporarily dropped. The above result can be restated as

$$\mathbf{q}^j = \mathbf{B}^{j+1}\mathbf{p}^j = \left(\mathbf{B}^j + \mathbf{C}^j\right)\mathbf{p}^j \quad \Rightarrow \quad \mathbf{C}^j\mathbf{p}^j = \mathbf{q}^j - \mathbf{B}^j\mathbf{p}^j, \tag{2.23}$$

where $\mathbf{C}^j$ is an additive update matrix and $\mathbf{B}^j$ is the last iterate. Based on the form of (2.23), new information to update the Hessian approximation is gained along only a single direction (i.e., $\mathbf{p}^j$) since it is assumed that $\mathbf{q}^j$ is small in the vicinity of the minimum. Therefore, the iterates, $\mathbf{q}^j$, should differ by a small amount. It has been suggested that this difference could be characterized by matrix corrections of low rank. The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [13, 14, 15, 16] (after the simultaneous independent developments in 1970) sets this correction to be a combination of the current gradient difference and current search direction as

$$\mathbf{C}^j = \frac{\mathbf{q}\mathbf{q}^T}{\mathbf{q}^T\mathbf{p}} - \frac{\mathbf{B}\mathbf{p}\mathbf{p}^T\mathbf{B}}{\mathbf{p}^T\mathbf{B}\mathbf{p}}, \tag{2.24}$$

where the jth update subscript is implicit for all quantities on the right-hand side. This form can be substituted in (2.23) to prove the equality. Note that the two components of $\mathbf{C}^j$ involve outer products of the vectors $\mathbf{q}^j$ and $\mathbf{p}^j$ with their respective selves creating the difference of two symmetric, rank-one matrices (preserving the symmetry of $\mathbf{B}^{j+1}$). As such, the BFGS scheme is known as a rank-two update method. The update is

$$\mathbf{B}^{j+1} = \mathbf{B}^j + \mathbf{C}^j, \tag{2.25}$$

where the initial $\mathbf{B}^0$ is usually set to the identity matrix, $\mathbf{I}$, and the new search direction can be generated by solving the linear system

$$\mathbf{B}^{j+1}\mathbf{p}^{j+1} = -\nabla\mathbf{f}(\mathbf{x}^{j+1}), \tag{2.26}$$

which enforces the Newton form. The BFGS Hessian algorithm update procedure is given as follows.

Step 1: Guess $\mathbf{x}^0$, set $\mathbf{B}^0 = \mathbf{I}$, and compute $\mathbf{p}^0 = -\nabla\mathbf{f}(\mathbf{x}^0)$.

Step 2: Compute $\mathbf{x}^1 = \mathbf{x}^0 + \alpha\mathbf{p}^0$.

Step 3: Evaluate $f(\mathbf{x}^1)$, $\nabla\mathbf{f}(\mathbf{x}^1)$, and $\mathbf{q}^0 = \nabla\mathbf{f}(\mathbf{x}^1) - \nabla\mathbf{f}(\mathbf{x}^0)$.

Step 4: Compute $\mathbf{C}^0 = \mathbf{C}(\mathbf{q}^0, \mathbf{p}^0, \mathbf{B}^0)$.

Step 5: Compute $\mathbf{B}^1 = \mathbf{B}^0 + \mathbf{C}^0$.

Step 6: Solve $\mathbf{B}^1\mathbf{p}^1 = -\nabla\mathbf{f}(\mathbf{x}^1)$ for $\mathbf{p}^1$.

Step 7: Go to Step 2 and iterate (i.e., compute $\mathbf{x}^2 = \mathbf{x}^1 + \alpha\mathbf{p}^1$).

Initially, $\mathbf{B}^0 = I$ is positive definite and the fact that it is updated with rank-one terms that are outer products should, in theory, maintain its symmetry. Dennis and Schnabel [17] discuss a Cholesky factorization for the BFGS method that ensures positive definiteness for this computation.

2.5 The Quadratic Programming Problem

2.5.1 Solving the Linear System of Unknowns

At this point, enough of the supporting cast is available to develop the RQP problem. To allow the use of the KKT conditions, it is assumed that the cost function can be expressed as a quadratic expression and that the equality and inequality constraints can be linearized. The statement of the quadratic problem (QP) for constrained optimization is as follows:

$$\underset{\mathbf{p}^j}{Minimize} \quad \Delta f^{j+1}(\mathbf{p}^j) = \nabla \mathbf{f}^{jT}\mathbf{p}^j + \frac{1}{2}\mathbf{p}^{jT}\mathbf{B}^j\mathbf{p}^j \tag{2.27}$$

subject to

$$\mathbf{g}(\mathbf{x}^j) + \nabla \mathbf{g}(\mathbf{x}^j)\mathbf{p}^j = [\mathbf{0}],$$
$$\mathbf{h}(\mathbf{x}^j) + \nabla \mathbf{h}(\mathbf{x}^j)\mathbf{p}^j \leq [\mathbf{0}].$$

This QP variant is one that yields an analytic solution. A sample quadratic cost surface and linearized constraint are contrasted against their nonlinear "parents" and are shown in Fig. 2.4.

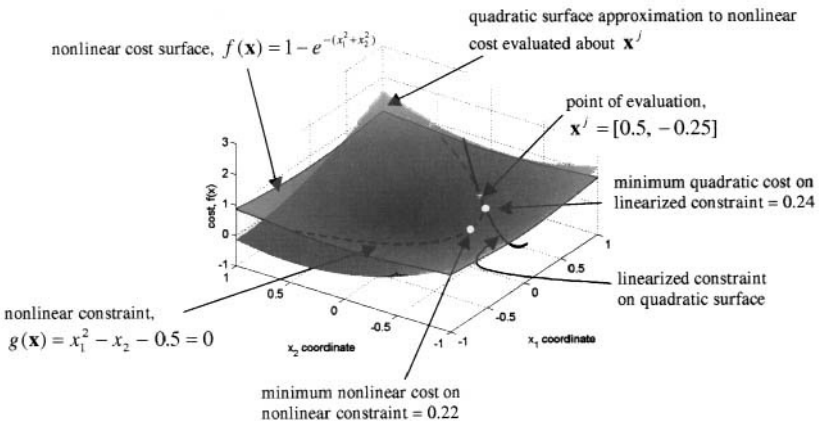


Figure 2.4. *QP features.*

To solve for the RQP search direction and the requisite Lagrange multipliers for a given iterate, $\mathbf{x}^j$, form the augmented cost function as

$$\Delta L^{j+1}(\mathbf{p}^j, \boldsymbol{\lambda}^{jT}, \boldsymbol{\nu}^{jT}, \mathbf{s}^j) = \nabla \mathbf{f}^{jT}\mathbf{p}^j + \frac{1}{2}\mathbf{p}^{jT}\mathbf{B}^j\mathbf{p}^j + \boldsymbol{\lambda}^{jT}\left(\mathbf{g}(\mathbf{x}^j) + \nabla \mathbf{g}(\mathbf{x}^j)\mathbf{p}^j\right)$$
$$+ \boldsymbol{\nu}^{jT}\left(\mathbf{h}(\mathbf{x}^j) + \nabla \mathbf{h}(\mathbf{x}^j)\mathbf{p}^j + (\mathbf{s}^j)^2\right). \tag{2.28}$$

This can be simplified by noting that only those parts of the inequality constraints that are on the boundary are active, $\mathbf{h}_{eq}(\mathbf{x}^j) = [\mathbf{0}]$, and affect the solution. $\mathbf{h}_{eq}$ is used to represent the active subset m' of the set of m inequality constraints. As the solution

progresses, the algorithm must test the inequalities and determine which are active. The active subset of $\mathbf{h}(\mathbf{x}^j)$ is lumped with the equality constraints, $\mathbf{g}(\mathbf{x})$, to form a composite set of constraints

$$\boldsymbol{\Psi} = \begin{bmatrix} g_i(\mathbf{x}),\ i = 1, \ldots, l \\ h_{eq_i}(\mathbf{x}),\ i = l+1, \ldots, l+m' \end{bmatrix} = [\mathbf{0}] \text{ for } i = 1, \ldots, l+m'. \tag{2.29}$$

Next, substitute (2.29) into (2.28) and neglect the inactive constraints (i.e., $(\mathbf{s}^j)^2 \neq 0$) to yield

$$\Delta L^{j+1}(\mathbf{p}^j, \boldsymbol{\eta}^j) = \nabla \mathbf{f}(\mathbf{x}^j)^T \mathbf{p}^j + \frac{1}{2} \mathbf{p}^{jT} \mathbf{B}^j \mathbf{p}^j + \boldsymbol{\eta}^{jT} \left(\boldsymbol{\Psi}(\mathbf{x}^j) + \nabla \boldsymbol{\Psi}(\mathbf{x}^j) \mathbf{p}^j \right), \tag{2.30}$$

where

$$\boldsymbol{\eta}^j = \begin{bmatrix} \boldsymbol{\lambda}^j \\ v_i^j,\ i = 1, \ldots, m' \end{bmatrix}.$$

Now, returning to the KKT conditions (2.18), an analytic form suitable for use in an iterative method can be derived using

$$\left. \frac{(\partial \Delta L^{j+1}(\mathbf{x}^j))}{\partial \mathbf{p}} \right|_{\mathbf{p}^j} = \nabla \mathbf{f}(\mathbf{x}^j) + \mathbf{B}^j \mathbf{p}^j + \left(\nabla \boldsymbol{\Psi}(\mathbf{x}^j) \right)^T \boldsymbol{\eta}^j = [\mathbf{0}] \qquad (n \text{ equations}), \tag{2.31a}$$

$$\left. \frac{(\partial \Delta L^{j+1}(\mathbf{x}^j))}{\partial \boldsymbol{\eta}} \right|_{\boldsymbol{\eta}^j} = \boldsymbol{\Psi}(\mathbf{x}^j) + \nabla \boldsymbol{\Psi}(\mathbf{x}^j) \mathbf{p}^j = [\mathbf{0}] \qquad (l+m' \text{ equations}). \tag{2.31b}$$

By dropping the j-iterate evaluation nomenclature, (2.31) can be rearranged into the linear system

$$\begin{bmatrix} \mathbf{B} & \nabla \boldsymbol{\Psi}^T \\ \nabla \boldsymbol{\Psi} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \boldsymbol{\eta} \end{bmatrix} = \begin{bmatrix} -\nabla \mathbf{f} \\ -\boldsymbol{\Psi} \end{bmatrix}. \tag{2.32}$$

Using a variant of the matrix inversion lemma [18] and assuming that the matrix $\nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T$ is invertible (if $n > l + m'$), an inverse,

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11}(n \times n) & \mathbf{A}_{12}(n \times q) \\ \mathbf{A}_{21}(q \times n) & \mathbf{A}_{22}(q \times q) \end{bmatrix},$$

of the system matrix in (2.32) can be found (where $q = l + m'$) such that

$$\begin{bmatrix} \mathbf{p} \\ \boldsymbol{\eta} \end{bmatrix} = \mathbf{A} \begin{bmatrix} -\nabla \mathbf{f} \\ -\boldsymbol{\Psi} \end{bmatrix}, \tag{2.33}$$

where

$$\begin{aligned} \mathbf{A}_{11} &= \mathbf{B}^{-1} - \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T (\nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T)^{-1} \nabla \boldsymbol{\Psi} \mathbf{B}^{-1}, \\ \mathbf{A}_{12} &= \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T (\nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T)^{-1}, \\ \mathbf{A}_{21} &= (\nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T)^{-1} \nabla \boldsymbol{\Psi} \mathbf{B}^{-1}, \\ \mathbf{A}_{22} &= -(\nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T)^{-1}. \end{aligned}$$

Then, η and $\mathbf{p}$ can be solved for as

$$\eta^j = (\nabla\Psi\mathbf{B}^{-1}\nabla\Psi^T)^{-1}(\Psi - \nabla\Psi\mathbf{B}^{-1}\nabla\mathbf{f}), \tag{2.34}$$

$$\mathbf{p}^j = -\mathbf{B}^{-1}(\nabla\Psi)^T(\nabla\Psi\mathbf{B}^{-1}\nabla\Psi^T)^{-1}\left(\Psi - \nabla\Psi\mathbf{B}^{-1}\nabla\mathbf{f}\right) - \mathbf{B}^{-1}\nabla\mathbf{f}, \tag{2.35}$$

where all items on the right-hand sides of (2.34) and (2.35) are evaluated about the $\mathbf{x}^j$ iterate. Note that the last term in (2.35) is the search direction that would result from an unconstrained Newton-type search update for the minimum (i.e., rearrange (2.26)) and the first term is the modification of that direction due to the constraints. The Hessian approximation, $\mathbf{B}$, is updated via the algorithm following (2.26).

2.5.2 A Merit Function for One-Dimensional Search

Knowing the search direction allows one to find the updated vector of decision variables via $\mathbf{x}^{j+1} = \mathbf{x}^j + \alpha\mathbf{p}^j$, except for the matter of which step size α is the best to choose. The previous discussion in Section 2.2 for unconstrained optimization centered on using the cost function, $f(\mathbf{x})$, as a merit function. In the original work by Powell [8], a merit form is proposed to include the cost as well as the constraints. This function includes the constraints through the use of penalty terms and is given as

$$\phi(\mathbf{x}^j, \boldsymbol{\mu}^j) = f(\mathbf{x}^j) + \sum_{i=1}^{l} \mu_i^j \left|g_i(\mathbf{x}^j)\right| + \sum_{i=l+1}^{l+m} \mu_i^j \left|\max(0, h_i(\mathbf{x}^j)\right|, \tag{2.36}$$

where the vector $\boldsymbol{\mu}^j$ is a set of positive weights to be determined and $|\cdot|$ indicates absolute value. Note that for the case of minimization, any violated constraint, $g_i(x) \neq 0$ or $h_i(x) > 0$, will result in increases or nonzero "penalties" in $\phi(\mathbf{x}, \boldsymbol{\mu})$ and therefore provides feedback mechanisms on the size of the step to traverse along $\mathbf{p}^j$ to produce a new $\mathbf{x}^{j+1}$. One uses (2.36) by guessing a value of α, computing $\mathbf{x}^{j+1}$ along $\mathbf{p}^j$, and evaluating $\phi(\mathbf{x}, \boldsymbol{\mu})$ for $\mathbf{x}^{j+1}$. In order to use this function more efficiently, researchers have developed techniques to set the values of $\boldsymbol{\mu}^j$ adaptively. Powell [8] suggested making the weights μ_i^j approximately equal to $|\eta_i^j|$. This would make them a function of the active constraints. A second suggestion, also due to Powell, was to retain some history on those constraints that may have become inactive near the solution and use the following form of the weights:

$$\mu_i^{j+1} = \max\left[|\eta_i^{j+1}|, \frac{1}{2}(\mu_i^j + |\eta_i^{j+1}|)\right], \quad i = 1, \ldots, l \qquad \text{(active constraints)}, \tag{2.37}$$

where μ_i^j is the previous iterate weight per constraint.

2.5.3 A Measure of Convergence and the Recursive Quadratic Programming Algorithm

In the unconstrained case, the "zeroing" of both the gradient, $\nabla\mathbf{f}(\mathbf{x}^j)$, and search direction, $\mathbf{p}^j$, was shown to be a useful condition for assessing convergence to $\mathbf{x}^*$. The KKT conditions in (2.31) provide the counterpart to this for the constrained case. The test for convergence

to the minimum for the set of active constraints can be restated from (2.5) as

$$||\nabla \mathbf{f}(\mathbf{x}^j) + \mathbf{B}^j \mathbf{p}^j + (\nabla \boldsymbol{\Psi}(\mathbf{x}^j))^T \boldsymbol{\eta}^j|| \leq \varepsilon_1, \quad ||\mathbf{p}^j|| \leq \varepsilon_2 \tag{2.38}$$

for suitable small numbers, ε_1 and ε_2.

The basic RQP algorithm sequence is as follows.

Step 1: Guess $\mathbf{x}^0$, set $\mathbf{B}^0 = \mathbf{I}$ (identity matrix), and set the iteration index to $j = 0$.

Step 2: Evaluate $f(\mathbf{x}^j)$, $\mathbf{g}(\mathbf{x}^j)$, and $\mathbf{h}(\mathbf{x}^j)$.

Step 3: Check the inequalities, $\mathbf{h}(\mathbf{x}^j)$, for the active set ($\mathbf{h}_{eq}(\mathbf{x}) = [h_i(\mathbf{x}^j)] \geq [\mathbf{0}]$, where the $>$ sign indicates violation) and form

$$\boldsymbol{\Psi}(\mathbf{x}) = \begin{bmatrix} \mathbf{g}(\mathbf{x}) \\ \mathbf{h}_{eq}(\mathbf{x}) \end{bmatrix}.$$

Step 4: Evaluate the gradients, $\nabla \mathbf{f}(\mathbf{x}^j)$ and $\nabla \boldsymbol{\Psi}(\mathbf{x}^j)$.

Step 5: Solve the QP problem:

a) $\boldsymbol{\eta}^j = (\nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T)^{-1} \left(\boldsymbol{\Psi} - \nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \mathbf{f}\right)$,

b) $\mathbf{p}^j = -\mathbf{B}^{-1}(\nabla \boldsymbol{\Psi})^T (\nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \boldsymbol{\Psi}^T)^{-1} (\boldsymbol{\Psi} - \nabla \boldsymbol{\Psi} \mathbf{B}^{-1} \nabla \mathbf{f}) - \mathbf{B}^{-1} \nabla \mathbf{f}$.

Step 6: If $j > 0$, then compute the update to $\mathbf{B}$ via $\mathbf{B}^j \mathbf{p}^j = -\nabla \mathbf{f}(\mathbf{x}^j)$.

Step 7: Check for convergence:

$$||\nabla \mathbf{f}(\mathbf{x}^j) + \mathbf{B}^j \mathbf{p}^j + (\nabla \boldsymbol{\Psi}(\mathbf{x}^j))^T \boldsymbol{\eta}^j|| \leq \varepsilon_1, \quad ||\mathbf{p}^j|| \leq \varepsilon_2.$$

Step 8: If convergence is satisfied, then stop. Otherwise, execute a one-dimensional search along $\mathbf{p}^j$ via the following:

a) Initially compute the penalty weights. On the first iteration, set $\mu_i^j = |\eta_i^j|$. Otherwise, set

$$\mu_i^j = \max\left[|\eta_i^j|, \frac{1}{2}(\mu_i^{j-1} + |\eta_i^j|)\right].$$

b) Next, solve for $\mathbf{x}^{j+1}$ by varying $\alpha = \alpha_k$ such that $\mathbf{x}_k^{j+1} = \mathbf{x}^j + \alpha_k \mathbf{p}^j$ to minimize

$$\phi(\mathbf{x}_k^{j+1}, \boldsymbol{\mu}^j) = f(\mathbf{x}_k^{j+1}) + \sum_{i=1}^{l} \mu_i^j |g_i(\mathbf{x}_k^{j+1})| + \sum_{i=l+1}^{l+m'} \mu_i^j |\max(0, (h(\mathbf{x})_k^{j+1}))|.$$

Step 9: Set $j = j + 1$ and go to Step 2.

This concludes the development of the RQP algorithm as developed by Powell [8]. Newer formulations, as implemented in MATLAB [19], have integrated other forms of constraints that include hard bounds on the decision variables, i.e., $\mathbf{x}_{lower} \leq \mathbf{x}^j \leq \mathbf{x}_{upper}$. Such bounds can always be included by varying the set size, $\alpha \mathbf{p}^j$, until the most seriously violated bound is satisfied.

Several examples of RQP follow to provide insight into the diversity of approaches afforded for mainstream engineering modeling and simulation.

For those wishing to dig deeper into the aforementioned concepts, as well as explore the expanse of material in the field of optimization, a very complete gateway site on the World Wide Web is kept by the Optimization Technology Center of Northwestern University at http://www.ece.northwestern.edu/OTC [20].

In Section 2.6, the MATLAB Optimization Toolbox [19] is used to facilitate the implementation of the RQP development.

2.6 Applications of Constrained Minimization

2.6.1 Implementation of the Recursive Quadratic Programming Algorithm

The function *fmincon* from the MATLAB Optimization Toolbox [19] was used to implement the RQP algorithm for several engineering applications. *fmincon* finds the minimum of a constrained nonlinear multivariable function. It attempts to accomplish the following problem statement:

$$\underset{\mathbf{x}^j}{Minimize} \qquad f(\mathbf{x})$$

subject to

$$\begin{aligned} \mathbf{h}(\mathbf{x}^j) &\leq [\mathbf{0}], \\ \mathbf{g}(\mathbf{x}^j) &= [\mathbf{0}], \\ \mathbf{x}_{lower} \leq \mathbf{x}^j &\leq \mathbf{x}_{upper}, \\ \mathbf{D} \cdot \mathbf{x}^j &\leq \mathbf{f}, \\ \mathbf{D}_{eq} \cdot \mathbf{x}^j &= \mathbf{b}_{eq}, \end{aligned} \tag{2.39}$$

where $\mathbf{b}$ and $\mathbf{b}_{eq}$ are fixed vectors, $\mathbf{x}_{lower}$ and $\mathbf{x}_{upper}$ are vectors of decision variable bounds, $\mathbf{D}$ and $\mathbf{D}_{eq}$ are fixed matrices, $\mathbf{h}(\mathbf{x}^j)$ and $\mathbf{g}(\mathbf{x}^j)$ are vectors of inequality and equality constraints, and $f(\mathbf{x}^j)$ is the scalar cost function. Callable routines are written by the user to return $f(\mathbf{x}^j)$, $\mathbf{h}(\mathbf{x}^j)$, and $\mathbf{g}(\mathbf{x}^j)$, which can be of arbitrary nonlinear form [19]. $\mathbf{x}^j$ is the vector of current decision variables for which to solve. Though *fmincon* provides the above linear constraint formulations (i.e., $\mathbf{D} \cdot \mathbf{x}^j \leq \mathbf{b}$), only the nonlinear forms will be exercised in the subsequent examples.

The Optimization Toolbox provides a variety of optimization methods. To enhance the operational commonality of these routines, entries in a common vector of user-specified parameters can be set to tailor such items as convergence tolerances, derivative computations, numbers of allowable computations, various bounds, and output specifications to the user's liking.

Three applications from the areas of welding, missile trajectory optimization, and satellite control have been solved to demonstrate the flexibility of RQP. The welding example (see Section 2.6.2) is an example of a maximization goal cast in *fmincon*'s minimization framework. This example is small enough in the number of decision variables that the structure of the optimization geometry can be depicted. The missile trajectory problem (see Section 2.6.3) provides a classic example of the decision variables representing parameterized open-loop control histories. The vehicle's flight control solution demonstrates nonintuitive behavior and is derived starting from remote initial variable guesses. The final example, involving satellite control (see Section 2.6.4), will demonstrate how a problem can be reshaped when the number of apparent active constraints is at least equal to the size of the decision vector (i.e., $l + m' \geq n$).

2.6.2 Optimizing Efficiencies in the Laser Welding Process

Laser welding has gained considerable acceptance in modern-day manufacturing due to its ability to concentrate the joining energy in a very confined space of interest while minimizing collateral heating. This makes it ideal for welding extremely small parts with precious little real estate between a weld joint and an adjacent precision mechanism that could be damaged by heating. In autogenous welding (which employs no filler metals, but fuses adjacent parts by heating regions of them to a molten state), the weld characteristics are governed by a set of decision variables; this is called the weld procedure. The procedure represents the settings on the device to produce a weld with specific output characteristics. This procedure contains constant $\mathbf{x}$ values for

1. laser output power, Q_o,
2. part travel speed, V, and
3. focusing lens characterized by spot diameter, D.

Q_o and V can vary continuously over fixed ranges, while D is typically fixed for a given lens. However, for the laser device considered here, a defocusing mechanism will be assumed present to allow a continuously varying D value over a prescribed range.

This example presents a nonlinear constrained optimization problem with three decision variables for which to solve and as such allows the depiction of the solution space. Surfaces representing the variation of the cost and constraints will be displayed as functions of the decision variables.

For the remainder of this discussion, all references to the weld procedure will imply the decision variable vector, $\mathbf{x} = (Q_o, V, D)$.

The model has four output states, $\mathbf{y}$, that are functions of the input weld procedure. These states correlate directly with measurements taken on over 100 welds [21, 22] and are as follows:

1. The energy transfer efficiency (η_t, dimensionless) is defined as the net heat input to the part divided by the energy produced by the power source.
2. The melting efficiency (η_m, dimensionless) is defined as the amount of heat required to just melt the fusion zone (molten area of joining) relative to the net heat input deposited in the part.

3. The weld width (W), in millimeters (mm) is derived from the assumption of a parabolic cross-sectional area, A, of the weld.

4. Weld penetration depth (P) (mm), is defined as the depth of the parabolic area.

This input-output relationship is displayed in Fig. 2.5. The procedure produces a weld of given heat efficiencies and a given size (W, P). A shield gas is used during the process to prevent contamination and weakening of the weld by the constituent gases in the atmosphere.

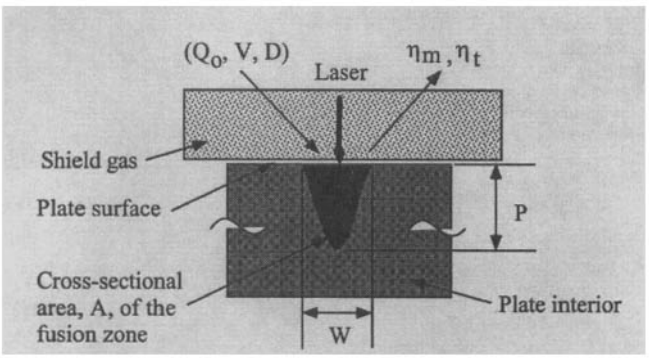


Figure 2.5. *Input-output model for laser welding.*

A semiempirical nonlinear algebraic model, $\mathbf{y} = \mathbf{s}(\mathbf{x})$, describing the states was developed from previous theoretical work and experimental results [21, 23] and is shown below:

$$P = \frac{c_1 \alpha Q_o}{V^{c_2} D^{c_3}}, \tag{2.40a}$$

$$\eta_t = c_4 - c_5^{\left[\frac{\pi}{2 \arctan \frac{c_6 D}{P}}\right]}, \tag{2.40b}$$

$$R_y = \frac{V Q_o \eta_t}{\alpha^2 \delta h}, \tag{2.40c}$$

$$\eta_m = 0.48 - 0.29 e^{\left[\frac{-R_y}{6.82}\right]} - 0.17 e^{\left[\frac{-R_y}{58.8}\right]}, \tag{2.40d}$$

$$A = \frac{R_y \eta_m \alpha^2}{V^2}, \tag{2.40e}$$

where R_y is an intermediate quantity, the Rykalin number [21, 22]; α and δh are the material-dependent thermal diffusivity and heat of enthalpy constants; and the coefficients c_1 to c_9 are constants derived through least-square fits of the aforementioned measurement data [23]. The final quantity, weld width, W, is based on a parabolic approximation of A, where

$$W = \frac{3A}{2P}. \tag{2.40f}$$

As mentioned previously, maximizing the amount of heat deposited in the weld while minimizing collateral heating effects is a much sought after manufacturing condition. In this vein, the maximization of melting efficiency, η_m, during the weld process is ideally

suited for this purpose. The goal will be to maximize η_m (which has a theoretical upper bound of 48%) while attaining (or constraining) a weld of specified dimensions, W and P. From the mathematics of the problem, this will entail maximizing R_y, which in turn requires maximizing V, Q_o, and/or η_t. Since η_t depends on D and P via an empirical relationship, rather simply stated variable descriptions yield decidedly nonlinear behavior. A plot of the problem space depicting the variation of η_m and regions of constant W and P as functions of the weld procedure is displayed in Fig. 2.6.

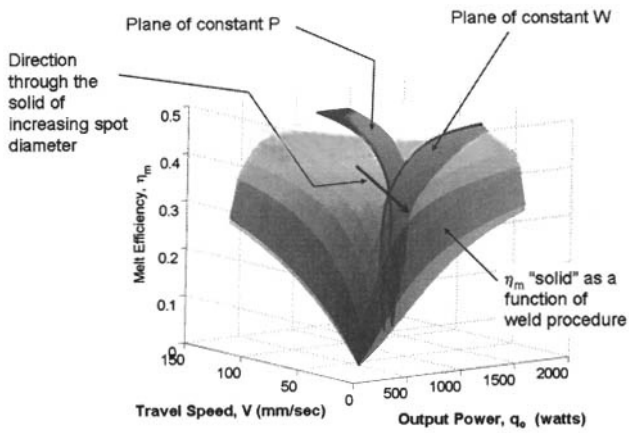


Figure 2.6. *Laser weld problem space.*

The chosen values for W and P in Fig. 2.6 are such that there is an entire family of weld procedures that can attain both values (satisfy both constraints) simultaneously. If a physically attainable weld, (W, P), has been specified according to the model in (2.40), then it is expected that the procedure solution lies along the intersection of the constraint planes to produce a family of Q_o, V pairs that yield the desired size weld. Tracing this intersection of planes upward (direction of decreasing D) to the lower bound of D generates the maximum η_m.

For this application, the problem statement is as follows:

$$\underset{\mathbf{x}^T=(Q_o,V,D)}{\textit{Minimize}} \quad -\eta_m$$

subject to

$$\Psi(\mathbf{x}) = \begin{bmatrix} P(Q_o, V, D) - P_{desired} \\ W(Q_o, V, D) - W_{desired} \end{bmatrix} = 0,$$

$$\begin{bmatrix} Q_{o_{min}} \\ V_{min} \\ D_{min} \end{bmatrix} \leq \begin{bmatrix} Q_o \\ V \\ D \end{bmatrix} \leq \begin{bmatrix} Q_{o_{max}} \\ V_{max} \\ D_{max} \end{bmatrix}. \tag{2.41}$$

Note that the performance measure is $-\eta_m$, indicating that the goal is to maximize a quantity within a minimization framework. Given that the problem statement has two

equality constraints, three decision variables will be sufficient to pose this problem (i.e., $(\nabla\Psi\mathbf{B}^{-1}\nabla\Psi^T)^{-1}$ should exist).

For the numerical example, a weld that has dimensions $W = P = 1$ mm in type-304 stainless steel, a high-strength metal used in the aerospace and auto industries, has been chosen. Limits on the weld procedure variables and materials values for type-304 stainless steel are

$$\begin{bmatrix} 75 \text{ watts} \\ 3 \text{ mm/sec} \\ 0.118 \text{ mm} \end{bmatrix} \leq \begin{bmatrix} Q_o \\ V \\ D \end{bmatrix} \leq \begin{bmatrix} 1600 \text{ watts} \\ 120 \text{ mm/sec} \\ 0.294 \text{ mm} \end{bmatrix},$$
$$\alpha(\text{m}^2/\text{sec}) = 5.18 \times 10^{-6}, \tag{2.42}$$
$$\delta h(\text{joules/m}^3) = 9.41 \times 10^9.$$

As one can see from the bounds in (2.42), these variables can span four to five orders of magnitude. *From initial trial optimization runs, it became clear that it would be necessary to scale both the decision variables and the outputs to their maximum values.* The original measurement data that constructed the model in (2.40) were refit with the scaling. The scaled model used in the computations follows:

$$p = \frac{1}{P_{max}}\left[\frac{c_1\alpha' q_o Q_{o_{max}}}{v^{c_2} d^{c_3}}\right],$$

$$\eta_t' = \frac{1}{\eta_{t_{max}}}\left(c_4 - c_5^{\left[\frac{\pi}{2\arctan\left(\frac{c_6 D}{P}\right)}\right]}\right),$$

$$R_y = \frac{v V_{max} q_o Q_{o_{max}} \eta_t' \eta_{t_{max}}}{\alpha^2 \delta h}, \tag{2.43}$$

$$\eta_m' = \frac{1}{\eta_{m_{max}}}\left(0.48 - 0.29e^{\left[\frac{-R_y}{6.82}\right]} - 0.17e^{\left[\frac{-R_y}{58.8}\right]}\right),$$

$$A = \frac{R_y \eta_m' \eta_{m_{max}} \alpha^2}{(v V_{max})^2},$$

$$w = \frac{1.5}{W_{max}}\left[\frac{A}{p P_{max}}\right],$$

where lowercase and primed quantities indicate scaled values of their uppercase and unprimed counterparts. The fit coefficients, c_i, and maximum values for scaling are shown in Table 2.1. Scaling factors for the weld procedure variables are the maximum values in (2.42). The mathematical description in (2.43) fits the measured weld data that were taken using argon shield gas.

Gradient evaluations of cost and constraint with respect to $\mathbf{x}$ were defaulted to finite difference approximations. Initial $\mathbf{x}$ values were simply chosen as midpoint values in the respective ranges. The use of *fmincon* produced the optimal solution in Table 2.2, shown by the black dot in Fig. 2.7.

Table 2.1. *Weld model fit parameters and variable maximums.*

Coefficient	Value	Unit
c_1	29.33	–
c_2	0.29	–
c_3	0.32	–
c_4	0.90	–
c_5	0.63	–
c_6	0.26	–
P_{max}	5	mm
$\eta_{t_{max}}$	0.95	–
$\eta_{m_{max}}$	0.51	–
W_{max}	15.09	mm
α_{max}	110×10^{-6}	m^2/sec

Table 2.2. *RQP results from fmincon.*

Quantity	Desired Value	Achieved Value	Unit
η_m	max ($\eta_m \leq 0.48$)	0.375	–
η_t	$\eta_t \leq 0.95$	0.68	–
P	1.00	1.00	mm
W	1.00	1.00	mm
Q_o	$75 \leq Q_o \leq 1600$	525.81	watts
V	$3 \leq V \leq 120$	21.34	mm/sec
D	$0.118 \leq D \leq 0.294$	0.118	mm

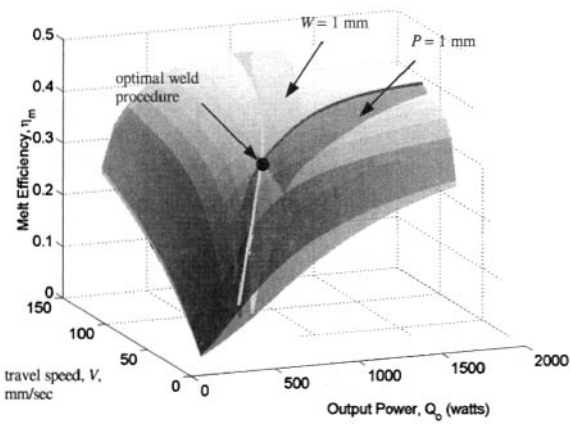


Figure 2.7. *Optimal weld procedure solution for* $W = P = 1$ mm *in type-*304 *stainless steel.*

The weld procedure solution, as surmised previously, lies along the intersection of constraint planes at the minimum value of D to maximize η_m. The solution for the efficiencies is acceptable and does not exceed the theoretical maximums, as shown in Table 2.2. The width and depth were achieved to a relative solution tolerance of 10^{-8}. Solved values for power and speed were intermediate to their respective limits, but the spot diameter

solution retreated to the decision variable lower limit, as foreseen. It is interesting to note that, according to the model, if the lens corresponding to $D = 0.294$ mm were used to achieve the same size weld, η_m would drop to 27% from the optimal 37.5%.

Plots of the convergence histories are shown in Fig. 2.8. The problem essentially converged in two iterations of the RQP algorithm, as shown in Fig. 2.8 (left plot), when the spot size was reduced to its lower bound. Iteration zero comprises the initial conditions. With one of the decision variables fixed at its lower bounds, two decision variables are left to satisfy two constraints. Thus, the remaining iterations display an algebraic solution.

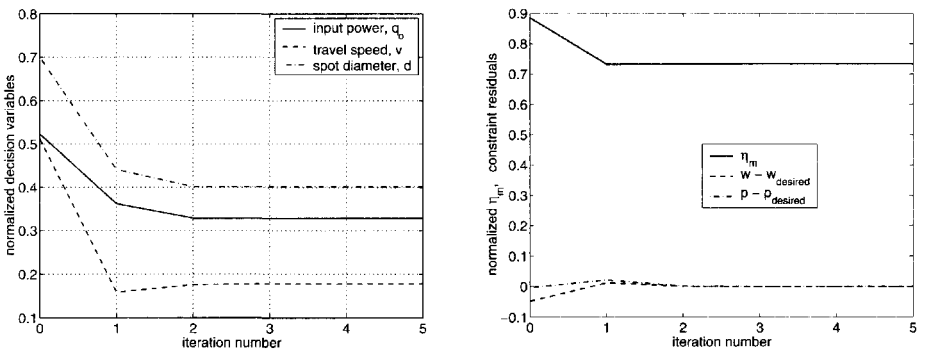


Figure 2.8. *RQP decision variable, cost, and constraint residual convergence histories for weld example.*

In Fig. 2.8 (right plot), the melting efficiency, η_m, cost has converged after one iteration and the constraint residuals have zeroed (i.e., equality constraints have been met) after two iterations. Though the cost was actually $-\eta_m$, the actual efficiency was plotted for clarity. Note that its initial value was reduced to allow satisfaction of the weld size constraints.

This small problem was solved without difficulty by *fmincon*'s application of the RQP algorithm. However, the problem involved disparate scales among the variables that needed to be addressed before the algorithm could be applied effectively.

2.6.3 Maximum Terminal Velocity for Hypersonic Standoff Missiles

A perplexing issue for military campaigns is how to dislodge enemies from caves, underground bunkers, and other well-hardened or deeply buried targets. The use of kinetic energy weapons is attractive because their energy varies as the square of velocity, so maximizing the velocity of a missile or bomb adds considerable energy to whatever explosive package it carries. In addition, the use of maximum velocity to penetrate deep into the ground may allow the coupling of a detonation to the surrounding soil to neutralize hardened targets via mechanical wave propagation.

The application suggested here proposes that a guided hypersonic missile be injected into the vicinity from a remote location via a ballistic missile carrier. Once near targets of interest, it might need to loiter in order to coordinate attacks with other assets, after which it would engage an assigned target.

This application seeks a solution to the optimal flight of an unpowered, gliding vehicle to specified final conditions (Fig. 2.9). This problem has been addressed extensively in the

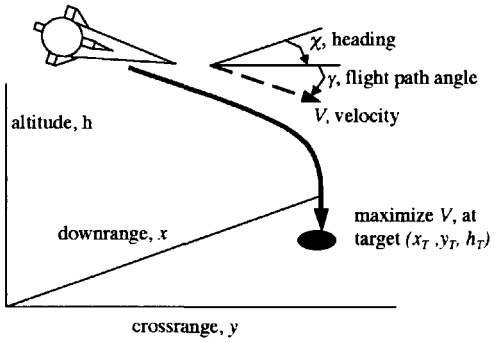


Figure 2.9. *Maximum terminal velocity scenario.*

literature in an attempt to find analytic feedback guidance solutions to maximize relevant metrics such as range and terminal velocity. For this example, a guidance solution would provide continually updated control directives based on measured position and flight path attitude with respect to a target position. To date, a general analytic solution has not been found, but the use of constrained optimization allows one to glean major insights into the behavior of this form of optimal flight [24].

The example in this section presents an application of trajectory optimization via the solution of decision variables that parameterize control histories. These histories transition the system between known trajectory boundary conditions while optimizing flight cost. Each decision variable represents a single value of a control input history at a unique time.

For this study, a skid-to-turn missile with independent, orthogonal controls to provide lift and side force will be considered. The skid-to-turn missile will be initially traveling near Mach 10 at 100,000 ft and, for the ranges covered and relevant performance information desired, can be adequately modeled as a point mass.

The equations of motion for a nonthrusting point mass traveling over a locally flat, nonrotating Earth at an altitude that is small with respect to the radius of the Earth can be expressed in state notation as $\dot{\mathbf{y}} = \mathbf{s}(\mathbf{y}, C_L, C_S, t)$ and in expanded form as

$$\begin{aligned}
\dot{x} &= V\cos\gamma\cos\chi, \\
\dot{y} &= V\cos\gamma\sin\chi, \\
\dot{h} &= V\sin\gamma, \\
\dot{V} &= -\frac{\rho V^2 S_R C_D}{2m} - g_s\sin\gamma, \\
\dot{\gamma} &= \frac{\rho V^2 S_R C_L}{2mV} - \left(\frac{g_s}{V} - \frac{V}{r_s}\right)\cos\gamma, \\
\dot{\chi} &= \frac{\rho V^2 S_R C_S}{mV\cos\gamma},
\end{aligned} \tag{2.44}$$

where x is downrange, y is crossrange, h is altitude, γ is the elevation flight path angle, χ is the heading angle, ρ is the atmospheric density, g_s is the gravitational constant at Earth's

surface, and r_s is Earth's radius. S_R is the vehicle reference area, m is the vehicle mass, and C_L, C_S, and C_D are nondimensional lift control, side-force control, and drag coefficients, respectively.

The following additional assumptions were employed: (1) an exponential atmosphere of $\rho = \rho_o e^{-h/\beta}$, where $\beta = 23{,}800$ ft is the scale height and $\rho_o = 0.002378$ slugs/ft^3 is the surface density, and (2) a parabolic drag polar for a symmetric skid-to-turn vehicle, consisting of a constant and an induced term, $C_D = C_{D_o} + \kappa(C_L^2 + C_S^2)$, where $C_{D_o} = 0.043$, $\kappa = 0.54$, and $\sqrt{C_{D_o}/\kappa}$ is the maximum lift-to-drag ratio. For this example, C_D will only be considered a function of the control inputs.

Given this model, the goal is to solve for the lift and side-force controls as functions of time, $C_L(t)$ and $C_S(t)$, that enable the missile to hit a fixed target while maintaining maximum terminal velocity flight. Since the constrained optimization algorithm has to operate on decision variables that appear as constants in the model, the controls will be changed from continuous forms to vectors of parameterized ones, $\mathbf{x} = [C_L(t_i)\ C_S(t_i)]$ for $i = 1, \ldots, n/2$, at discrete times, as shown in Fig. 2.10. The RQP algorithm will alter $\mathbf{x}$ (depicted by the solid dots) until the terminal velocity cost is maximized and the miss-distance constraints are satisfied to specified tolerances.

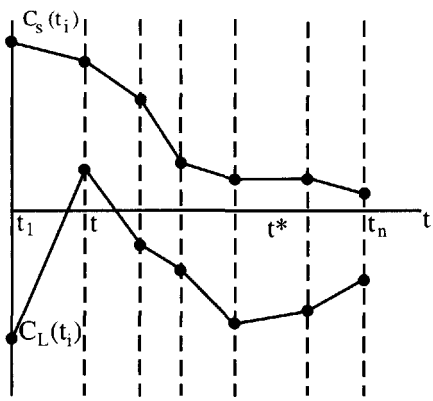


Figure 2.10. *Parameterized lift and side-force control histories.*

The function evaluations, $f(\mathbf{x}^j)$, $\mathbf{h}(\mathbf{x}^j)$, and $\mathbf{g}(\mathbf{x}^j)$, will rely on numerical integration of the state model to a final time, t_f, at which point the final velocity and the residuals between the achieved and desired targets are computed. Since it is not feasible to provide individual control coefficients for every possible time, $\hat{t}$, in Fig. 2.10, linear interpolation within $C_L(t_i)$ and $C_S(t_i)$ will be used to provide $C_L(\hat{t})$ and $C_S(\hat{t})$. The state model is integrated with a fixed-step, fourth-order Runge–Kutta (RK) scheme coded in MATLAB [25]. Model-history generation is divided into 100 time steps, each of which consists of the controls being evaluated at the beginning of the step with the aforementioned interpolation and the RK scheme being used to advance the history to the next step.

One final change is needed to complete the parameterization. Since the final time of the trajectory is unknown, a method is needed to change it, as the changing control coefficients generated by the RQP algorithm cause the point mass vehicle to fly different trajectories and, therefore, different times to the known target. This change is enabled by

normalizing the discrete times by the current t_f. As such, the normalized time span for integration is always 0 to 1 (assuming $t_1 = 0$). The parameterization is completed by adding the current t_f as the final decision variable for which to solve to give the complete decision variable vector,

$$\mathbf{x} = \left[C_L(t_i),\ C_S(t_i) \text{ for } i = 1, \ldots, (n-1)/2, t_f\right], \text{ where } \tau_i = t_i/t_f, \tag{2.45}$$

for n total parameters (i.e., decision variables). The model equations accommodate this change to normalized time by using the relation

$$\dot{\mathbf{y}} = \frac{d\mathbf{y}}{dt} = \frac{d\mathbf{y}}{d\tau}\frac{d\tau}{dt} = \frac{d\mathbf{y}}{d\tau}\frac{1}{t_f} = \mathbf{s}(\mathbf{y}, \mathbf{x}, \tau),$$

$$\frac{d\mathbf{y}}{d\tau} = \dot{\mathbf{y}}' = t_f\mathbf{s}(\mathbf{y}', \mathbf{x}, \tau), \text{ where } \mathbf{y}'(1) = \mathbf{y}'(0) + \int_0^1 \dot{\mathbf{y}}' d\tau, \tag{2.46}$$

where $\mathbf{y}'$ is the time-normalized vector of states.

Now that the parameterization of the model is complete, a formal statement of the constrained optimization application will be made. The goal is to minimize the performance index

$$f(\mathbf{x}) = -V(\tau = 1) \tag{2.47a}$$

subject to

$$\dot{\mathbf{y}}' = \mathbf{s}(\mathbf{y}', \tau) \tag{2.47b}$$

and constraints

$$\mathbf{g}(\mathbf{y}'(\mathbf{x})) = \begin{bmatrix} x'(1) - x_{target} \\ y'(1) - y_{target} \\ h'(1) - h_{target} \end{bmatrix} = 0. \tag{2.47c}$$

The function evaluations will need to provide the performance metric and constraint residual valuations, $f(\mathbf{x}^j)$ and $\mathbf{g}(\mathbf{y}'(\mathbf{x}^j))$, for the current decision vector, $\mathbf{x}^j$. (The use of the nomenclature, $\mathbf{g}(\mathbf{y}'(\mathbf{x}^j))$, is a slight departure from the constraint usage to this point, $\mathbf{g}(\mathbf{x}^j)$, and is meant to imply that the constraint equations are functions of $\mathbf{x}^j$ via their inclusion in the state equations, $\mathbf{y}(\mathbf{x}^j, \tau)$.)

The hypersonic flight problem employs variables of widely disparate magnitudes: $x'(\tau)$, $y'(\tau) \sim \mathcal{O}(10^7)$; $h' \sim \mathcal{O}(10^6)$; $V' \sim \mathcal{O}(10^5)$; and $\gamma'(\tau)$, $\chi'(\tau)$, $C_L(\tau)$, $C_S(\tau) \sim \mathcal{O}(1)$. Note that the $\sim \mathcal{O}()$ notation means "on the order of." Based on this, one should expect nonuniformities in the hyperspace shapes (i.e., greater than three dimensions) of the cost contours. This makes convergence to a solution highly sensitive to initial guesses and more than likely will result in the optimization solution chattering due to the irregular contours as the algorithm tries to update $\mathbf{x}^j$. (The Rosenbrock problem is the quintessential demonstration of this effect on a small problem scale.) Initial attempts to use "raw" velocity for cost and physical miss distances in the three dimensions as constraint residuals resulted in nonconvergent behavior with nonsmooth control histories being produced.

Because of this, the cost and constraint residual computations will be scaled to suitable maximum values in an attempt to "circularize" the problem cost hyperspace as a function of the decision variables. As a result of this, the normalized values of cost and the equality constraint residuals on the target position in three dimensions, $\hat{f}(\mathbf{x})$ and $\hat{\mathbf{g}}(\mathbf{y}'(\mathbf{x}))$, will be $\mathcal{O}(1)$. In addition, to shorten the run time, the three normalized equality constraints will be combined into a single inequality constraint to allow trajectory termination within a specified radius, R, of the target. The problem statement with these changes for normalized cost, constraint, and state equations is as follows:

Minimize the performance index

$$\hat{f}(\mathbf{x}) = -V_f / V_o \tag{2.48a}$$

subject to

$$\dot{\mathbf{y}}' = \mathbf{s}(\mathbf{y}', \mathbf{x}, \tau) \tag{2.48b}$$

and constraint

$$\hat{\mathbf{h}}(\mathbf{y}'(\mathbf{x})) = \frac{1}{l}\left(\sqrt{(x'_f - x_T)^2 + (y'_f - y_T)^2 + (h'_f - h_T)^2} - R\right) \leq 0, \tag{2.48c}$$

where

$$l = \sqrt{(x_T - x'_o)^2 + (y_T - y'_o)^2 + (h_T - h'_o)^2}, \tag{2.48d}$$

which is the straight line range to the target. The subscripts f, o, and T indicate the final values (at $\tau = 1$), the known initial values (at $\tau = 0$), and the known target coordinates, respectively.

fmincon was used to implement the RQP optimization algorithm. Seven equally spaced decision variables over normalized time were chosen to parameterize the control histories for $C_L(t_i)$ and $C_S(t_i)$. Combining these with t_f resulted in a vector of 15 (n) decision variables being used. Gradient evaluations of cost and constraints with respect to $\mathbf{x}^j$ were defaulted to finite difference approximations.

The results are based on the tabulated boundary conditions in Table 2.3 (as a footnote, 1 nautical mile (nm) = 6076 ft). Flight initiates from a straight and level hypersonic initial condition and terminates at various crossranges on the surface.

Table 2.3. *Boundary conditions for maximum-velocity trajectories.*

Variable	Symbol	Initial condition	Terminal condition	Unit
Downrange	x	0	80	nm
Crossrange	y	0	20, 40, 60, 80	nm
Altitude	h	100,000	0	ft
Velocity	V	11,300	maximum	ft/sec
Flight path angle	γ	0	unconstrained	deg
Heading angle	χ	0	unconstrained	deg

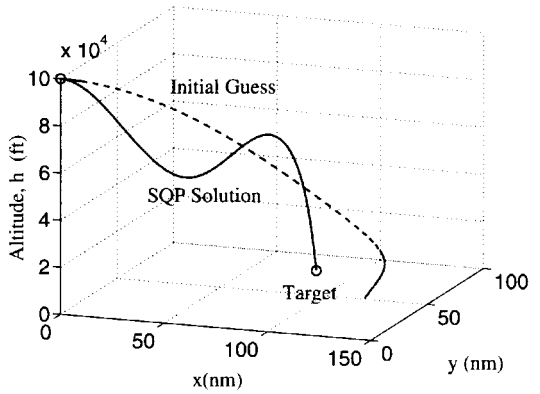


Figure 2.11. *Vehicle trajectory motion for initial guess and RQP solution.*

The most severe terminal condition, 80-nm crossrange, was chosen for detailed examination (see Figs. 2.11 and 2.12). The inequality constraint on the target was that the trajectory was to terminate when the final position was within a distance to target of $R = 150$ ft. The plots compare the physical trajectories, controls, and velocity histories generated by initially constant value sets of controls and those due to the substantially different constrained optimization solution. The initial control guesses generate a smooth helical motion toward the target that falls far short of the target "in crossrange" by over 20 nm. The optimized solution commands the vehicle on an initial leg that dives into the atmosphere. The flight continues with a section of lofting flight and terminates with a descent to the target. The velocities in Fig. 2.12 (left plot) show that the optimal trajectory immediately starts decreasing with respect to the initial guess but ends up with both a higher terminal value by about 2000 ft/sec (maximum velocity = 5600 ft/sec) and, not surprisingly, a shorter trajectory time, 98 sec versus the initial guess of 115 sec.

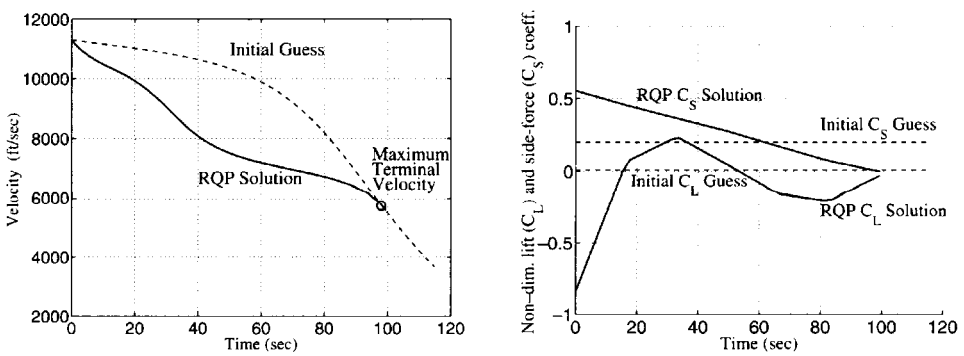


Figure 2.12. *Maximum terminal velocity history (left plot) and lift and side-force histories (right plot) for* 80-nm *crossrange case.*

Finally, the results in Fig. 2.12 (see right plot) compare the constant history guesses and optimized controls. While the side force linearly decreases to zero during the trajectory to achieve the necessary heading change, it is the lift that orchestrates the optimal programming. The lift history, $C_L(t)$, mimics the physical trajectory by initially depressing the trajectory, then using a lofting maneuver, and finally redepressing the trajectory before bleeding off nearly all of its magnitude.

This behavior may strike one as nonintuitive until the velocity and heading state equations are reexamined. If one considers the aerodynamic force dominant over the centrifugal and gravitational forces deep in the atmosphere, then the velocity equation reduces to the first term in (2.44). These equations are restated as

$$\dot{V} = -\frac{\rho V^2 S_R C_D}{2m} = -\frac{\rho V^2 S_R \left[C_{D_o} + \kappa(C_L^2 + C_S^2)\right]}{2m},$$
$$\dot{\chi} = \frac{\rho V^2 S_R C_S}{mV \cos\gamma}. \tag{2.49}$$

Since all variable quantities in the velocity equation are positive, velocity will do nothing but decrease. It is incumbent upon the control scheduling to manage that decrease in the most favorable terminal value manner possible.

Note that ρ appears to the first power in the velocity state equation, while C_L and C_S in the induced drag term are quadratic. Exerting significant C_S at an elevated altitude and therefore reducing ρ for any length of time to attain a given heading change, $\dot{\chi}$, incurs a significant velocity (i.e., drag) penalty. A better method would be to incur a linear velocity penalty from a ρ increase by depressing the trajectory. This would then allow a smaller C_S to attain the same $\dot{\chi}$.

The lift scheduling subsequently lofts the trajectory in such a manner that the rest of the heading change can be accomplished while attaining a higher altitude, a lower ρ, and smaller magnitude C_L and C_S. The final part of the trajectory maneuvers the vehicle to be nearly aligned with the target, and the control forces make final, ever-decreasing corrections to place the vehicle within the target circle.

In Fig. 2.13, the convergence results are given for the chosen settings, number of decision variables, and model implementation. The RQP method was able to start from a crude guess of the decision variables, as shown in Fig. 2.11. However, unlike the welding example, the number of variables plus the complexity of the computations to produce a cost and constraint residual were such that run time and number of iterations to converge were large. Since 15 decision variable histories would probably not be visually separable on a single grid, Fig. 2.13 (left plot) shows a history of the single parameter t_f. Though 500 RQP iterations were run, the solution has essentially converged after 300, varying t_f from a maximum of 120 to its final value of 98. Fig. 2.13 (right plot) shows the normalized cost and constraint residual. Note that the constraint was satisfied after 25 iterations (i.e., $h(\mathbf{x}^j) \approx 0$), though the cost was continually, but slowly, improving (maximum velocity = 5600 ft/sec).

In Fig. 2.14, a composite of results for 20-, 40-, 60-, and 80-nm crossranges for an 80-nm downrange are given. In Fig. 2.14 (left plot), the physical trajectories show increased

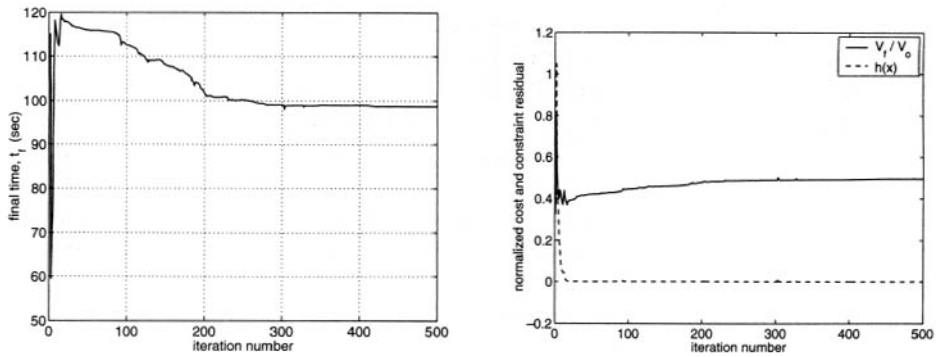


Figure 2.13. *Convergence histories for missile guidance example.*

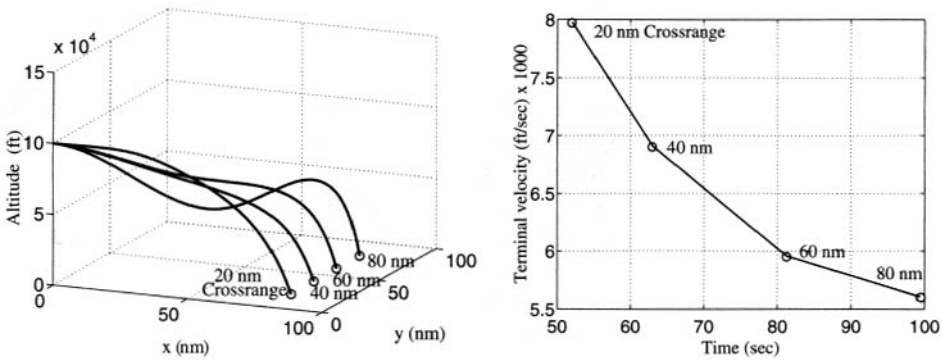


Figure 2.14. *Maximum terminal velocity results for multiple crossranges.*

trajectory shaping with crossrange using altitude changes to gain ease of turning. Fig. 2.14 (right plot) shows the maximum terminal velocities as a function of crossrange. Note that the predominant loss in terminal velocity occurs from turning to target crossranges of 20 to 60 nm, though visually the trajectory shaping is most evident for the 80-nm case.

As stated previously, the search for an optimal feedback law to provide continuous control inputs, $C_L(t)$ and $C_S(t)$, to hit a target at maximum velocity has been unsuccessful due to the inherent nonlinearities in the equations of motion. From observations of the results of the parameterized constrained suboptimal solutions, the quest for such an algorithm must demonstrate that the altitude/controls management tradeoff for drag minimization and ease of turning is embedded within its formulation.

This application provided an example with no analytic solution, but examination of the constrained optimal solutions can provide researchers with information on relevant control inputs and causal behavior. Smooth trajectory behavior was obtained using relatively few decision variables to approximate continuous control. However, time to converge was very sensitive to the number of variables and the model complexity. Scaling the cost and

constraint quantities was again necessary to gain favorable behavior for numerical output over widely different variable ranges.

2.6.4 Optimal Slew Maneuvers for Space Telescope Attitude Control

The study in this section presents an example of a constrained nonlinear optimization problem that is ill posed for the number of decision variables available for solution. A course of action is suggested to formulate the problem within the standard framework.

The availability of high-quality long-range sensors and cameras has spawned unprecedented growth in remote sensing from orbit in fields as diverse as natural resource management, urban development, mapping, and reconnaissance. To achieve such ambitious missions assumes a system with not only range and resolution but a means to acquire areas of interest from orbit on demand.

In this application of constrained optimization, an example of satellite attitude control for telescope pointing is considered. This can be modeled as the control of a two-gimbaled system executing azimuth-elevation slewing maneuvers while fixed to an inertial base [26]. A simple control scheme would be to time fixed-amplitude bang-bang maneuvers to steer the telescope to a prescribed angular orientation.

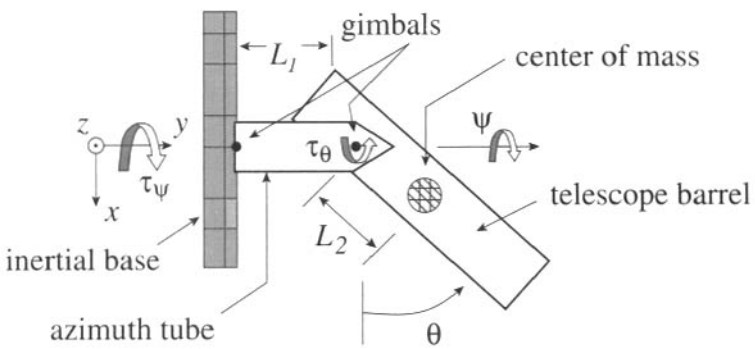


Figure 2.15. *Satellite attitude control schematic.*

The telescope attitude is assumed controlled by small thrusters, which set its azimuth and elevation angles, ψ and θ, as shown in Fig. 2.15. L_1 and L_2 are the telescope fixed base-to-elevation-gimbal-point distance and the elevation-gimbal-point-to-telescope-barrel center of mass (COM) distance. x, y, and z refer to an inertial coordinate system moving parallel to the base of the satellite at a constant velocity. The temporal equations of motion for describing the angular movements (ψ, θ) of this system are given by

$$\begin{aligned}\tau_\psi = &\left[I_{21} + I_{12}(\sin\theta)^2 + (I_{22} + m_2 L_2^2)(\cos\theta)^2\right]\ddot{\psi} \\ &+ \left[I_{21} - (I_{22} + m_2 L_2^2)\right]\sin(2\theta)\dot{\psi}\dot{\theta},\end{aligned} \tag{2.50a}$$

$$\tau_\theta = (I_{32} + m_2 L_2^2)\ddot{\theta} - \frac{1}{2}\left[I_{21} - (I_{22} + m_2 L_2^2)\right]\sin(2\theta)\dot{\psi}^2, \tag{2.50b}$$

where the I_{ij} terms correspond to elements of the inertia tensor for the telescope barrel and azimuth tube, m_2 is the mass of the barrel, and τ_ψ and τ_θ correspond to the torques about the azimuth (normal to inertial base) and elevation (gimbal line through azimuth tube to telescope) axes, respectively. The mass of the telescope is $m_2 = 60$ lbs/32.2 ft/sec^2 $= 1.86$ slugs. The system inertia tensor is given as

$$\mathbf{I} = \frac{1}{a_1}\begin{bmatrix} 0 & 606 & 0 \\ 1023 & 1550 & 0 \\ 0 & 1536 & 0 \end{bmatrix} \text{lb-in}^2$$

$$= \begin{bmatrix} 0 & 0.1307 & 0 \\ 0.2206 & 0.3343 & 0 \\ 0 & 0.3313 & 0 \end{bmatrix} \text{slug-ft}^2 \quad \text{and} \quad L_2 = 0,$$

where $a_1 = 32.2$ ft/sec^2 $\times$ 144 in^2/ft^2.

The description of the input torques, $\tau_\psi(t)$ and $\tau_\theta(t)$, as functions of time, t, includes the following characteristics:

1. bang-bang profiles for a single period that possess a single interior switch;
2. not necessarily equal total periods, t_ψ and t_θ, or even equal pre- and postswitch segment times;
3. identical equisided amplitudes of 150 oz-in separated at the switch points.

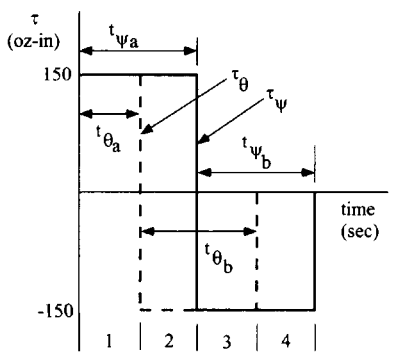


Figure 2.16. *Sample torque control period profiles.*

A plot of this torque behavior is shown in Fig. 2.16. For the sake of generality, a sample of the torque profiles for unequal control timings is displayed. A complete control timing for the given torque periods is the sum of parts before and after the respective switches, where

$$t_\psi = t_{\psi_a} + t_{\psi_b},$$
$$t_\theta = t_{\theta_a} + t_{\theta_b}. \tag{2.51}$$

In order to generate an angular motion time history, the equations of motion are rearranged to first-order form for the states:

$$\begin{aligned} y_1 &= \psi, \\ y_2 &= \dot{\psi}, \\ y_3 &= \theta, \\ y_4 &= \dot{\theta}. \end{aligned} \tag{2.52}$$

These definitions provide the state equations, $\dot{\mathbf{y}} = \mathbf{s}(\mathbf{y}, \boldsymbol{\tau}, t)$, for the vectors of states, $\mathbf{y}(t) = y_{1-4}(t)$, with embedded torques, $\boldsymbol{\tau}(t) = \tau_{\psi,\theta}(t)$. The first-order differential state equations are given by

$$\begin{aligned} \dot{y}_1 &= y_2 = \dot{\psi}, \\ \dot{y}_2 &= \ddot{\psi}, \\ \dot{y}_3 &= y_4 = \dot{\theta}, \\ \dot{y}_4 &= \ddot{\theta}, \end{aligned} \tag{2.53}$$

where $\ddot{\psi}$ and $\ddot{\theta}$ are obtained by solving the relations in (2.50). These are integrated with the MATLAB RK scheme *ode*45 [27] using interpolation within the current set of timings for the torques $\tau_\psi(\hat{t})$ and $\tau_\theta(\hat{t})$ at any given $\hat{t}$.

Integration of a complete two-axis slew maneuver is segmented at the switch points and endpoints. The segment numbering 1 to 4 at the bottom of Fig. 2.16 indicates that, for the current sample set of torque period timings, integration is done in four parts, separated at these segment breaks. Initial conditions for subsequent segments are the end conditions of the previous segment.

Since amplitude is constant, the timings for the segment breaks, t_{ψ_a}, t_{ψ_b}, t_{θ_a}, and t_{θ_b}, are the only items that vary, so they become the constant decision variables or parameters, $\mathbf{x}$, of the problem. Function evaluations for cost and constraint evaluations must account for the various permutations, where the parameter time segments of $\tau_\psi(t)$ and $\tau_\theta(t)$ may change their ordering and/or durations with respect to one another.

Using this model, the problem is to find the two periods of bang-bang motion for $\tau_\psi(t)$ and $\tau_\theta(t)$ that would slew the telescope from a set of initial angles, ψ_o and θ_o, to a set of final angles, $\psi_f = \psi(t_\chi)$ and $\theta_f = \theta(t_\theta)$, in a rest-to-rest maneuver in minimum overall time. For this problem, *rest-to-rest* will be defined as $\dot{\psi}_f = \dot{\theta}_f = 0$, with zero initial angular velocities assumed. One item of interest in the timings is that, typically, $t_\psi \neq t_\theta$. Although the final angular velocities will be zeroed, the final accelerations will not be considered. It will remain to be seen how much the shorter time final angular position drifts while waiting for the longer one to come to zero velocity.

The problem of optimal slew time will be posed as a constrained optimization problem with the four time segment components, $x = [t_{\psi_a}, t_{\psi_b}, t_{\theta_a}, t_{\theta_b}]$, of the constant amplitude bang-bang torque periods, $\tau_\psi(t)$ and $\tau_\theta(t)$, as the decision variables for which to solve. The problem has a cost of optimal overall final time (which is to be defined) and four constraints, two on final angular position and two on final angular velocity, which must be addressed in the context of the optimization problem statement. Since there are four constraints and four parameters, the problem must be restated to ensure that the matrix $\left(\nabla\boldsymbol{\Psi}\mathbf{B}^{-1}\nabla\boldsymbol{\Psi}\right)^{-1}$ exists.

To meet this requirement, an augmented cost, including a penalty function, will be fashioned from both a composite final time and the angular velocity constraints. The optimization problem will be stated as follows:

Minimize the performance index

$$f(\mathbf{x}) = \sqrt{t_\psi^2 + t_\theta^2} + W\sqrt{\dot{\psi}^2(t_\psi) + \dot{\theta}^2(t_\theta)} \tag{2.54a}$$

subject to

$$\dot{\mathbf{y}} = \mathbf{s}(\mathbf{y}, \boldsymbol{\tau}(\mathbf{x}), t) \tag{2.54b}$$

and constraints

$$\mathbf{g}(\mathbf{y}(\mathbf{x})) = \begin{bmatrix} \psi(t_\psi) - \psi_{desired} \\ \theta(t_\theta) - \theta_{desired} \end{bmatrix} = [\mathbf{0}]. \tag{2.54c}$$

This problem statement allows the positional constraints to be expressed within the paradigm of the RQP formulation. A weight, W, of 100 (arrived at arbitrarily) was used to couple the penalty function of resultant final angular velocity to the composite-time cost. Note that this penalty is minimum where the final angular velocities are zero. The performance and constraint specifications are not unique for the goals of this application and a variety of other combinations may be equally useful. As an example, the final composite-time and end constraints could have been evaluated at whichever current period, t_ψ or t_θ, is longer. (The use of $\mathbf{g}(\mathbf{y}(\mathbf{x}))$ is identical to its use in the missile example and is meant to imply that the decision variables are embedded in the state model, $\mathbf{y}(\mathbf{x}^j, \tau)$.)

Routines were written to evaluate $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{y}(\mathbf{x}))$ in which the state equations, $\dot{\mathbf{y}} = \mathbf{s}(\mathbf{y}, \boldsymbol{\tau}(\mathbf{x}), t)$, are embedded along with the logic to manage the numerical integration for the various orderings of the parameterized torque period components ($\mathbf{x} = [t_{\psi_a}, t_{\psi_b}, t_{\theta_a}, t_{\theta_b}]$). Gradient evaluations of performance and constraints used finite difference approximations. Again, the routine *fmincon* from the MATLAB's Optimization Toolbox was used to solve for the optimal times.

Nine sets of desired angle boundary conditions, $[(\psi, \theta)_{o_i}, (\psi, \theta)_{f_i}]$ for $i = 1, \ldots, 9$, were selected for optimal slew-time computations. Rest-to-rest maneuvers dictated that initial and final angular velocities were to be zero. Initial decision variable (or parameter) guesses for $\mathbf{x}$ were chosen to overshoot what was expected to be the minimum. The velocity state initial conditions were $\dot{\psi}_o = \dot{\theta}_o = 0$. Each parameter was bounded according to $0.1 \leq x_i \leq 1.5$ sec.

With regard to scaling, since $y_i \approx 1$ and the angles and angular rates are handled internally in radian units that are approximately equal to one, there was no need to scale the cost or constraint specifications.

Fig. 2.17 depicts angular position (left plot) and velocity (right plot) results for a slew between 0° and 40° on both axes. The solid lines show the azimuth histories, where the thin solid line is generated from an initial guessed set $\mathbf{x}^0 = [0.7, 0.7, 0.71, 0.71]$ sec and the thick line shows the results obtained from the RQP optimization. The dashed lines are the counterparts for the elevation histories. For both angles, the final boundary conditions for the guessed parameters are significantly different than the desired. The final times have been

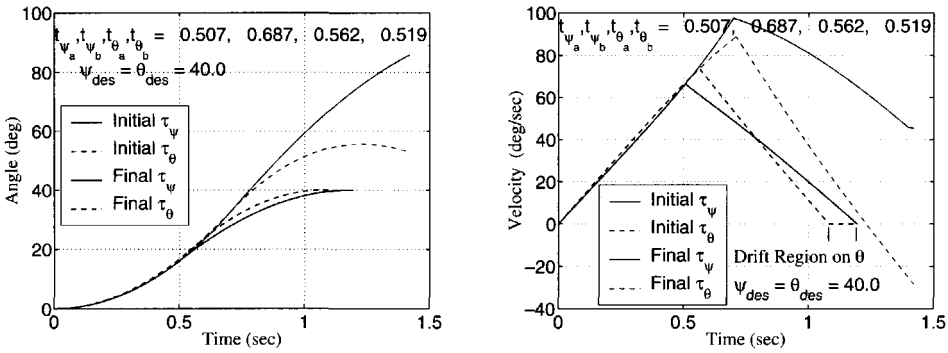


Figure 2.17. *Angular position (left plot) and velocity slew (right plot) histories for* 40° *traverses on both axes.*

decreased from about 1.4 to 1.2 sec via RQP. Satisfaction of the constraints, $\mathbf{g}(\mathbf{y}(\mathbf{x})) = [\mathbf{0}]$, was typically accomplished to 10^{-7} degrees or better.

The angular position (see Fig. 2.17, left plot) shows an asymptotic approach to satisfy the desired final values. The angular velocity (see Fig. 2.17, right plot) clearly shows the difference in optimized final times in the lower right-hand corner, where the elevation slew finishes over 0.1 sec before the azimuth one. The drift in elevation was 0.00225° during this time. As an example of the necessity to account for the variability in the segment times in the problem setup, note that even though the total elevation slew, θ, finished first, the first segment of the azimuth slew, ψ, finished before its elevation counterpart.

Fig. 2.18 shows the convergence histories of decision variables, cost, and constraint residuals. The cost was normalized by the weight, 100, so that it and the residuals could appear on one plot. The units of the constraint residuals are radians. Iteration zero is the initial condition.

This example has essentially converged in 10 iterations. The t_{ψ_a} and t_{θ_b} segments are changed the most from their initial conditions. The time cost with penalty on the final

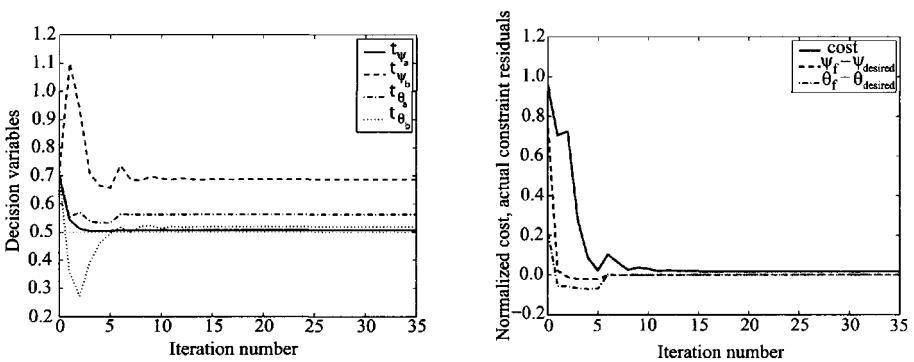


Figure 2.18. *RQP decision variable, cost, and constraint residual convergence histories for telescope example.*

velocities is reduced to a fraction of its initial value. After being reduced significantly during the first iteration, the constraint residuals are effectively "zeroed," indicating that the desired final angles have been achieved.

A summary of the results for the nine cases is given in Table 2.4.

Table 2.4. *RQP satellite slew solutions.*

Case	ψ_o (deg)	θ_o (deg)	ψ_f (deg)	θ_f (deg)	t_{ψ_a} (sec)	t_{ψ_b} (sec)	t_{θ_a} (sec)	t_{θ_b} (sec)	drift (deg)
1	0	0	20	20	0.385	0.422	0.389	0.379	−2.821e−5
2	0	0	30	30	0.457	0.552	0.481	0.457	−3.512e−4
3	0	0	40	40	0.507	0.687	0.562	0.519	−2.248e−3
4	0	20	20	40	0.382	0.471	0.395	0.373	−7.215e−4
5	0	10	40	50	0.494	0.731	0.570	0.514	−4.809e−3
6	0	20	0.25	0.25	0.046	0.046	0.043	0.043	6.182e−8
7	0	20	1	21	0.091	0.092	0.086	0.086	−4.916e−7
8	0	20	10	25	0.285	0.298	0.194	0.188	−0.02218
9	0	20	20	30	0.396	0.435	0.277	0.260	−0.1075

For the set of mass properties in (2.50), the RQP algorithm has deduced that maneuvers on the order of 40° per angular movement take slightly over 1 sec to accomplish, while those of 1° magnitude or less are completed in under 0.1 sec. Drift proved to be greatest for those cases of moderate-to-large-angle slews (>5°), where θ_o was initially large.

This example has demonstrated one course of action to provide a solution to a constrained optimization problem that was ill posed from the number of constraints versus available decision variables. The tactic taken here was to augment the cost with a penalty function involving two of the constraints. The cost itself was cast as a resultant of the individual axes' slew times, and, as mentioned earlier, is just one interpretation of optimality. Obviously, this application lends itself to numerous formulations. This is both a blessing from the standpoint of tailoring the result and a curse from the viewpoint of "beauty being in the eye of the beholder."

2.7 Summary

RQP has proved to be a flexible approach for solving the general nonlinear constrained optimization problem. Three examples of constrained optimization arising from diverse applications have been presented. Once the problems were tuned, executions proceeded routinely, although at various rates, to their respective solutions. This tuning touched directly on two areas of interest, scaling and initial conditions, and indirectly took advantage of a third, the lack of requirements for neighboring solutions. For the first, it was common to all of the examples either to compute in a regime that was naturally scaled, as in the satellite problem, or to introduce it as done in the welding and missile guidance examples. This tends to prevent the generation of decision variables that may cause discontinuous, oscillatory, or extreme control behavior by searching in an irregular hyperspace. Scaling as applied here is more art than science. An algorithm that is less sensitive to this phenomenon or includes it in a natural manner would be beneficial.

Since the RQP algorithm uses the BFGS algorithm to refine the Hessian, it embodies Newton-like convergence properties. Typically, one can expect a convergence rate that approaches quadratic. (Gill, Murray, and Wright [9] have noted that, as a rule of thumb, a quadratic convergence rate implies that the number of correct digits in the components of $\mathbf{x}^j$ increases by two at each iterate.) However, being a local search algorithm, RQP needs good initial decision variable guesses in order to ensure convergence. As the size of the decision variable hyperspace increases, this becomes even more critical, as seen in the missile guidance example. Robustness of the solution process to initial condition, $\mathbf{x}^0$, would be a sought-after attribute.

For the self-contained problems demonstrated here, RQP has been shown to be a robust solver. However, information gained from the solution of one problem statement cannot be used for a neighboring problem, where there are changes to initial conditions on state variables involved, say, in an integration over time. Such changes would affect subsequent cost evaluations and constraint residual computations. There is no mechanism to incorporate these changes and the problem would need to be resolved. An algorithm that could provide a solution structure to incorporate numerical changes to varying initial conditions or constraint bound levels would be extremely useful in an extensive design study, where significant tradeoffs, gleaned from numerous runs, have to be refined.

This page intentionally left blank

Chapter 3
Introduction to Dynamic Programming

3.1 Introduction

The goal of functional optimization problems is to find the function of time that minimizes the scalar cost functional. Most direct methods of optimization, such as recursive quadratic programming (RQP), transform the calculus of variations problem into a suboptimal parameter optimization problem. These direct method algorithms are of the order N^3 or N^2 in the optimization parameters (i.e., parameterized functions of time with interpolation or series approximations) and n in the state variables. The indirect methods of optimization attempt to solve the "true" optimization problem by solving the two-point boundary value problem (TPBVP). The problem with indirect methods is that they are extremely sensitive to the initial guess of the initial conditions of the costate equations. As a result, direct methods are often used to initialize indirect methods when solving functional optimization problems.

Dynamic programming (DP) is the best of both worlds. It solves the direct problem, which is less sensitive to the initial guess, and provides a discrete-time approximation to the optimal function of time since the DP algorithms are of order N in the optimization parameters. However, the cost of this linear behavior in N is an order n^2 or n^3 in the state variables. This chapter describes the basics of discrete dynamic programming (DDP) and demonstrates the ability of DDP to find smooth input shaping profiles and optimal trajectories from zero initial guesses.

3.2 Discrete Dynamic Programming

The details of a DDP algorithm for input shaping and trajectory optimization are presented in this section for linear problems. Constraints are dealt with approximately by using a simple penalty approach. DDP algorithms for unconstrained nonlinear problems [28], nonlinear problems with equality constraints [29], and problems with a combination of equality and inequality constraints [31] will be discussed in detail in Chapter 4, as well as being introduced to the reader by the last example in this chapter.

The principle of optimality [32, 33, 34] is the basis of DP. An optimal sequence of decisions in a multistage decision process problem has the property that whatever the initial

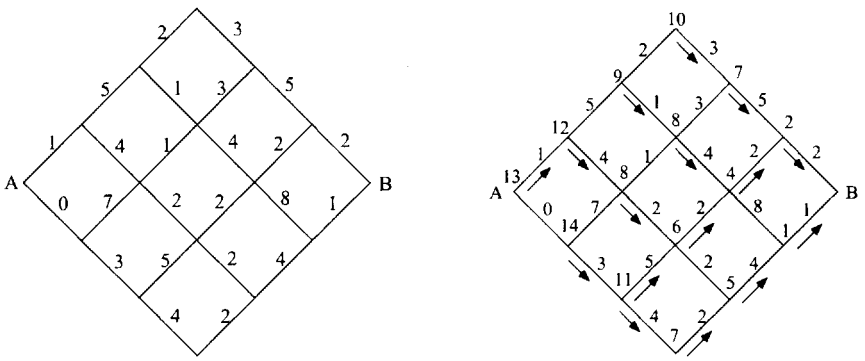


Figure 3.1. *DP concepts are illustrated by a simple example (left diagram) and its solution (right diagram).*

stage, state, and decision, the remaining decisions must constitute an optimal sequence of decisions for the remaining problem, with the stage and state resulting from the first decision considered as initial conditions.

A typical application of DP is the problem of traveling from point A to point B in Fig. 3.1 (left diagram). Movement is only allowed from left to right and the cost of traveling from one point on the grid to another is given by the number at the edge connecting the two points. The goal is to find the path from A to B that minimizes the total cost.

The number by each point in the grid in Fig. 3.1 (right diagram) is the cost of the lowest-cost path from that point to B. These numbers are obtained recursively by moving backward from B to A and applying the principle of optimality. The arrows in Fig. 3.1 (right diagram) indicate the direction to be taken from each point to minimize the total cost of getting to B. The best path from A to B is seen to have a cost of 13. The path moves *uddduud*, where *u* denotes up to the right and *d* denotes down to the right.

An application of DP that leads to a DDP algorithm is the discrete-time optimal control of a planar two-link rigid robot (see Fig. 3.2). In this example, each stage refers to a discrete point in time. For example, the ith stage refers to time t_i, the $(i + 1)$th stage to time t_{i+1}, etc. The state refers to the configuration of the system at a particular point in time. In this example, the state at the ith stage consists of the joint angles and joint angular rates at the

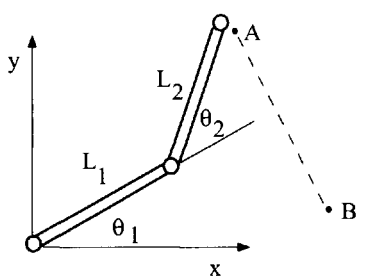


Figure 3.2. *Discrete-time optimal control of a planar two-link rigid robot.*

time t_i. Decisions are the actions taken at a given stage that transform the system from one state to another. In this example, the decision at the ith stage consists of the torques applied to the robot's joints at time t_i.

3.2.1 Application of Discrete Dynamic Programming to Data Smoothing

A straightforward application of DP to data smoothing produces a DDP solution for linear problems. This class of linear problems is defined as the minimization of the objective function

$$J(x_k, u_k) = \frac{1}{2}\sum_{k=1}^{n}\left((x_k - d_k)^2 + \lambda u_k^2\right) \tag{3.1a}$$

subject to

$$x_{k+1} = x_k + h_k u_k \qquad (k = 1, \ldots, N-1). \tag{3.1b}$$

In (3.1), x_k is the value of the approximating function (a linear spline) evaluated at time t_k. d_k is the value of the data point at time t_k. $\lambda \geq 0$ is a smoothing parameter. u_k is the input at time t_k and $h_k = t_{k+1} - t_k$. The input u_k can be interpreted as the first derivative of the linear spline for $t_k \leq t \leq t_{k+1}$ (see Fig. 3.3). The number of stages equals N, which is the number of optimization parameters, u_k. The state and decision at the kth stage are given by x_k and u_k, respectively. Define the optimal value function as

$$f_i(x_i) = \min_{(u_i, \ldots, u_N)}\left[\frac{1}{2}\sum_{k=i}^{N}\left((x_k - d_k)^2 + \lambda u_k^2\right)\right]. \tag{3.2}$$

That is, $f_i(x_i)$ is the value of (3.2) for an optimal sequence of decisions (inputs) in terms of the *initial* state x_i. Here, the number of stages is equal to $N - i + 1$. The solution strategy will be to obtain $f_i(x_i)$ from $f_{i+1}(x_{i+1})$ by using DP. In other words, the solution to the $(N - k + 1)$th stage problem will be obtained from that for the $(N - k)$th stage problem.

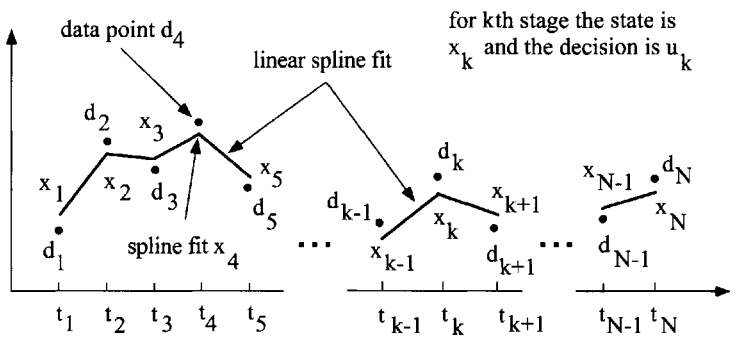


Figure 3.3. *Linear spline curve fit.*

This is where the idea of an embedded solution is made manifest. In mathematical form, the principle of optimality states that

$$f_i(x_i) = \min_{u_i} \left[\frac{(x_i - d_i)^2 + \lambda u_i^2}{2} + f_{i+1}(x_i + h_i u_i) \right]. \tag{3.3}$$

One postulates that the optimal value function can be expressed as

$$f(x_i) = \zeta_i + v_i x_i + w_i \frac{x_i^2}{2}. \tag{3.4}$$

Substituting (3.4) into (3.3) yields

$$\begin{aligned} \zeta_i + v_i x_i + \frac{w_i x_i^2}{2} = \min_{u_i} \Bigg[& \frac{(x_i - d_i)^2 + \lambda u_i^2}{2} + \zeta_{i+1} \\ & + v_{i+1}(x_i + h_i u_i) + \frac{w_{i+1}(x_i + h_i u_i)^2}{2} \Bigg]. \end{aligned} \tag{3.5}$$

The value of u_i that minimizes (3.5) is readily obtained by differentiating the right-hand side of (3.5) with respect to u_i and equating the result to zero. This leads to

$$u_i = \frac{-h_i(v_{i+1} + w_{i+1} x_i)}{\lambda + h_i^2 w_{i+1}}. \tag{3.6}$$

Substituting (3.6) into (3.5) and equating terms of equal degree in x_i yields the recursive equations

$$\zeta_i = \zeta_{i+1} + \frac{d_i^2}{2} - \frac{h_i^2 v_{i+1}^2}{2(\lambda + h_i^2 w_{i+1})}, \tag{3.7}$$

$$v_i = -d_i + \frac{\lambda v_{i+1}}{\lambda + h_i^2 w_{i+1}}, \tag{3.8}$$

$$w_i = 1 + \frac{\lambda w_{i+1}}{\lambda + h_i^2 w_{i+1}}. \tag{3.9}$$

Given the values for ζ_N, v_N, and w_N, one can recursively work backward by using (3.7)–(3.9) to obtain

$$f_1(x_1) = \zeta_1 + v_1 x_1 + w_1 \frac{x_1^2}{2}. \tag{3.10}$$

Examination of (3.2) reveals that (3.10) provides the minimum value of the objective function given by (3.1a) in terms of the initial state x_i. Minimizing f_i with respect to the initial state yields

$$x_1 = \frac{-v_1}{w_1}. \tag{3.11}$$

This is the value for the initial state that minimizes (3.1a).

To determine ζ_N, v_N, and w_N, (3.2) and (3.4) are combined, giving

$$\zeta_N + v_N x_N + w_N \frac{x_N^2}{2} = \min_{u_i} \left[\frac{(x_N - d_N)^2 + \lambda u_N^2}{2} \right]. \tag{3.12}$$

The right-hand side of (3.12) is minimized for $u_N = 0$. Substituting $u_N = 0$ into (3.12) and equating terms of equal degree in x_N yields

$$\zeta_N = \frac{d_N^2}{2}, \tag{3.13}$$

$$v_N = -d_N, \tag{3.14}$$

$$w_N = 1. \tag{3.15}$$

Summary of the discrete dynamic programming algorithm for data smoothing

- Calculate and store v_N and w_N by using (3.14) and (3.15).
- Calculate and store v_i and w_i for $i = N - 1, \ldots, 1$ by using (3.8) and (3.9).
- Calculate the initial state x_1 by using (3.11).
- Calculate u_i and x_{i+1} for $i = 1, \ldots, N - 1$ by using (3.6) and (3.1b).

Example 3.1. *Given* $N = 501$, $t_k = (k - 1)/500$, $\lambda = 0.001$, *and* $d_k = \sin(2\pi t_k) + e_k$, *where* e_k *is a uniformly distributed random variable between* -0.2 *and* 0.2, *what happens as* λ *goes to* 0 *and as* λ *goes to* ∞?

Solution

The results of this example are shown in Fig. 3.4. The DDP algorithm is applied to the noisy sinusoidal data shown along with the data-smoothed results in Fig. 3.4. As the parameter λ goes to 0, the smoothed data fit approaches the noisy sinusoidal data. λ going to ∞ will cause u_i to go to 0 (see (3.6)), which results in a least-squares fit to the data.

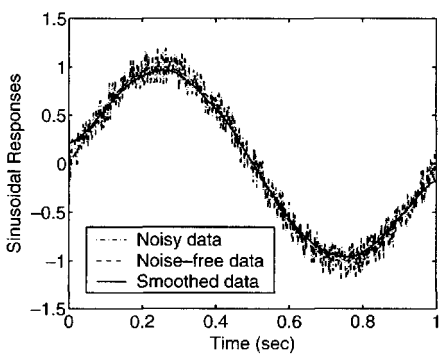


Figure 3.4. *DP for data smoothing application results for Example* 3.1.

3.2.2 Application of Discrete Dynamic Programming to Discrete-Time Optimal Control Problems

The problems of real interest are input shaping and trajectory optimization, which can be formulated as discrete-time optimal control problems. The general nonlinear initial value problem is defined as

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_1) = \mathbf{x}_1,$$

where $\mathbf{x} \in R^n$ is the state and $\mathbf{u} \in R^m$ is the input. The input is assumed to be discretized temporally as

$$\mathbf{u}(t) = \mathbf{u}_k, \quad t_k \leq t \leq t_{k+1},$$

for $k = 1, \ldots, N-1$, where the $\mathbf{u}_k$'s are the optimization parameters. Given the existence and uniqueness of the solution to the initial value problem, adjacent states in time can be related as

$$\mathbf{x}_{k+1} = \mathbf{g}_k(\mathbf{x}_k, \mathbf{u}_k), \tag{3.16}$$

where $\mathbf{x}_k = \mathbf{x}(t_k)$.

The discrete-time optimal control problem is stated as follows: Given the initial state $\mathbf{x}_1$, find the input $\mathbf{u}_k$ that minimizes the objective function

$$\Gamma(\mathbf{x}_k, \mathbf{u}_k) = \sum_{k=1}^{N} \Gamma_k(\mathbf{x}_k, \mathbf{u}_k) \tag{3.17}$$

subject to (3.16).

Consider a subset of discrete-time optimal control problems in which the dynamics are linear and the objective function is quadratic in the inputs and states:

$$\mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{B}_k \mathbf{u}_k, \tag{3.18}$$

$$\Gamma(\mathbf{x}_k, \mathbf{u}_k) = \sum_{k=1}^{N} \eta_k + \mathbf{x}_k^T \mathbf{y}_k + \mathbf{u}_k^T \mathbf{z}_k + \frac{1}{2}\left[\mathbf{x}_k^T \mathbf{Q}_k \mathbf{x}_k + 2\mathbf{x}_k^T \mathbf{R}_k \mathbf{u}_k + \mathbf{u}_k^T \mathbf{S}_k \mathbf{u}_k\right]. \tag{3.19}$$

As was done previously for the data smoothing problem, define the optimal value function as

$$\Lambda_i(\mathbf{x}_i) = \min_{(\mathbf{u}_i, \ldots, \mathbf{u}_N)} \sum_{k=i}^{N} \eta_k + \mathbf{x}_k^T \mathbf{y}_k + \mathbf{u}_k^T \mathbf{z}_k + \frac{1}{2}\left[\mathbf{x}_k^T \mathbf{Q}_k \mathbf{x}_k + 2\mathbf{x}_k^T \mathbf{R}_k \mathbf{u}_k + \mathbf{u}_k^T \mathbf{S}_k \mathbf{u}_k\right]. \tag{3.20}$$

Application of the principle of optimality yields

$$\Lambda_i(\mathbf{x}_i) = \min_{\mathbf{u}_i} \left\{ \eta_i + \mathbf{x}_i^T \mathbf{y}_i + \mathbf{u}_i^T \mathbf{z}_i + \frac{1}{2}\left[\mathbf{x}_i^T \mathbf{Q}_i \mathbf{x}_i + 2\mathbf{x}_i^T \mathbf{R}_i \mathbf{u}_i + \mathbf{u}_i^T \mathbf{S}_i \mathbf{u}_i\right] + \Lambda_{i+1}\left[\mathbf{A}_i \mathbf{x}_i + \mathbf{B}_i \mathbf{u}_i\right] \right\}. \tag{3.21}$$

To be consistent with the use of a quadratic objective function, one expresses the optimal value function as

$$\Lambda_i(\mathbf{x}_i) = \zeta_i + \mathbf{x}_i^T \mathbf{v}_i + \frac{1}{2}\mathbf{x}_i^T \mathbf{W}_i \mathbf{x}_i. \tag{3.22}$$

Substituting (3.22) into (3.21) yields

$$\begin{aligned}\zeta_i + \mathbf{x}_i^T \mathbf{v}_i + \frac{\mathbf{x}_i^T \mathbf{W}_i \mathbf{x}_i}{2} = \min_{\mathbf{u}_i} \Big\{ & \eta_i + \zeta_{i+1} + \mathbf{x}_i^T \mathbf{h}_{4i} \\ & + \mathbf{u}_i^T \mathbf{h}_{5i} + \frac{1}{2}\left[\mathbf{x}_i^T \mathbf{H}_{1i} \mathbf{x}_i + 2\mathbf{x}_i^T \mathbf{H}_{2i} \mathbf{u}_i + \mathbf{u}_i^T \mathbf{H}_{3i} \mathbf{u}_i\right] \Big\},\end{aligned} \tag{3.23}$$

where

$$\begin{aligned}\mathbf{H}_{1i} &= \mathbf{Q}_i + \mathbf{A}_i^T \mathbf{W}_{i+1} \mathbf{A}_i, \\ \mathbf{H}_{2i} &= \mathbf{R}_i + \mathbf{A}_i^T \mathbf{W}_{i+1} \mathbf{B}_i, \\ \mathbf{H}_{3i} &= \mathbf{S}_i + \mathbf{B}_i^T \mathbf{W}_{i+1} \mathbf{B}_i, \\ \mathbf{h}_{4i} &= \mathbf{y}_i + \mathbf{A}_i^T \mathbf{v}_{i+1}, \\ \mathbf{h}_{5i} &= \mathbf{z}_i + \mathbf{B}_i^T \mathbf{v}_{i+1}.\end{aligned}$$

Differentiating the right-hand side of (3.23) with respect to $\mathbf{u}_i$ and then setting it equal to zero results in

$$\mathbf{u}_i = -\mathbf{H}_{3i}^{-1}\left[\mathbf{H}_{2i}^T \mathbf{x}_i + \mathbf{h}_{5i}\right]. \tag{3.24}$$

Finally, substituting (3.24) for $\mathbf{u}_i$ into (3.23) and equating terms of equal degree in $\mathbf{x}_i$ results in the following recursive equations:

$$\zeta_i = \zeta_{i+1} + \eta_i - \frac{1}{2}\mathbf{h}_{5i}^T \mathbf{H}_{3i}^{-1} \mathbf{h}_{5i},$$

$$\mathbf{v}_i = \mathbf{h}_{4i} - \mathbf{H}_{2i}\mathbf{H}_{3i}^{-1}\mathbf{h}_{5i}, \tag{3.25}$$

$$\mathbf{W}_i = \mathbf{H}_{1i} - \mathbf{H}_{2i}\mathbf{H}_{3i}^{-1}\mathbf{H}_{2i}^T. \tag{3.26}$$

The *initial* values for the recursive equations above are given by

$$\zeta_N = \eta_N,$$

$$\mathbf{v}_N = \mathbf{y}_N, \tag{3.27}$$

$$\mathbf{W}_N = \mathbf{Q}_N. \tag{3.28}$$

If the initial state $\mathbf{x}_i$ is unspecified, then minimization of (3.17) yields

$$\mathbf{x}_1 = -\mathbf{W}_1^{-1}\mathbf{v}_1. \tag{3.29}$$

Summary of the DDP algorithm for linear optimal control problems

The procedure for solving the discrete-time linear optimal control problem is as follows:

1. Calculate $\mathbf{v}_N$ and $\mathbf{W}_N$ by using (3.27) and (3.28).
2. Calculate $\mathbf{v}_i$ and $\mathbf{W}_i$ recursively for $i = N-1$ to $i = 1$ by using (3.25) and (3.26) and store the matrices $\mathbf{H}_{3i}^{-1}$, $\mathbf{H}_{2i}^{T}$, and $\mathbf{H}_{3i}^{-1}\mathbf{h}_{5i}$ in the process.
3. If the initial conditions are unspecified, then calculate $\mathbf{x}_1$ by using (3.29).
4. Calculate $\mathbf{u}_i$ and $\mathbf{x}_i$ recursively for $i = 1$ to $i = N-1$ by using (3.24) and (3.18), respectively.

The computational requirements for implementing the DDP algorithm are as follows:

1. Order Nn^3 operations are required to solve a problem.
2. Order Nnm storage is required to solve a problem.
3. Constraints can be accommodated by using penalty functions.

3.2.3 Implementation Details

Detail 1: Determination of state transition matrices found in (3.18). Consider a linear, time-invariant system written in first-order form as

$$\dot{\mathbf{x}} = \mathbf{E}\mathbf{x} + \mathbf{F}\mathbf{u}. \tag{3.30}$$

Recall that the solution of (3.30) can be written in terms of the matrix exponential [35] as

$$\mathbf{x}(t) \equiv \phi(t; t_0, \mathbf{x}_0, \mathbf{u}) = e^{\mathbf{E}(t-t_0)}\mathbf{x}_0 + \int_{t_0}^{t} e^{\mathbf{E}(t-\tau)}\mathbf{F}\mathbf{u}(\tau)d\tau,$$

where $\mathbf{x}_0$ is the state at time t_0. Setting

$$\begin{aligned} t_0 &= t_k, \\ t &= t_{k+1}, \\ h &= t_{k+1} - t_k, \end{aligned}$$

one obtains

$$\mathbf{x}_{k+1} = e^{\mathbf{E}h}\mathbf{x}_k + \int_0^h e^{\mathbf{E}(h-\tau)}d\tau\mathbf{F}\mathbf{u}_k. \tag{3.31}$$

Comparison of (3.18) and (3.31) implies

$$\mathbf{A}_k = e^{\mathbf{E}h}, \tag{3.32}$$

$$\mathbf{B}_k = \left[\int_0^h e^{\mathbf{E}(h-\tau)}d\tau\mathbf{F}\right]. \tag{3.33}$$

Example 3.2. *Find the state transition matrices for a cubic spline given as*

$$\mathbf{x}_k = [f(t_k) \quad \dot{f}(t_k) \quad \ddot{f}(t_k)]^T \quad \textit{and} \quad u_k = \dddot{f}(t_k).$$

The matrices $\mathbf{E}$ *and* $\mathbf{F}$ *for the continuous-time system are*

$$\mathbf{E} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{F} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Solution

$$e^{\mathbf{E}t} = \begin{bmatrix} 1 & t & \frac{t^2}{2} \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix},$$

$$\left[\int_0^h e^{\mathbf{E}(h-\tau)} d\tau \mathbf{F}\right] = \begin{bmatrix} \frac{h^3}{6} \\ \frac{h^2}{2} \\ h \end{bmatrix}.$$

Thus,

$$\mathbf{A}_k = \begin{bmatrix} 1 & h & \frac{h^2}{2} \\ 0 & 1 & h \\ 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{B}_k = \begin{bmatrix} \frac{h^3}{6} \\ \frac{h^2}{2} \\ h \end{bmatrix}.$$

Comments

- The MATLAB function *c2d* of the Control System Toolbox can be used to convert state-space models from continuous time to discrete time for linear time-invariant models. That is, *c2d* can be used to calculate the transition matrices $\mathbf{A}_k$ and $\mathbf{B}_k$ (see (3.32) and (3.33)) based on the continuous-time state-space model (see (3.30)).
- The transition matrices $\mathbf{A}_k$ and $\mathbf{B}_k$ can also be calculated directly from closed-form solutions. This point is illustrated in Example 3.3.
- Calculation of the transition matrices for time-varying or nonlinear systems may require the use of a numerical integration scheme, such as the Runge–Kutta (RK) algorithm.

Example 3.3. *Find the state transition matrices for a single-link rigid robot with*

$$\mathbf{x}_k = [\theta(t_k) \quad \dot{\theta}(t_k)]^T \quad \textit{and} \quad u_k = M_k.$$

The equation of motion for this system is given by

$$J\ddot{\theta} = M.$$

Solution

For $t_k \leq t \leq t_{k+1}$,

$$\dot{\theta}(t) = \dot{\theta}(t_k) + \frac{u_k(t - t_k)}{J}, \tag{3.34}$$

$$\theta(t) = \theta(t_k) + \dot{\theta}(t_k)(t - t_k) + \frac{u_k}{2J}(t - t_k)^2. \tag{3.35}$$

Substituting $t_k = t_{k+1}$ into (3.34) and (3.35) gives

$$\mathbf{A}_k = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}, \tag{3.36}$$

$$\mathbf{B}_k = \frac{1}{J}\begin{bmatrix} \frac{h^2}{2} \\ h \end{bmatrix}, \tag{3.37}$$

where $h = t_{k+1} - t_k$.

Detail 2: Enforcement of constraints with penalty functions. Consider the problem of minimizing the function

$$G(x, y) = x^2 + y^2$$

subject to the constraint

$$\Psi_1 = x + y - 1 = 0.$$

One can propose to solve the constrained minimization problem by solving a related unconstrained problem that includes the constraint equation in the objective function.

Define the augmented function as

$$\begin{aligned} \bar{G} &= G + p\Psi \\ &= x^2 + y^2 + p(x + y - 1)^2. \end{aligned}$$

Minimization of $\bar{G}$ leads to the solution

$$x = y = \frac{p}{2p + 1}.$$

Comment

Penalty functions provide a convenient means for converting constrained optimization problems to unconstrained problems. The major drawback of penalty functions is that their use

may lead to numerical difficulties (ill conditioning) as the penalty parameter takes on *large* values. This will be solved later by employing a homotopy scheme to enforce the penalty function.

Example 3.4. *Solve the following problem related to the discrete-time optimal control of a single-link rigid robot.*

Minimize

$$\Gamma = \frac{1}{2}\sum_{k=1}^{N} u_k^2$$

subject to

$$\begin{aligned}\theta(0) &= 0,\\ \dot{\theta}(0) &= 0,\\ \theta(T) &= \theta_F,\\ \dot{\theta}(T) &= 0.\end{aligned}$$

Solution

Defining $h = T/(N-1)$ and $t_k = (k-1)/h$, one can use the state transition matrices given in (3.36) and (3.37). In order to enforce the constraints on the final states, consider the problem of minimizing

$$\bar{\Gamma} = \frac{1}{2}\sum_{k=1}^{N} u_k^2 + p[\mathbf{x}_N - \mathbf{x}_F]^T[\mathbf{x}_N - \mathbf{x}_F], \tag{3.38}$$

where

$$\mathbf{x}_F = [\theta_F \;\; 0]^T. \tag{3.39}$$

Examination of (3.38) and (3.39) reveals that the only nonzero terms to be used in the DP algorithm are

$$\begin{aligned}S_k &= 1 \qquad \text{for} \qquad k = 1, \ldots, N,\\ \eta_N &= p\frac{\theta_F^2}{2},\\ \mathbf{y}_N &= -p\mathbf{x}_F,\\ \mathbf{W}_N &= p\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}.\end{aligned}$$

For this example, consider the problem in which $J = 1.5$, $T = 1.2$, $N = 201$, $\theta_F = \pi/2$, and $p = 1 \times 10^6$. The input (torque, left plot) and states (joint angle and joint angle rates, right plot) are plotted as functions of time in Fig. 3.5. Note that the constraints on the states at the final time are met satisfactorily by using the penalty function.

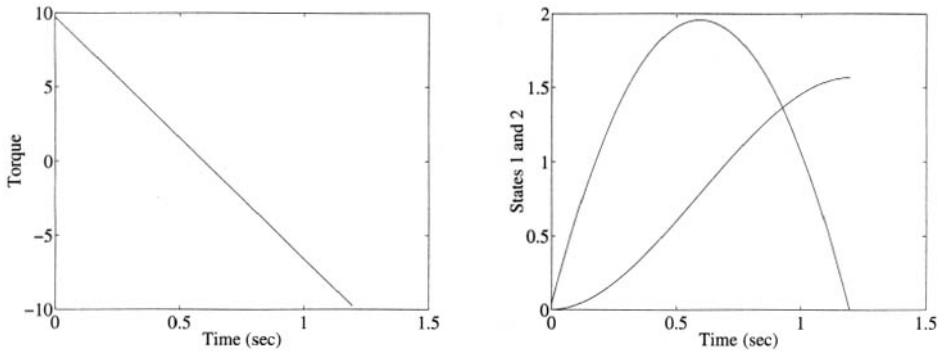


Figure 3.5. *Torque (left plot) and joint/joint angle rate (right plot) time histories for Example* 3.4.

Example 3.5. *Solve the discrete-time optimal control problem of a single-link rigid robot with a modified state equation and objective function given as*

$$\mathbf{x}_k = [\theta(t_k) \quad \dot{\theta}(t_k) \quad M_k]^T \quad \textit{and} \quad u_k = \dot{M}_k.$$

Minimize

$$\Gamma = \frac{1}{2}\sum_{k=1}^{N} u_k^2$$

subject to

$$\begin{aligned} \theta(0) &= 0, \\ \dot{\theta}(0) &= 0, \\ M(0) &= 0, \\ \theta(T) &= \theta_F, \\ \dot{\theta}(T) &= 0, \\ M(T) &= 0. \end{aligned}$$

Solution

This problem is solved in the same manner as Example 3.4 with the following differences:

$$\mathbf{A}_k = \begin{bmatrix} 1 & h & \frac{h^2}{2J} \\ 0 & 1 & \frac{h}{J} \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{B}_k = \begin{bmatrix} \frac{h^3}{6J} \\ \frac{h^2}{2J} \\ h \end{bmatrix},$$

$$\mathbf{x}_F = \begin{bmatrix} \theta_F \\ 0 \\ 0 \end{bmatrix}, \qquad \mathbf{W}_N = p \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

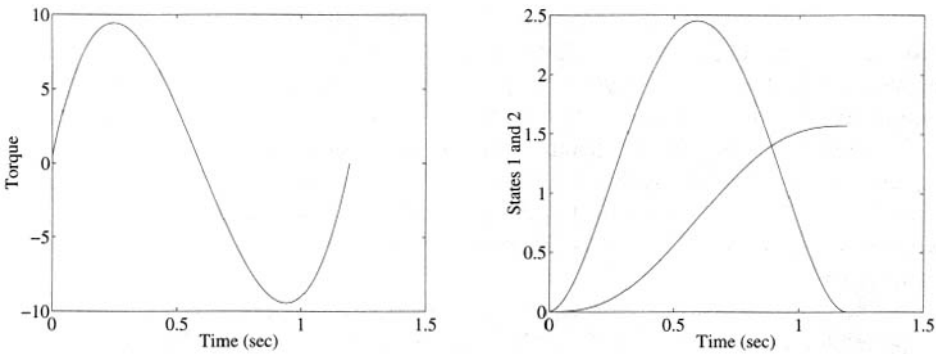


Figure 3.6. *Torque (left plot) and joint/joint angle rate (right plot) time histories for Example* 3.5.

The results are plotted in Fig. 3.6 (torque, left plot; joint angles and rates, right plot) for this example and were obtained by using a value of $p = 1 \times 10^8$ for the penalty parameter. Note that the torque time history (see Fig. 3.6, left plot) is smoother than the one for Example 3.4. The MATLAB standalone code that implements DDP for the discrete-time optimal control solutions of Examples 3.4 and 3.5 is located in Section B.7 of Appendix B.

3.3 A Nonlinear Optimal Control Problem with Constraints

At this point, it is time to discuss the necessary extensions to the DDP algorithm for linear problems to solve nonlinear constrained problems. In Section 3.2.2, one had a glimpse of some of the issues with solving nonlinear problems:

1. no closed-form solution of the nonlinear dynamic models (see (3.16)),
2. no systematic way to impose constraints,
3. no systematic way to handle arbitrary cost functionals (i.e., minimum time),
4. no systematic way to pick initial starting points.

The following example is based on excerpts from Dohrmann and Robinett [36][1] and will be used to address these issues and set the stage for Chapter 4, where DDP algorithms are developed to solve nonlinear constrained optimal control problems.

The precision pointing of imaging satellites for nonproliferation has become an area of current interest. Imaging of ground targets for such applications leads to pointing accuracy

[1]Revised and reprinted by permission of the Institute of Electrical and Electronics Engineers, Inc. (IEEE).

requirements on the order of microradians. There are many contributors to pointing angle error, but vibrational disturbances caused by flexible solar array support structures can be a major problem. In order to alleviate this problem, input shaping has been proposed to help minimize the residual vibration of the solar arrays after a three-dimensional slew maneuver.

Three example problems dealing with a simple spacecraft model are provided. The model consists of a rigid bus with two attached beams to model the effects of flexible solar panels. The effect of misalignment of the principal mass axes and the principal planes for bending is investigated. The results display interesting symmetries previously observed for planar maneuvers [7].

The first step is to develop equations of motion for a flexible spacecraft subjected to applied forces and moments. Flexible deformations are assumed to be small relative to the overall length of the spacecraft. The angular velocity magnitude for overall rigid body motion is also assumed to be small relative to the lowest structural natural frequency. Under these assumptions, the governing equations for the rigid body and flexible motions can be decoupled. Furthermore, the equations can be assembled in a straightforward manner using finite element analysis results.

The spacecraft is idealized as a system of interconnected particles, each of mass m^i ($i = 1, \ldots, N$), as shown in Fig. 3.7. Also shown in the figure are a floating reference frame, B, and an inertial frame, A. Orthogonal, dextral sets of unit vectors, $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ and $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$, are fixed in B and A, respectively.

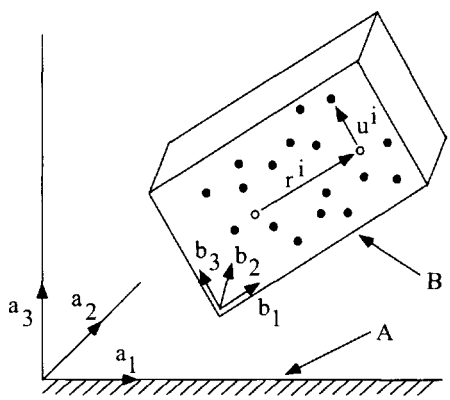


Figure 3.7. *Sketch of system of particles and reference frames.*

The angular velocity vector of B in A is denoted by $\boldsymbol{\omega}$. The position vector from the origin O of B to the ith particle when the system is undeformed is denoted by $\mathbf{r}^i$. The displacement vector of the ith particle from its undeformed position, $\mathbf{u}^i$, is assumed to be a function of the generalized coordinates, $q_1, \ldots, q_n$. The notational convention is adopted herein that, for any vector, $\mathbf{v}$, one has $v_k \equiv \mathbf{v} \cdot \mathbf{b}_k$ for $k = 1, 2, 3$.

The position of O and the orientation of B in A depend on the particular choice of the floating frame. It is convenient to use the so-called Bückens frame [37]. This frame is

defined by the constraint equations

$$\sum_{i=1}^{N} m^i(\mathbf{r}^i + \mathbf{u}^i) = 0 \tag{3.40}$$

and

$$\sum_{i=1}^{N} m^i(\mathbf{r}^i \times \mathbf{u}^i) = 0. \tag{3.41}$$

The constraints given by (3.40) and (3.41) are easily accommodated by the finite element method. Moreover, the nonrigid body, free-free modes of a structure automatically satisfy these conditions. This fact allows one to describe flexible motions in terms of the free-free mode shapes calculated from a finite element analysis.

The angular momentum, $\mathbf{H}$, of the system about its center of mass (COM), O, is defined as

$$\mathbf{H} = \sum_{i=1}^{N} m^i(\mathbf{r}^i + \mathbf{u}^i) \times \mathbf{v}^i, \tag{3.42}$$

where $\mathbf{v}^i$ denotes the velocity of the ith particle in A. The system kinetic energy T is defined as

$$T = \frac{1}{2} \sum_{i=1}^{N} m^i \mathbf{v}^i \cdot \mathbf{v}^i. \tag{3.43}$$

Using a basic kinematical relationship, one obtains

$$\mathbf{v}^i = \mathbf{v}^{\mathrm{O}} + \dot{\mathbf{u}}^i + \boldsymbol{\omega} \times (\mathbf{r}^i + \mathbf{u}^i), \tag{3.44}$$

where $\mathbf{v}^{\mathrm{O}}$ is the velocity of O in A and $(\dot{})$ denotes the time derivative in B.

The angular momentum principle states that the time derivative of $\mathbf{H}$ in A is equal to the net moment, $\mathbf{M}$, acting on the system about O. Thus,

$$\dot{\mathbf{H}} + \boldsymbol{\omega} \times \mathbf{H} = \mathbf{M}. \tag{3.45}$$

Substituting (3.42) and (3.44) into (3.45), accounting for the constraints given by (3.40) and (3.41), and linearizing the result yields

$$\begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{12} & I_{22} & I_{23} \\ I_{13} & I_{23} & I_{33} \end{bmatrix} \begin{pmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{pmatrix} = \begin{pmatrix} M_1 \\ M_2 \\ M_3 \end{pmatrix}, \tag{3.46}$$

where

$$
\begin{aligned}
I_{11} &= \sum_{i=1}^{N} m^i \left[(r_2^i)^2 + (r_3^i)^2\right], \\
I_{12} &= -\sum_{i=1}^{N} m^i r_1^i r_2^i, \\
I_{22} &= \sum_{i=1}^{N} m^i \left[(r_3^i)^2 + (r_1^i)^2\right], \\
I_{23} &= -\sum_{i=1}^{N} m^i r_2^i r_3^i, \\
I_{33} &= \sum_{i=1}^{N} m^i \left[(r_1^i)^2 + (r_2^i)^2\right], \\
I_{13} &= -\sum_{i=1}^{N} m^i r_3^i r_1^i.
\end{aligned} \tag{3.47}
$$

Notice that (3.46) is equivalent to the linear form of Euler's equations and is decoupled from translational and flexible motions.

An assumed displacement expansion of the form

$$
u_k^i = \sum_{j=1}^{n} q_j \phi_k^{ji} \tag{3.48}
$$

is used to account for spacecraft flexibility. Defining the vector, $\boldsymbol{\phi}^j$, as

$$
\boldsymbol{\phi}^j = [\phi_1^{j1} \quad \phi_2^{j1} \quad \phi_3^{j1} \quad \cdots \quad \phi_1^{jN} \quad \phi_2^{jN} \quad \phi_3^{jN}]^T, \tag{3.49}
$$

the strain energy, U, of the spacecraft is expressed as the quadratic form

$$
U = \frac{1}{2}\mathbf{q}^T \bar{\mathbf{K}} \mathbf{q}, \tag{3.50}
$$

where

$$
\mathbf{q} = [q_1 \quad \cdots \quad q_n]^T, \tag{3.51}
$$

$$
\bar{\mathbf{K}}_{lm} = \boldsymbol{\phi}^{lT} \mathbf{K} \boldsymbol{\phi}^m, \tag{3.52}
$$

and $\mathbf{K}$ is the stiffness matrix associated with a finite model of the spacecraft.

The equations governing flexible motion are obtained from the Lagrangian equations

$$
\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} + \frac{\partial D}{\partial \dot{q}_j} + \frac{\partial U}{\partial q_j} = Q_j \qquad (j = 1, \ldots, n), \tag{3.53}
$$

where D is the Rayleigh dissipation function, assumed to be of the classical form

$$D = \frac{1}{2}\dot{\mathbf{q}}^T \bar{\mathbf{C}} \dot{\mathbf{q}}. \tag{3.54}$$

The generalized force, Q_j, in (3.53) is given by

$$Q_j = \boldsymbol{\phi}^{jT} \mathbf{f}, \tag{3.55}$$

where

$$\mathbf{f} = [f_1^1 \quad f_2^1 \quad f_3^1 \quad \cdots \quad f_1^N \quad f_2^N \quad f_3^N]^T \tag{3.56}$$

and f_k^i denotes the force acting on the ith particle in the direction $\mathbf{b}_k$.

Substituting (3.43), (3.44), (3.48), and (3.50) into (3.53), accounting for the constraints given by (3.40) and (3.41), and linearizing the result yields

$$\bar{\mathbf{M}}\ddot{\mathbf{q}} + \bar{\mathbf{C}}\dot{\mathbf{q}} + \bar{\mathbf{K}}\mathbf{q} = [Q_1 \quad \cdots \quad Q_n]^T, \tag{3.57}$$

where

$$\bar{\mathbf{M}}_{lm} = \sum_{i=1}^{N} \sum_{k=1}^{3} m^i \phi_k^{li} \phi_k^{mi}. \tag{3.58}$$

When nonrigid body, free-free modes of the spacecraft are used for the ϕ's in (3.48), the matrices, $\bar{\mathbf{M}}$ and $\bar{\mathbf{K}}$, are diagonal. Under the additional assumption of modal damping, the matrix, $\bar{\mathbf{C}}$, is also diagonal. In this case, (3.57) assumes the simple form

$$m_{(j)(j)}\ddot{q}_j + c_{(j)(j)}\dot{q}_j + k_{(j)(j)}q_j = Q_j \qquad (j = 1, \ldots, n). \tag{3.59}$$

In summary, the equations of motion governing the rigid body and flexible motions are given by (3.46) and (3.59), respectively. All of the coefficients appearing in these equations can be obtained in a straightforward manner from a finite element analysis of the spacecraft. Note that these equations continue to hold when concentrated moments and rotational inertias are included in the development. The orientation of B in A is governed by the kinematical differential equations

$$\dot{\epsilon}_1 = \frac{1}{2}(\omega_1\epsilon_4 - \omega_2\epsilon_3 + \omega_3\epsilon_2), \tag{3.60}$$

$$\dot{\epsilon}_2 = \frac{1}{2}(\omega_1\epsilon_3 + \omega_2\epsilon_4 - \omega_3\epsilon_1), \tag{3.61}$$

$$\dot{\epsilon}_3 = \frac{1}{2}(-\omega_1\epsilon_2 + \omega_2\epsilon_1 + \omega_3\epsilon_4), \tag{3.62}$$

$$\dot{\epsilon}_4 = -\frac{1}{2}(\omega_1\epsilon_1 + \omega_2\epsilon_2 + \omega_3\epsilon_3), \tag{3.63}$$

where $\epsilon_1, \ldots, \epsilon_4$ are Euler parameters [38].

A sketch of the spacecraft model used in the example problems is shown in Fig. 3.8. The spacecraft bus is modeled as a rigid body of mass m_b with principal mass moments of inertia, I_1, I_2, and I_3. Unit vectors, $\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$, are fixed in the bus and are aligned with the principal axes. Two massless beam elements of length L with tips of mass m_p are used to model flexible solar panels. The axes of the undeformed beams are assumed to be aligned with the c_1 direction and to pass through the bus COM. The first principal plane of bending for the beams is defined by a vector normal to $\mathbf{n}$. The normal to the second principal plane of bending is then defined as the cross product, $\mathbf{c}_1 \times \mathbf{n}$. Flexural rigidities for bending in the two principal planes are denoted by $(EI)_1$ and $(EI)_2$. Moments M_1, M_2, and M_3 are applied to the spacecraft bus in the $\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$ directions, respectively. Each moment is subject to the inequality constraints $-C_k \leq M_k \leq C_k$ for $k = 1, 2, 3$.

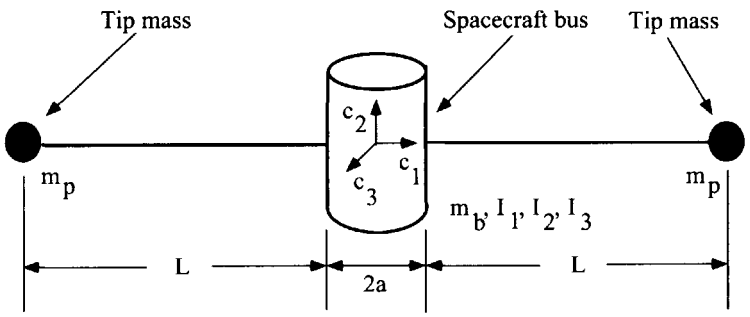


Figure 3.8. *Sketch of model used in examples.*

The example problems deal with rest-to-rest maneuvers whereby the spacecraft is slewed from an initial orientation defined by $\epsilon_1 = \epsilon_2 = \epsilon_3 = 0$ and $\epsilon_4 = 1$ to a final orientation defined by $\epsilon_k = \epsilon_{kf}$ for $k = 1, \ldots, 4$. The goal is to reorient the spacecraft and bring it to a quiescent state in a specified time, T, while minimizing the performance index

$$\Gamma = \int_0^T \left(M_1^2 + M_2^2 + M_3^2\right) dt. \tag{3.64}$$

Since closed-form solutions to (3.60)–(3.63) do not exist in general, it is necessary to use numerical procedures to solve the optimal control problem. In particular, a fourth-order RK algorithm is used to obtain the equivalent of (3.16). To solve the following example problems, a DDP algorithm, which will be developed in Chapter 4, will be used. The salient features of the DDP algorithm are

1. quadratic convergence near the solution;
2. inequality constraints for inputs accounted for exactly;
3. equality constraints for states imposed using a quadratic penalty function;
4. optimality conditions determined from definiteness of $p \times p$ matrices;
5. order Nnp storage required for each iteration;
6. order Nn^3 operations required for each iteration.

In the above description, n is the number of states, p is the number of inputs, and N is the number of time steps used to discretize the problem.

Values for the dimensionless parameters used in the examples are provided in Table 3.1. Two antisymmetric bending modes are retained in the assumed displacement expansion (see (3.48)).

Table 3.1. *Parameters used in examples.*

Parameter	Value
$\frac{a}{L}$	0.50
$\frac{m_p}{m_b}$	0.06
$\frac{I_2}{I_1}$	0.50
$\frac{I_3}{I_1}$	1.00
$m_p\frac{(a+L)^2}{I_1}$	1.38
$\frac{(EI)_2}{(EI)_1}$	3.00
$\frac{(EI)_1T^2}{(m_pL^3)}$	150
$\frac{CT^2}{(I_2+2m_p(a+L)^2)}$	8.00

For this simple model, these are the only two modes excited by the applied moments. The free-free mode shapes for the examples are mass matrix normalized and were calculated using the commercial finite element code MSC.NASTRAN. Results presented in Figs. 3.9–3.11 are plotted as functions of the dimensionless time variable, $r = t/T$. The generalized coordinates, q_1 and q_2, are associated with deformation nominally in the first and second principal planes of bending, respectively.

The first example is concerned with a maneuver in which reorientation of the spacecraft can be accomplished simply by slewing about the 2-axis of the bus.

In this example, the normal $\mathbf{n}$ is chosen as $\mathbf{c}_3$ so that the principal planes of bending are aligned with the principal axes of the spacecraft. The final orientation of the satellite is defined by the Euler parameters $\epsilon_{1f} = \epsilon_{3f} = 0$, $\epsilon_{2f} = \sin(\theta/2)$, and $\epsilon_{4f} = \cos(\theta/2)$, where $\theta = 1.7$ rad. This reorientation corresponds to a rotation of 1.7 rad about the 2-axis of the bus. Plots of the dimensionless moments $u_k = M_k/C$ $(k = 1, 2, 3)$, angular velocity measure numbers, Euler parameters, and generalized coordinates are shown in Fig. 3.9. As expected, only u_2 is nonzero and out-of-plane vibrations are absent, as indicated by the constant zero value of q_2. Notice also the antisymmetry of u_2 about $\tau = 1/2$.

The second example is identical to the first with the exception that the normal is $\mathbf{n} = (\mathbf{c}_2 + \mathbf{c}_3)/\sqrt{2}$. This choice for $\mathbf{n}$ corresponds to a 45° misalignment of the principal planes of bending with the principal axes of the spacecraft. Plots of the results are shown in Fig. 3.10. Notice that both out-of-plane moments and vibrations occur. Note that it is still possible to achieve the rest-to-rest maneuver simply with a moment about the 2-axis of the bus, but this is not optimal. It may also occur that, as the angle of rotation θ is increased, the maneuver is possible only with nonzero values for M_1 and M_3. Otherwise, the inequality constraints on M_2 may be prohibitively stringent. Although it is difficult to see the behavior of the other two inputs from the figure, u_1 is symmetric and u_3 is antisymmetric about $\tau = 1/2$.

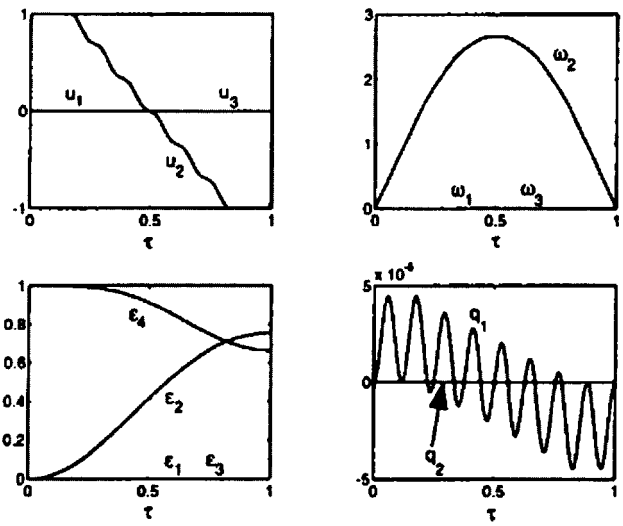


Figure 3.9. *Numerical simulation results for first example. (From Dohrmann and Robinett* [36]. © *Institute of Electrical and Electronics Engineers, Inc.)*

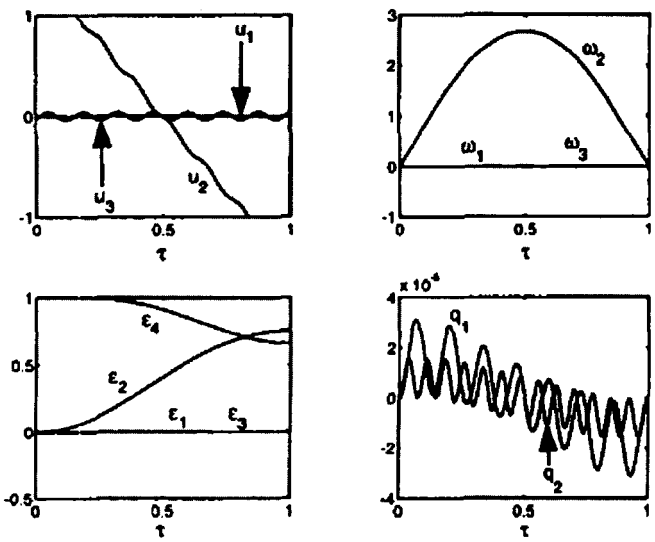


Figure 3.10. *Numerical simulation results for second example. (From Dohrmann and Robinett* [36]. © *Institute of Electrical and Electronics Engineers, Inc.).*

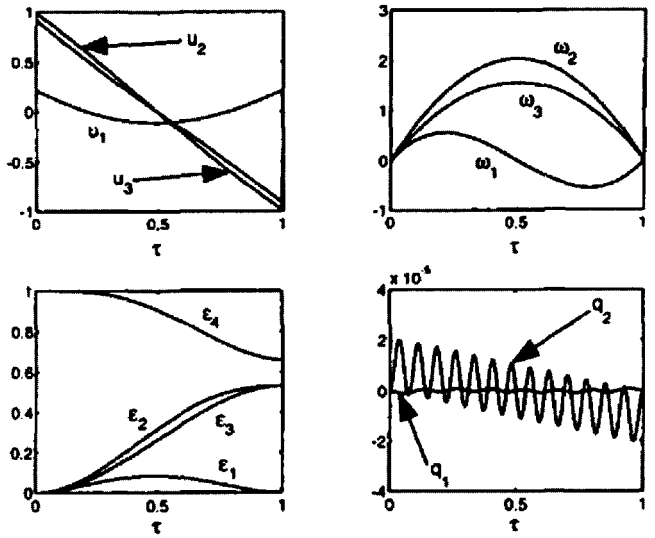


Figure 3.11. *Numerical simulation results for third example. (From Dohrmann and Robinett* [36]. © *Institute of Electrical and Electronics Engineers, Inc.)*

The third example is identical to the second with the exception that the final orientation is defined by the Euler parameters $\epsilon_{1f} = 0, \epsilon_{2f} = \epsilon_{3f} = \sin(\theta/2)/\sqrt{2}$, and $\epsilon_{4f} = \cos(\theta/2)$, where $\theta = 1.7$ rad. This example examines a situation in which the maneuver could be accomplished by slewing about a principal axis of bending not aligned with a principal axis of the spacecraft. If the optimal maneuver were such, then ω_1 would be zero for all times. This, however, is clearly not the case, as is evident from the results presented in Fig. 3.11. Notice also the symmetries of the inputs about $\tau = 1/2$. One final observation: All of the initial guesses of the torque profiles were zero since a homotopy scheme was used to enforce the constraints. Arbitrary cost functionals will be discussed in Chapter 4.

3.4 Summary

This chapter introduced the concepts of the principle of optimality and DP. These concepts were used to create a DDP algorithm to solve linear optimization problems. Several numerical implementation details were discussed via several example problems. Finally, extensions to the DDP algorithm for linear problems that are necessary to solve nonlinear constrained problems were discussed via an input torque shaping problem for three-dimensional slew maneuvers of flexible spacecraft. The discussion of these extensions has set the stage for Chapter 4.

This page intentionally left blank

Chapter 4
Advanced Dynamic Programming

4.1 Introduction

Several desirable attributes of dynamic programming (DP) provide the reasons for using these algorithms. Here is a summary:

- The number of numerical operations is linear in the number of optimization parameters, which enables one to solve for a smooth input/control history. This is in contrast to SQP/RQP, where the number of operations is cubic or quadratic.
- The combination of discrete dynamic programming (DDP) and homotopy enables one to initialize the optimization problem with a zero initial guess. This means that the algorithm eliminates the sensitivity to the initial guess.
- DDP enables one to attack the true optimization problem with a direct method instead of initializing an indirect method with a direct method to obtain the same results.

This chapter expands upon these attributes by directly enforcing equality and inequality constraints on the states and controls. In Chapter 3, these constraints were handled with penalty functions, which often resulted in mediocre performance. The equality constraints are imposed by Lagrange multipliers, while the inequality constraints are enforced with a primal-dual interior point method. Before discussing these enhancements, this chapter begins, as the previous chapter finished, with an advanced application of DDP to the real-time guidance of the multiple launch rocket system (MLRS). The development in Section 4.2 is originally from Dohrmann, Eisler, and Robinett [39, 40].[2]

4.2 A Dynamic Programming Approach to Rocket Guidance Problems

This example applies the DP approach to a burnout-to-apogee guidance of precision munitions problem. It demonstrates an innovative way to solve the "optimal return" problem,

[2]Revised and reprinted by permission of the American Institute of Aeronautics and Astronautics, Inc.

which produces a real-time feedback guidance algorithm that is analogous to the linear quadratic regulator (LQR) problem. The solution of the optimal return from any point is the real strength of DDP.

The use of aided navigation via satellite updates promises unparalleled, cost-effective, and accurate systems for both civilian and defense purposes. Contemporary defense systems routinely integrate conventional inertial measurement unit (IMU) technology with global positioning system (GPS) updating to produce impressive targeting results. This has been most notable for standoff munitions such as cruise missiles and guided bombs [41, 42, 43]. A recent study by Singh, Fraysee, and Kohli [44], was directed at fabricating the IMU/GPS combination in solid-state form able to withstand artillery gun setback loads. The present study seeks to capitalize on the cost-effectiveness issue by formulating a guidance algorithm for a field munition solely reliant on offboard navigation aids. A major drawback of using only offboard sources, which this work addresses, is that the update signals can be obscured by electronic jamming. Given that this issue is addressed here, it is further assumed that retrofits of existing, extensively used, ballistic munitions with low-cost navigation and actuation hardware add-ons would be justified by significantly lowering the costs of defended target kills via increased accuracy.

Precision and accuracy are the two major issues in munitions targeting. Important factors of both are the effects of launch offsets and winds. Although this study is concerned primarily with the former, wind effects can be accounted for by using more complex aerodynamic models. The method presented here remains essentially the same regardless of the aerodynamic model used.

As stated previously, jamming of aided navigation signals by the adversary must be taken into account for the viability of the proposed scheme. If not, the current IMU/GPS-style integration must be relied upon, which entails a significant cost penalty. The approach taken here is to apply guidance during the initial phase of flight, where jamming effects are

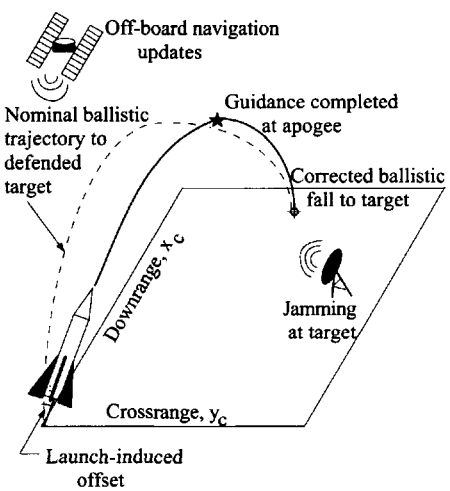


Figure 4.1. *One possible guided rocket scenario.*

minimal, and to complete all guidance updates and reestablish nominal ballistic accuracy before descent to the target (see Fig. 4.1). To date, previous work for burnout-to-apogee guidance appears nonexistent. The flight regime for this munition is below 50,000 ft and supersonic. In this regime, aerodynamic and gravity forces are of comparable magnitude and both change significantly along the flight path. As such, none of the usual simplifying approximations to the equations of motion for atmospheric flight can be made [45] and an analytic solution cannot be obtained. Nonlinear programming (NP) solutions are certainly viable alternatives to generate control histories for a problem involving both boundary conditions and interior constraints. However, these solutions would be of an open-loop rather than a feedback form. NP solutions would need to be computed for a large number of different trajectories, with highly involved interpolations required to keep this number manageable and to span a useful guidance "space."

As an alternative, a feedback control scheme based on the method of DP is presented in this study. DP offers the following benefits:

1. It retains all relevant physics of the problem in computing guidance commands.

2. It accommodates a variety of different launch offsets.

3. It provides a set of feedback gains for optimal return from any point in the guided portion of the trajectory.

The price of this capability is the necessity of computing and storing a matrix of feedback gains at as many update points as one feels are necessary to provide acceptable performance. Given that contemporary field munition operations require on-line targeting computations to account for such variables as weather conditions, it is felt that the proposed computational requirements would be compatible with such exercises. In addition, to minimize the attendant actuation hardware costs, the guidance scheme would ideally generate commands of minimal magnitude. This would allow considerable latitude in the use of internal [46] versus external methods for attitude control.

In the interest of simplicity and to highlight the salient features of the proposed algorithm, a retrofitted munition capable of independent lift and side-force generation is assumed (this method can be generalized to an oriented lift-plane control system as well). Representative aerodynamics for an MLRS [47] are used. First, the physical model is developed, followed by the guidance control problem formulation and the DP solution. Finally, simulation results are shown for a single set of feedback gains applied to two sets of launch errors.

4.2.1 Physical Model

In this study, an artillery rocket is assumed to have unguided launch and propulsive phases. Guidance commences after motor burnout. The physical model for the guidance portion of the rocket flight is that of an unpowered point mass flying over a spherical, nonspinning Earth at an altitude that is small with respect to the radius of Earth. The equations

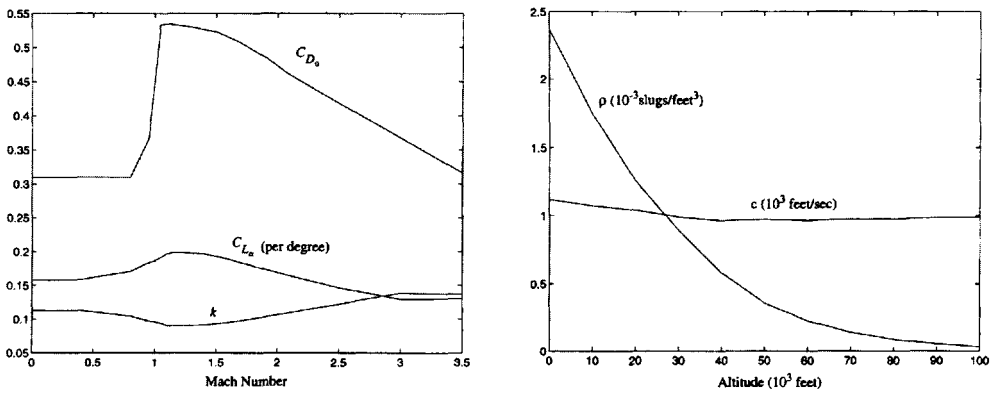


Figure 4.2. *Aerodynamic (left plot) and atmospheric (right plot) data. (From Dohrmann, Eisler, and Robinett* [39]. *Reprinted by permission of the American Institute of Aeronautics and Astronautics, Inc.)*

of motion are

$$\begin{aligned}
\dot{x}_c &= V\cos\gamma\cos\Psi,\\
\dot{y}_c &= V\cos\gamma\sin\Psi,\\
\dot{z}_c &= V\sin\gamma,\\
\dot{V} &= \frac{-\rho V^2 C_D S}{2m} - g\sin\gamma,\\
\dot{\gamma} &= \frac{\rho V^2 C_L S}{2mV} - \left(\frac{g}{V} - \frac{V}{r}\right)\cos\gamma,\\
\dot{\Psi} &= \frac{\rho V^2 C_S S}{2mV\cos\gamma}.
\end{aligned} \tag{4.1}$$

The coefficients, C_L and C_S, are assumed linear over a small aerodynamic angle range and related to angles of attack, α, and sideslip, β, via $C_L = C_{L_\alpha}\alpha$ and $C_S = C_{L_\alpha}\beta$, assuming an aerodynamically symmetric vehicle. Drag variations with lift and side force are fitted as a quadratic function according to $C_D = C_{D_o} + k(C_L^2 + C_S^2)$, where C_{D_o} and k are Mach number dependent and plotted with C_{L_α} in Fig. 4.2 (left plot). The density and speed of sound are tabulated for a standard atmosphere and are shown in Fig. 4.2 (right plot). This model complexity was felt necessary due to the fact that the munition will be flying in the transonic to moderate-supersonic regime, where aerodynamic changes are significant. More complex aerodynamic models that account for current weather conditions could also be accommodated.

4.2.2 Guidance Problem

The input $\mathbf{u}$ for the guidance problem is chosen as

$$\mathbf{u} = [\dot{C}_L \quad \dot{C}_S]^T. \tag{4.2}$$

The physical model given by (4.1) along with (4.2) can be expressed in the standard form

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \qquad \mathbf{x}(t_1) = \mathbf{x}_1, \tag{4.3}$$

where $\mathbf{x}$ is the state defined as

$$\mathbf{x} = \begin{bmatrix} x_c & y_c & z_c & V & \gamma & \Psi & C_L & C_S \end{bmatrix}^T. \tag{4.4}$$

The nominal trajectory $\hat{\mathbf{x}}(t)$ corresponds to the input $\mathbf{u}(t) = \mathbf{0}$ and the initial conditions

$$\hat{\mathbf{x}}(t_1) = [\hat{x}_c(t_1) \quad \hat{y}_c(t_1) \quad \hat{z}_c(t_1) \quad \hat{V}(t_1) \quad \hat{\gamma}(t_1) \quad \hat{\Psi}(t_1) \quad 0 \quad 0]^T. \tag{4.5}$$

It is assumed that the nominal trajectory hits the target at the final time T. That is,

$$\begin{aligned} \hat{x}_c(T) &= x_{ct}, \\ \hat{y}_c(T) &= y_{ct}, \\ \hat{z}_c(T) &= z_{ct}, \end{aligned} \tag{4.6}$$

where (x_{ct}, y_{ct}, z_{ct}) are the coordinates of the target.

In practice, it is unlikely that the nominal trajectory will hit the target because the correct initial conditions cannot be enforced exactly and because the accuracy of the aerodynamic model is limited. The approach taken here is to control the portion of the flight between the times t_1 and t_N ($t_N < T$) in such a manner that the corrected ballistic trajectory of the munition is directed toward the target. In the present study, t_N is set equal to the time at apogee. It is assumed for $t \geq t_N$ that navigation signals are jammed and the lift and side-force coefficients are set to zero.

The guidance problem is stated as follows. Find the input, $\mathbf{u}(t)$, that minimizes the functional

$$\Gamma(\mathbf{u}) = \lambda \begin{bmatrix} (x_c(t_f) - x_{ct})^T & (y_c(t_f) - y_{ct})^T \end{bmatrix} \begin{bmatrix} (x_c(t_f) - x_{ct}) \\ (y_c(t_f) - y_{ct}) \end{bmatrix} + \int_{t_1}^{t_f} [\mathbf{u}(t)]^T [\mathbf{u}(t)]\, dt \tag{4.7}$$

subject to (4.3) and the constraints

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}(t) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{for} \quad t = t_1 \quad \text{and} \quad t_N \leq t \leq t_f, \tag{4.8}$$

where t_f denotes the time of impact. The term premultiplied by the scalar, $\lambda > 0$, in (4.7) is equal to the square of the distance between the point of impact and the target. Thus, λ directly affects the tradeoff between accuracy and control effort as measured by the integral term in (4.7). For $\lambda = 0$, the input is zero for all times, resulting in a ballistic trajectory. In the limit as $\lambda \to \infty$, the resulting trajectory hits the target. The previous statement implicitly assumes that the physical model and measurements of the initial conditions are exact. This assumption clearly is not valid in practice, so care must be taken in choosing an appropriate value for λ. The topic of choosing λ is discussed further in Section 4.2.5. Equation (4.8) states that the lift and side-force coefficients are zero initially and zero during the portion of the trajectory between times t_N and t_f.

Notice that the time at impact, t_f (final time), in (4.7) is unspecified and need not equal the time at impact, T, for the nominal trajectory. In Section 4.2.3, the guidance problem is transformed into a standard form in which the final time is specified and the constraints given by (4.8) are satisfied automatically.

4.2.3 Transformation of the Guidance Problem

It proves useful to introduce the ballistic trajectory $\bar{\mathbf{x}}(t)$ from t_N and t_f corresponding to the input $\mathbf{u}(t) = \mathbf{0}$ and initial conditions at $t = t_N$ of

$$\bar{\mathbf{x}}(t_N) = \begin{bmatrix} \bar{x}_c(t_N) & \bar{y}_c(t_N) & \bar{z}_c(t_N) & \bar{V}(t_N) & \bar{\gamma}(t_N) & \bar{\Psi}(t_N) & 0 & 0 \end{bmatrix}^T. \tag{4.9}$$

Defining t_f as the time at impact for $\bar{\mathbf{x}}(t)$ and

$$\Delta t_f = t_f - T, \tag{4.10}$$

$$\Delta \mathbf{x}(t_N) = \bar{\mathbf{x}}(t_N) - \hat{\mathbf{x}}(t_N), \tag{4.11}$$

the following linear approximation in Δt_f and $\Delta \mathbf{x}(t_N)$ is obtained:

$$\bar{z}_c(t_f) = z_{ct} + \mathbf{C}\Delta \mathbf{x}(t_N) + d\Delta t_f, \tag{4.12}$$

$$\begin{bmatrix} \bar{x}_c(t_f) \\ \bar{y}_c(t_f) \end{bmatrix} = \begin{bmatrix} x_{ct} \\ y_{ct} \end{bmatrix} + \mathbf{E}\Delta \mathbf{x}(t_N) + \mathbf{F}\Delta t_f. \tag{4.13}$$

The last two columns of the sensitivity matrices, $\mathbf{C}$ and $\mathbf{E}$, in (4.12) and (4.13) can be set equal to zero since the last two elements of $\Delta \mathbf{x}(t_N)$ are zero. All of the sensitivity matrices in (4.12) and (4.13) can be determined from numerical integration of the aerodynamic model along with finite difference calculations. Another option is to use an automatic differentiation program to obtain these terms. In this study, finite difference calculations were used.

Setting $\bar{z}_c(t_f)$ equal to z_{ct} in (4.12) leads to

$$\Delta t_f = -\mathbf{C}\frac{\Delta \mathbf{x}(t_N)}{d}. \tag{4.14}$$

Combining (4.13) and (4.14), one obtains

$$\begin{bmatrix} \bar{x}_c(t_f) - x_{ct} \\ \bar{y}_c(t_f) - y_{ct} \end{bmatrix} = \mathbf{G}\Delta \mathbf{x}(t_N), \tag{4.15}$$

where

$$\mathbf{G} = \mathbf{E} - \frac{1}{d}\mathbf{F}\mathbf{C}. \tag{4.16}$$

Equation (4.15) is a key relation that will be used to transform the guidance problem to an equivalent standard form.

Returning for a moment to the controlled portion of the trajectory, the input is assumed to be discretized for any time $t_1 \leq t \leq t_N$ as

$$\mathbf{u}(t) = \mathbf{u}_k \quad \text{for} \quad t_k \leq t < t_{k+1} \quad \text{and} \quad k = 1, \ldots, N-1, \tag{4.17}$$

where

$$t_k = t_1 + h(k-1) \tag{4.18}$$

and

$$h = \frac{(t_N - t_1)}{(N-1)}. \tag{4.19}$$

The input is set equal to zero for $t \geq t_N$.

From its definition (see (4.9)), it is clear that $\bar{\mathbf{x}}(t)$ satisfies (4.8) for $t_n \leq t \leq t_f$. Thus, without loss of generality, one can consider $\mathbf{x}(t) = \bar{\mathbf{x}}(t)$ for $t_N \leq t \leq t_f$. Consequently, substitution of (4.15) and (4.17) into (4.7) yields

$$\Gamma(\mathbf{u}_k) = \lambda \tilde{\mathbf{x}}_N^T \mathbf{G}^T \mathbf{G} \tilde{\mathbf{x}}_N + h \sum_{k=1}^{N-1} \mathbf{u}_k^T \mathbf{u}_k, \tag{4.20}$$

where

$$\tilde{\mathbf{x}}_k = \mathbf{x}(t_k) - \hat{\mathbf{x}}(t_k), \quad k = 1, \ldots, N. \tag{4.21}$$

In order to satisfy (4.8) at $t = t_N$, the seventh and eighth elements of $\tilde{\mathbf{x}}_N$ must equal zero. Thus, for an input of the form given by (4.17), integration of (4.2) between the limits of t_{N-1} and t_N implies that

$$\mathbf{u}_{N-1} = -\mathbf{D}\tilde{\mathbf{x}}_{N-1}, \tag{4.22}$$

where

$$\mathbf{D} = \frac{1}{h} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.23}$$

Linearizing the dynamics about the nominal trajectory, one obtains

$$\tilde{\mathbf{x}}_{k+1} = \mathbf{A}\tilde{\mathbf{x}}_k + \mathbf{B}_k \mathbf{u}_k, \quad k = 1, \ldots, N-1, \tag{4.24}$$

where the state transition matrices $\mathbf{A}_k$ and $\mathbf{B}_k$ can be determined with recourse to a numerical integration scheme (refer to Section A.1 in Appendix A). Substituting (4.22) and (4.24) with $k = N-1$ into (4.20) yields

$$\Gamma(\mathbf{u}_k) = \tilde{\mathbf{x}}_{N-1}^T \mathbf{W}_{N-1} \tilde{\mathbf{x}}_{N-1} + h \sum_{k=1}^{N-2} \mathbf{u}_k^T \mathbf{u}_k, \tag{4.25}$$

where

$$\mathbf{W}_{N-1} = \lambda \left(\mathbf{A}_{N-1} - \mathbf{B}_{N-1}\mathbf{D}\right)^T \mathbf{G}^T \mathbf{G} \left(\mathbf{A}_{N-1} - \mathbf{B}_{N-1}\mathbf{D}\right) + h\mathbf{D}^T\mathbf{D}. \tag{4.26}$$

In summary, the original guidance problem has been transformed from minimization of a functional with an unspecified final time to minimization of the function given by (4.25) with a specified final time.

4.2.4 Dynamic Programming Solution

Minimization of (4.25) subject to (4.24) can be accomplished in a straightforward manner using the DP method. Detailed explanations of the theory and application of DP are available in several texts [48, 49]. The algorithm for the guidance problem of interest is summarized in the following two-step procedure.

Step 1: Starting with $\mathbf{W}_{N-1}$ given by (4.26), calculate $\mathbf{W}_k$ recursively for $k = N-2, \ldots, 1$ from

$$\mathbf{W}_k = \mathbf{A}_k^T \mathbf{W}_{k+1} \mathbf{A}_k - \mathbf{H}_{2k} \mathbf{H}_{3k}^{-1} \mathbf{H}_{2k}^T, \tag{4.27}$$

where

$$\mathbf{H}_{2k} = \mathbf{A}_k^T \mathbf{W}_{k+1} \mathbf{B}_k, \tag{4.28}$$

$$\mathbf{H}_{3k} = h \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \mathbf{B}_k^T \mathbf{W}_{k+1} \mathbf{B}_k. \tag{4.29}$$

In the process, store the feedback gain matrices

$$\mathbf{K}_k = \mathbf{H}_{3k}^{-1} \mathbf{H}_{2k}^T. \tag{4.30}$$

All of these calculations are performed before launching the munition.

Step 2: Calculate the inputs $\mathbf{u}_1, \ldots, \mathbf{u}_{N-1}$ from the equation

$$\mathbf{u}_k = -\mathbf{K}_k \tilde{\mathbf{x}}_k \tag{4.31}$$

and $\mathbf{u}_{N-1}$ from (4.22). The $\tilde{\mathbf{x}}_k$ terms in (4.22) and (4.31) are calculated by substituting the measured and nominal trajectories into (4.21). The constraint of zero lift and side-force coefficients at the beginning of the trajectory is satisfied by setting the elements in rows 7 and 8 of $\tilde{\mathbf{x}}_1$ equal to zero. As the inputs are calculated, C_L and C_S are determined by integrating (4.2). The result is

$$\begin{bmatrix} C_L(t_{k+1}) \\ C_S(t_{k+1}) \end{bmatrix} = \begin{bmatrix} C_L(t_k) \\ C_S(t_k) \end{bmatrix} + h\mathbf{u}_k, \quad k = 1, \ldots, N-1, \tag{4.32}$$

and, between the times t_k and t_{k+1}, C_L and C_S vary linearly.

An attractive feature of the control law given by (4.31) is that it provides optimal return from any point in the guided portion of the trajectory. That is, the control law is optimal in the sense that it minimizes the function

$$\Gamma_k(\tilde{\mathbf{x}}_k, \mathbf{u}_i) = \tilde{\mathbf{x}}_{N-1}^T \mathbf{W}_{N-1} \tilde{\mathbf{x}}_{N-1} + h \sum_{i=k}^{N-2} \mathbf{u}_i^T \mathbf{u}_i \tag{4.33}$$

subject to (4.24) for all values of $\tilde{\mathbf{x}}_k$, not just the one on the optimal trajectory for the original launch offset $\tilde{\mathbf{x}}_1$. Thus, if at times intermediate to t_1 and t_N the trajectory is "bumped" away

from the optimal owing to modeling, measurement, or actuation errors, the control law is attractive. However, success of the algorithm ultimately depends on having adequate physical models and measurement systems. As a final note, the control law presented above is equivalent to what one would obtain using discrete linear-quadratic guidance [49] owing to the use of linearized dynamics, a nominal trajectory that hits the target, and the quadratic form of the function given by (4.25).

4.2.5 Numerical Simulation Results

The MLRS-type vehicle for this study has a burnout mass of 14.25 slugs and a reference area of 0.4356 ft^2. Boundary conditions for the nominal trajectory are shown in Table 4.1. Both the nominal and the controlled trajectories were obtained by numerically integrating the nonlinear differential equations given by (4.3) using a fixed-step, fourth-order Runge–Kutta (RK) method [50]. The apogee altitude is approximately 41,000 ft and Mach number variations are from 1.05 to 2.33. The nominal trajectory was discretized in the time domain with 500 time steps.

Table 4.1. *Boundary conditions for the nominal trajectory.*

Variable	Initial condition	Final value	Unit
x_c	0	98,270	ft
y_c	0	0	ft
z_c	7390	0	ft
V	2532	1332	ft/sec
γ	48.64	−65.13	deg
Ψ	0	0	deg

Feedback gains for the control scheme were generated by choosing $\lambda/h = 1 \times 10^{-6}$. The states $\mathbf{x}_1, \ldots, \mathbf{x}_N$ ($N = 213$) for the controlled portion of the trajectory, $t_1 \leq t \leq t_N$, were obtained using the inputs generated from (4.31) and (4.22). The corrected ballistic trajectory for $t > t_N$ was obtained by setting both the lift and side-force coefficients to zero.

Two different sets of launch offsets were considered: (1) γ and Φ offsets of 0.36° and 1° from nominal at burnout, respectively, and (2) γ and Φ offsets of −0.64° and −0.75°, respectively. In Fig. 4.3, the aerodynamic control angles (α, β) for the two cases computed use the same set of feedback gains. These angles were obtained from the equations $\alpha = C_L/C_{L_\alpha}$ and $\beta = C_S/C_{L_\alpha}$ (see Section 4.2.1) using the calculated values for C_L and C_S. Notice in both cases that the angular magnitudes are well below 1°. Figure 4.3 shows the point mass trajectories for the two cases (middle plots). In Fig. 4.3 (middle left plot), the controlled and uncontrolled trajectories as well as the nominal are plotted. Note that even for the small γ and Φ offsets at burnout, the target miss distance, as shown by the uncontrolled trajectory, can be substantial (>1700 ft). The controlled trajectory for case 2 is plotted on an expanded crossrange scale in Fig. 4.3 (middle right plot). The miss distances for cases 1 and 2 were approximately 7 ft and 5 ft, respectively. Velocity (bottom left plot) and horizontal heading angle (bottom right plot) time histories for the two cases are shown in Fig. 4.3. Notice that the horizontal heading angle is a constant after apogee in both cases.

As was mentioned previously, selection of the scalar parameter λ (see (4.7)) is an important consideration in the practical implementation of the proposed guidance scheme.

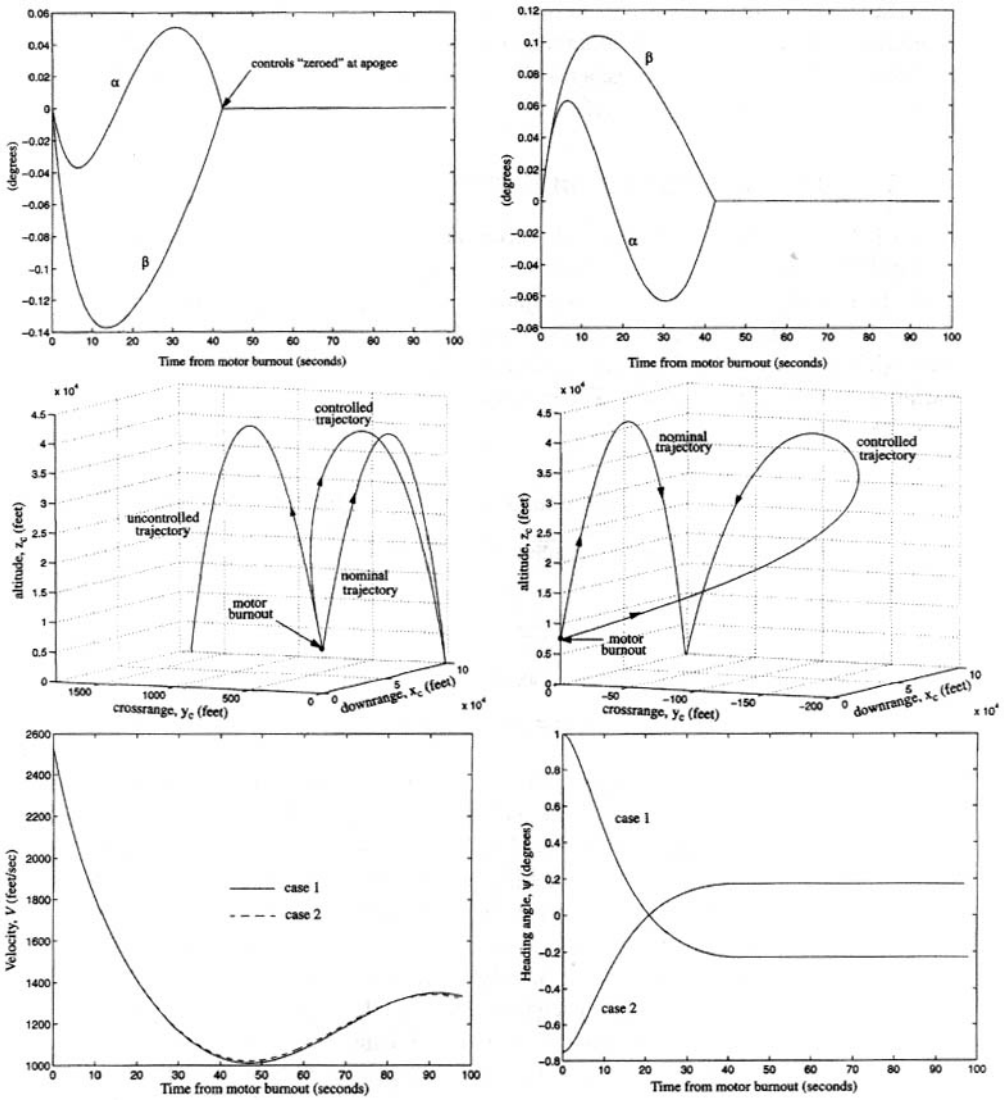


Figure 4.3. *Results are shown for the angle of attack, α, and sideslip, β, time histories for cases* 1 *(upper left plot) and* 2 *(upper right plot), the trajectory comparisons for cases* 1 *(middle left plot) and* 2 *(middle right plot), and the velocity (bottom left plot) and heading angle (bottom right plot) time histories. (From Dohrmann, Eisler, and Robinett* [39, 40]. *Reprinted by permission of the American Institute of Aeronautics and Astronautics, Inc.)*

One option is to choose a relatively large value for λ such that the corrected ballistic trajectory impact is very close to the target. Indeed, such an approach would lead to desirable results in computer simulations. In practice, however, choosing a large value for λ could lead to undesirable performance of the control system.

In the limit as $\lambda \to \infty$, the feedback gains (see (4.30)) asymptotically approach a finite fixed value. As the time step for discretization, h, becomes very small, the gains for the portion of the trajectory near $t = t_N$ become very large. A similar situation is present with proportional navigation [51] schemes, where feedback gains become infinite as the time-to-go approaches zero. One consequence of large gains is that disturbances caused by modeling or measurement errors can lead to undesirably large control inputs near $t = t_N$.

In selecting an appropriate value for λ, one must balance the competing goals of targeting accuracy and control system insensitivity to disturbances. One starting point is to choose λ as small as possible such that the specified targeting requirements are met for the anticipated range of launch offsets. The value of λ could then be increased depending on the accuracy of the measurement system and physical model. Although the topic of choosing the value of λ is not addressed in complete detail by this study, it is hoped that some of the important issues involved have been conveyed.

4.3 Sequential Quadratic Programming Implementations for Equality-Constrained Optimal Control

This section is based on excerpts from Dohrmann and Robinett [29, 30][3] and describes extensions and modifications of the basic DDP algorithm developed in Chapter 3 to directly enforce equality constraints. In particular, efficient SQP implementations are presented for equality-constrained, discrete-time, optimal control problems. The algorithms are applicable to problems with either fixed or free final times. Constraints are enforced using a Lagrange multiplier approach whereby inconsistent or redundant constraints are readily identified. Problem solutions are obtained by iteratively solving a series of constrained quadratic programs. The number of mathematical operations required for each iteration is proportional to the number of discrete times N. This is in contrast to a conventional SQP method for which this number is proportional to N^3 or N^2 for quasi-Newton methods. The algorithms exhibit quadratic convergence under the same conditions as those for SQP and simplify to an existing DP approach when there are no constraints and the final time is fixed. Two simple test problems and three application problems are used to demonstrate the algorithms. The application examples include a satellite dynamics problem, a pursuit problem with a nonholonomic constraint, and a set of brachistochrone problems involving viscous friction.

4.3.1 Fixed Final Time Problems

Consider the initial value problem

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(t_1) = \mathbf{x}_1, \tag{4.34}$$

[3]Revised and reprinted by permission of Kluwer Academic/Plenum Publishers, Inc.

where $\mathbf{x} \in R^n$ is the state and $\mathbf{u} \in R^m$ is the control. The control is assumed to be discretized temporally as

$$\mathbf{u}(t) = \mathbf{u}_i, \quad t_i \leq t < t_{i+1}, \tag{4.35}$$

for $i = 1, \ldots, N-1$, where $t_1, \ldots, t_N$ are given. Given the existence and uniqueness of the solution to the initial value problem, adjacent states in time can be related as

$$\mathbf{x}_{i+1} = \mathbf{g}_i(\mathbf{x}_i, \mathbf{u}_i) \tag{4.36}$$

for $i = 1, \ldots, N-1$, where $\mathbf{x}_i = \mathbf{x}(t_i)$.

The equality-constrained, discrete-time, optimal control problem P1 is stated as follows. Given the initial state $\mathbf{x}_1$, find the controls $\mathbf{u}_1, \ldots, \mathbf{u}_{N-1}$ that minimize the function

$$\Gamma = \sum_{i=1}^{N-1} \Gamma_i(\mathbf{x}_i, \mathbf{u}_i) + \Gamma_N(\mathbf{x}_N) \tag{4.37}$$

subject to (4.36) and the constraints

$$\bar{\mathbf{c}}_i(\mathbf{x}_{i+1}, \mathbf{u}_i) = 0 \text{ and } \bar{\mathbf{c}}_N(\mathbf{x}_N) = 0 \tag{4.38}$$

for $i = 1, \ldots, N-1$, where $\bar{\mathbf{c}}_i \in R^{n_i}$. The functions $\mathbf{g}_i$, Γ_i, and $\bar{\mathbf{c}}_i$ are all assumed to be twice differentiable.

An approximate solution to P1 can be obtained by introducing a penalty function to enforce the constraints. For example, the solution to the problem of minimizing the function

$$L_\rho = \sum_{i=1}^{N-1} \left[\Gamma_i(\mathbf{x}_i, \mathbf{u}_i) + \frac{\rho}{2} \bar{\mathbf{c}}_i^T(\mathbf{x}_{i+1}, \mathbf{u}_i) \bar{\mathbf{c}}_i(\mathbf{x}_{i+1}, \mathbf{u}_i) \right] + \Gamma_N(\mathbf{x}_N) + \frac{\rho}{2} \bar{\mathbf{c}}_N^T(\mathbf{x}_N) \bar{\mathbf{c}}_N(\mathbf{x}_N) \tag{4.39}$$

subject to (4.36) converges to the solution of P1 under mild conditions as the scalar parameter $\rho > 0$ approaches ∞ [9]. The use of penalty functions to enforce constraints often has its merits, but can lead to ill conditioning of the problem.

As an alternative to using penalty functions, the algorithm presented in this section uses a Lagrange multiplier technique to enforce the constraints. Solutions to P1 can be found with this algorithm by solving a series of constrained quadratic programming subproblems. The distinguishing feature of this algorithm is the efficient manner in which the subproblems are solved.

Consider the Lagrangian function, L, associated with (4.37)–(4.38) and defined as

$$L = \sum_{i=1}^{N-1} \left[\Gamma_i(\mathbf{x}_i, \mathbf{u}_i) - \bar{\mathbf{c}}_i^T(\mathbf{x}_{i+1}, \mathbf{u}_i) \boldsymbol{\lambda}_i \right] + \Gamma_N(\mathbf{x}_N) + \bar{\mathbf{c}}_N^T(\mathbf{x}_N) \boldsymbol{\lambda}_N, \tag{4.40}$$

where $\boldsymbol{\lambda}_i \in R^{\bar{n}_i}$ is a vector of Lagrange multipliers. With the goal of approximating the Lagrangian as a quadratic function of the controls and Lagrange multipliers, perturbations of variables $\tilde{\mathbf{x}}_i$, $\tilde{\mathbf{u}}_i$, and $\tilde{\boldsymbol{\lambda}}_i$ are introduced and defined as

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i - \hat{\mathbf{x}}_i, \tag{4.41}$$

$$\tilde{\mathbf{u}}_i = \mathbf{u}_i - \hat{\mathbf{u}}_i, \tag{4.42}$$

$$\tilde{\boldsymbol{\lambda}}_i = \boldsymbol{\lambda}_i - \hat{\boldsymbol{\lambda}}_i, \tag{4.43}$$

where $\hat{\mathbf{x}}_i$, $\hat{\mathbf{u}}_i$, and $\hat{\boldsymbol{\lambda}}_i$ are the nominal values of $\mathbf{x}_i$, $\mathbf{u}_i$, and $\boldsymbol{\lambda}_i$, respectively.

Expanding (4.36) as a Taylor series about $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{u}}_i$ yields

$$\tilde{x}_{i+1}^{\alpha} = a_i^{\alpha\beta}\tilde{x}_i^{\beta} + b_i^{\alpha\beta}\tilde{u}_i^{\beta} + \frac{1}{2}(c_i^{\alpha\beta\gamma}\tilde{x}_i^{\beta}\tilde{x}_i^{\gamma} + 2d_i^{\alpha\beta\gamma}\tilde{x}_i^{\beta}\tilde{u}_i^{\gamma} + e_i^{\alpha\beta\gamma}\tilde{u}_i^{\beta}\tilde{u}_i^{\gamma}) + \mathcal{O}(3), \tag{4.44}$$

where $\mathcal{O}(3)$ denotes cubic and higher-order terms of the perturbed controls, and

$$a_i^{\alpha\beta} = \frac{\partial g_i^{\alpha}}{\partial x_i^{\beta}}, \tag{4.45}$$

$$b_i^{\alpha\beta} = \frac{\partial g_i^{\alpha}}{\partial u_i^{\beta}}, \tag{4.46}$$

$$c_i^{\alpha\beta\gamma} = \frac{\partial^2 g_i^{\alpha}}{\partial x_i^{\beta}\partial x_i^{\gamma}}, \tag{4.47}$$

$$d_i^{\alpha\beta\gamma} = \frac{\partial^2 g_i^{\alpha}}{\partial x_i^{\beta}\partial u_i^{\gamma}}, \tag{4.48}$$

$$e_i^{\alpha\beta\gamma} = \frac{\partial^2 g_i^{\alpha}}{\partial u_i^{\beta}\partial u_i^{\gamma}}. \tag{4.49}$$

The partial derivatives in (4.45)–(4.49) are evaluated at the nominal values of their arguments.

It proves useful to expand $\tilde{\mathbf{x}}_i$ in ascending powers of the perturbed controls as

$$\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}_{il} + \tilde{\mathbf{x}}_{iq} + \mathcal{O}(3), \tag{4.50}$$

where $\tilde{\mathbf{x}}_{il}$ and $\tilde{\mathbf{x}}_{iq}$ denote linear and quadratic functions of the perturbed controls, respectively. That is, $\tilde{x}_{il}^{\alpha} = \sum_{j=1}^{i-1}\mu_{ij}^{\alpha\beta}\tilde{u}_j^{\beta}$ and $\tilde{x}_{iq}^{\alpha} = \sum_{k=1}^{i-1}\pi_{ijk}^{\alpha\beta\gamma}\tilde{u}_j^{\beta}\tilde{u}_k^{\gamma}$. Substituting (4.50) into (4.44) and equating terms of equal order yields

$$\tilde{\mathbf{x}}_{i+1,l} = \mathbf{A}_i\tilde{\mathbf{x}}_{il} + \mathbf{B}_i\tilde{\mathbf{u}}_i, \tag{4.51}$$

$$\tilde{x}_{i+1,q}^{\alpha} = a_i^{\alpha\beta}\tilde{x}_{iq}^{\beta} + \frac{1}{2}(c_i^{\alpha\beta\gamma}\tilde{x}_{il}^{\beta}\tilde{x}_{il}^{\gamma} + 2d_i^{\alpha\beta\gamma}\tilde{x}_{il}^{\beta}\tilde{u}_i^{\gamma} + e_i^{\alpha\beta\gamma}\tilde{u}_i^{\beta}\tilde{u}_i^{\gamma}). \tag{4.52}$$

Expanding (4.40) as a Taylor series about $\mathbf{x}_i = \hat{\mathbf{x}}_i$, $\mathbf{u}_i = \hat{\mathbf{u}}_i$, and $\boldsymbol{\lambda}_i = \hat{\boldsymbol{\lambda}}_i$, making use of (4.51)–(4.52), and neglecting cubic and higher-order terms yields the following quadratic approximation of L:

$$L_q = \sum_{i=1}^{N-1}\Big[\kappa_i + (\tilde{\mathbf{x}}_{il} + \tilde{\mathbf{x}}_{iq})^T\mathbf{y}_i + \tilde{\mathbf{u}}_i^T\mathbf{z}_i + \frac{1}{2}\left(\tilde{\mathbf{x}}_{il}^T\mathbf{Q}_i\tilde{\mathbf{x}}_{il} + 2\tilde{\mathbf{x}}_{il}^T\mathbf{R}_i\tilde{\mathbf{u}}_i + \tilde{\mathbf{u}}_i^T\mathbf{S}_i\tilde{\mathbf{u}}_i\right)$$
$$- (\bar{\mathbf{A}}\tilde{\mathbf{x}}_{il} + \bar{\mathbf{B}}_i\tilde{\mathbf{u}}_i + \bar{\mathbf{c}}_i)^T\tilde{\boldsymbol{\lambda}}_i\Big] + \Gamma_N(\hat{\mathbf{x}}_N) + (\tilde{\mathbf{x}}_{Nl} + \tilde{\mathbf{x}}_{Nq})^T\mathbf{y}_N + \frac{1}{2}\tilde{\mathbf{x}}_{Nl}^T\mathbf{Q}_N\tilde{\mathbf{x}}_{Nl}, \tag{4.53}$$

where

$$\bar{\mathbf{A}}_i = \bar{\mathbf{D}}_i \mathbf{A}_i, \tag{4.54}$$

$$\bar{\mathbf{B}}_i = \bar{\mathbf{H}}_i + \bar{\mathbf{D}}_i \mathbf{B}_i, \tag{4.55}$$

$$\bar{\mathbf{c}}_i = \bar{\mathbf{c}}_i(\hat{\mathbf{x}}_{i+1}, \hat{\mathbf{u}}_i), \tag{4.56}$$

$$\kappa_i = \Gamma_i(\hat{\mathbf{x}}_i, \hat{\mathbf{u}}_i) - \bar{\mathbf{c}}_i^T \hat{\boldsymbol{\lambda}}_i, \tag{4.57}$$

$$\mathbf{y}_i = \frac{\partial \Gamma_i}{\partial x_i^\alpha} - \bar{\mathbf{A}}_i^T \hat{\boldsymbol{\lambda}}_i, \tag{4.58}$$

$$\mathbf{z}_i = \frac{\partial \Gamma_i}{\partial u_i^\alpha} - \bar{\mathbf{B}}_i^T \hat{\boldsymbol{\lambda}}_i, \tag{4.59}$$

$$q_i^{\alpha\beta} = \frac{\partial^2 \Gamma_i}{\partial x_i^\alpha \partial x_i^\beta} - \hat{\lambda}_i^\gamma \bar{g}_{i1}^{\gamma\alpha\beta}, \tag{4.60}$$

$$r_i^{\alpha\beta} = \frac{\partial^2 \Gamma_i}{\partial x_i^\alpha \partial u_i^\beta} - \hat{\lambda}_i^\gamma \bar{g}_{i2}^{\gamma\alpha\beta}, \tag{4.61}$$

$$s_i^{\alpha\beta} = \frac{\partial^2 \Gamma_i}{\partial u_i^\alpha \partial u_i^\beta} - \hat{\lambda}_i^\gamma \bar{g}_{i3}^{\gamma\alpha\beta}, \tag{4.62}$$

$$\bar{d}_i^{\alpha\beta} = \frac{\partial \bar{c}_i^\alpha}{\partial x_{i+1}^\beta}, \tag{4.63}$$

$$\bar{h}_i^{\alpha\beta} = \frac{\partial \bar{c}_i^\alpha}{\partial u_i^\beta}, \tag{4.64}$$

$$\bar{g}_{i1}^{\alpha\beta\gamma} = \bar{d}_i^{\alpha\eta} c_i^{\eta\beta\gamma} + \frac{\partial^2 \bar{c}_i^\alpha}{\partial x_{i+1}^\eta \partial x_{i+1}^\epsilon} a_i^{\eta\beta} a_i^{\epsilon\gamma}, \tag{4.65}$$

$$\bar{g}_{i2}^{\alpha\beta\gamma} = \bar{d}_i^{\alpha\eta} d_i^{\eta\beta\gamma} + \frac{\partial^2 \bar{c}_i^\alpha}{\partial x_{i+1}^\eta \partial x_{i+1}^\epsilon} a_i^{\eta\beta} b_i^{\epsilon\gamma} + \frac{\partial^2 \bar{c}_i^\alpha}{\partial x_{i+1}^\eta \partial u_i^\gamma} a_i^{\eta\beta}, \tag{4.66}$$

$$\bar{g}_{i3}^{\alpha\beta\gamma} = \bar{d}_i^{\alpha\eta} e_i^{\eta\beta\gamma} + \frac{\partial^2 \bar{c}_i^\alpha}{\partial x_{i+1}^\eta \partial x_{i+1}^\epsilon} b_i^{\eta\beta} b_i^{\epsilon\gamma} + 2\frac{\partial^2 \bar{c}_i^\alpha}{\partial x_{i+1}^\eta \partial u_{i+1}^\gamma} b_i^{\eta\beta} + \frac{\partial^2 \bar{c}_i^\alpha}{\partial u_i^\beta \partial u_i^\gamma}. \tag{4.67}$$

The partial derivatives in (4.58)–(4.67) are evaluated at the nominal value of their arguments.

Let

$$L_{qi} = \sum_{l=1}^{N-1} \Big[\kappa_k + (\tilde{\mathbf{x}}_{kl} + \tilde{\mathbf{x}}_{kq})^T \mathbf{y}_k + \tilde{\mathbf{u}}_k^T \mathbf{z}_k + \frac{1}{2} \left(\tilde{\mathbf{x}}_{kl}^T \mathbf{Q}_k \tilde{\mathbf{x}}_{kl} + 2\tilde{\mathbf{x}}_{kl}^T \mathbf{R}_k \tilde{\mathbf{u}}_k + \tilde{\mathbf{u}}_k^T \mathbf{S}_k \tilde{\mathbf{u}}_k \right) - \left(\bar{\mathbf{A}}_k \tilde{\mathbf{x}}_{kl} + \bar{\mathbf{B}}_k \tilde{\mathbf{u}}_k + \bar{\mathbf{c}}_k \right)^T \tilde{\boldsymbol{\lambda}}_k \Big] + \Gamma_N(\hat{\mathbf{x}}_N) + \left(\tilde{\mathbf{x}}_{Nl} + \tilde{\mathbf{x}}_{Nq} \right)^T \mathbf{y}_N + \frac{1}{2} \tilde{\mathbf{x}}_{Nl}^T \mathbf{Q}_N \tilde{\mathbf{x}}_{Nl} \tag{4.68}$$

for $i = 1, \ldots, N$. Expanding the sum on the right-hand side of (4.68) yields

$$L_{qi} = \kappa_i + \left(\tilde{\mathbf{x}}_{il} + \tilde{\mathbf{x}}_{iq} \right)^T \mathbf{y}_i + \tilde{\mathbf{u}}_i^T \mathbf{z}_i + \frac{1}{2} \left(\tilde{\mathbf{x}}_{il}^T \mathbf{Q}_i \tilde{\mathbf{x}}_{il} + 2\tilde{\mathbf{x}}_{il}^T \mathbf{R}_i \tilde{\mathbf{u}}_i + \tilde{\mathbf{u}}_i^T \mathbf{S}_i \tilde{\mathbf{u}}_i \right) - \left(\bar{\mathbf{A}}_i \tilde{\mathbf{x}}_{il} + \bar{\mathbf{B}}_i \tilde{\mathbf{u}}_i + \bar{\mathbf{c}}_i \right)^T \tilde{\boldsymbol{\lambda}}_i + L_{q,i+1}. \tag{4.69}$$

4.3.2 Derivation of Recursive Equations

It is shown inductively that, if L_{qi} is stationary with respect to $\tilde{\mathbf{u}}_i, \ldots, \tilde{\mathbf{u}}_{N-1}$ and $\tilde{\boldsymbol{\lambda}}_{ir}, \ldots, \tilde{\boldsymbol{\lambda}}_{N-1,r}$, then L_{qi} can be expressed as

$$L_{qi} = \zeta_i + \tilde{\mathbf{x}}_{il}^T \boldsymbol{\nu}_i + \frac{1}{2}\tilde{\mathbf{x}}_{il}^T \mathbf{W}_i \tilde{\mathbf{x}}_{il} + \tilde{\mathbf{x}}_{iq}^T \mathbf{p}_i - (\bar{\mathbf{A}}_{in}\tilde{\mathbf{x}}_{il} + \bar{\mathbf{c}}_{in})^T \tilde{\boldsymbol{\lambda}}_{in}, \tag{4.70}$$

where

$$\begin{bmatrix} \tilde{\boldsymbol{\lambda}}_{in} \\ \tilde{\boldsymbol{\lambda}}_{ir} \\ \vdots \\ \tilde{\boldsymbol{\lambda}}_{N-1,r} \end{bmatrix} = \hat{\mathbf{A}}_i \begin{bmatrix} \tilde{\boldsymbol{\lambda}}_i \\ \vdots \\ \tilde{\boldsymbol{\lambda}}_{N-1} \end{bmatrix}, \tag{4.71}$$

$\tilde{\boldsymbol{\lambda}}_{in} \in R^{\bar{n}_{in}}$, and the matrix $\hat{\mathbf{A}}_i \in R^{\bar{n}_{is} \times \bar{n}_{is}}$ is orthogonal if and only if $\bar{n}_{is} > 0$. Nonsingularity of $\hat{\mathbf{A}}_i$ implies that L_{qi} is stationary with respect to $\tilde{\boldsymbol{\lambda}}_{ir}, \ldots, \tilde{\boldsymbol{\lambda}}_{N-1,r}$ and $\tilde{\boldsymbol{\lambda}}_{in}$.

The steps of the proof ultimately lead to (4.98)–(4.102), which form the basis of the proposed algorithm. The derivation begins by substituting (4.51)–(4.52) into (4.70) for $i \leftarrow i + 1$. Then, substituting the result into (4.69) gives

$$\begin{aligned} L_{qi} = {} & \tilde{\mathbf{x}}_{il}^T \mathbf{h}_{4i} + \tilde{\mathbf{u}}_i^T \mathbf{h}_{5i} + \tilde{\mathbf{x}}_{iq}^T \left(\mathbf{y}_i + \mathbf{A}_i^T \mathbf{p}_{i+1}\right) + \frac{1}{2}\left(\tilde{\mathbf{x}}_{il}^T \mathbf{H}_{1i} \tilde{\mathbf{x}}_{il} + 2\tilde{\mathbf{x}}_{il}^T \mathbf{H}_{2i} \tilde{\mathbf{u}}_i + \tilde{\mathbf{u}}_i^T \mathbf{H}_{3i} \tilde{\mathbf{u}}_i\right) \\ & - \left(\bar{\mathbf{A}}_{it}\tilde{\mathbf{x}}_{il} + \bar{\mathbf{B}}_{it}\tilde{\mathbf{u}}_i + \bar{\mathbf{c}}_{it}\right)^T \tilde{\boldsymbol{\lambda}}_{it} + \kappa_i + \zeta_{i+1}, \end{aligned} \tag{4.72}$$

where

$$\mathbf{H}_{1i} = \mathbf{Q}_{pi} + \mathbf{A}_i^T \mathbf{W}_{i+1} \mathbf{A}_i, \tag{4.73}$$

$$\mathbf{H}_{2i} = \mathbf{R}_{pi} + \mathbf{A}_i^T \mathbf{W}_{i+1} \mathbf{B}_i, \tag{4.74}$$

$$\mathbf{H}_{3i} = \mathbf{S}_{pi} + \mathbf{B}_i^T \mathbf{W}_{i+1} \mathbf{B}_i, \tag{4.75}$$

$$\mathbf{h}_{4i} = \mathbf{y}_i + \mathbf{A}_i^T \boldsymbol{\nu}_{i+1}, \tag{4.76}$$

$$\mathbf{h}_{5i} = \mathbf{z}_i + \mathbf{B}_i^T \boldsymbol{\nu}_{i+1}, \tag{4.77}$$

$$q_{pi}^{\alpha\beta} = q_i^{\alpha\beta} + p_{i+1}^{\gamma} c_i^{\gamma\alpha\beta}, \tag{4.78}$$

$$r_{pi}^{\alpha\beta} = r_i^{\alpha\beta} + p_{i+1}^{\gamma} d_i^{\gamma\alpha\beta}, \tag{4.79}$$

$$s_{pi}^{\alpha\beta} = s_i^{\alpha\beta} + p_{i+1}^{\gamma} e_i^{\gamma\alpha\beta}, \tag{4.80}$$

$$\bar{\mathbf{A}}_{it} = \begin{bmatrix} \bar{\mathbf{A}}_i \\ \bar{\mathbf{A}}_{i+1,n}\mathbf{A}_i \end{bmatrix}, \tag{4.81}$$

$$\bar{\mathbf{B}}_{it} = \begin{bmatrix} \bar{\mathbf{B}}_i \\ \bar{\mathbf{A}}_{i+1,n}\mathbf{B}_i \end{bmatrix}, \tag{4.82}$$

$$\bar{\mathbf{c}}_{it} = \begin{bmatrix} \bar{\mathbf{c}}_i \\ \bar{\mathbf{c}}_{i+1,n} \end{bmatrix}, \tag{4.83}$$

$$\tilde{\boldsymbol{\lambda}}_{it} = \begin{bmatrix} \tilde{\boldsymbol{\lambda}}_i \\ \tilde{\boldsymbol{\lambda}}_{i+1,n} \end{bmatrix}. \tag{4.84}$$

Note that L_{qi} given by (4.72) is stationary with respect to $\tilde{\mathbf{u}}_{i+1}, \ldots, \tilde{\mathbf{u}}_{N-1}$ and $\tilde{\boldsymbol{\lambda}}_{i+1,r}, \ldots, \tilde{\boldsymbol{\lambda}}_{N-1,r}$ since (4.70) was used for $L_{q,i+1}$ in (4.69).

Let

$$\tilde{\boldsymbol{\lambda}}_{it} = \mathbf{Q}_{ir}\tilde{\boldsymbol{\lambda}}_{ir} + \mathbf{Q}_{in}\tilde{\boldsymbol{\lambda}}_{in}, \tag{4.85}$$

$$\tilde{\mathbf{u}}_i = \bar{\mathbf{Q}}_{ir}\tilde{\mathbf{u}}_{ir} + \bar{\mathbf{Q}}_{in}\tilde{\mathbf{u}}_{in}, \tag{4.86}$$

where the columns of $\mathbf{Q}_{ir}$ and $\bar{\mathbf{Q}}_{ir}$ span the range spaces of $\bar{\mathbf{B}}_{it}$ and $\bar{\mathbf{B}}_{it}^T$, the columns of $\mathbf{Q}_{in}$ and $\bar{\mathbf{Q}}_{in}$ span the null spaces of $\bar{\mathbf{B}}_{it}^T$ and $\bar{\mathbf{B}}_{it}$, and the matrices $[\mathbf{Q}_{ir} \quad \mathbf{Q}_{in}]$ and $[\bar{\mathbf{Q}}_{ir} \quad \bar{\mathbf{Q}}_{in}]$ are orthogonal. The matrices $\mathbf{Q}_{ir}$, $\mathbf{Q}_{in}$, $\bar{\mathbf{Q}}_{ir}$, and $\bar{\mathbf{Q}}_{in}$ can be obtained from the orthogonal-triangular (QR) factorizations of $\bar{\mathbf{B}}_{it}$ and $\bar{\mathbf{B}}_{it}^T$.

Substituting (4.85) and (4.86) into (4.72); setting the gradients with respect to $\tilde{\mathbf{u}}_{ir}$, $\tilde{\mathbf{u}}_{in}$, and $\tilde{\boldsymbol{\lambda}}_{ir}$ equal to zero; solving for $\tilde{\mathbf{u}}_{ir}$, $\tilde{\mathbf{u}}_{in}$, and $\tilde{\boldsymbol{\lambda}}_{ir}$; and substituting the results into (4.86) yield

$$\tilde{\mathbf{u}}_i = -\left(\bar{\mathbf{E}}_i\tilde{\mathbf{x}}_{il} + \bar{\mathbf{f}}_i\right), \tag{4.87}$$

$$\tilde{\boldsymbol{\lambda}}_{ir} = \left(\bar{\mathbf{G}}_i\tilde{\mathbf{x}}_{il} + \bar{\mathbf{h}}_i\right), \tag{4.88}$$

where

$$\bar{\mathbf{E}}_i = \bar{\mathbf{Q}}_{ir}\bar{\mathbf{R}}_{ir} + \bar{\mathbf{Q}}_{in}\bar{\mathbf{R}}_{in}, \tag{4.89}$$

$$\bar{\mathbf{f}}_i = \bar{\mathbf{Q}}_{ir}\bar{\mathbf{s}}_{ir} + \bar{\mathbf{Q}}_{in}\bar{\mathbf{s}}_{in}, \tag{4.90}$$

$$\bar{\mathbf{G}}_i = \left(\mathbf{Q}_{ir}^T\bar{\mathbf{B}}_{it}\bar{\mathbf{Q}}_{ir}\right)^{-1}\bar{\mathbf{Q}}_{ir}^T\left(\mathbf{H}_{2i}^T - \mathbf{H}_{3i}\bar{\mathbf{E}}_i\right), \tag{4.91}$$

$$\bar{\mathbf{h}}_i = \left(\mathbf{Q}_{ir}^T\bar{\mathbf{B}}_{it}\bar{\mathbf{Q}}_{ir}\right)^{-1}\bar{\mathbf{Q}}_{ir}^T\left(\mathbf{h}_{5i} - \mathbf{H}_{3i}\bar{\mathbf{f}}_i\right), \tag{4.92}$$

$$\bar{\mathbf{R}}_{ir} = \left(\mathbf{Q}_{ir}^T\bar{\mathbf{B}}_{it}\bar{\mathbf{Q}}_{ir}\right)^{-1}\bar{\mathbf{Q}}_{ir}^T\bar{\mathbf{A}}_{it}, \tag{4.93}$$

$$\bar{\mathbf{s}}_{ir} = \left(\mathbf{Q}_{ir}^T\bar{\mathbf{B}}_{it}\bar{\mathbf{Q}}_{ir}\right)^{-1}\mathbf{Q}_{ir}^T\bar{\mathbf{c}}_{it}, \tag{4.94}$$

$$\bar{\mathbf{R}}_{in} = \left(\bar{\mathbf{Q}}_{in}^T\mathbf{H}_{3i}\bar{\mathbf{Q}}_{in}\right)^{-1}\bar{\mathbf{Q}}_{in}^T\left(\mathbf{H}_{2i}^T - \mathbf{H}_{3i}\bar{\mathbf{Q}}_{ir}\bar{\mathbf{R}}_{ir}\right), \tag{4.95}$$

$$\bar{\mathbf{s}}_{in} = \left(\bar{\mathbf{Q}}_{in}^T\mathbf{H}_{3i}\bar{\mathbf{Q}}_{in}\right)^{-1}\bar{\mathbf{Q}}_{in}^T\left(\mathbf{h}_{5i} - \mathbf{H}_{3i}\bar{\mathbf{Q}}_{ir}\bar{\mathbf{s}}_{ir}\right). \tag{4.96}$$

One arrives at (4.70) by substituting (4.85) and (4.87)–(4.88) into the right-hand side of (4.72) with

$$\zeta_i = \kappa_i + \zeta_{i+1} - \bar{\mathbf{f}}_i^T\mathbf{h}_{5i} + \frac{1}{2}\bar{\mathbf{f}}_i^T\mathbf{H}_{3i}\bar{\mathbf{f}}_i, \tag{4.97}$$

$$\boldsymbol{\nu}_i = \mathbf{h}_{4i} - \bar{\mathbf{E}}_i^T\mathbf{h}_{5i}^T - \mathbf{H}_{2i}\bar{\mathbf{f}}_i + \bar{\mathbf{E}}_i^T\mathbf{H}_{3i}\bar{\mathbf{f}}_i, \tag{4.98}$$

$$\mathbf{W}_i = \mathbf{H}_{1i} - \bar{\mathbf{E}}_i^T\mathbf{H}_{2i}^T - \mathbf{H}_{2i}\bar{\mathbf{E}}_i + \bar{\mathbf{E}}_i^T\mathbf{H}_{3i}\bar{\mathbf{E}}_i, \tag{4.99}$$

$$\mathbf{p}_i = \mathbf{y}_i + \mathbf{A}_i^T\mathbf{p}_{i+1}, \tag{4.100}$$

$$\bar{\mathbf{A}}_{in} = \mathbf{Q}_{in}^T\bar{\mathbf{A}}_{it}, \tag{4.101}$$

$$\bar{\mathbf{c}}_{in} = \mathbf{Q}_{in}^T\bar{\mathbf{c}}_{it}. \tag{4.102}$$

To summarize, if $\bar{\mathbf{Q}}_{in}^T\mathbf{H}_{3i}\bar{\mathbf{Q}}_{in}$ is nonsingular or has zero rows and columns, then L_{qi} is stationary with respect to $\tilde{\mathbf{u}}_i, \ldots, \tilde{\mathbf{u}}_{N-1}$ and $\tilde{\boldsymbol{\lambda}}_{ir}, \ldots, \tilde{\boldsymbol{\lambda}}_{N-1,r}$ provided $\tilde{\mathbf{u}}_i$ and $\tilde{\boldsymbol{\lambda}}_{ir}$ are chosen according to (4.87) and (4.88).

The starting values for the recursive equations are obtained by substituting (4.70) into (4.68), setting $i = N$, and equating coefficients of like terms. The result is

$$\zeta_N = \Gamma_N(\hat{\mathbf{x}}_N), \tag{4.103}$$

$$\nu_N = \mathbf{y}_N, \tag{4.104}$$

$$\mathbf{W}_N = \mathbf{Q}_N, \tag{4.105}$$

$$\mathbf{p}_N = \mathbf{y}_N, \tag{4.106}$$

$$\bar{\mathbf{A}}_{Nn} = [\ ], \tag{4.107}$$

$$\bar{\mathbf{c}}_{Nn} = [\ ], \tag{4.108}$$

where [ ] denotes a matrix with zero rows.

It remains to show that the matrix $\hat{\mathbf{A}}_i$ in (4.71) is orthogonal if $\bar{n}_{rs} > 0$. Setting $i \leftarrow i + 1$ in (4.71) and making use of (4.84) and (4.85), one arrives at (4.71), where

$$\hat{\mathbf{A}}_i = \begin{bmatrix} \mathbf{Q}_i^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{A}}_{i+1} \end{bmatrix} \tag{4.109}$$

and

$$\mathbf{Q}_i = [\mathbf{Q}_{in} \quad \mathbf{Q}_{ir}]. \tag{4.110}$$

The terms $\mathbf{I}$ and $\mathbf{0}$ appearing in (4.109) denote identity matrices and matrices of zeros of appropriate dimensions, respectively. If $\bar{n}_{is} > 0$, $\hat{\mathbf{A}}_i$ is orthogonal because both matrices on the right-hand side of (4.109) are orthogonal. The recursive equation (see (4.109)) is initialized by setting $\hat{\mathbf{A}}_N = [\ ]$. This completes the inductive proof of (4.70) and (4.71).

4.3.3 Constraint Equations

With the goal of determining the consistency of the constraints, it proves useful to express $\tilde{\boldsymbol{\lambda}}_{in}$ as

$$\tilde{\boldsymbol{\lambda}}_{in} = \hat{\mathbf{C}}_i \begin{bmatrix} \tilde{\boldsymbol{\lambda}}_i \\ \vdots \\ \tilde{\boldsymbol{\lambda}}_{N_1} \end{bmatrix}. \tag{4.111}$$

Notice from comparison of (4.71) and (4.111) that $\hat{\mathbf{C}}_i$ consists of the first $\bar{n}_{in}$ rows of $\hat{\mathbf{A}}_i$. Thus, it follows from (4.109) that

$$\hat{\mathbf{C}}_i = \mathbf{Q}_{in}^T \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{C}}_{i+1} \end{bmatrix}. \tag{4.112}$$

Recursive (4.112) is initialized by

$$\hat{\mathbf{C}}_N = [\ ]. \tag{4.113}$$

With reference to (4.70), since each element of $\tilde{\mathbf{x}}_{il}$ is equal to zero for $i = 1$, the stationarity of L_q with respect to $\tilde{\boldsymbol{\lambda}}_{1n}$ requires the norm of $\bar{\mathbf{c}}_{1n}$ to equal zero. If this is not the case, then

the constraints are inconsistent. That is, the linear approximation of the constraints given by (4.38) cannot be satisfied. If the norm of $\bar{\mathbf{c}}_{1n}$ is equal to zero, then the constraints are consistent, but $\bar{n}_{1n}$ of them are redundant. Inconsistent or redundant constraints are present whenever $\bar{n}_{1n}$ and the leading dimension of $\bar{\mathbf{c}}_{1n}$ are not equal to zero. Such constraints must be removed in order to apply the algorithm. As a final note, a practical constraint qualification [9] is satisfied if $\bar{n}_{1n} = 0$ since this indicates that the constraints are linearly independent.

The rows of the matrix $\hat{\mathbf{C}}_1$ indicate the linear combination of original constraints that are either inconsistent or redundant. Based on the information contained in $\hat{\mathbf{C}}_1$, one can determine which of the original constraints to remove. Again, $\bar{n}_{1n}$ must equal zero in order for the constraints to be consistent and nonredundant.

4.3.4 Algorithm for P1

The algorithm for calculating $\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_{N-1}$ and $\tilde{\boldsymbol{\lambda}}_1, \ldots, \tilde{\boldsymbol{\lambda}}_{N-1}$ that make L_q, stationary is summarized below.

Step 1: Set $\hat{\mathbf{x}}_1 = \mathbf{x}_1$ and calculate $\hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_N$ from (4.36) using the given values for $\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_{N-1}$.

Step 2: Calculate $\boldsymbol{\nu}_N$, $\mathbf{W}_N$, $\mathbf{p}_N$, $\bar{\mathbf{A}}_{Nn}$, $\bar{\mathbf{c}}_{Nn}$, and $\hat{\mathbf{C}}_N$ from (4.104)–(4.108) and (4.113), respectively.

Step 3: Calculate $\boldsymbol{\nu}_i$, $\mathbf{W}_i$, $\mathbf{p}_i$, $\bar{\mathbf{A}}_{in}$, $\bar{\mathbf{c}}_{in}$, and $\hat{\mathbf{C}}_i$ from (4.98)–(4.102) and (4.112), respectively, for $i = N-1$ to $i = 1$, storing $\bar{\mathbf{E}}_i$, $\bar{\mathbf{f}}_i$, $\bar{\mathbf{G}}_i$, $\bar{\mathbf{h}}_i$, $\mathbf{Q}_{ir}$, and $\mathbf{Q}_{in}$ in the process.

Step 4: A value of $\bar{n}_{1n}$ greater than zero indicates the presence of inconsistent or redundant constraints. Such constraints must be identified and removed based on the information contained in matrix $\hat{\mathbf{C}}_1$. Return to the beginning of Step 3 once these constraints are removed.

Step 5: Calculate $\tilde{\mathbf{u}}_i$, $\tilde{\boldsymbol{\lambda}}_{ir}$, $\tilde{\boldsymbol{\lambda}}_{it}$, $\tilde{\boldsymbol{\lambda}}_i$, $\tilde{\boldsymbol{\lambda}}_{i+1,n}$, and $\tilde{\mathbf{x}}_{i+1}$ for $i = 1$ to $i = N-1$ from (4.87), (4.88), (4.85), (4.84), and (4.51), respectively, starting with $\tilde{\mathbf{x}}_{1l} = 0$ and $\tilde{\boldsymbol{\lambda}}_{1n} = [\ ]$.

This algorithm can be used iteratively to find a local solution, denoted by $(\mathbf{u}_i^*, \boldsymbol{\lambda}_i^*)$, that makes L stationary by setting $\hat{\mathbf{u}}_i \leftarrow \hat{\mathbf{u}}_i + \tilde{\mathbf{u}}_i$ and $\hat{\boldsymbol{\lambda}}_i \leftarrow \hat{\boldsymbol{\lambda}}_i + \tilde{\boldsymbol{\lambda}}_i$ after Step 5 and returning to Step 1. Such an approach converges quadratically to the solution if $(\hat{\mathbf{u}}_i, \hat{\boldsymbol{\lambda}}_i)$ is sufficiently close to $(\mathbf{u}_i^*, \boldsymbol{\lambda}_i^*)$ under the same conditions as those for SQP [9]. Moreover, the number of mathematical operations required for each is only proportional to N. This is in contrast to a conventional SQP approach in which this number is proportional to N^3. Sufficient conditions for the converged solution to be a strong local minimum of the constrained problem are satisfied if the matrices $\bar{\mathbf{Q}}_{in}^T \mathbf{H}_{3i} \bar{\mathbf{Q}}_{in}$ are either positive definite or have zero rows and columns for $i = 1, \ldots, N-1$.

Note that the present algorithm simplifies to the DP algorithm of Dunn and Bertsekas [28] in the absence of constraints. Indeed, for $\bar{n}_i + \bar{n}_{in} = 0$, (4.89)–(4.90) and

(4.98)–(4.99) simplify to

$$\tilde{\mathbf{E}}_i = \mathbf{H}_{3i}^{-1}\mathbf{H}_{2i}^T, \tag{4.114}$$

$$\bar{\mathbf{f}}_i = \mathbf{H}_{3i}^{-1}\mathbf{h}_{5i}, \tag{4.115}$$

$$\boldsymbol{v}_i = \mathbf{h}_{4i} - \mathbf{H}_{2i}\mathbf{H}_{3i}^{-1}\mathbf{h}_{5i}, \tag{4.116}$$

$$\mathbf{W}_i = \mathbf{H}_{1i} - \mathbf{H}_{2i}\mathbf{H}_{3i}^{-1}\mathbf{H}_{2i}^T. \tag{4.117}$$

Equations (4.100), (4.116), and (4.117) are equivalent to (11b), (27h), and (27j) of Dunn and Bertsekas [28].

4.3.5 Free Final Time Problems

The equality-constrained, discrete-time, optimal control problem P2 with free final time is stated as follows. Given the initial state $\mathbf{x}_1$ and an unspecified time step Δt, find the controls $\mathbf{u}_1, \ldots, \mathbf{u}_{N-1}$ that minimize the function

$$\Gamma = \sum_{i=1}^{N-1} \Gamma_i(\mathbf{x}_i, \Delta t, \mathbf{u}_i) + \Gamma_N(\mathbf{x}_N, \Delta t) \tag{4.118}$$

subject to (4.36) and (4.38). As with problem P1, the controls are assumed to be discretized according to (4.35). The times $t_1, \ldots, t_N$, however, are given by

$$t_k = (k-1)\Delta t. \tag{4.119}$$

The algorithm for an iterative step in the solution of P2 is obtained with a simple modification to the algorithm for P1. The first step is to augment the state vectors with the time step and set $n \leftarrow n+1$. That is, $\mathbf{x}_i$ is replaced by $[\mathbf{x}_i^T \ \Delta t]^T$ for $i = 1, \ldots, N$. In a similar manner, $\tilde{\mathbf{x}}_i$ is replaced by $[\mathbf{x}_i^T \ \Delta t]^T$, where $\tilde{\Delta t} = \Delta t - \hat{\Delta t}$. Once this is done, P2 is identical in form to P1 with the exception that the nth element of $\mathbf{x}_1$ is not specified.

Setting $i = 1$ in (4.70) and noting that all the elements of $\tilde{\mathbf{x}}_{1l}$ and $\tilde{\mathbf{x}}_{1q}$ are zero except for $\tilde{\mathbf{x}}_{1l}^n$, which equals Δt, one obtains

$$L_{q1} = \zeta_1 + \tilde{\Delta t} v_1^n + \frac{1}{2}\tilde{\Delta t} w_1^{nn} \tilde{\Delta t} - \left(\bar{\mathbf{e}}_{1n}\tilde{\Delta t} + \bar{\mathbf{c}}_{1n}\right)^T \tilde{\boldsymbol{\lambda}}_{1n}, \tag{4.120}$$

where $\bar{\mathbf{e}}_{1n}$ is the nth column of $\bar{\mathbf{A}}_{1n}$. If $\bar{n}_{1n} > 1$ or $\bar{n}_{1n} = 1$ and $\bar{\mathbf{e}}_{1n} = \mathbf{0}$, then the constraints are either inconsistent or redundant. Such constraints must be removed in order to apply the algorithm.

Setting the gradients of L_{q1} with respect to $\tilde{\Delta t}$ and $\tilde{\boldsymbol{\lambda}}_{1n}$ equal to zero yields

$$\begin{bmatrix} w_1^{nn} & -\bar{\mathbf{e}}_{1n} \\ -\bar{\mathbf{e}}_{1n} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{\Delta t} \\ \tilde{\boldsymbol{\lambda}}_{1n} \end{bmatrix} = \begin{bmatrix} -v_1^n \\ \bar{\mathbf{c}}_{1n} \end{bmatrix}. \tag{4.121}$$

Notice that the matrix on the left-hand side of (4.121) is nonsingular for $\bar{n}_{in} = 0$ only if $w_1^{nn} \neq 0$. For $\bar{n}_{in} = 1$, this matrix is nonsingular as long as $\bar{\mathbf{e}}_{1n} \neq \mathbf{0}$, as assumed above.

4.3.6 Algorithm for P2

The algorithm for calculating $\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_{n-1}, \tilde{\boldsymbol{\lambda}}_1, \ldots, \tilde{\boldsymbol{\lambda}}_{N-1}$, and $\tilde{\Delta}t$ that make L_q stationary is summarized below.

Step 1: Augment the state vectors $\mathbf{x}_i$ for $i = 1, \ldots, N$ with the time step Δt, as described above.

Step 2: Rewrite (4.36)–(4.38) in terms of the augmented state vectors.

Step 3: Carry out Steps 1 and 2 of the algorithm for P1.

Step 4: Calculate $\boldsymbol{\nu}_i, \mathbf{W}_i, \mathbf{p}_i, \bar{\mathbf{A}}_{in}, \bar{\mathbf{c}}_{in}$, and $\hat{\mathbf{C}}_i$ from (4.98)–(4.102) and (4.112), respectively, for $i = N-1$ to $i = 1$, storing $\bar{\mathbf{E}}_i, \bar{\mathbf{f}}_i, \bar{\mathbf{G}}_i, \bar{\mathbf{h}}_i, \mathbf{Q}_{ir}$, and $\mathbf{Q}_{in}$ in the process.

Step 5: If ($\bar{n}_{1n} = 0$ or $\bar{n}_{1n} = 1$) and $\bar{\mathbf{e}}_{1n} \neq \mathbf{0}$, proceed to Step 6. If ($\bar{n}_{1n} > 0$ or $\bar{n}_{1m} = 1$) and $\bar{\mathbf{e}}_{1n} = \mathbf{0}$, then the constraints are either inconsistent or redundant. Identify and remove these constraints based on the information contained in $\hat{\mathbf{C}}_1$ and return to the beginning of Step 4.

Step 6: Calculate $\tilde{\Delta}t$ and $\tilde{\boldsymbol{\lambda}}_{1n}$ by solving (4.121).

Step 7: Calculate $\tilde{\mathbf{u}}_i, \tilde{\boldsymbol{\lambda}}_{ir}, \tilde{\boldsymbol{\lambda}}_{it}, \tilde{\boldsymbol{\lambda}}_i, \tilde{\boldsymbol{\lambda}}_{i+1,n}$, and $\tilde{\mathbf{x}}_{i+1,l}$ for $i = 1$ to $i = N-1$ from (4.87), (4.88), (4.85), (4.84), and (4.51), respectively, starting with $\tilde{x}^{\alpha}_{il} = 0$ for $j \neq n$ and $\tilde{x}^{n}_{1l} = \tilde{\Delta}t$.

This algorithm can be used iteratively to find a local solution denoted by $(\mathbf{u}_i^*, \boldsymbol{\lambda}_i^*, \Delta t^*)$ that makes L stationary by setting $\hat{\mathbf{u}}_i \leftarrow \hat{\mathbf{u}}_i + \tilde{\mathbf{u}}_i$, $\hat{\boldsymbol{\lambda}}_i \leftarrow \hat{\boldsymbol{\lambda}}_i + \tilde{\boldsymbol{\lambda}}_i$, and $\hat{\Delta}t \leftarrow \hat{\Delta}t + \tilde{\Delta}t$ after Step 7 and returning to Step 1. Such an approach converges quadratically to the solution if $(\hat{\mathbf{u}}_i, \hat{\boldsymbol{\lambda}}_i, \hat{\Delta}t)$ is sufficiently close to $(\mathbf{u}_i^*, \boldsymbol{\lambda}_i^*, \Delta t^*)$ under the same conditions as those for SQP. As with the algorithm for P1, the number of mathematical operations required for each iteration is proportional to N. Sufficient conditions for the converged solution to be a strong local minimum of the constrained problem are satisfied if w_1^{nn} is positive and the matrices $\bar{\mathbf{Q}}_{in}^T \mathbf{H}_{3i} \bar{\mathbf{Q}}_{in}$ are either positive definite or have zero rows and columns for $i = 1, \ldots, N-1$.

4.3.7 Practical Issues

Notice from (4.85) and (4.86) that the algorithm for P1 requires the determination of matrix rank. Care must be taken when determining rank on a digital computer because of roundoff errors. Practical methods for calculating the rank of a matrix numerically are discussed by Golub and Van Loan [52].

The quadratic convergence of the algorithms presented is an appealing feature, but this convergence is achieved only if the nominal values for the controls and Lagrange multipliers are sufficiently close to the solution. Poor initial estimates can lead to iterates that sometimes do not converge. Therefore, a strategy for obtaining initial estimates of the nominal controls and Lagrange multipliers may be required.

Previous work has demonstrated the usefulness of penalty functions to enforce constraints for problems of practical interest. One approach is to initially set the nominal values

of all the controls to zero and select a value of ρ, say ρ_1, such that an algorithm for unconstrained minimization of (4.39) converges. By setting the nominal values of the controls equal to those for the converged solution corresponding to ρ_1, a new unconstrained problem with penalty parameter $\rho_2 > \rho_1$ is solved. This process is repeated until the constraints are satisfied to within a specified tolerance.

At a certain point in the above process, one can switch from using an unconstrained algorithm with penalty functions to one of the algorithms presented here. This simply involves setting the nominal values for the controls equal to those of the last converged solution. Estimates for the Lagrange multipliers are given by Gill, Murray, and Wright [9] as

$$\hat{\boldsymbol{\lambda}}_i = -\hat{\rho}\hat{\mathbf{c}}_i \tag{4.122}$$

for $i = 1, \ldots, N-1$, where $\hat{\rho}$ and $\hat{\mathbf{c}}_i$ are the values of ρ and $\bar{\mathbf{c}}_i$ for the last converged solution.

A variety of different homotopy schemes [53, 54] can also be used to obtain solutions to P1 and P2. For example, the algorithm for P1 can be used to solve a series of problems in which (4.38) is replaced by

$$\alpha\bar{\mathbf{c}}_i(\mathbf{x}_{i+1}, \mathbf{u}_i) + (1-\alpha)\hat{\mathbf{c}}_i(\mathbf{x}_{i+1}, \mathbf{u}_i) = \mathbf{0}. \tag{4.123}$$

The idea is to set the nominal values of the controls and Lagrange multipliers equal to zero and construct the functions $\hat{\mathbf{c}}_i$ such that a convergent solution is obtained for α equal to zero. The scalar α is then increased and the problem is resolved using the controls and Lagrange multipliers corresponding to the converged solution for the previous value of α. This process is repeated until α reaches the value of unity. It is clear from comparison of (4.38) and (4.123) that the solution corresponding to $\alpha = 1$ is identical to that of the original problem. Similar homotopy schemes involving modifications of (4.36) and (4.37) can also be used.

A third approach, which is often the simplest, is to use common sense in the selection of the nominal controls to the problem. With such an approach, the physical understanding of the problem is used to guide the selection process. This approach is used for the final example problem (Example 4.5).

4.3.8 Example Problems

Five example problems are used to demonstrate the algorithms for P1 and P2. The first two (Examples 4.1 and 4.2) are simple test problems that show the equivalence of results obtained from the present algorithms with a conventional SQP step. The third example (Example 4.3) deals with optimal open-loop control for reorientation of a rigid satellite. The fourth example (Example 4.4) is a pursuit problem involving a nonholonomic constraint. The algorithm for P2 is applied to a brachistochrone problem with viscous friction in the final example (Example 4.5).

Example 4.1. *The first example problem has* $n = 3$, $m = 1$, $N = 5$, $\bar{n}_i = 0$ *for* $i = 1, \ldots, N-2$, *and* $\bar{n}_{N-1} = 2$.

Furthermore, $\hat{u}_i = 0.02$ and $\hat{\lambda}_i = 0.1$ for $i = 1, \ldots, N-1$ and $\mathbf{x}_1 = [0.2,\ 0.1,\ 0.3]^T$. Equations (4.36)–(4.38) are given specifically by

$$\mathbf{x}_{i+1} = \frac{1}{10}\left\{\begin{bmatrix} 10 & -3 & 7 \\ 6 & 9 & -4 \\ 3 & 6 & -5 \end{bmatrix} \sin \mathbf{x}_i + \begin{bmatrix} 6 \\ 2 \\ 9 \end{bmatrix} \sin u_i + \sin u_i \sin \mathbf{x}_i\right\}, \tag{4.124}$$

$$\Gamma = \frac{1}{2}\left[\sum_{i=1}^{N-1}\left[\left(\mathbf{h}\mathbf{x}_i - \frac{i}{10}\right)^2 + u_i^2\right] + \left(\mathbf{h}\mathbf{x}_N - \frac{N}{10}\right)^2\right], \tag{4.125}$$

$$\bar{\mathbf{c}}_{N-1}(\mathbf{x}_N, u_{N-1}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \sin \mathbf{x}_N + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u_{N-1} + \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} u_{N-1} \sin \mathbf{x}_N, \tag{4.126}$$

where $\mathbf{h} = [0.2\ \ 0.1\ \ 0.3]$.

Values of $\tilde{\mathbf{u}}_i$ and $\tilde{\boldsymbol{\lambda}}_{N-1}$ obtained with a single iteration of the algorithm for P1 are compared with a conventional SQP step in Table 4.2. Notice that the two sets of results are identical to machine precision. Also included in the table are the results obtained by applying the algorithm in an iterative manner. The quadratic convergence of the iterates is clear.

Table 4.2. *Numerical results for Example* 4.1.

	Method		Iteration			
Variable	Algorithm for P1	SQP step	2	3	4	5
$\tilde{u}_1$	−0.20669135095700	−0.20669135095700	7.14e−3	−3.61e−4	−2.46e−7	6.63e−14
$\tilde{u}_2$	−0.11554620940361	−0.11554620940361	8.42e−3	7.45e−4	−3.13e−7	−1.52e−14
$\tilde{u}_3$	0.18989202918926	0.18989202918926	−2.81e−2	3.76e−4	−2.32e−7	−4.81e−14
$\tilde{u}_4$	−0.17429325658688	−0.17429325658688	−6.23e−3	−2.65e−4	5.08e−7	1.82e−13
$\tilde{\lambda}_4^1$	1.55756737022431	1.55756737022431	−1.07e−0	2.51e−2	−1.75e−6	4.20e−13
$\tilde{\lambda}_4^2$	−3.51442816577612	−3.51442816577613	7.03e−1	−1.24e−2	3.67e−6	1.62e−12

The SQP step was obtained as follows. Let

$$\hat{e}^{\alpha\beta} = \frac{\partial^2 L}{\partial \bar{u}^\alpha \bar{u}^\beta}, \tag{4.127}$$

$$\hat{f}^\alpha = \frac{\partial L}{\partial \bar{u}^\alpha}, \tag{4.128}$$

where L is the Lagrangian defined by (4.40) and

$$\bar{\mathbf{u}} = [\mathbf{u}_1^T \quad \boldsymbol{\lambda}_1^T \cdots \mathbf{u}_{N-1}^T \quad \boldsymbol{\lambda}_{N-1}^T]^T. \tag{4.129}$$

The SQP step is given by

$$\Delta\bar{\mathbf{u}} = -\hat{\mathbf{E}}^{-1}\hat{\mathbf{f}}. \tag{4.130}$$

Equation (4.130) was formulated and solved with the aid of the symbolic math software MAPLE® [55] using 20-digit accuracy.

Example 4.2. *This example is used to demonstrate the algorithm for P2.*

Equations (4.36) and (4.118) are given specifically by

$$\mathbf{x}_{i+1} = \frac{1}{10}\left\{\begin{bmatrix} 10 & -3 & 7 \\ 6 & 9 & -4 \\ 3 & 6 & -5 \end{bmatrix} \sin \mathbf{x}_i + \begin{bmatrix} 6 \\ 2 \\ 9 \end{bmatrix} \sin u_i + \sin u_i \sin \mathbf{x}_i \right.$$

$$\left. + \Delta t \sin \mathbf{x}_i + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \frac{\Delta t \sin u_i}{20} \right\}, \tag{4.131}$$

$$\Gamma = \frac{1}{2}\left[\sum_{i=1}^{N-1}\left[\left(\mathbf{hx}_i - \frac{i}{10}\right)^2 + u_i^2\right] + \left(\mathbf{hx}_N - \frac{N}{10} + (\Delta t)^2\right)^2\right]. \tag{4.132}$$

The same constraints, nominal values for controls and Lagrange multipliers, and initial conditions as for the first example are used in this example. The nominal value of the time step, $\hat{\Delta}t$, is set equal to 0.1.

Values of $\tilde{\mathbf{u}}_i$, $\tilde{\boldsymbol{\lambda}}_i$, and $\tilde{\Delta}t$ obtained with a single iteration of the algorithm for P2 are compared with a conventional SQP step in Table 4.3. As with Example 4.1, the two sets of results are identical to machine precision. Also included in Table 4.3 are the results obtained by applying the algorithm iteratively.

Table 4.3. *Numerical results for Example* 4.2.

	Method		Iteration			
Variable	Algorithm for P1	SQP step	2	3	4	5
$\tilde{u}_1$	−0.20465960544149	−0.20465960544149	1.15e−2	−7.78e−4	−1.01e−6	3.40e−13
$\tilde{u}_2$	−0.11133892367840	−0.11133892367840	1.98e−3	1.74e−3	−1.23e−6	−1.16e−12
$\tilde{u}_3$	0.19325841005861	0.19325841005861	−3.54e−2	3.76e−4	6.77e−8	2.35e−12
$\tilde{u}_4$	−0.17462005066592	−0.17462005066592	−2.33e−3	−4.30e−4	1.19e−6	2.07e−12
$\tilde{\lambda}_4^1$	1.57770620682180	1.57770620682180	−1.13e−0	3.07e−2	−5.52e−6	1.89e−11
$\tilde{\lambda}_4^2$	−3.51579406694943	−3.51579406694943	7.47e−1	−1.53e−2	6.07e−6	−2.04e−11
$\tilde{\Delta}t$	−0.11589019511086	−0.11589019511086	−8.62e−2	8.38e−4	8.19e−6	−1.83e−12

Example 4.3. *The third example deals with a satellite dynamics problem considered by Junkins and Turner* [56].

The system involves a rigid satellite initially undergoing a tumbling motion. The problem is to determine the torques that bring the satellite to rest in a specified time t_f while minimizing the integral

$$J = \frac{1}{2}\int_0^{t_f} \left[T_1^2(t) + T_2^2(t) + T_3^2(t)\right] dt, \tag{4.133}$$

where T_1, T_2, and T_3 are the torques acting about the respective body-fixed principal axes. Moreover, the satellite is required to have a specified orientation at time t_f.

The kinematical equations associated with the orientation of the satellite are given by

$$\dot{\epsilon}_1 = \frac{1}{2}(\omega_1\epsilon_4 - \omega_2\epsilon_3 + \omega_3\epsilon_2), \tag{4.134}$$

$$\dot{\epsilon}_2 = \frac{1}{2}(\omega_1\epsilon_3 + \omega_2\epsilon_4 - \omega_3\epsilon_1), \tag{4.135}$$

$$\dot{\epsilon}_3 = \frac{1}{2}(-\omega_1\epsilon_2 + \omega_2\epsilon_1 + \omega_3\epsilon_4), \tag{4.136}$$

$$\dot{\epsilon}_4 = -\frac{1}{2}(\omega_1\epsilon_1 + \omega_2\epsilon_2 + \omega_3\epsilon_3), \tag{4.137}$$

where $\epsilon_1, \ldots, \epsilon_4$ are Euler parameters [38] and ω_1, ω_2, and ω_3 are the angular rates of the satellite about its principal axes. The dynamical equations associated with the motion of the satellite are given by

$$\dot{\omega}_1 = \frac{[(I_2 - I_3)\omega_2\omega_3 + T_1]}{I_1}, \tag{4.138}$$

$$\dot{\omega}_2 = \frac{[(I_3 - I_1)\omega_3\omega_1 + T_2]}{I_2}, \tag{4.139}$$

$$\dot{\omega}_3 = \frac{[(I_1 - I_2)\omega_1\omega_2 + T_3]}{I_3}, \tag{4.140}$$

where I_1, I_2, and I_3 are the principal moments of inertia.

In Junkins and Turner [56], Pontryagin's variational principle was used to derive the associated two-point boundary value problem (TPBVP) for this problem. The TPBVP consisted of 14 coupled differential equations that were solved numerically using a fixed-step, fourth-order RK method. A shooting method and a relaxation process were used to obtain a solution that satisfied all 14 boundary conditions. Special consideration was required in constructing the algorithm to solve the TPBVP because the Euler parameters satisfy the constraint equation

$$\epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 + \epsilon_4^2 = 0. \tag{4.141}$$

The algorithm for P1 is used to obtain an approximate solution to the problem by defining

$$\mathbf{x} = [\epsilon_1 \quad \epsilon_2 \quad \epsilon_3 \quad \epsilon_4 \quad \omega_1 \quad \omega_2 \quad \omega_3]^T, \tag{4.142}$$

$$\mathbf{u} = [T_1 \quad T_2 \quad T_3]^T, \tag{4.143}$$

$$t_i = \frac{(i-1)t_f}{N-1}, \tag{4.144}$$

$$\Gamma = \frac{1}{2}\sum_{i=1}^{N-1} \mathbf{u}_i^T \mathbf{u}_i. \tag{4.145}$$

The constraints to satisfy are $x_N^j = \epsilon_j(t_f)$ and $x_N^{j+4} = \omega_j(t_f)$ for $j = 1, 2, 3$. The constraint $x_N^4 = \epsilon_4(t_f)$ is automatically satisfied because of (4.141). The initial conditions,

Table 4.4. *Problem definition and results for Example* 4.3.

Variable	$t = 0$	$t = t_f$	Unit
ϵ_1	0	0.70106	–
ϵ_2	0	0.09230	–
ϵ_3	0	0.56098	–
ϵ_4	0	0.43047	–
ω_1	0.010	0	rad/sec
ω_2	0.005	0	rad/sec
ω_3	0.001	0	rad/sec
I_1	I_2	I_3	Unit
1×10^6	833333	916667	kg-m^2
Iteration	1	2	3
$\lvert\tilde{\mathbf{u}}\rvert$	5.06e3	1.09e3	2.03e2
Iteration	4	5	6
$\lvert\tilde{\mathbf{u}}\rvert$	5.26e0	1.26e−2	2.38e−8

final-time constraints, and mass properties for the problem are given in Table 4.4. The integer N is set equal to 101 and $t_f = 100$. Nominal values for the controls and Lagrange multipliers are all set equal to zero.

Time histories of the Euler parameters, angular rates, and torques are shown in Figs. 4.4 and 4.5, respectively. It is clear from these figures that all seven constraints are satisfied and the solution is smoothly varying. Values of the norm $|\tilde{\mathbf{u}}|$ defined as

$$|\tilde{\mathbf{u}}| = \sqrt{\sum_{i=1}^{N-1} \tilde{\mathbf{u}}_i^T \tilde{\mathbf{u}}_i} \tag{4.146}$$

are shown in Table 4.4 for six iterations of the algorithm for P1. Notice that the norms quadratically converge to zero.

Comparison of Figs. 4.4 and 4.5 with those of the cited reference showed significant differences. Values of the performance index J given by (4.133) were also significantly different ($J \approx 1.86\text{e}7$ for the present approach, $J \approx 3.60\text{e}8$ for Junkins and Turner [56]). The reason for these differences is that the final values of the Euler parameters used by Junkins and Turner [56] are opposite in sign to those shown in Table 4.4. Interestingly, both sets of Euler parameters physically correspond to the same orientation. The difference is that the present approach requires less effort, as measured by J, to accomplish the maneuver. Both solutions provide local minima to J, but the present approach appears to provide a close approximation to the global minimum.

A simple analogy to help explain the difference is that of rotating a glass of water about its vertical axis so that the final orientation is 90° counterclockwise from the original orientation. One approach is to rotate the glass 90° counterclockwise. Another approach is to rotate the glass 270° clockwise. The same final orientation is achieved in both cases, but the effort required by the former approach is less than the latter.

The right-hand sides of (4.36) and (4.45)–(4.49) had to be evaluated in order to obtain the results for this example. In Examples 4.4 and 4.5, closed-form expressions for these terms can be derived. For this example, these terms were determined numerically because

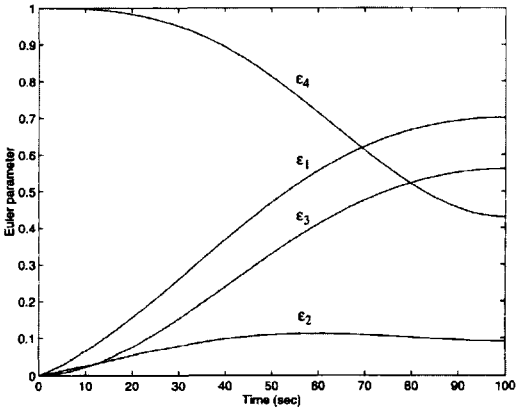


Figure 4.4. *Euler parameter time histories for Example* 4.3. *(From Dohrmann and Robinett* [29]. *Reprinted by permission of Kluwer Academic/Plenum Publishers.)*

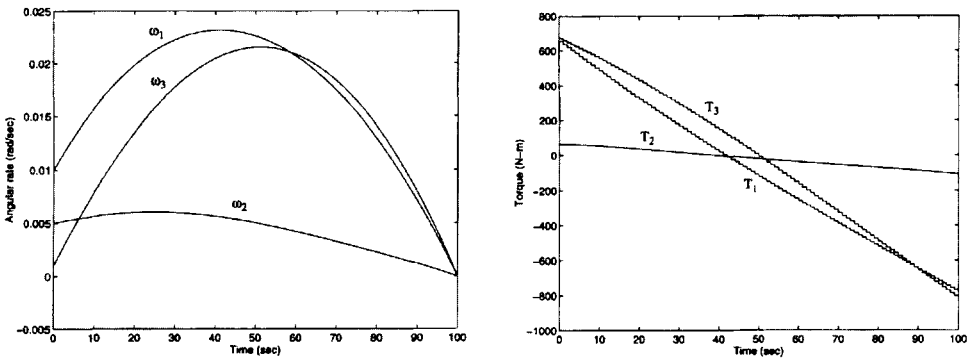


Figure 4.5. *Angular rate (left plot) and torque (right plot) time histories for Example* 4.3. *(From Dohrmann and Robinett* [29]. *Reprinted by permission of Kluwer Academic/Plenum Publishers.)*

analytical solutions to (4.134)–(4.140) only exist for special cases of initial conditions and controls. The numerical approach used for determining these terms, based on a fourth-order RK method, is given in Section A.2 of Appendix A.

As a final note, the direct optimization approach of the present algorithm offers a viable alternative to formulating and solving the TPBVPs associated with certain optimal control problems. Solving TPBVPs is by no means a simple task and can often be fraught with numerical difficulties. The state and costate differential equations of the TPBVP may be stiff even though the governing equations of the physical system are nonstiff. Direct optimization procedures avoid this situation and its attendant difficulties. As demonstrated in this example, direct optimization techniques may also require less effort to obtain estimates

of the solution that lead to convergent iterates. The main point to be made is that the present algorithms (direct optimization approaches) clearly warrant consideration when selecting a method to solve an optimal control problem.

Example 4.4. *The fourth example is a pursuit problem involving a nonholonomic constraint.*

A target, initially at the origin of a planar Cartesian coordinate system, moves along the x coordinate axis at a constant speed, V. A pursuing object is required to hit the target at a specified final time, t_f. During the time interval $0 < t \leq t_f/2$, the velocity vector of the pursuing object must point toward the target for imaging purposes. The problem is to determine the time histories of the forces acting on the pursuing object that satisfy the constraints of the problem while minimizing the integral

$$J = \frac{1}{2}\int_0^{t_f} \left[f_x^2(t) + f_y^2(t)\right] dt, \tag{4.147}$$

where f_x and f_y are the forces acting in the x and y coordinate directions, respectively. The equations of motion for the pursuing object are given by

$$m_o \ddot{x}_c = f_x, \tag{4.148}$$

$$m_o \ddot{y}_c = f_y, \tag{4.149}$$

where m_o is the mass and x_c and y_c are the coordinates of the object. The pointing constraint during $0 < t \leq t_f/2$ is expressed as

$$\dot{x}_c y_c - \dot{y}_c(x_c - Vt) = 0. \tag{4.150}$$

This constraint is nonholonomic because an integrating function cannot be found that turns it into an exact differential [57].

The algorithm for P1 is used to obtain an approximate solution to the problem by defining

$$\mathbf{x} = [x_c \quad y_c \quad \dot{x}_c \quad \dot{y}_c]^T, \tag{4.151}$$

$$\mathbf{u} = [f_x \quad f_y]^T, \tag{4.152}$$

$$t_i = \frac{(i-1)t_f}{N-1}, \tag{4.153}$$

$$\Gamma = \frac{1}{2}\sum_{i=1}^{N-1} \mathbf{u}_i^T \mathbf{u}_i. \tag{4.154}$$

Integration of (4.148) and (4.149) between the limits t_i and t_{i+1} yields

$$\mathbf{x}_{i+1} = \begin{bmatrix} 1 & 0 & h & 0 \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x}_i + \frac{1}{m_o} \begin{bmatrix} \frac{h^2}{2} & 0 \\ 0 & \frac{h^2}{2} \\ h & 0 \\ 0 & h \end{bmatrix} \mathbf{u}_i, \tag{4.155}$$

where $h = t_{i+1} - t_i$.

The constraints to satisfy are $x_i^3 x_i^2 - x_i^4(x_i^1 - Vt_i) = 0$ for $i = 1, \ldots, (N-1)/2$, $x_N^1 = Vt_f$, and $x_N^2 = 0$. The parameters defining the problem are $m_o = 1$, $t_f = 1$, $V = 1$, and $N = 101$. Nominal values for the controls and Lagrange multipliers are all set to zero.

The trajectory of the pursuing object and the applied forces are shown in Fig. 4.6. The circle in Fig. 4.6 (left plot) corresponds to $t = t_f/2$, at which time the target has coordinates of $(1/2, 0)$. The tangent to the trajectory at this point is drawn as a dashed line. Notice that the tangent intersects the target as required by the pointing constraint. Removal of the pointing constraint for $t > t_f/2$ leads to an abrupt change in the applied forces, as shown in Fig. 4.6 (right plot). For $t > t_f/2$, an analytical solution to the problem can be expressed in terms of the state vector at $t = t_f/2$. This solution has the applied forces varying linearly from nonzero values at $t = t_f/2$ to zero at $t = t_f$. These forces are closely approximated by those shown in Fig. 4.6 (right plot). Five iterations were used to obtain the results shown.

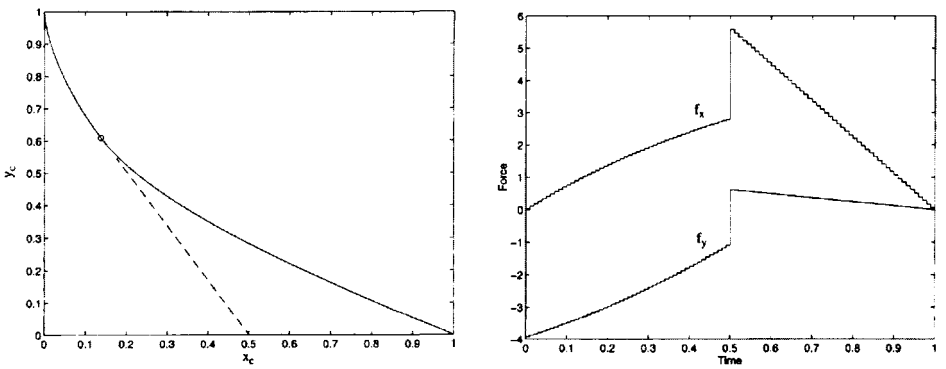


Figure 4.6. *Trajectory of pursuing body (left plot) and the force time histories (right plot) for Example* 4.4. *(From C.R. Dohrmann and R.D. Robinett* [30].*)*

Example 4.5. *The final example deals with a set of brachistochrone problems involving viscous friction.*

A particle, initially at rest at the origin of a planar Cartesian coordinate system, moves along a curve that terminates at (x_f, y_f). The particle is subjected to a gravitational force acting in the positive y coordinate direction and a viscous frictional force opposing the motion. The problem is to determine the curve that minimizes the traversal time from the origin to (x_f, y_f).

The equations governing the motion of the particle are

$$\dot{x}_c = V \cos\theta, \tag{4.156}$$

$$\dot{y}_c = V \sin\theta, \tag{4.157}$$

$$m_p \dot{V} = m_p g \sin\theta - cV, \tag{4.158}$$

where x_c and y_c are the coordinates of the particle, m_p is its mass, V is its velocity tangent to the path, g is the acceleration of gravity, c is the viscous friction coefficient, and θ is

the angle from the x coordinate axis to the tangent of the path. With a suitable change of variables, the governing equations can be rewritten as

$$\dot{x}_c = V\cos\theta, \tag{4.159}$$

$$\dot{y}_c = V\sin\theta, \tag{4.160}$$

$$\dot{V} = \sin\theta - cV. \tag{4.161}$$

The algorithm for P2 is used to obtain an approximate solution to the problem by defining

$$\mathbf{x} = [x_c \quad y_c \quad V \quad \Delta t]^T, \tag{4.162}$$

$$u = \theta, \tag{4.163}$$

$$\Gamma = x_N^4. \tag{4.164}$$

Integrating (4.159)–(4.161) between the limits t_i and $t_i + \Delta t$ yields

$$x_{i+1}^1 = x_i^1 + \cos u_i \frac{(x_i^4 \sin u_i + ab)}{c}, \tag{4.165}$$

$$x_{i+1}^2 = x_i^2 + \sin u_i \frac{(x_i^4 \sin u_i + ab)}{c}, \tag{4.166}$$

$$x_{i+1}^3 = \frac{\sin u_i}{c} + a\ \exp(-cx_i^4), \tag{4.167}$$

$$x_{i+1}^4 = x_i^4, \tag{4.168}$$

where

$$a = x_i^3 - \frac{\sin u_i}{c}, \tag{4.169}$$

$$b = 1 - \exp(-cx_i^4). \tag{4.170}$$

The terminal constraints to satisfy are $x_N^1 = x_f$ and $x_N^2 = y_f$. The parameters defining the problem are $x_f = y_f = 1$ and $N = 101$. A range of different values for the viscous friction parameter c was considered. Nominal values for the controls were set equal to the angle from the x coordinate axis to the line connecting the origin and (x_f, y_f). Nominal values of zero were assigned to the two Lagrange multipliers and $\hat{\Delta}t$ was set equal to the time for the straight line path.

The minimum-time paths for three different values of c are shown in Fig. 4.7 (left plot). The path corresponding to $c = 0.01$ is indistinguishable from the classical solution to the brachistochrone problem without friction. The parametric solution to the friction-free problem is given by the cycloid [58] as

$$x_c(\tau) = \frac{k^2}{2}(2\tau - \sin 2\tau), \tag{4.171}$$

$$y_c(\tau) = \frac{k^2}{2}(1 - \cos 2\tau), \tag{4.172}$$

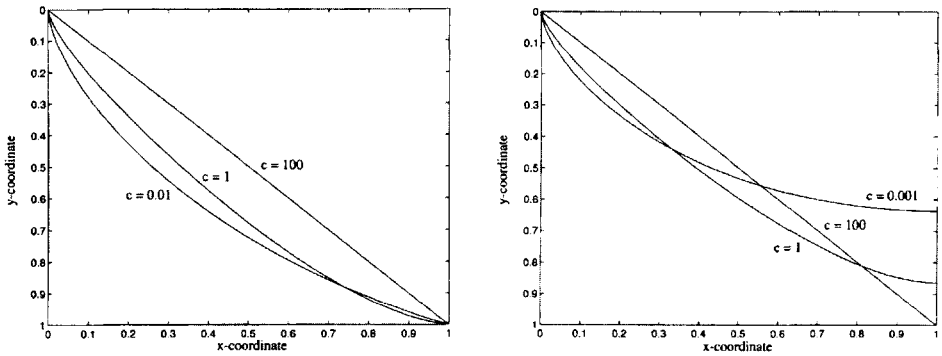


Figure 4.7. *Trajectories for the brachistochrone problem (see Example 4.5) with specified endpoint (left plot) and with specified final x coordinate (right plot). (From Dohrmann and Robinett [29]. Reprinted by permission of Kluwer Academic/Plenum Publishers.)*

where $k^2 \approx 1.146$ for the specified values of x_f and y_f. For the path corresponding to $c = 100$, the particle quickly reaches a terminal velocity for which the projection of the gravitational force along the path equals the frictional force. The velocity of the particle is nearly constant for $t > 0$ and its path is closely approximated by a straight line. Six iterations were used to obtain the results shown.

Results for a brachistochrone problem in which the constraint on the final y coordinate of the particle is removed are shown in Fig. 4.7 (right plot). The path corresponding to $c = 0.001$ is indistinguishable from the classical friction-free solution in which $k^2 = 2x_f/\pi$ in (4.171) and (4.172). The results shown for $c = 100$ suggest that the minimum-time path to reach a specified x coordinate for large values of c is a straight line making a 45° angle with the horizontal. The same observation follows from a simple analysis of the system.

In each of the previous three examples (see Examples 4.3–4.5), the controls were discretized by 100 time increments of equal length. Higher-accuracy solutions are obtained by simply increasing the number of discrete times, N, by a suitable amount. A key advantage of the present algorithms is that the number of mathematical operations required for each iteration grows only linearly with N.

4.4 A Dynamic Programming Method for Constrained Optimal Control

The last part of this chapter discusses the modifications to the DDP algorithm to directly enforce inequality constraints. This section is based on excerpts from Dohrmann and Robinett [31].[4] In particular, this method is based on an efficient algorithm for solving the subproblems of SQP. By using an interior point method to accommodate inequality constraints, a modification of an existing algorithm for equality-constrained problems can be used iteratively to solve the subproblems. Two test problems and two application prob-

[4]Revised and reprinted by permission of Kluwer Academic/Plenum Publishers.

lems are presented. The application examples include a rest-to-rest maneuver of a flexible structure and a constrained brachistochrone problem.

Optimization problems with inequality constraints generally require more sophisticated methods of solution than their unconstrained or equality-constrained counterparts. A challenging aspect of such problems is determining which of the inequality constraints are binding at the solution. Several methods can be used to determine the binding constraints in optimal control problems, including, among others, active set, projected Newton, and interior point methods. With any of these methods, a nonlinear program can be approximated by a series of quadratic subproblems with linear constraints. Under suitable conditions, the solutions to these subproblems converge to the solution of the original nonlinear program. Such is the essence of SQP. Active set methods can suffer from the limitation that the number of constraints added or removed from the working set is small. Consequently, the number of iterations required to solve an optimal control problem with many binding constraints may be prohibitive. Projected Newton methods can be used effectively to solve problems with control inequalities, but their application to state inequalities is not straightforward. Interior point methods have recently gained popularity for solving linear programs and have been applied to discrete-time optimal control as well. The method presented in this section provides an efficient means of solving the quadratic subproblems of SQP using a combined DP and primal-dual interior point method.

4.4.1 Problem Formulation

Consider the initial value problem

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \qquad \mathbf{x}(t_1) = \mathbf{x}_1, \tag{4.173}$$

where $\mathbf{x} \in R^n$ is the state and $\mathbf{u} \in r^m$ is the control. The control is assumed to be discretized temporally as

$$\mathbf{u}(t) = \mathbf{u}_i, \qquad t_i \leq t < t_{i+1}, \tag{4.174}$$

for $i = 1, \ldots, N-1$, where $t_1, \ldots, t_N$ are given. Given the solution to the initial value problem, adjacent states in time are related as

$$\mathbf{x}_{i+1} = \mathbf{g}_i(\mathbf{x}_i, \mathbf{u}_i) \tag{4.175}$$

for $i = 1, \ldots, N-1$, where $\mathbf{x}_i = \mathbf{x}(t_i)$.

The discrete-time optimal control problem P1 is stated as follows. Find the controls $\mathbf{u}_1, \ldots, \mathbf{u}_{N-1}$ and the initial state $\mathbf{x}_1$ that minimize the function

$$\Gamma = \sum_{i=1}^{N-1} \Gamma_i(\mathbf{x}_i, \mathbf{u}_i) + \Gamma_N(\mathbf{x}_N) \tag{4.176}$$

subject to (4.175) and the constraints

$$\mathbf{c}_{iE}(\mathbf{x}_i, \mathbf{u}_i) \quad \text{and} \quad \mathbf{c}_{NE}(\mathbf{x}_N) = \mathbf{0} \tag{4.177}$$

and

$$\mathbf{c}_{iI}(\mathbf{x}_i, \mathbf{u}_i) \leq \mathbf{0} \quad \text{and} \quad \mathbf{c}_{NI}(\mathbf{x}_N) \leq \mathbf{0} \tag{4.178}$$

for $i = 1, \ldots, N-1$. The functions $\mathbf{g}_i$, Γ_i, $\mathbf{c}_{iE}$, and $\mathbf{c}_{iI}$ are all assumed to be twice differentiable.

4.4.2 Sequential Quadratic Programming Subproblem Formulation

Consider the Lagrangian function L associated with (4.176)–(4.178) and defined as

$$L = \sum_{i=1}^{N-1} \Gamma_i(\mathbf{x}_i, \mathbf{u}_i) + \mathbf{c}_{iE}^T(\mathbf{x}_i, \mathbf{u}_i)\boldsymbol{\lambda}_{iE} + \mathbf{c}_{iI}^T(\mathbf{x}_i, \mathbf{u}_i)\boldsymbol{\lambda}_{iI}$$

$$+ \Gamma_N(\mathbf{x}_N) + \mathbf{c}_{NE}^T(\mathbf{x}_N)\boldsymbol{\lambda}_{NE} + \mathbf{c}_{NI}^T(\mathbf{x}_N)\boldsymbol{\lambda}_{NI}, \tag{4.179}$$

where $\boldsymbol{\lambda}_{iE}$ and $\boldsymbol{\lambda}_{iI}$ are vectors of Lagrange multipliers. With the goal of approximating the Lagrangian as a quadratic function, perturbation variables $\bar{\mathbf{u}}_i$, $\bar{\mathbf{x}}_1$, $\bar{\boldsymbol{\lambda}}_{iE}$, and $\bar{\boldsymbol{\lambda}}_{iI}$ are introduced and defined as

$$\begin{aligned} \bar{\mathbf{u}}_i &= \mathbf{u}_i - \hat{\mathbf{u}}_i, \\ \bar{\mathbf{x}}_1 &= \mathbf{x}_1 - \hat{\mathbf{x}}_1, \\ \bar{\boldsymbol{\lambda}}_{iE} &= \boldsymbol{\lambda}_{iE} - \hat{\boldsymbol{\lambda}}_{iE}, \\ \bar{\boldsymbol{\lambda}}_{iI} &= \boldsymbol{\lambda}_{iI} - \hat{\boldsymbol{\lambda}}_{iI}, \end{aligned} \tag{4.180}$$

where $\hat{\mathbf{u}}_i$, $\hat{\mathbf{x}}_i$, $\hat{\boldsymbol{\lambda}}_{iE}$, and $\hat{\boldsymbol{\lambda}}_{iI}$ are the nominal values of $\mathbf{u}_i$, $\mathbf{x}_i$, $\boldsymbol{\lambda}_{iE}$, and $\boldsymbol{\lambda}_{iI}$, respectively.

Expanding (4.175) and (4.179) as a Taylor series about $\hat{\mathbf{u}}_i$, $\hat{\mathbf{x}}_1$, $\hat{\boldsymbol{\lambda}}_{iE}$, and $\hat{\boldsymbol{\lambda}}_{iI}$ yields the following quadratic approximation of L [29]:

$$\begin{aligned} L_q = \sum_{i=1}^{N-1} \Bigg[& \nu_i + \bar{\mathbf{x}}_i^T \mathbf{y}_i + \bar{\mathbf{u}}_i^T \mathbf{z}_i + \frac{1}{2}\left(\bar{\mathbf{x}}_i^T \mathbf{Q}_i \bar{\mathbf{x}}_i + 2\bar{\mathbf{x}}_i^T \mathbf{R}_i \bar{u}_i + \bar{\mathbf{u}}_i^T \mathbf{S}_i \bar{\mathbf{u}}_i\right) \\ & + \left(\mathbf{A}_{iE}\bar{\mathbf{x}}_i + \mathbf{B}_{iE}\bar{\mathbf{u}}_i + \hat{\mathbf{c}}_{iE}\right)^T \boldsymbol{\lambda}_{iE} + \left(\mathbf{A}_{iI}\bar{\mathbf{x}}_i + \mathbf{B}_{iI}\bar{\mathbf{u}}_i + \hat{\mathbf{c}}_{iI}\right)^T \boldsymbol{\lambda}_{iI} \Bigg] \\ & + \Gamma_N + \bar{\mathbf{x}}_N^T \mathbf{y}_N + \frac{1}{2}\bar{\mathbf{x}}_N^T \mathbf{Q}_N \bar{\mathbf{x}}_N + \left(\mathbf{A}_{NE}\bar{\mathbf{x}}_N + \hat{\mathbf{c}}_{NE}\right)^T \boldsymbol{\lambda}_{NE} + \left(\mathbf{A}_{NI}\bar{\mathbf{x}}_N + \hat{\mathbf{c}}_{NI}\right)^T \boldsymbol{\lambda}_{NI}, \end{aligned} \tag{4.181}$$

where ν_i is a constant,

$$\bar{\mathbf{x}}_{i+1} = \mathbf{A}_i \bar{\mathbf{x}}_i + \mathbf{B}_i \bar{\mathbf{u}}_i, \tag{4.182}$$

and

$$\hat{\mathbf{c}}_{iE} = \mathbf{c}_{iE}(\hat{\mathbf{x}}_i, \hat{\mathbf{u}}_i), \tag{4.183}$$

$$\hat{\mathbf{c}}_{iI} = \mathbf{c}_{iI}(\hat{\mathbf{x}}_i, \hat{\mathbf{u}}_i), \tag{4.184}$$

$$y_i^\alpha = \frac{\partial \Gamma_i}{\partial x_i^\alpha}, \tag{4.185}$$

$$z_i^\alpha = \frac{\partial \Gamma_i}{\partial u_i^\alpha}, \tag{4.186}$$

$$a_i^{\alpha\beta} = \frac{\partial g_i^\alpha}{\partial x_i^\beta}, \tag{4.187}$$

$$b_i^{\alpha\beta} = \frac{\partial g_i^\alpha}{\partial u_i^\beta}, \tag{4.188}$$

$$a_{iE}^{\alpha\beta} = \frac{\partial c_{iE}^\alpha}{\partial x_i^\beta}, \tag{4.189}$$

$$b_{iE}^{\alpha\beta} = \frac{\partial c_{iE}^\alpha}{\partial u_i^\beta}, \tag{4.190}$$

$$a_{iI}^{\alpha\beta} = \frac{\partial c_{iI}^\alpha}{\partial x_i^\beta}, \tag{4.191}$$

$$b_{iI}^{\alpha\beta} = \frac{\partial c_{iI}^\alpha}{\partial u_i^\beta}, \tag{4.192}$$

$$q_i^{\alpha\beta} = \frac{\partial^2 \Gamma_i}{\partial x_i^\alpha \partial x_i^\beta} + \frac{\partial^2 c_{iE}^\gamma}{\partial x_i^\alpha \partial x_i^\beta}\hat{\lambda}_{iE}^\gamma + \frac{\partial^2 c_{iI}^\gamma}{\partial x_i^\alpha \partial x_i^\beta}\hat{\lambda}_{iI}^\gamma + \frac{\partial^2 g_i^\gamma}{\partial x_i^\alpha \partial x_i^\beta} p_{i+1}^\gamma, \tag{4.193}$$

$$r_i^{\alpha\beta} = \frac{\partial^2 \Gamma_i}{\partial x_i^\alpha \partial u_i^\beta} + \frac{\partial^2 c_{iE}^\gamma}{\partial x_i^\alpha \partial u_i^\beta}\hat{\lambda}_{iE}^\gamma + \frac{\partial^2 c_{iI}^\gamma}{\partial x_i^\alpha \partial u_i^\beta}\hat{\lambda}_{iI}^\gamma + \frac{\partial^2 g_i^\gamma}{\partial x_i^\alpha \partial u_i^\beta} p_{i+1}^\gamma, \tag{4.194}$$

$$s_i^{\alpha\beta} = \frac{\partial^2 \Gamma_i}{\partial u_i^\alpha \partial u_i^\beta} + \frac{\partial^2 c_{iE}^\gamma}{\partial u_i^\alpha \partial u_i^\beta}\hat{\lambda}_{iE}^\gamma + \frac{\partial^2 c_{iI}^\gamma}{\partial u_i^\alpha \partial u_i^\beta}\hat{\lambda}_{iI}^\gamma + \frac{\partial^2 g_i^\gamma}{\partial u_i^\alpha \partial u_i^\beta} p_{i+1}^\gamma. \tag{4.195}$$

The partial derivatives in (4.185)–(4.195) are all evaluated at the nominal values of their arguments. The terms $\mathbf{p}_i$ appearing in (4.193)–(4.195) are obtained from the recursive equation

$$\mathbf{p}_i = \mathbf{y}_i + \mathbf{A}_{iE}^T \hat{\boldsymbol{\lambda}}_{iE} + \mathbf{A}_{iI}^T \hat{\boldsymbol{\lambda}}_{iI} + \mathbf{A}_i^T \mathbf{p}_{i+1} \tag{4.196}$$

starting with

$$\mathbf{p}_N = \mathbf{y}_N + \mathbf{A}_{NE}^T \hat{\boldsymbol{\lambda}}_{NE} + \mathbf{A}_{NI}^T \hat{\boldsymbol{\lambda}}_{NI}. \tag{4.197}$$

The SQP subproblem P2 is stated as follows. Find the controls $\bar{\mathbf{u}}_1, \ldots, \bar{\mathbf{u}}_{N-1}$ and the initial state $\bar{\mathbf{x}}_1$ that minimize the function

$$\Gamma_q = \sum_{i=1}^{N-1} \left[\Gamma_i + \bar{\mathbf{x}}_i^T \mathbf{y}_i + \bar{\mathbf{u}}_i^T \mathbf{z}_i + \frac{1}{2}\left(\bar{\mathbf{x}}_i^T \mathbf{Q}_i \bar{\mathbf{x}}_i + 2\mathbf{x}_i^T \mathbf{R}_i \bar{\mathbf{u}}_i + \bar{\mathbf{u}}_i^T \mathbf{S}_i \bar{\mathbf{u}}_i \right) \right]$$

$$+ \Gamma_n + \bar{\mathbf{x}}_N^T \mathbf{y}_N + \frac{1}{2} \bar{\mathbf{x}}_N^T \mathbf{Q}_N \bar{\mathbf{x}}_N \tag{4.198}$$

subject to (4.182) and the constraints

$$\mathbf{A}_{iE}\bar{\mathbf{x}}_i + \mathbf{B}_{iE}\bar{\mathbf{u}}_i + \hat{\mathbf{c}}_{iE} = \mathbf{0} \quad \text{and} \quad \mathbf{A}_{NE}\bar{\mathbf{x}}_N + \hat{\mathbf{c}}_{NE} = \mathbf{0} \tag{4.199}$$

and

$$\mathbf{A}_{iI}\bar{\mathbf{x}}_i + \mathbf{B}_{iI}\bar{\mathbf{u}}_i + \hat{\mathbf{c}}_{iI} \leq \mathbf{0} \quad \text{and} \quad \mathbf{A}_{NI}\bar{\mathbf{x}}_N + \hat{\mathbf{c}}_{NI} \leq \mathbf{0}. \tag{4.200}$$

4.4.3 Interior Point Methods

As a brief introduction to primal-dual interior point methods, one can follow the development of Wright [60] and consider the following general convex quadratic program:

$$\min_{\mathbf{z}} \frac{1}{2}\mathbf{z}^T\mathbf{Q}\mathbf{z} + \mathbf{z}^T\mathbf{c} \quad \text{subject to} \quad \mathbf{H}\mathbf{z} + \mathbf{h} = \mathbf{0}, \quad \mathbf{G}\mathbf{z} + \mathbf{g} \leq \mathbf{0}, \tag{4.201}$$

where $\mathbf{Q}$ is a symmetric positive semidefinite matrix. The first-order Karush–Kuhn–Tucker (KKT) conditions for this quadratic program are

$$\mathbf{Q}\mathbf{z} + \mathbf{H}^T\boldsymbol{\zeta} + \mathbf{G}^T\boldsymbol{\lambda} = -\mathbf{c}, \tag{4.202}$$

$$\mathbf{H}\mathbf{z} = -\mathbf{h}, \tag{4.203}$$

$$\mathbf{G}\mathbf{z} + \mathbf{s} = -\mathbf{g}, \tag{4.204}$$

$$\mathbf{s} \geq \mathbf{0}, \quad \boldsymbol{\lambda} \geq \mathbf{0}, \quad \mathbf{s}^T\boldsymbol{\lambda} = 0, \tag{4.205}$$

where $\mathbf{s}$ is a vector of positive slack variables of dimension n_2.

Let the values of $\mathbf{z}$, $\boldsymbol{\zeta}$, $\boldsymbol{\lambda}$, and $\mathbf{s}$ at the kth iteration be denoted by $\mathbf{z}^k$, $\boldsymbol{\zeta}^k$, $\boldsymbol{\lambda}^k$, and $\mathbf{s}^k$, respectively. The infeasible interior point method for this system starts at a point $(\mathbf{z}^0, \boldsymbol{\zeta}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ that has $\boldsymbol{\lambda}^0 > \mathbf{0}$ and $\mathbf{s}^0 > \mathbf{0}$ (*interior* to the nonnegative orthant) but is possibly *infeasible* with respect to (4.202)–(4.204). All iterates $(\mathbf{z}^k, \boldsymbol{\zeta}^k, \boldsymbol{\lambda}^k, \mathbf{s}^k)$ retain the positivity properties $\boldsymbol{\lambda}^k > \mathbf{0}$ and $\mathbf{s}^k > \mathbf{0}$, but the infeasibilities and the complementarity gap defined by

$$\mu_k = (\boldsymbol{\lambda}^k)^T\frac{\mathbf{s}^k}{n_2} \tag{4.206}$$

are gradually reduced to zero as $k \to \infty$. Each step of the algorithm is a modified Newton step for the system of equations given by (4.202)–(4.204) and the complementarity conditions $\lambda^\beta s^\beta = 0$ for $\beta = 1, \ldots, n_2$. The system of linear equations solved at each iteration is of the form

$$\begin{bmatrix} \mathbf{Q} & \mathbf{H}^T & \mathbf{G}^T & \mathbf{0} \\ \mathbf{H} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{G} & \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{S}^k & \boldsymbol{\Lambda}^k \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{z}} \\ \tilde{\boldsymbol{\zeta}} \\ \tilde{\boldsymbol{\lambda}} \\ \tilde{\mathbf{s}} \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_1^k \\ -\mathbf{r}_2^k \\ -\mathbf{r}_3^k \\ \mathbf{d}_k \end{bmatrix}, \tag{4.207}$$

where

$$\mathbf{r}_1^k = \mathbf{Q}\mathbf{z}^k + \mathbf{H}^T\boldsymbol{\zeta}^k + \mathbf{G}^T\boldsymbol{\lambda}^k + \mathbf{c}, \tag{4.208}$$

$$\mathbf{r}_2^k = \mathbf{H}\mathbf{z}^k + \mathbf{h}, \tag{4.209}$$

$$\mathbf{r}_3^k = \mathbf{G}\mathbf{z}^k + \mathbf{s}^k + \mathbf{g}, \tag{4.210}$$

and $\mathbf{S}^k$ and $\boldsymbol{\Lambda}^k$ are diagonal matrices defined by $\mathbf{S}^k = \text{diag}(\mathbf{s}^k)$ and $\boldsymbol{\Lambda}^k = \text{diag}(\boldsymbol{\lambda}^k)$. Updated values of the unknowns are obtained from an equation of the form

$$(\mathbf{z}^{k+1}, \boldsymbol{\zeta}^{k+1}, \boldsymbol{\lambda}^{k+1}, \mathbf{s}^{k+1}) = (\mathbf{z}^k, \boldsymbol{\zeta}^k, \boldsymbol{\lambda}^k, \mathbf{s}^k) + \alpha_k(\tilde{\mathbf{z}}, \tilde{\boldsymbol{\zeta}}, \tilde{\boldsymbol{\lambda}}, \tilde{\mathbf{s}}) \tag{4.211}$$

for some $\alpha_k \in (0, 1]$. The value used for $\mathbf{d}_k$ in (4.207) depends on specific details of the algorithm.

From (4.207), one obtains

$$\tilde{\mathbf{s}} = (\boldsymbol{\Lambda}^k)^{-1}(\mathbf{d}_k - \mathbf{S}^k\tilde{\boldsymbol{\lambda}}). \tag{4.212}$$

Substituting (4.212) into (4.207) yields

$$\begin{bmatrix} \mathbf{Q} & \mathbf{H}^T & \mathbf{G}^T \\ \mathbf{H} & \mathbf{0} & \mathbf{0} \\ \mathbf{G} & \mathbf{0} & -(\boldsymbol{\Lambda}^k)^{-1}\mathbf{S}^k \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{z}} \\ \tilde{\boldsymbol{\zeta}} \\ \tilde{\boldsymbol{\lambda}} \end{bmatrix} = -\begin{bmatrix} \mathbf{r}_1^k \\ \mathbf{r}_2^k \\ \mathbf{r}_3^k + (\boldsymbol{\Lambda}^k)^{-1}\mathbf{d}_k \end{bmatrix}. \tag{4.213}$$

In order to relate the quadratic program given by (4.201) to P2, let

$$\mathbf{z} = \begin{bmatrix} \bar{\mathbf{u}}_1 \\ \vdots \\ \bar{\mathbf{u}}_{N-1} \\ \bar{\mathbf{x}}_1 \end{bmatrix}, \tag{4.214}$$

$$\boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\lambda}_{1E} \\ \vdots \\ \boldsymbol{\lambda}_{NE} \end{bmatrix}, \tag{4.215}$$

$$\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_{1I} \\ \vdots \\ \boldsymbol{\lambda}_{NI} \end{bmatrix}, \tag{4.216}$$

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}_{1I} \\ \vdots \\ \mathbf{s}_{NI} \end{bmatrix}, \tag{4.217}$$

$$\mathbf{d}_k = \begin{bmatrix} \mathbf{d}_{1k} \\ \vdots \\ \mathbf{d}_{Nk} \end{bmatrix}. \tag{4.218}$$

In the light of (4.182) and (4.198)–(4.201), it is clear that the matrices $\mathbf{Q}$, $\mathbf{H}$, and $\mathbf{G}$ associated with P2 will generally be dense. Rather than solve (4.213) directly, the approach taken here is to modify an existing algorithm to solve a related problem with the identical solution.

4.4.4 Equivalent Problem Formulation

Consider the Lagrangian function L_q associated with (4.201) and defined as

$$L_q(\mathbf{z}, \boldsymbol{\zeta}, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{z}^T\mathbf{Q}\mathbf{z} + \mathbf{z}^T\mathbf{c} + \boldsymbol{\zeta}^T(\mathbf{H}\mathbf{z} + \mathbf{h}) + \boldsymbol{\lambda}^T(\mathbf{G}\mathbf{z} + \mathbf{g}). \tag{4.219}$$

Introducing the change of variables

$$\mathbf{z} = \mathbf{z}^k + \tilde{\mathbf{z}}, \tag{4.220}$$

$$\boldsymbol{\zeta} = \boldsymbol{\zeta}^k + \tilde{\boldsymbol{\zeta}}, \tag{4.221}$$

$$\boldsymbol{\lambda} = \boldsymbol{\lambda}^k + \tilde{\boldsymbol{\lambda}} \tag{4.222}$$

into (4.219), one can verify that the solution of the equations

$$\frac{\partial L_q^k}{\partial \tilde{\mathbf{z}}} = \mathbf{0}, \tag{4.223}$$

$$\frac{\partial L_q^k}{\partial \tilde{\boldsymbol{\zeta}}} = \mathbf{0}, \tag{4.224}$$

$$\frac{\partial L_q^k}{\partial \tilde{\boldsymbol{\lambda}}} = \mathbf{0} \tag{4.225}$$

is identical to the solution of (4.213), where

$$L_q^k(\tilde{\mathbf{z}}, \tilde{\boldsymbol{\zeta}}, \boldsymbol{\lambda}) = L_q(\mathbf{z}^k + \tilde{\mathbf{z}}, \boldsymbol{\zeta}^k + \tilde{\boldsymbol{\zeta}}, \boldsymbol{\lambda}^k + \tilde{\boldsymbol{\lambda}})$$

$$- \frac{1}{2}\tilde{\boldsymbol{\lambda}}^T(\boldsymbol{\Lambda}^T)^{-1}\mathbf{S}^k\tilde{\boldsymbol{\lambda}} + \tilde{\boldsymbol{\lambda}}^T[\mathbf{s}^k + (\boldsymbol{\Lambda}^k)^{-1}\mathbf{d}_k]. \tag{4.226}$$

That is, the values of $\tilde{\mathbf{z}}$, $\tilde{\boldsymbol{\zeta}}$, and $\tilde{\boldsymbol{\lambda}}$ that make L_q^k stationary also satisfy (4.213).

Let

$$\bar{\mathbf{u}}_i = \mathbf{u}_i^k + \tilde{\mathbf{u}}_i, \tag{4.227}$$

$$\bar{\mathbf{x}}_i = \mathbf{x}_i^k + \tilde{\mathbf{x}}_i, \tag{4.228}$$

$$\boldsymbol{\lambda}_{iE} = \boldsymbol{\lambda}_{iE}^k + \tilde{\boldsymbol{\lambda}}_{iE}, \tag{4.229}$$

$$\boldsymbol{\lambda}_{iI} = \boldsymbol{\lambda}_{iI}^k + \tilde{\boldsymbol{\lambda}}_{iI}, \tag{4.230}$$

$$\mathbf{s}_{iI} = \mathbf{s}_{iI}^k + \tilde{\mathbf{s}}_{iI}, \tag{4.231}$$

where the superscript k denotes the iteration number. By associating the quadratic program of (4.201) with P2, it follows from (4.181)–(4.182), (4.219), and (4.226) that

$$L_q^k = \sum_{i=1}^{N-1} \Big[v_i^k + \tilde{\mathbf{x}}_i^T \mathbf{y}_i^k + \tilde{\mathbf{u}}_i^T \mathbf{z}_i^k + \frac{1}{2}\left(\tilde{\mathbf{x}}_i^T \mathbf{Q}_i \tilde{\mathbf{x}}_i + 2\tilde{\mathbf{x}}_i^T \mathbf{R}_i \tilde{\mathbf{u}}_i + \tilde{\mathbf{u}}_i^T \mathbf{S}_i \tilde{\mathbf{u}}_i\right)$$

$$+ \left(\mathbf{A}_{iE}\tilde{\mathbf{x}}_i + \mathbf{B}_{iE}\tilde{\mathbf{u}}_i + \mathbf{c}_{iE}^k\right)^T \tilde{\boldsymbol{\lambda}}_{iE} + \left(\mathbf{A}_{iI}\tilde{\mathbf{x}}_i + \mathbf{B}_{iI}\tilde{\mathbf{u}}_i + \mathbf{c}_{iI}^k\right)^T \tilde{\boldsymbol{\lambda}}_{iI} - \frac{1}{2}\tilde{\boldsymbol{\lambda}}_{iI}^T \mathbf{G}_i^k \tilde{\boldsymbol{\lambda}}_{iI} \Big] + \tilde{\mathbf{x}}_N^T \mathbf{y}_N^k$$

$$+ \frac{1}{2}\tilde{\mathbf{x}}_N^T \mathbf{Q}_N \tilde{\mathbf{x}}_N + \left(\mathbf{A}_{NE}\tilde{\mathbf{x}}_N + \mathbf{c}_{NE}^k\right)^T \tilde{\boldsymbol{\lambda}}_{NE} + \left(\mathbf{A}_{NI}\tilde{\mathbf{x}}_N + \mathbf{c}_{NI}^k\right)^T \tilde{\boldsymbol{\lambda}}_{NI} - \frac{1}{2}\tilde{\boldsymbol{\lambda}}_{NI}^T \mathbf{G}_N^k \tilde{\boldsymbol{\lambda}}_{NI} \tag{4.232}$$

and

$$\tilde{\mathbf{x}}_{i+1} = \mathbf{A}_i \tilde{\mathbf{x}}_i + \mathbf{B}_i \tilde{\mathbf{u}}_i, \tag{4.233}$$

$$\tilde{\mathbf{x}}_{i+1}^k = \mathbf{A}_i \mathbf{x}_i^k + \mathbf{B}_i \mathbf{u}_i^k, \tag{4.234}$$

where v_i^k is a constant and

$$\mathbf{y}_i^k = \mathbf{y}_i + \mathbf{Q}_i \mathbf{x}_i^k + \mathbf{R}_i \mathbf{u}_i^k + \mathbf{A}_{iE}^T \boldsymbol{\lambda}_{iE}^k + \mathbf{A}_{iI}^T \boldsymbol{\lambda}_{iI}^k, \tag{4.235}$$

$$\mathbf{z}_i^k = \mathbf{z}_i + \mathbf{R}_i^T \mathbf{x}_i^k + \mathbf{S}_i \mathbf{u}_i^k + \mathbf{B}_{iE}^T \boldsymbol{\lambda}_{iE}^k + \mathbf{B}_{iI}^T \boldsymbol{\lambda}_{iI}^k, \tag{4.236}$$

$$\mathbf{c}_{iE}^k = \mathbf{A}_{iE}\mathbf{x}_i^k + \mathbf{B}_{iE}\mathbf{u}_i^k + \hat{\mathbf{c}}_{iE}, \tag{4.237}$$

$$\mathbf{c}_{iI}^k = \mathbf{A}_{iI}\mathbf{x}_i^k + \mathbf{B}_i\mathbf{u}_i^k + \hat{\mathbf{c}}_{iI} + \mathbf{s}_{iI}^K + [\mathrm{diag}(\boldsymbol{\lambda}_{iI}^k)]^{-1}\mathbf{d}_{ik}, \tag{4.238}$$

$$\mathbf{G}_i^k = [\mathrm{diag}(\boldsymbol{\lambda}_{iI}^k)]^{-1}\ \mathrm{diag}(\mathbf{s}_{iI}^k). \tag{4.239}$$

The matrix $\mathbf{G}_i^k$ may be expressed as

$$\mathbf{G}_i^k = \begin{bmatrix} \mathbf{P}_{ib} & \mathbf{P}_{if} \end{bmatrix} \begin{bmatrix} \mathbf{G}_{ib} & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_{if} \end{bmatrix} \begin{bmatrix} \mathbf{P}_{ib}^T \\ \mathbf{P}_{if}^T \end{bmatrix}, \tag{4.240}$$

where $[\mathbf{P}_{ib} \quad \mathbf{P}_{if}]$ is a permutation matrix chosen such that the diagonal elements of the second matrix on the right-hand side of (4.240) are sorted in ascending numerical order. The number of columns in $\mathbf{P}_{ib}$ equals the number of diagonal elements in $\mathbf{G}_i^k$ that are less than unity.

It proves useful to expand $\tilde{\boldsymbol{\lambda}}_{iI}$ as

$$\tilde{\boldsymbol{\lambda}}_{iI} = \mathbf{P}_{ib}\tilde{\boldsymbol{\lambda}}_{ib} + \mathbf{P}_{if}\tilde{\boldsymbol{\lambda}}_{if}. \tag{4.241}$$

Separation of $\tilde{\boldsymbol{\lambda}}_{iI}$ into contributions due to $\tilde{\boldsymbol{\lambda}}_{ib}$ and $\tilde{\boldsymbol{\lambda}}_{if}$ is done to avoid ill conditioning of matrices used in the solution of the equivalent problem. Substituting (4.241) for $\tilde{\boldsymbol{\lambda}}_{iI}$ into (4.232) and setting the gradient of L_q^k with respect to $\tilde{\boldsymbol{\lambda}}_{if}$ equal to zero yields

$$\tilde{\boldsymbol{\lambda}}_{if} = \mathbf{G}_{if}^{-1}\mathbf{P}_{if}^T\left(\mathbf{A}_{iI}\tilde{\mathbf{x}}_i + \mathbf{B}_{iI}\tilde{\mathbf{u}}_i + \mathbf{c}_{iI}^k\right). \tag{4.242}$$

Substituting (4.242) for $\tilde{\boldsymbol{\lambda}}_{if}$ into (4.241) and the result into (4.232) yields

$$L_q^k = \sum_{i=1}^{N-1} \Big[\bar{v}_i^k + \tilde{\mathbf{x}}_i^T \bar{\mathbf{y}}_i + \tilde{\mathbf{u}}_i^T \bar{\mathbf{z}}_i + \frac{1}{2}\left(\tilde{\mathbf{x}}_i^T \bar{\mathbf{Q}}_i \tilde{\mathbf{x}}_i + 2\tilde{\mathbf{x}}_i^T \bar{\mathbf{R}}_i \tilde{\mathbf{u}}_i + \tilde{\mathbf{u}}_i^T \bar{\mathbf{S}}_i \tilde{\mathbf{u}}_i\right) + \left(\bar{\mathbf{A}}_i \tilde{\mathbf{x}}_i + \bar{\mathbf{B}}_i \tilde{\mathbf{u}}_i + \bar{\mathbf{c}}_i\right)^T \tilde{\boldsymbol{\lambda}}_i - \frac{1}{2}\tilde{\boldsymbol{\lambda}}_i^T \bar{\mathbf{G}}_i \tilde{\boldsymbol{\lambda}}_i \Big] + \tilde{\mathbf{x}}_N^T \bar{\mathbf{y}}_N + \frac{1}{2}\tilde{\mathbf{x}}_N^T \bar{\mathbf{Q}}_N \tilde{\mathbf{x}}_N + \left(\bar{\mathbf{A}}_N \tilde{\mathbf{x}}_N + \bar{\mathbf{c}}_N\right)^T \tilde{\boldsymbol{\lambda}}_N - \frac{1}{2}\tilde{\boldsymbol{\lambda}}_N^T \bar{\mathbf{G}}_N \tilde{\boldsymbol{\lambda}}_N, \tag{4.243}$$

where $\bar{v}_i^k$ is a constant and

$$\bar{\mathbf{y}}_i = \mathbf{y}_i^k + \mathbf{A}_{iI}^T \bar{\mathbf{P}}_i \mathbf{c}_{iI}^k, \tag{4.244}$$

$$\bar{\mathbf{z}}_i = \mathbf{z}_i^k + \mathbf{B}_{iI}^T \bar{\mathbf{P}}_i \mathbf{c}_{iI}^k, \tag{4.245}$$

$$\bar{\mathbf{P}}_i = \mathbf{P}_{if} \mathbf{G}_{if}^{-1} \mathbf{P}_{if}^T, \tag{4.246}$$

$$\bar{\mathbf{Q}}_i = \mathbf{Q}_i + \mathbf{A}_{iI}^T \bar{\mathbf{P}}_i \mathbf{A}_{iI}, \tag{4.247}$$

$$\bar{\mathbf{R}}_i = \mathbf{R}_i + \mathbf{A}_{iI}^T \bar{\mathbf{P}}_i \mathbf{B}_{iI}, \tag{4.248}$$

$$\bar{\mathbf{S}}_i = \mathbf{S}_i + \mathbf{B}_{iI}^T \bar{\mathbf{P}}_i \mathbf{B}_{iI}, \tag{4.249}$$

$$\bar{\mathbf{A}}_i = \begin{bmatrix} \mathbf{A}_{iE} \\ \mathbf{P}_{ib}^T \mathbf{A}_{iI} \end{bmatrix}, \tag{4.250}$$

$$\bar{\mathbf{B}}_i = \begin{bmatrix} \mathbf{B}_{iE} \\ \mathbf{P}_{ib}^T \mathbf{B}_{iI} \end{bmatrix}, \tag{4.251}$$

$$\bar{\mathbf{c}}_i = \begin{bmatrix} \mathbf{c}_{iE}^k \\ \mathbf{P}_{ib}^T \mathbf{c}_{iI}^k \end{bmatrix}, \tag{4.252}$$

$$\tilde{\boldsymbol{\lambda}}_i = \begin{bmatrix} \tilde{\boldsymbol{\lambda}}_{iE} \\ \tilde{\boldsymbol{\lambda}}_{ib} \end{bmatrix}, \tag{4.253}$$

$$\bar{\mathbf{G}}_i = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_{ib} \end{bmatrix}. \tag{4.254}$$

A modification of the DP algorithm [29] is presented in Section A.3 of Appendix A to solve the equivalent problem. This algorithm can be used to determine the controls $\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_{N-1}$; the initial state $\tilde{\mathbf{x}}_1$; and the Lagrange multipliers $\tilde{\boldsymbol{\lambda}}_1, \ldots, \tilde{\boldsymbol{\lambda}}_N$ that make L_q^k stationary subject to (4.233). Equations (4.233) and (4.253) are then used to calculate the states $\tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_N$ and the Lagrange multipliers $\tilde{\boldsymbol{\lambda}}_{iE}$ and $\tilde{\boldsymbol{\lambda}}_{ib}$ for $i = 1, \ldots, N$. Values of $\tilde{\boldsymbol{\lambda}}_{if}$ calculated from (4.242) are then used to determine $\tilde{\boldsymbol{\lambda}}_{iI}$ from (4.241). The slack variables $\tilde{\mathbf{s}}_{iI}$ are given by the equation

$$\tilde{\mathbf{s}}_{iI} = -\left[\mathbf{A}_{iI}\left(\mathbf{x}_i^k + \tilde{\mathbf{x}}_i\right) + \mathbf{B}_{iI}\left(\mathbf{u}_i^k + \tilde{\mathbf{u}}_i\right) + \hat{\mathbf{c}}_{iI} + \mathbf{s}_i^k\right]. \tag{4.255}$$

Finally, the previously determined quantities are assembled as

$$\tilde{\mathbf{z}} = \begin{bmatrix} \tilde{\mathbf{u}}_1 \\ \vdots \\ \tilde{\mathbf{u}}_{N-1} \\ \tilde{\mathbf{x}}_1 \end{bmatrix}, \tag{4.256}$$

$$\tilde{\boldsymbol{\zeta}} = \begin{bmatrix} \tilde{\boldsymbol{\lambda}}_{1E} \\ \vdots \\ \tilde{\boldsymbol{\lambda}}_{NE} \end{bmatrix}, \tag{4.257}$$

$$\tilde{\boldsymbol{\lambda}} = \begin{bmatrix} \tilde{\boldsymbol{\lambda}}_{1I} \\ \vdots \\ \tilde{\boldsymbol{\lambda}}_{NI} \end{bmatrix}, \tag{4.258}$$

$$\tilde{\mathbf{s}} = \begin{bmatrix} \tilde{\mathbf{s}}_{1I} \\ \vdots \\ \tilde{\mathbf{s}}_{NI} \end{bmatrix}. \tag{4.259}$$

4.4.5 Details of Algorithm

An algorithm is presented in this section for solving P2 based on an adaptation of Mehrotra's predictor-corrector algorithm. The algorithm uses a common step length for both primal and dual variables and is based largely on material taken from Wright [61].

The affine scaling "predictor" direction $(\tilde{\mathbf{z}}_a, \tilde{\boldsymbol{\zeta}}_a, \tilde{\boldsymbol{\lambda}}_a, \tilde{\mathbf{s}}_a)$ is defined as the solution to (4.207) with

$$\mathbf{d}_k = -\text{diag}(\boldsymbol{\lambda}^k)\,\text{diag}(\mathbf{s}^k)\mathbf{e}, \tag{4.260}$$

where

$$\mathbf{e} = [1, 1, \ldots, 1]^T. \tag{4.261}$$

With reference to (4.238), this direction is obtained by setting

$$\mathbf{c}_{iI}^k = \mathbf{A}_{iI}\mathbf{x}_i^k + \mathbf{B}_{iI}\mathbf{u}_i^k + \hat{\mathbf{c}}_{iI} \tag{4.262}$$

and solving the equivalent problem. The step length α_a and complementarity gap μ_a for the affine scaling direction are defined as

$$\alpha_a = \arg\ \max\{\alpha \in [0, 1] | \boldsymbol{\lambda}^k + \alpha\tilde{\boldsymbol{\lambda}}_a \geq \mathbf{0}, \quad \mathbf{s}^k + \alpha\tilde{\mathbf{s}}_a \geq \mathbf{0}\}, \tag{4.263}$$

$$\mu_a = (\boldsymbol{\lambda}^k + \alpha_a\tilde{\boldsymbol{\lambda}}_a)^T \frac{1}{n_2}\left(\mathbf{s}^k + \alpha_a\tilde{\mathbf{s}}_a\right). \tag{4.264}$$

The centering parameter σ is chosen as

$$\sigma = \left(\frac{\mu_a}{\mu_k}\right)^3, \tag{4.265}$$

where μ_k is given by (4.206).

The combined predictor-centering-corrector (PCC) direction $(\tilde{\mathbf{z}}_p, \tilde{\boldsymbol{\zeta}}_p, \tilde{\boldsymbol{\lambda}}_p, \tilde{\mathbf{s}}_p)$ is defined as the solution to (4.207) with

$$\mathbf{d}_k = -\text{diag}(\boldsymbol{\lambda}^k)\,\text{diag}(\mathbf{s}^k) - \text{diag}(\tilde{\boldsymbol{\lambda}}_a)\,\text{diag}(\tilde{\mathbf{s}}_a)\mathbf{e} + \sigma\mu_k\mathbf{e}. \tag{4.266}$$

With reference to (4.238), this direction is obtained by setting

$$\mathbf{c}_{iI}^k = \mathbf{A}_{iI}\mathbf{x}_i^k + \mathbf{B}_{iI}\mathbf{u}_i^k + \hat{\mathbf{c}}_{iI} + [\text{diag}(\boldsymbol{\lambda}_{iI}^k)]^{-1}[\sigma\mu_k\mathbf{e} - \text{diag}(\tilde{\boldsymbol{\lambda}}_{ai})\,\text{diag}(\tilde{\mathbf{s}}_{ai})\mathbf{e}] \tag{4.267}$$

and solving the equivalent problem. The maximum step length for this direction is given by

$$\alpha_{max} = \arg\ \max\{\alpha \in [0, 1] | \boldsymbol{\lambda}_k + \alpha\tilde{\boldsymbol{\lambda}}_p \geq \mathbf{0}, \quad \mathbf{s}^k + \alpha\tilde{\mathbf{s}}_p \geq \mathbf{0}\}. \tag{4.268}$$

Given $\delta > 0, k = 0$, and $(\mathbf{z}^0, \boldsymbol{\zeta}^0, \boldsymbol{\lambda}^0, \mathbf{s}^0)$ with $\boldsymbol{\lambda}^0 > \mathbf{0}$ and $\mathbf{s}^0 > \mathbf{0}$, the following algorithm can be used to solve P2.

Step 1: Calculate the residual vector $\mathbf{r}^k$ defined as

$$\mathbf{r}^k = \sqrt{(\mathbf{r}_1^k)^2 + (\mathbf{r}_2^k)^2 + (\mathbf{r}_3^k)^2 + (\mathbf{r}_4^k)^2}, \tag{4.269}$$

where expressions for $\mathbf{r}_1^k, \ldots, \mathbf{r}_4^k$ are given in Section A.3 of Appendix A. If $\mathbf{r}^k < \delta$, then proceed to Step 5.

Step 2: Calculate the affine scaling direction $(\tilde{\mathbf{z}}_a, \tilde{\boldsymbol{\zeta}}_a, \tilde{\boldsymbol{\lambda}}_a, \tilde{\mathbf{s}}_a)$ and the scalars α_a, μ_a, and σ (see (4.262)–(4.265)).

Step 3: Calculate the combined PCC direction $(\tilde{\mathbf{z}}_p, \tilde{\boldsymbol{\zeta}}_p, \tilde{\boldsymbol{\lambda}}_p, \tilde{\mathbf{s}}_p)$ and the scalar α_{max} (see (4.267)–(4.268)).

Step 4: Calculate $(\mathbf{z}^{z+1}, \boldsymbol{\zeta}^{z+1}, \boldsymbol{\lambda}^{k+1}, \mathbf{s}^{k+1})$ from the equation

$$(\mathbf{z}^{k+1}, \boldsymbol{\zeta}^{k+1}, \boldsymbol{\lambda}^{k+1}, \mathbf{s}^{k+1}) = (\mathbf{z}^k, \boldsymbol{\zeta}^k, \boldsymbol{\lambda}^k, \mathbf{s}^k) + \alpha(\tilde{\mathbf{z}}_p, \tilde{\boldsymbol{\zeta}}_p, \tilde{\boldsymbol{\lambda}}_p, \tilde{\mathbf{s}}_p), \tag{4.270}$$

where

$$\alpha = \min(0.995\ \alpha_{max}, 1). \tag{4.271}$$

Set $k \leftarrow k + 1$ and return to Step 1.

Step 5: Calculate the search direction $\tilde{\mathbf{u}}_i$, $\tilde{\mathbf{x}}_1$, $\tilde{\boldsymbol{\lambda}}_{iE}$, $\tilde{\boldsymbol{\lambda}}_{iI}$, and $\tilde{\mathbf{s}}_{iI}$ by solving the equivalent problem with $\mathbf{c}_{iI}^k$ given by (4.262), $\bar{\mathbf{P}}_i = \mathbf{0}$, and $\mathbf{G}_{ib} = \mathbf{0}$ (see (4.246) and (4.254)). Calculate

$$\mathbf{u}_i^* = \mathbf{u}_i^k + \tilde{\mathbf{u}}_i, \tag{4.272}$$

$$\mathbf{x}_1^* = \mathbf{x}_1^k + \tilde{\mathbf{x}}_1, \tag{4.273}$$

$$\boldsymbol{\lambda}_{iE}^* = \boldsymbol{\lambda}_{iE}^k + \tilde{\boldsymbol{\lambda}}_{iE}, \tag{4.274}$$

$$\boldsymbol{\lambda}_{iI}^* = \boldsymbol{\lambda}_{iI}^k + \tilde{\boldsymbol{\lambda}}_{iI}, \tag{4.275}$$

$$\mathbf{s}_{iI}^* = \mathbf{s}_{iI}^k + \tilde{\mathbf{s}}_{iI}. \tag{4.276}$$

If $\boldsymbol{\lambda}_{iI}^* \geq \mathbf{0}$ and $\mathbf{s}_{iI}^* \geq \mathbf{0}$ for $i = 1, \ldots, N$, then exit the algorithm. Otherwise, return to Step 2.

Step 5 solves an equality-constrained problem in which a subset of the inequality constraints are treated as equality constraints. To elaborate, consider the ratio $r_i^\beta = (s_{iI}^k)^\beta/(\lambda_{iI}^k)^\beta$ for a specific value of i and β. If $r_i^\beta < 1$, then the constraint is treated as an equality constraint. If $r_i^\beta \geq 1$, then the constraint is ignored. Checking the conditions $\boldsymbol{\lambda}_{iI}^* \geq \mathbf{0}$ and $\mathbf{s}_{iI}^* \geq \mathbf{0}$ in Step 5 ensures that the Lagrange multipliers and slack variables for the inequality constraints satisfy (4.205). It is assumed that strict complementarity holds at the solution to P2. That is, each element of the vector $\boldsymbol{\lambda}_{iI} + \mathbf{s}_{iI}$ is greater than zero for $i = 1, \ldots, N$. Finally, note that if the solution set to P2 is empty, then the norms of the search directions approach infinity as $k \to \infty$ [61].

In order to apply the algorithm described above, it is necessary to specify the values of $\mathbf{u}_i^0$, $\mathbf{x}_1^0$, $\boldsymbol{\lambda}_{iE}^0$, $\boldsymbol{\lambda}_{iI}^0$, and $\mathbf{s}_{iI}^0$. The approach taken here is to set

$$\mathbf{u}_i^0 = \mathbf{0}, \quad \mathbf{x}_1^0 = \mathbf{0}, \quad \boldsymbol{\lambda}_{iE}^0 = \hat{\boldsymbol{\lambda}}_{iE}, \tag{4.277}$$

and

$$\boldsymbol{\lambda}_{iI}^0 = \max(\hat{\boldsymbol{\lambda}}_{iI}, \varepsilon\mathbf{e}), \qquad \mathbf{s}_{iI}^0 = \max(-\hat{\mathbf{c}}_{iI}, \varepsilon\mathbf{e}), \tag{4.278}$$

where ε is a positive scalar and the function "max" is applied componentwise in (4.278). This approach satisfies the positivity requirements $\boldsymbol{\lambda}_{iI}^0 > \mathbf{0}$ and $\mathbf{s}_{iI}^0 > \mathbf{0}$.

The values of $\mathbf{u}_i^*$, $\mathbf{x}_1^*$, $\boldsymbol{\lambda}_{iE}^*$, and $\boldsymbol{\lambda}_{iI}^*$ calculated in Step 5 of the algorithm are used to update the nominal values of the controls and Lagrange multipliers as follows:

$$\hat{\mathbf{u}}_i \leftarrow \hat{\mathbf{u}}_i + \alpha^*\mathbf{u}_i^*, \tag{4.279}$$

$$\hat{\mathbf{x}}_1 \leftarrow \hat{\mathbf{x}}_1 + \alpha^*\mathbf{x}_1^*, \tag{4.280}$$

$$\hat{\boldsymbol{\lambda}}_{iE} \leftarrow (1 - \alpha^*)\hat{\boldsymbol{\lambda}}_{iE} + \alpha^*\boldsymbol{\lambda}_{iE}^*, \tag{4.281}$$

$$\hat{\boldsymbol{\lambda}}_{iI} \leftarrow (1 - \alpha^*)\hat{\boldsymbol{\lambda}}_{iI} + \alpha^*\boldsymbol{\lambda}_{iI}^*. \tag{4.282}$$

SQP algorithms typically select the step length parameter $\alpha^* \in (0, 1]$ to obtain a decrease in a specified merit or penalty function [62]. In all of the example problems, α^* is set equal to unity for simplicity.

The process of formulating and solving the SQP subproblems and updating the nominal values of the controls and Lagrange multipliers is repeated until convergence to the solution of P1 is obtained. The reader is referred to Fletcher [63] for specific details on the convergence of SQP.

Recall in (4.201) that the matrix $\mathbf{Q}$ is assumed positive semidefinite. For certain problems, it may occur that the quadratic subproblems do not have well-defined solutions during early SQP iterations. Several options are available to deal with such problems, but it was found that a simple continuation method sufficed for the final example problem. The basic idea is to solve one or more related problems, each with well-posed quadratic subproblems. By varying a continuation parameter, the solutions of the related problems converge to the solution of P1. More details of this process are provided in the final example problem (Example 4.9).

4.4.6 Example Problems

Four example problems are included to demonstrate the method presented in this chapter. The first example (Example 4.6) shows the equivalence of the method with a less efficient standard approach. The second example (Example 4.7) is taken from the literature and solved using two different approaches. The remaining two examples (Examples 4.8 and 4.9) involve a rest-to-rest maneuver of a flexible structure and a constrained brachistochrone problem with viscous friction. In all the examples, the scalar parameters are $\delta = 10^{-6}$, $\varepsilon = 10^{-2}$, and $c_{max} = 10^4$.

Example 4.6. *Here,* $N = 5$, $n = 1$, $m = 1$, *and*

$$\Gamma = \sum_{i=1}^{N-1} u_i^2,$$

$$x_{i+1} = x_i - 2u_i + 0.1x_i^2,$$

$$c_{1E} = x_1 - 1,$$

$$c_{NE} = \sin(x_N) - 0.1,$$

$$\mathbf{c}_{3I} = \begin{bmatrix} x_3 + \sin(x_3) \\ -x_3 - 1 \end{bmatrix}.$$

Table 4.5. *Comparison of results for Example* 4.6.

	AS Direction		PCC Direction	
Variable	DPIP	Maple	DPIP	Maple
u_1	0.362038893	0.362038893	0.495872371	0.495872371
u_2	0.297099200	0.297099200	0.408219902	0.408219902
u_3	0.342882735	0.342882735	0.139130362	0.139130362
u_4	0.273592193	0.273592193	0.110021456	0.110021456
λ_{NE}	0.912541602	0.912541602	0.227850406	0.227850406
λ_{3I^1}	−0.117621239	−0.117621239	0.133978825	0.133978825
λ_{3I^2}	−0.006195758	−0.006195758	0.000446434	0.000446434
s_{3I^1}	0.048810619	0.048810619	0.827532125	0.827532125
s_{3I^2}	−1.471780678	−1.471780678	−2.018434432	−2.018434432

In this example, the nominal values of the controls and Lagrange multipliers are all set equal to 0.2 and $\hat{x}_1 = 1$. The affine scaling (AS) and combined PCC directions calculated in Steps 2 and 3 of the DP/interior point (DPIP) algorithm are shown in Table 4.5 for the first iteration. Also shown in the table are the same directions calculated using a conventional approach in which (4.207) is formulated and solved using the symbolic math software MAPLE. The agreement between the two sets of results is evident.

Example 4.7 (see Wright [59])). *Here,* $n = 4$, $m = 1$, *and*

$$\Gamma = \mathbf{x}_N^T \mathbf{x}_N,$$

$$t_i = 4.2\frac{(i-1)}{(N-1)},$$

$$\dot{\mathbf{x}} = \begin{bmatrix} -0.5 & 5 & 0 & 0 \\ -5 & -0.5 & 0 & 0 \\ 0 & 0 & -0.6 & 10 \\ 0 & 0 & -10 & -0.6 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} u,$$

$$\mathbf{c}_{1E} = \mathbf{x}_1 - [10 \quad 10 \quad 10 \quad 10]^T,$$

$$\mathbf{c}_{NI} = \mathbf{x}_N - [1 \quad 1 \quad 1 \quad 1]^T,$$

$$\mathbf{c}_{iI} = [(u_i - 1) \quad (-u_i - 1)]^T$$

for $i = 1, \ldots, N-1$.

The differential equations for this problem are integrated to obtain

$$\mathbf{x}_{i+1} = \begin{bmatrix} \mathbf{A}(0.5, 5, \Delta t) & \mathbf{0} \\ \mathbf{0} & \mathbf{A}(0.6, 10, \Delta t) \end{bmatrix} \mathbf{x}_i + \begin{bmatrix} \mathbf{B}(0.5, 5, \Delta t) \\ \mathbf{B}(0.6, 10, \Delta t) \end{bmatrix} u_i,$$

where $\Delta t = 4.2/(N-1)$ and

$$\mathbf{A}(a, b, \Delta t) = \exp(-a\Delta t) \begin{bmatrix} \cos b\Delta t & \sin b\Delta t \\ -\sin b\Delta t & \cos b\Delta t \end{bmatrix},$$

$$\mathbf{B}(a, b, \Delta t) = \frac{1}{a^2 + b^2} \begin{bmatrix} b - \exp(-a\Delta t)(a \sin b\Delta t + b \cos b\Delta t) \\ a + \exp(-a\Delta t)(b \sin b\Delta t - a \cos b\Delta t) \end{bmatrix}.$$

The solution to this problem has a bang-bang control with eight switching times. Results for this example were obtained by setting all the nominal values of the controls and Lagrange multipliers equal to zero. The number of PCC iterations and the calculated values of Γ are shown in Table 4.6 for different values of N. Only a single SQP iteration is required because the state transition equation is linear and Γ is a quadratic function. Control time histories for $N = 201$ are shown in Fig. 4.8 after the second, fourth, and sixth PCC iterations.

The exact solution to this problem can be obtained by solving a related equality-constrained problem. The basic idea is to constrain the controls to their previously calculated bounding values while allowing the time step to be different for each of the time intervals in which the control is held constant. This can be accomplished by augmenting the state

Table 4.6. *Results for Example* 4.7 *using linear state transition equations.*

N	PCC iterations	Γ
101	8	1.01266
201	9	1.00630
401	10	1.00391
801	9	1.00355
1601	12	1.00351
3201	13	1.00349

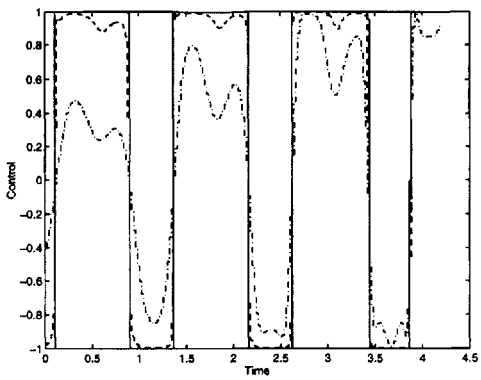


Figure 4.8. *Control time histories for Example* 4.7 *with* $N = 201$. *Results shown after two (dash-dot line), four (dashed line), and six (solid line) PCC iterations. (From Dohrmann and Robinett* [31]. *Reprinted by permission of Kluwer Academic/Plenum Publishers.)*

vector, $\mathbf{x}_i$, with the discrete time, t_i, replacing the control, u_i, with the time step, $t_{i+1} - t_i$, and setting $N = 9$. The definition of the related problem is

$$\Gamma = \mathbf{x}_N^T \mathbf{x}_N - (x_N^5)^2,$$

$$\mathbf{c}_{1E} = \mathbf{x}_1 - [10 \quad 10 \quad 10 \quad 10 \quad 0]^T,$$

$$c_{NE} = x_N^5 - 4.2,$$

$$\mathbf{x}_{i+1} = \begin{bmatrix} \mathbf{A}(0.5, 5, u_i) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}(0.6, 10, u_i) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 \end{bmatrix} \mathbf{x}_i + \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ 1 \end{bmatrix} u_i + \begin{bmatrix} \mathbf{B}(0.5, 5, u_i) \\ \mathbf{B}(0.6, 10, u_i) \\ \mathbf{0} \end{bmatrix} \hat{\mathbf{d}}_i,$$

where $\{\hat{\mathbf{d}}_i\} = \{-1, 1, -1, 1, -1, 1, -1, 1\}$. Because the state transition equation is nonlinear, more than a single SQP iteration is required to solve the problem.

Calculated values of the switching times and Γ are shown in Table 4.7. These results were obtained with three SQP iterations. The nominal values of the controls were calculated using the switching times for the original problem with $N = 201$. Also shown in Table 4.7 are the results of Longsdon [64] as reported by Wright [59]. The close agreement of the two sets of results is evident.

Table 4.7. *Comparison of results for Example* 4.7.

Nonlinear approach, $N = 9, \Gamma = 1.003475$		Longsdon [64], $\Gamma = 1.00347$	
Switching time	u	Switching time	u
0.00000	−1.0	0.00000	−1.0
0.11098	1.0	0.11198	1.0
0.89916	−1.0	0.89979	−1.0
1.36573	1.0	1.36428	1.0
2.16827	−1.0	2.16960	−1.0
2.62051	1.0	2.62063	1.0
3.43653	−1.0	3.43619	−1.0
3.87534	1.0	3.87530	1.0

Example 4.8. *This example involves a rest-to-rest maneuver of a flexible structure modeled by two point masses (m_1 and m_2) connected by a spring (k).*

The first mass is subjected to a force whose time derivative is denoted by u. The goal is to translate the structure, initially at rest, a specified distance d, bringing it back to rest at the final time T. Here,

$$\Gamma = \sum_{i=1}^{N-1} (u_i)^2,$$

$$t_i = (i-1)\frac{T}{(N-1)},$$

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\frac{k}{m_1} & \frac{k}{m_1} & 0 & 0 & \frac{1}{m_1} \\ \frac{k}{m_2} & -\frac{k}{m_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u,$$

$$\mathbf{c}_{1E} = \mathbf{x}_1 - [0 \quad 0 \quad 0 \quad 0 \quad 0]^T,$$

$$\mathbf{c}_{NE} = \mathbf{x}_N - [d \quad d \quad 0 \quad 0 \quad 0]^T,$$

$$\mathbf{c}_{iI} = \left[(x_i^5 - f_{max}) \quad (-x_i^5 - f_{max})\right]^T$$

for $i = 1, \ldots, N$. The equality constraints require the structure to be quiescent with an applied force of zero at the initial and final times. In addition, the displacement of the two masses at the final time is required to equal d. The inequality constraints limit the applied force magnitude to maximum values of f_{max}. The differential equations for this problem are integrated exactly to obtain their discrete-time counterpart (see (4.175)), but these results are not reported.

In this example, $m_1 = 1$, $m_2 = 1$, $k = 3$, $T = 6$, $f_{max} = 1$, and $N = 201$. The force time history for $d = 3.5$ is shown in Fig. 4.9 (left plot) along with the corresponding displacement and velocity time histories of the two masses (right plot). These results were obtained with one SQP and eleven PCC iterations.

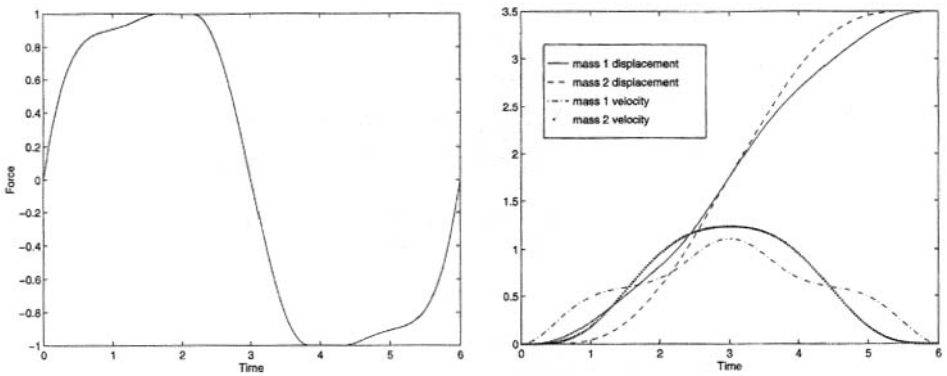


Figure 4.9. *Control, displacement, and velocity time histories for Example* 4.8. *(From Dohrmann and Robinett* [31]. *Reprinted by permission of Kluwer Academic/Plenum Publishers.)*

Example 4.9 (see Dohrmann and Robinett [29] for a similar problem without inequality constraints). *Here,* $N = 201$, $n = 4$, $m = 1$, *and*

$$\Gamma = x_1^4,$$
$$t_i = (i-1)x_1^4,$$
$$x_{i+1}^1 = x_i^1 + \cos u_i \frac{\left(x_i^4 \sin u_i + ab\right)}{c},$$
$$x_{i+1}^2 = x_i^2 + \sin u_i \frac{\left(x_i^4 \sin u_i + ab\right)}{c},$$
$$x_{i+1}^3 = \frac{\sin u_i}{c} + a \exp(-cx_i^4),$$
$$x_{i+1}^4 = x_i^4,$$
$$\mathbf{c}_{1E} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{x}_1,$$
$$\mathbf{c}_{NE} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{x}_N - \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$
$$c_{iI} = x_i^2 - a_1 x_i^1 - a_2$$

for $i = 1, \ldots, N$, *where* c *is a viscous damping coefficient and*

$$a = x_i^3 - \frac{\sin u_i}{c},$$
$$b = 1 - \exp(-cx_i^4).$$

The first three elements of the state vector are the coordinates and speed of a particle in a gravitational field, while the fourth element is the time step for the problem. The control

is the angle from the x coordinate axis to the tangent of the particle's path. The goal is to determine the path that minimizes the traversal time for a particle initially at rest at the origin to the position with coordinates (x_f, y_f).

The nominal values of the controls and time step are set equal to those for the straight line path from $(0, 0)$ to (x_f, y_f). That is,

$$\hat{u}_i = \arctan\left(\frac{y_f}{x_f}\right), \quad i = 1, \ldots, N-1,$$

and $\hat{x}_1^4 = T/(N-1)$, where T is the solution to the nonlinear equation

$$y_f\,(cT + \exp(-cT) - 1) - c^2\left(x_f^2 + y_f^2\right) = 0.$$

If the nominal values of the Lagrange multipliers associated with $\mathbf{c}_{NE}$ are set equal to zero, then the first SQP subproblem does not have a well-defined solution. To address this problem, consider a related problem with Γ replaced by

$$\Gamma = \alpha x_1^4 + (1-\alpha)\sum_{i=1}^{N-1}\left(u_i - \arctan\left(\frac{y_f}{x_f}\right)\right)^2,$$

where $\alpha \in [0, 1]$ is a continuation parameter. Notice that the solutions for $\alpha = 0$ and $\alpha = 1$ correspond to the straight line and minimum-time paths, respectively.

The controls and Lagrange multipliers for the solution to the problem with $\alpha = 0.5$ were used as nominal values for the problem with $\alpha = 1$. Minimum-time paths for $x_f = 1$, $y_f = 1$, $a_1 = 1$, $a_2 = 0.1$, and $N = 101$ are shown in Fig. 4.10 for three different values of c. A convergence summary for the case of $\alpha = 1$ and $c = 1$ is provided in Table 4.8. Note that if the parameter ε (see (4.278)) is reduced to a smaller value after the first SQP iteration, then the number of PCC iterations for subsequent SQP iterations is reduced. The numbers of PCC iterations required with $\varepsilon = 10^{-6}$ for SQP iterations 2 through 6 are shown in parentheses in the final column of Table 4.8. These results show that an adaptive choice for the parameter ε could enhance the performance of the algorithm.

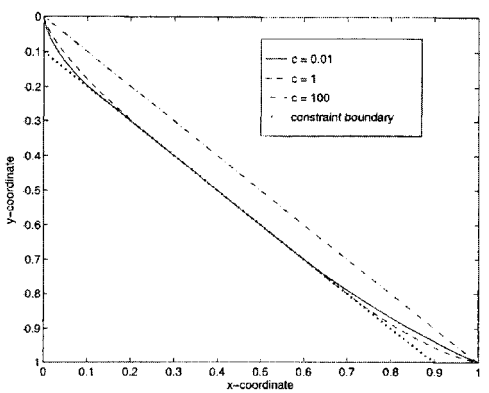


Figure 4.10. *Minimum-time paths for Example* 4.9. *(From Dohrmann and Robinett* [31]. *Reprinted by permission of Kluwer Academic/Plenum Publishers.)*

Table 4.8. *Convergence summary for Example* 4.9.

SQP iteration	$\sqrt{\sum_{i=1}^{N-1} \bar{u}_i^2}$	PCC iterations
1	3.6608	8
2	2.5688	8(5)
3	1.1650	7(4)
4	3.8820e−2	7(3)
5	7.0132e−4	7(1)
6	4.2931e−8	7(1)

4.5 Summary

In this chapter, DP attributes were developed for the direct enforcement of equality and inequality constraints on the states and controls. Numerous examples were provided to demonstrate the application of the DP algorithms, and several constructive aerospace problems were reviewed.

A DP approach was presented to help correct for launch offsets of precision-guided munitions, operating with low-cost satellite navigation updates against defended targets. This approach provided guidance commands up to the time of apogee and was shown to be unaffected by the jamming of external navigation updates subsequent to this time. A guidance problem with an unspecified final time was successfully transformed to a standard form amenable to the DP method. The feedback gains computed using the guidance algorithm were shown to be robust for a range of small-angle launch offsets. The DP approach required only a single pass to compute gains and could be done quickly to meet the needs of a battle environment.

Next, two DP algorithms were presented for solving equality-constrained, discrete-time optimal control problems. The algorithms are applicable to problems with either specified or free final times. Constraint enforcement was accomplished using a Lagrange multiplier approach, whereby inconsistent or redundant constraints were easily identified. Solutions were obtained with the DP algorithms by solving a series of constrained quadratic programming subproblems. The distinguishing feature of the DP algorithms was the efficiency with which the subproblems were solved.

In addition, updates to controls, Lagrange multipliers, and the final time for a problem were found by setting the gradients of the quadratic approximation to the Lagrangian equal to zero. Consequently, the DP algorithms yielded results that were identical to a more conventional SQP step and were quadratically convergent under the same conditions. The number of mathematical operations required for each iteration is proportional to the number of discrete times N. This is in contrast to a conventional SQP method, in which this number is proportional to N^3. In the absence of constraints, the DP algorithm for specified final time problems simplified to the DP approach of Dunn and Bertsekas [28].

Next, two example problems demonstrated the equivalence of the DP algorithms' results with those from a conventional SQP method. Quadratic convergence of the iterations was also shown. The proficiency of the DP algorithms was further demonstrated with the successful solution of three diversified application problems. Results for a satellite dynamics problem compared favorably with those obtained using a different approach. A pursuit problem with a nonholonomic path constraint and an endpoint constraint was also solved.

In the second example, a set of brachistochrone problems with a viscous friction component was solved. This example demonstrated the application of the DP algorithm for *free* final time problems.

Finally, an efficient DP algorithm was presented that solves inequality constrained discrete-time optimal control problems. The DP algorithm solved the quadratic subproblems of SQP using a combined DPIP method. The number of mathematical operations required for each iteration is proportional to the number of discrete times N. The DP algorithm is applicable to problems with either *fixed* or *free* final times. It was applied successfully to a minimum-time problem. In contrast to other interior point algorithms, the present approach avoids the introduction of adjoint variables associated with the state transition equation. Consequently, only the controls and initial state need to be considered as unknowns in the problem.

In Chapter 5, a series of case studies are reviewed that best illustrate how to apply the DPIP algorithm. Solutions to both robotic and aerospace problems are demonstrated.

This page intentionally left blank

Chapter 5
Applied Case Studies

5.1 Introduction

This chapter presents a series of significant case studies that illustrate the procedures for applying the dynamic programming/interior point (DPIP) method algorithm. The theoretical foundation and algorithm development were presented in the previous two chapters. Each case study has been developed following a specific format that includes an introduction, model development, optimization design(s), implementation issues, numerical simulation results, and discussion (summary/conclusions). These case studies draw on popular themes and applications in the robotics and aerospace disciplines. Where applicable, the DPIP algorithm is compared to more conventional numerical optimization techniques such as the recursive quadratic programming (RQP) algorithm. The goal is to focus on a variety of applications that may be adapted easily to the reader's own interests and problems. In addition, the case studies discuss algorithm implementation issues, and MATLAB script files are provided in Appendix B for the reader to be able to reproduce the numerical results where applicable. This chapter includes six case studies.

Section 5.2: Case Study 1: Rotary Jib Crane. This case study demonstrates how to generate open-loop input trajectories for a rotary jib crane. Both DPIP and RQP optimization designs are performed. Numerical simulations confirm the designs. Further investigation includes the effects of nonlinear dynamics on achieving accurate results particularly on the hardware implementation.

Section 5.3: Case Study 2: Slewing Flexible Link. This case study demonstrates how to generate open-loop input trajectories for a slewing flexible link. Several DPIP designs are performed. Both ideal open-loop torque profiles and closed-loop profiles are investigated. For the closed-loop profile, an RQP design that was validated on experimental hardware [66] is compared with an equivalent DPIP design.

Section 5.4: Case Study 3: Flexible Planar Arm. This case study demonstrates how to generate feedforward trajectories for a flexible planar robot arm. DPIP optimization is employed to investigate minimum effort, minimum effort with bounds, minimum-time, and minimum torque-rate solutions. Numerical simulations confirm the designs.

Section 5.5: Case Study 4: Minimum-Time Maneuvers of Rigid Systems. This case study demonstrates how to determine time-optimal maneuvers for a rigid symmetric spacecraft. A simple time-optimal double integrator problem is studied first, which is also representative of an eigenaxis rotation maneuver. Numerical simulations are performed to confirm the solution.

Section 5.6: Case Study 5: Maximum-Radius Orbit Transfer. This case study demonstrates how to transfer a rocket vehicle from a given initial circular orbit to the largest possible circular orbit. The transfer orbit length of time is fixed, along with the rocket thrust, while trying to find the thrust direction history for the maximum-radius orbit transfer. Numerical simulations help to determine the time histories.

Section 5.7: Case Study 6: PUMA 560 Industrial Manipulator. This case study demonstrates how to generate feedforward trajectories for the gross positioning of an industrial manipulator. DPIP optimization is employed to generate minimum effort with both torque- and velocity-bounded solutions. Numerical simulations are given for the DPIP design time histories.

The main goal of this chapter is to provide explicit steps in a case study format that demonstrate how to implement and use DPIP methods to solve for optimized trajectory solutions that are applied to dynamical systems. The remainder of this chapter presents each case study in more detail.

5.2 Case Study 1: Rotary Jib Crane

5.2.1 Introduction

In the construction and transportation industries, cranes are a popular piece of equipment with multiple degrees of freedom (DOFs) that may include variable load-line lengths, jib (usually cart/trolley) lengths, and boom angles [65]. Typically, the payload pendulum oscillations are low frequency. To prevent the excitation of spherical pendulum modes, the point-to-point payload maneuvers are performed slowly, which contributes to high construction and transportation costs. This case study details a general method for applying command shaping to various multiple DOF cranes such that the payload moves to a specified point without residual oscillations. A dynamic programming (DP) method is used for general command shaping for optimal maneuvers [29, 31]. Computationally, the DP approach requires order-N calculations to arrive at a solution, where N is the number of discretizations of the input commands. This feature is exploited for the crane command shaping problem, allowing for rapid calculation of command histories. These results are compared to near-optimal solutions, where the commands are formulated based on an acceleration pulse-coast-pulse profile. The discrete pulse shapes are required due to the specific rotary jib crane experimental hardware setup. The pulse on-time, coast off-time, and acceleration amplitude are chosen as the solution to a parameter optimization problem that employs an RQP technique. Numerical simulation results and experimental verification for a constant length rotary jib crane are reviewed for the RQP optimization design procedures [66]. Numerical simulation results are reviewed for the DP optimization design procedures.

The objective of this case study is to design a feedforward open-loop control trajectory for a rotary jib crane that uses numerical optimization techniques. The following steps are included.

Step 1: Derive the equations of motion for a fixed-length rotary jib crane.

Step 2: Develop a numerical simulation of the rotary jib crane.

Step 3: Solve the trajectory optimization problem using RQP design.

Step 4: Solve the trajectory optimization problem using DP design.

Step 5: Compare the numerical simulations for the optimization approaches.

Step 6: Verify the optimization design for real rotary jib crane hardware.

Step 7: Summarize the case study findings.

5.2.2 Model Development

The experimental hardware test setup is shown in Fig. 5.1 (left picture). The rotary jib crane parameters are defined in Table 5.1. The system consists of a hardware control computer, a stepper motor used to rotate the jib crane, and a swinging payload or pendulum. Various

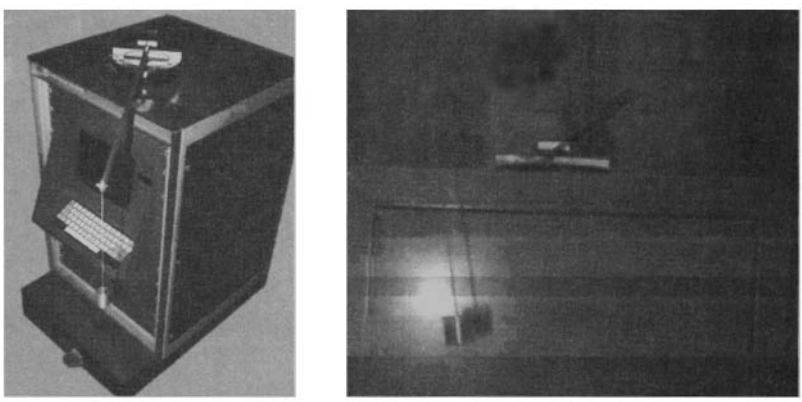


Figure 5.1. *Rotary jib crane hardware setup and corresponding in-transit* 90° *slewing motion (courtesy of Sandia National Laboratories).*

Table 5.1. *Rotary jib crane physical parameters*

Description	Symbol	Value	Unit
Length of boom	l	0.659	m
Gravity	g	9.81	m/s^2
Length of pendulum cable	d	0.251	m
Pendulum mass	m	0.05	kg

trajectories can be implemented via the keyboard terminal. To test a typical trajectory, the values for the overall trajectory position, velocity, and acceleration are entered and executed. This causes the rotary jib crane to move from the initial position to some final position. In Fig. 5.1 (right picture), the rotary jib crane is viewed in motion, starting a 90° swing going from right to left. During the maneuver, the pendulum is expected to oscillate. At the completion of the maneuver, the goal is to minimize the residual oscillations of the pendulum. The rotary jib crane is modeled with a single prescribed hub input γ and two dynamic (θ, ϕ) DOFs. The geometry, kinematics, and dynamics are derived next. In addition, the dynamic equations of motion are linearized and nondimensionalized.

The geometry and the kinematic frames are shown in Fig. 5.2. In the initial configuration, all the prime axes are colocated at the end of the boom. The pendulum is kinematically described with two Euler angles, starting at the end of the boom, and rotating about the $\mathbf{y}'$ axis, an amount $-\theta$ in the radial direction (relative to the boom), then rotating about the positive $\mathbf{x}'$ axis an amount ϕ in the tangential direction. In addition, to compensate inertially, since the boom is rotating about the $\mathbf{z}^A$ axis with a specified input γ, Chasle's theorem is used to compensate in the body axis coordinate system for this movement. Superscripts 0, A, 1, 2, 3 describe what coordinate system one is working with, starting with the inertial frame and going to the tip frame, defined as

$$\begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix}^0, \quad \begin{Bmatrix} x \\ y \\ z \end{Bmatrix}^A, \quad \begin{Bmatrix} x \\ y \\ z \end{Bmatrix}^1, \quad \begin{Bmatrix} x' \\ y' \\ z' \end{Bmatrix}^2, \quad \begin{Bmatrix} x'' \\ y'' \\ z'' \end{Bmatrix}^3.$$

The Euler transformation matrices between coordinate systems are

$$^1\mathbf{R}_2 = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix},$$

$$^2\mathbf{R}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix},$$

$$^1\mathbf{R}_3 = {}^1\mathbf{R}_2 \quad {}^2\mathbf{R}_3 = \begin{bmatrix} c\theta & s\theta s\phi & s\theta c\phi \\ 0 & c\phi & -s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix},$$

where shorthand notation is being used for $\sin = s$ and $\cos = c$.

The pendulum center of mass (COM) is expressed as a vector in frame 3, but we need to express it in frame 1:

$$\mathbf{r}_3^3 = \begin{Bmatrix} 0 \\ 0 \\ -d \end{Bmatrix}^3,$$

$$\mathbf{r}_3^1 = {}^1\mathbf{R}_3\mathbf{r}_3^3 = \begin{bmatrix} c\theta & s\theta s\phi & s\theta c\phi \\ 0 & c\phi & -s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \begin{Bmatrix} 0 \\ 0 \\ -d \end{Bmatrix}^3 = \begin{Bmatrix} -ds\theta c\phi \\ ds\phi \\ -dc\theta c\phi \end{Bmatrix}^1.$$

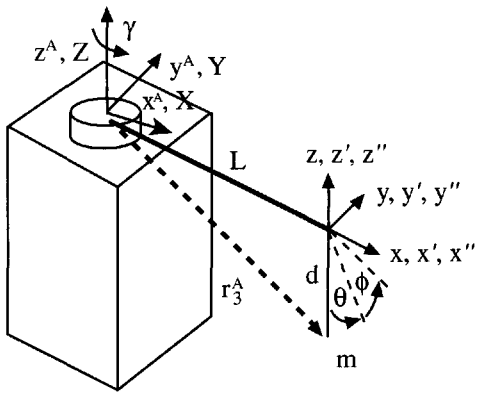


Figure 5.2. *Mathematical schematic of rotary jib crane kinematics.*

The orientation between frames A and 1 requires only the identity matrix. The position vector is specified as the length l from the hub to the tip of the boom in the x direction. To express $\mathbf{r}$ in the A frame, the following operation is performed:

$$\mathbf{r}_3^A = \mathbf{l}^A + {}^A\mathbf{R}_1 \mathbf{r}_3^1 = \begin{Bmatrix} l - s\theta c\phi d \\ s\phi d \\ -c\theta c\phi d \end{Bmatrix}^A.$$

The body frame rotation velocity $\boldsymbol{\Omega}$ is defined as

$$\boldsymbol{\Omega} = \begin{Bmatrix} 0 \\ 0 \\ \dot{\gamma} \end{Bmatrix}.$$

Chasle's theorem [57] is defined as

$$\left.\frac{d\mathbf{r}_3^A}{dt}\right|_{\text{nonrotating}} = \left.\frac{d\mathbf{r}_3^A}{dt}\right|_{\text{rotating}} + \boldsymbol{\Omega} \times \mathbf{r}_3^A.$$

Application of the theorem yields

$$\left.\frac{d\mathbf{r}_3^A}{dt}\right|_{\text{nonrotating}} = \begin{Bmatrix} -c\theta c\phi d\dot{\theta} + s\theta s\phi d\dot{\phi} - s\phi d\dot{\gamma} \\ c\phi d\dot{\phi} + l\dot{\gamma} - s\theta c\phi d\dot{\gamma} \\ s\theta c\phi d\dot{\theta} + c\theta s\phi d\dot{\phi} \end{Bmatrix}.$$

From this point, the *nonrotating* and *rotating* subscripts are dropped.

Define the kinetic energy as

$$T = \frac{1}{2} m \left(\frac{d\mathbf{r}_3^A}{dt}\right) \cdot \left(\frac{d\mathbf{r}_3^A}{dt}\right).$$

Next, perform the substitutions to yield

$$T = \frac{1}{2} m[(-c\theta c\phi d\dot{\theta} + s\theta s\phi d\dot{\phi} - s\phi d\dot{\gamma})^2 + (c\phi d\dot{\phi} + l\dot{\gamma} - s\theta c\phi d\dot{\gamma})^2 + (s\theta c\phi d\dot{\theta} + c\theta s\phi d\dot{\phi})^2].$$

Squaring and simplifying gives

$$T = \frac{1}{2}mc^2\phi d^2\dot{\theta}^2 + \frac{1}{2}md^2\dot{\phi}^2 + \frac{1}{2}m\left[(l - s\theta c\phi d)^2 + s^2\phi d^2\right]\dot{\gamma}^2$$
$$+ m\left[c\phi dl - s\theta d^2\right]\dot{\phi}\dot{\gamma} + mc\theta c\phi s\phi d^2\dot{\theta}\dot{\gamma}.$$

The potential energy is defined as

$$V = mgd(1 - c\theta c\phi).$$

Compute the Lagrangian as

$$L = T - V = \frac{1}{2}mc^2\phi d^2\dot{\theta}^2 + \frac{1}{2}md^2\dot{\phi}^2 + \frac{1}{2}m\left[(l - s\theta c\phi d)^2 + s^2\phi d^2\right]\dot{\gamma}^2$$
$$+ m\left[c\phi dl - s\theta d^2\right]\dot{\phi}\dot{\gamma} + mc\theta c\phi s\phi d^2\dot{\theta}\dot{\gamma} - mgd(1 - c\theta c\phi).$$

Now, apply the Lagrangian for the θ-DOF dynamics as

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = 0,$$

which yields

$$\frac{\partial L}{\partial \theta} = -mldc\theta c\phi\dot{\gamma}^2 + ms\theta c\theta c^2\phi d^2\dot{\gamma}^2 - mc\theta d^2\dot{\phi}\dot{\gamma} - ms\theta c\phi s\phi d^2\dot{\theta}\dot{\gamma} - mgds\theta c\phi$$
$$\frac{\partial L}{\partial \dot{\theta}} = mc^2\phi d^2\dot{\theta} + mc\theta c\phi s\phi d^2\dot{\gamma},$$
$$\left(\frac{d}{dt}\right)\frac{\partial L}{\partial \dot{\theta}} = -2mc\phi s\phi d^2\dot{\theta}\dot{\phi} + mc^2\phi d^2\ddot{\theta} - ms\theta c\phi s\phi d^2\dot{\gamma}\dot{\theta} - mc\theta s^2\phi d^2\dot{\gamma}\dot{\phi}$$
$$+ mc\theta c\phi^2 d^2\dot{\gamma}\dot{\phi} + mc\theta c\phi s\phi d^2\ddot{\gamma}.$$

Assembling all the terms yields

$$mc^2\phi d^2\ddot{\theta} - 2mc\phi s\phi d^2\dot{\theta}\dot{\phi} - mc\theta s^2\phi d^2\dot{\gamma}\dot{\phi} + mc\theta c^2\phi d^2\dot{\gamma}\dot{\phi} + mc\theta c\phi s\phi d^2\ddot{\gamma} + mldc\theta c\phi\dot{\gamma}^2$$
$$-ms\theta c\theta c^2\phi d^2\dot{\gamma}^2 + mc\theta d^2\dot{\phi}\dot{\gamma} + mgds\theta c\phi = 0.$$

Next, apply the Lagrangian for the ϕ-DOF dynamics as

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right) - \frac{\partial L}{\partial \phi} = 0,$$

which yields

$$\frac{\partial L}{\partial \phi} = -mc\phi s\phi d^2\dot{\theta}^2 + mlds\theta s\phi\dot{\gamma}^2 - ms^2\theta c\phi s\phi d^2\dot{\gamma}^2 + ms\phi c\phi d^2\dot{\gamma}^2$$
$$- ms\phi dl\dot{\phi}\dot{\gamma} - mc\theta s^2\phi d^2\dot{\theta}\dot{\gamma} + mc\theta c^2\phi d^2\dot{\theta}\dot{\gamma} - mgdc\theta s\phi,$$
$$\frac{\partial L}{\partial \dot{\phi}} = md^2\dot{\phi} + m[c\phi dl - s\theta d^2]\dot{\gamma},$$
$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}} = md^2\ddot{\phi} - ms\phi dl\dot{\gamma}\dot{\phi} + mc\phi dl\ddot{\gamma} - mc\theta d^2\dot{\gamma}\dot{\theta} - ms\theta d^2\ddot{\gamma}.$$

Assembling the terms yields

$$md^2\ddot{\phi} + mc\phi dl\ddot{\gamma} - mc\theta d^2\dot{\gamma}\dot{\theta} - ms\theta d^2\ddot{\gamma} + mc\phi s\phi d^2\dot{\theta}^2 - mlds\theta s\phi\dot{\gamma}^2 + md^2s^2\theta c\phi s\phi\dot{\gamma}^2$$
$$- md^2s\phi c\phi\dot{\gamma}^2 + md^2c\theta s^2\phi\dot{\theta}\dot{\gamma} - md^2c\theta c^2\phi\dot{\theta}\dot{\gamma} + mgdc\theta s\phi = 0.$$

Next, linearize the above equations with the approximations $\sin\theta \approx \theta$, $\cos\theta \approx 1$, and $\sin\phi \approx \phi$, $\cos\phi \approx 1$. The results are as follows.

The θ-DOF equation of motion is

$$md^2\ddot{\theta} - 2md^2\phi\dot{\theta}\dot{\phi} - md^2\phi^2\dot{\gamma}\dot{\phi} + md^2\dot{\gamma}\dot{\phi} + md^2\phi\ddot{\gamma}$$
$$+ mld\dot{\gamma}^2 - md^2\dot{\gamma}^2\theta + md^2\dot{\gamma}\dot{\phi} + mgd\theta = 0$$

and the ϕ-DOF equation of motion is

$$md^2\ddot{\phi} + mdl\ddot{\gamma} - md^2\dot{\gamma}\dot{\theta} - md^2\ddot{\gamma}\theta + md^2\phi\dot{\theta}^2 - mld\theta\phi\dot{\gamma}^2 + md^2\theta^2\phi\dot{\gamma}^2$$
$$- md^2\dot{\gamma}^2\phi + md^2\phi^2\dot{\theta}\dot{\gamma} - md^2\dot{\gamma}\dot{\theta} + mgd\phi = 0.$$

All nonlinear terms, such as $\phi\dot{\theta}\dot{\phi}$, $\phi^2\dot{\phi}$, $\phi\dot{\theta}^2$, $\phi\theta$, $\theta^2\phi$, and $\phi^2\dot{\theta}$, are assumed small and therefore neglected.

Define the stiffness diagonal term as

$$k_{diag} = mgd - md^2\dot{\gamma}^2.$$

Then, the final linearized equations are presented in matrix form as

$$\begin{bmatrix} md^2 & 0 \\ 0 & md^2 \end{bmatrix}\begin{Bmatrix} \ddot{\theta} \\ \ddot{\phi} \end{Bmatrix} + \begin{bmatrix} 0 & 2md^2\dot{\gamma} \\ -2md^2\dot{\gamma} & 0 \end{bmatrix}\begin{Bmatrix} \dot{\theta} \\ \dot{\phi} \end{Bmatrix}$$
$$+ \begin{bmatrix} k_{diag} & md^2\ddot{\gamma} \\ -md^2\ddot{\gamma} & k_{diag} \end{bmatrix}\begin{Bmatrix} \theta \\ \phi \end{Bmatrix} = \begin{Bmatrix} -mld\dot{\gamma}^2 \\ -mdl\ddot{\gamma} \end{Bmatrix}.$$

Nondimensionalizing yields the final form fixed-length rotary jib crane dynamics as

$$\begin{Bmatrix} \ddot{\theta} \\ \ddot{\phi} \end{Bmatrix} + \begin{bmatrix} 0 & 2\dot{\gamma} \\ -2\dot{\gamma} & 0 \end{bmatrix}\begin{Bmatrix} \dot{\theta} \\ \dot{\phi} \end{Bmatrix} + \begin{bmatrix} (\frac{g}{d} - \dot{\gamma}^2) & \ddot{\gamma} \\ -\ddot{\gamma} & (\frac{g}{d} - \dot{\gamma}^2) \end{bmatrix}\begin{Bmatrix} \theta \\ \phi \end{Bmatrix} = \begin{Bmatrix} -\frac{l}{d}\dot{\gamma}^2 \\ -\frac{l}{d}\ddot{\gamma} \end{Bmatrix}. \tag{5.1}$$

Develop a numerical simulation of the rotary jib crane

The MathWorks MATLAB software [27] was used to develop the numerical simulation. A user-supplied driver script code called *jibsim.m* calls the MATLAB Runge–Kutta (RK) *ode*23 function, which calls the user-supplied equations of motion function *dyn*1.*m*. The current parameters being simulated are stored in a script file called *params.m*. This function is called during the simulation and the desired results are plotted using the script file *jibplt.m*. In addition, a nonlinear dynamic model was developed and defined in function *dyn*2.*m*. The script files for the simulation and equations of motion are provided in Section B.1 of Appendix B.

5.2.3 Optimization Designs

The goal of this section is to design an input-shaped command for the rotary jib crane to produce a rest-to-rest, residual–oscillation-free maneuver. Three separate specified final-angle ($\gamma_{desired}$) maneuvers are reviewed. For the last maneuver, the nonlinear equations of motion were required to achieve acceptable performance for the goal. Two methods for the input-shaped command generation are considered. Initially, an RQP optimization method parameterized a pulse-coast-pulse profile by determining the pulse on-time, coast off-time, and acceleration amplitude while minimizing a cost function that penalizes the residual payload oscillations. In contrast, a DP approach is considered where the hub motion is found by minimizing the prescribed hub acceleration $\ddot{\gamma}$ effort while simultaneously enforcing the payload oscillation constraints. The advantages of the new DP approach are reviewed. The problem formulations are discussed next.

Recursive quadratic programming optimization approach

The rotary jib crane constrained optimization problem was set up using the previously derived linearized equations of motion. The trajectory optimization problem was solved using the RQP algorithm `VF02AD` [67]. An acceleration pulse-coast-pulse profile is employed as the general input shaping function. The profile is characterized by three separate parameters. The first two parameters define the acceleration on-time, t_A (pulse), and off-time, t_C (coast). The last parameter defines the acceleration magnitude, A. In addition, there is the potential of seven end constraints, defined as

$$\theta(t_f) = \dot{\theta}(t_f) = \gamma(t_f) - \gamma_{desired} = \dot{\gamma}(t_f) = \phi(t_f) = \dot{\phi}(t_f) = t_f - t_d(= 2t_A + t_C) = 0. \tag{5.2}$$

For three parameters, one can have up to two constraints. For most of the results, the radial angle and angle rate have been selected as the constraints and the remaining end conditions were set up as penalty weighting functions within the performance index, but not necessarily all used. The performance index is defined as

$$J(\xi) = W_1\phi(t_f)^2 + W_2\dot{\phi}(t_f)^2 + W_3(t_f - t_d)^2 + W_4(\gamma(t_f) - \gamma_{desired})^2 + W_5\dot{\gamma}(t_f)^2, \tag{5.3}$$

with the two equality constraints set as

$$\Psi_1(\xi) = \theta(t_f) - 0 \tag{5.4}$$

and

$$\Psi_2(\xi) = \dot{\theta}(t_f) - 0. \tag{5.5}$$

Dynamic programming approach

For the DP solution, the performance index

$$J = \int_0^{t_f} \ddot{\gamma}^2 dt \tag{5.6}$$

was minimized, subject to (5.1), initially at rest, with the initial conditions given as

$$\phi(0) = \theta(0) = \dot{\phi}(0) = \dot{\theta}(0) = \gamma(0) = \dot{\gamma}(0) = 0 \tag{5.7}$$

and the equality constraints given by

$$\phi(t_f) = \theta(t_f) = \gamma(t_f) - \gamma_{desired} = \dot{\phi}(t_f) = \dot{\theta}(t_f) = \dot{\gamma}(t_f) = 0. \tag{5.8}$$

The final time was set to $t_f = 2.2$ sec. The numerical simulation results are reviewed in Section 5.2.5.

To increase the robustness to parameter variation, the third derivative of the hub angle (jerk) is used in the cost function, which results in smoothing the trajectories for position, velocity, and acceleration, and is defined as

$$J = \int_0^{t_f} \left(\dddot{\gamma}\right)^2 dt. \tag{5.9}$$

The initial acceleration $\ddot{\gamma}(0) = 0$ and equality constraint $\ddot{\gamma}(t_f) = 0$ are added to (5.7) and (5.8), respectively.

5.2.4 Implementation Issues

The DP numerical solution uses a group of MATLAB codes [68] written to numerically determine optimal controls for generally nonlinear dynamical systems with both equality and inequality constraints. The program implementation for the DPIP method supports the previously developed theoretical foundations (see Chapter 4).

The initial guess to start an iterative process can frequently be crucial. One attractive feature of the DPIP is that it appears to be less sensitive to the initial guess than other optimization techniques. However, for nonlinear systems, this can still be a problem. A simple five-step procedure was established to help the numerical solution converge to the optimal controls.

Step 1: Establish an appropriate resolution of discrete optimal control values for the problem.

Step 2: Start the initial guess u_i for the nonlinear problem with small arbitrary values. Several variations for the arbitrary values may be tried.

Step 3: If Step 2 is unsuccessful, start the initial guess u_i for the linear problem with small arbitrary values. Several variations for the arbitrary values may be tried.

Step 4: Upon successful solution of the linear problem, use the linear optimal control solution as the initial guess to the nonlinear problem.

Step 5: If unsuccessful in Step 4, then resort to homotopy methods [53, 54, 69] to gradually move from the linear solution to the nonlinear solution.

The first four steps of this procedure were successfully applied for several of the DP optimization runs.

The DPIP code requires four user-supplied problem specific functions as input [68]. These four functions define the nonlinear differential state equations, the cost function to be minimized, the state and control variable equality constraints, and the state and control variable inequality constraints. A driver routine is also included that helps define problem specifics and other iteration specific logistics. For the first DP optimization run (see Table 5.3), these functions are *exjcrane*1.*m*, the rotary jib crane driver script code; *esjcrane*1.*m*, the linear dynamics script code; *cojcrane*1.*m*, the cost function script code; *eqjcrane*1.*m*, the equality constraints script code; and *injcrane*1.*m*, the inequality constraints script code. The script files are listed in Section B.1 of Appendix B.

In all of the RQP optimization designs, the RQP optimization code `VF02AD` [67] is employed to determine the optimal trajectories. For further details of these runs, see [66].

5.2.5 Numerical Simulation Results

Three different optimization runs are performed for each design. For the RQP optimization runs, 125°, 90°, and large-angle 165° trajectories were determined. For the last run, the trajectories that investigate both a linear and a nonlinear dynamic model are reviewed. The summarized results, which include the final parameters for each run, are given in Table 5.2. The Table 5.2 entries for the RQP optimization results include the run number, the model used, the desired hub angle, the pulse amplitude, the pulse on-time, the pulse coast-time, and the final time. For the DPIP optimization runs, 90°, 165°, and 165° robustness variations were performed. Several iterations that required the linear models to start the nonlinear solutions are discussed. The summarized results are given in Table 5.3. The Table 5.3 entries for the DPIP optimization results include the run number, the model used, the desired hub angle, the initial u_i guess, the number of iterations performed, and the final convergence values. The number of discrete optimal control parameters was set to $N = 201$ for all DPIP runs. The RQP and DPIP solutions are compared for both the 90° and 165° cases. The numerical results are reviewed in detail next. In addition, the RQP optimization trajectories have been implemented and tested on the actual Sandia experimental hardware.

Table 5.2. *VF02AD RQP optimization results.*

Run #	1	2	3a	3b
Model	linear	linear	linear	nonlinear
$\gamma_{desired}$	125°	90°	165°	165°
A	1.276200	1.864744	2.900	2.704710
t_A	0.906770	0.826938	0.655	0.745183
t_C	0.446654	0.591552	0.830	0.688098
t_f	2.260200	2.245400	2.140	2.178500

Table 5.3. *DPIP optimization results.*

Run #	1	2a	2b	2c	3a	3b	3c
Model	linear	n.l.	linear	n.l.	n.l.	linear	n.l.
$\gamma_{desired}$	90°	165°	165°	165°	165°	165°	165°
u_i guess	$1.0\,\forall\, i$	$1.0\,\forall\, i$	$1.0\,\forall\, i$	$u_{2b_{result}}$	$1.0\,\forall\, i$	$1.0\,\forall\, i$	$u_{3b_{result}}$
Iteration #	4	7	4	3	2	4	2
Convergence	$6.86e^{-6}$	NaN	$2.08e^{-7}$	$2.52e^{-9}$	NaN	$2.30e^{-6}$	$2.35e^{-7}$

Recursive quadratic programming optimization 125° trajectory

To start the optimizer, initial guesses for the three parameters were determined by selecting an acceleration value and intuitively tuning the acceleration pulse-on time and acceleration pulse-off time parameters until the residual oscillations were close to zero. Then, by making several runs with the optimizer, the sensitivities of the weights for the constraints were determined. By iteratively fine-tuning the weights, the closest optimal trajectory that meets all the constraints and end conditions was achieved. Table 5.4 lists the initial conditions used for the 125° run to determine the three parameters.

Table 5.4. *Initial conditions for* 125° *VF02AD RQP optimization run.*

Variable	Value	Comments
x_f	126.89	deg
t_f	2.21	sec
acc	10^{-7}	convergence criteria
nstep	600	# RK steps
t_A	0.8	params(1)
A	1.9	params(2)
t_C	0.6	params(3)

The performance index defined in (5.3) set $W_1 = 100$, $W_2 = 10$, and $W_3 = W_4 = W_5 = 0$. The constraints from (5.4) and (5.5) were used. The final parameters upon convergence are given in Table 5.2, run #1. The MATLAB simulation employs the final parameters to generate numerical results for the pendulum tangential ϕ and radial θ oscillations, tangential $\dot{\phi}$ and radial $\dot{\theta}$ angle rates, hub acceleration $\ddot{\gamma}$, hub velocity $\dot{\gamma}$, and hub position γ. The plots are shown in Fig. 5.3. Extensive computer runs and iterations were required to determine a satisfactory RQP-optimized trajectory that performed well on the experimental setup [66].

Recursive quadratic programming versus dynamic programming/interior point optimization 90° trajectory

In the 90° trajectory, the simulation design started with very small arbitrary values for the parameters, $t_A = 0.2$, $A = 0.4$, and $t_C = 0.2$. All of the constraints were put into the performance index and only the final distance was used as a hard constraint. After many iterations, and all the weights set to 0.1, a solution converged that had some residual oscillations. The optimizer was reconfigured to the same architecture used for the 125° trajectory and started with the values found previously for t_A, A, and t_C as the initial conditions. The final parameter values, after convergence, were resolved from the optimizer and recorded in Table 5.2, run #2. The use of these parameter values resulted in zero residual oscillations. In addition, many other trajectories were found with different variations of weights and code settings. All these RQP runs showed similar results for both the simulation and the hardware runs.

For the DPIP case, the optimization run was started with an arbitrary guess and converged in four iterations (see Table 5.3). The results for pendulum swing angles and rates, hub angles, velocities, and accelerations are compared for both optimization results in Fig. 5.4. Good correlation exists between the RQP and DPIP optimization results. For all of the DPIP results, in general, smooth trajectories were produced, with a minimum number of iterations for convergence.

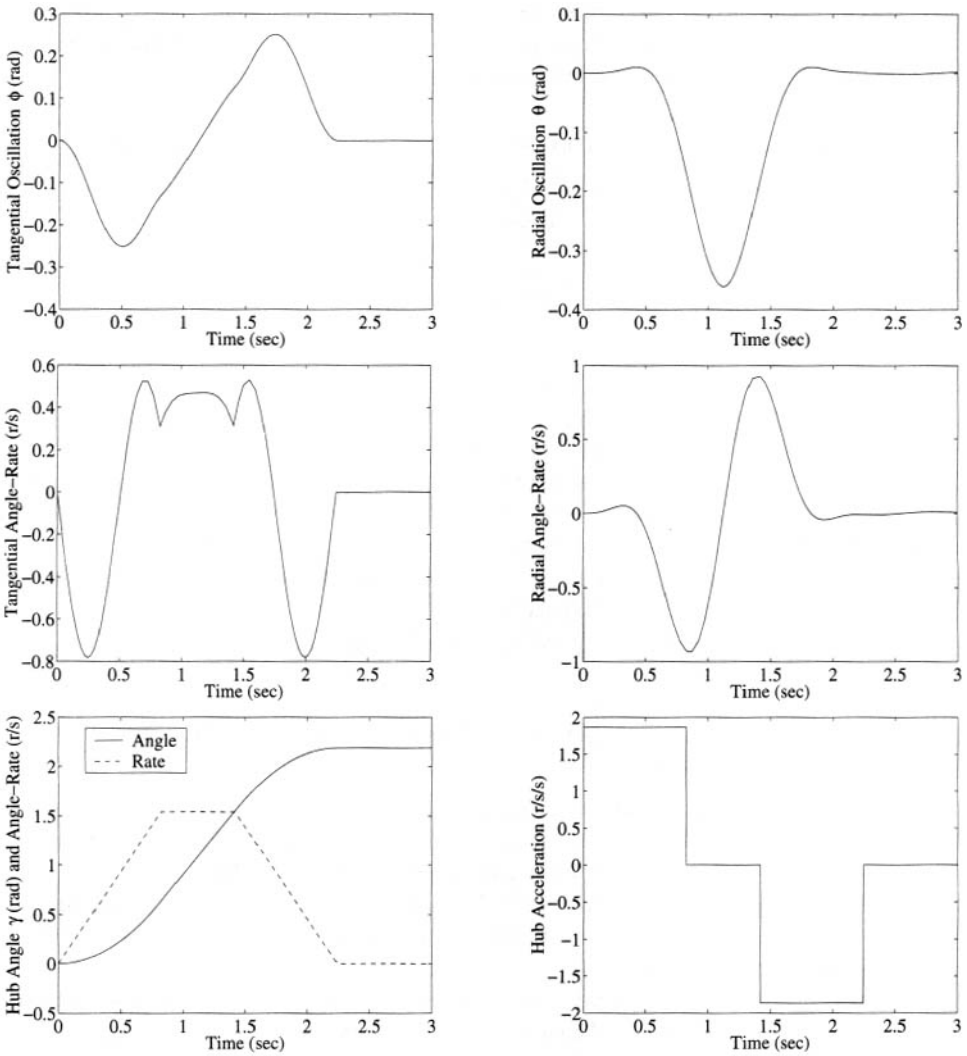


Figure 5.3. *RQP optimization* 125° *trajectory results.*

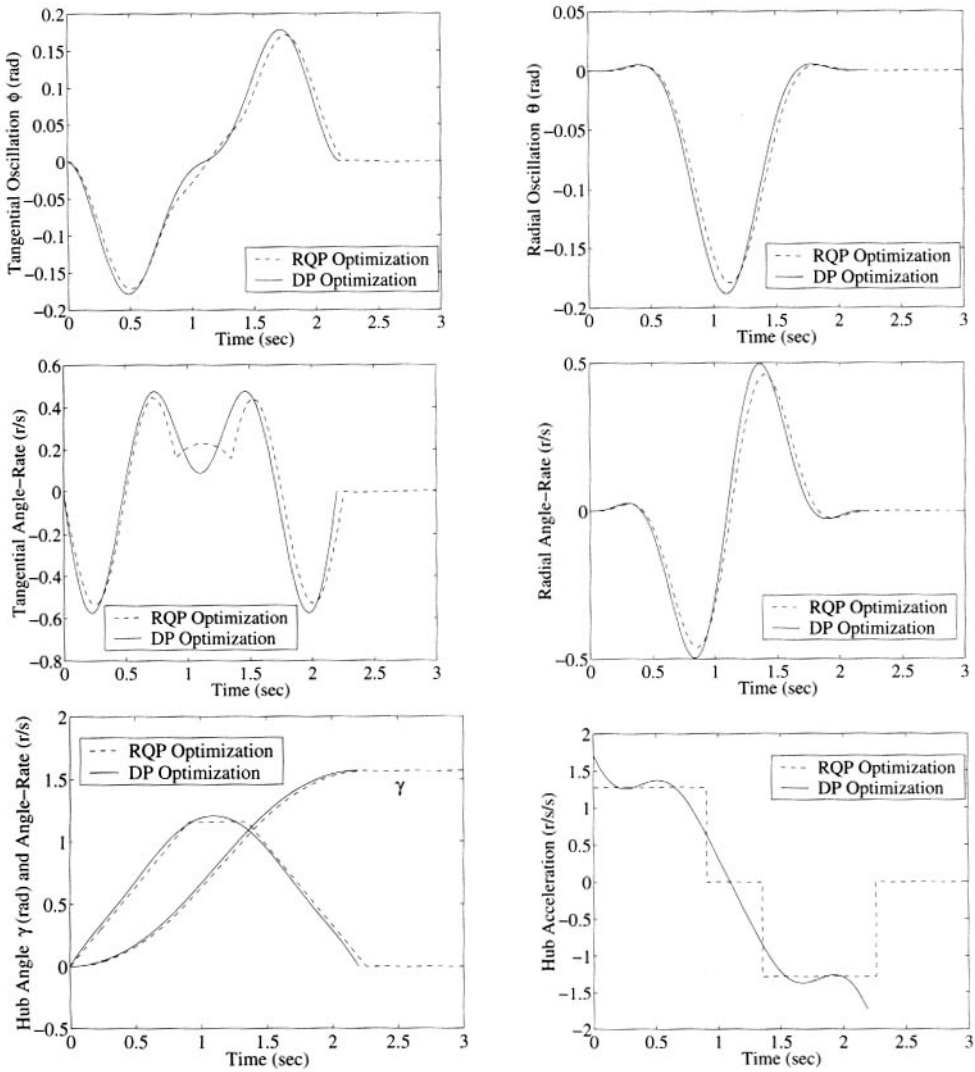


Figure 5.4. *RQP versus DP optimization 90° trajectory results.*

Recursive quadratic programming versus dynamic programming/interior point optimization 165° trajectory—Nonlinear dynamics

A trajectory move was designed to produce large transient pendulum swings. In this case, a large angle is defined to be on the order of 35° or greater. An acceleration just below the limit was targeted. Upon manually tuning the other two parameters, a *near*-optimal solution was reached with the resulting parameters given in Table 5.2, run #3a.

These same results were simulated using nonlinear dynamics and produced large residual oscillations. The comparisons between linear and nonlinear dynamic models are plotted in Fig. 5.5. Both tangential and radial oscillations and angle rates show large discrepancies and significant residual oscillations between the linear and nonlinear results. For large-angle swings, nonlinear dynamics are required to achieve satisfactory performance.

The final parameter results from Table 5.2, run #3a (linear solution), were used as the initial guess, along with nonlinear dynamics, to run the optimizer `VF02AD`. The parameters given in Table 5.2, run #3b, resulted. These parameters were simulated and the results plotted in Fig. 5.6. Shown in the plots are the tangential and radial oscillations, the tangential and radial angle rates, the hub angle and angle rate, and the hub angle accelerations.

For the DPIP case, the first four steps discussed in Section 5.2.4 were followed. The first run started with an arbitrary guess and used nonlinear dynamics. After seven iterations,

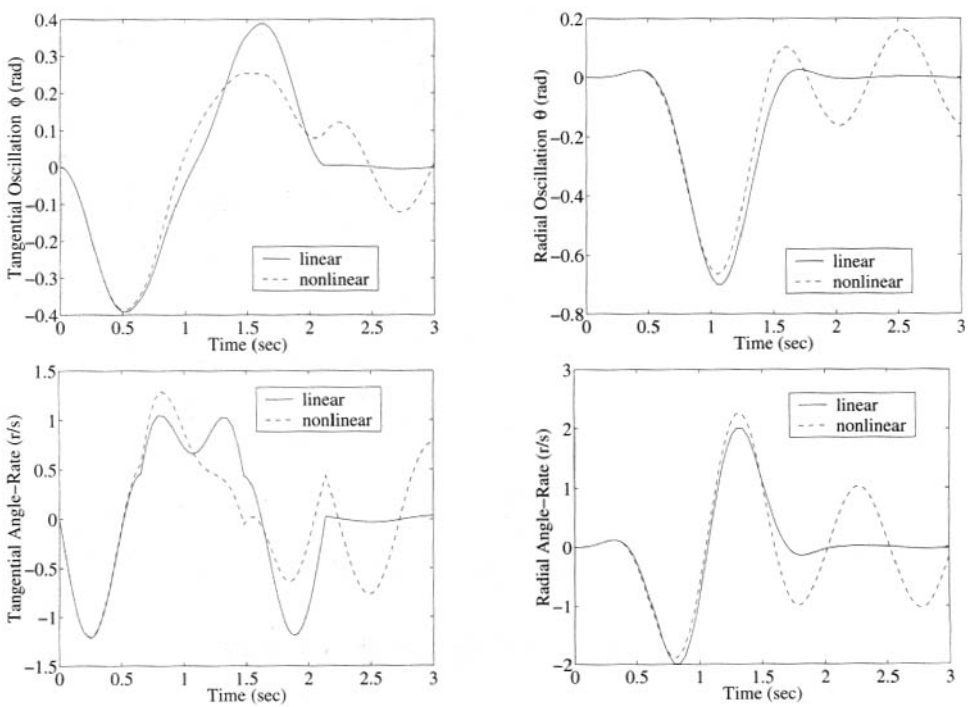


Figure 5.5. *RQP optimization linear versus nonlinear model comparisons for* 165° *trajectory results.*

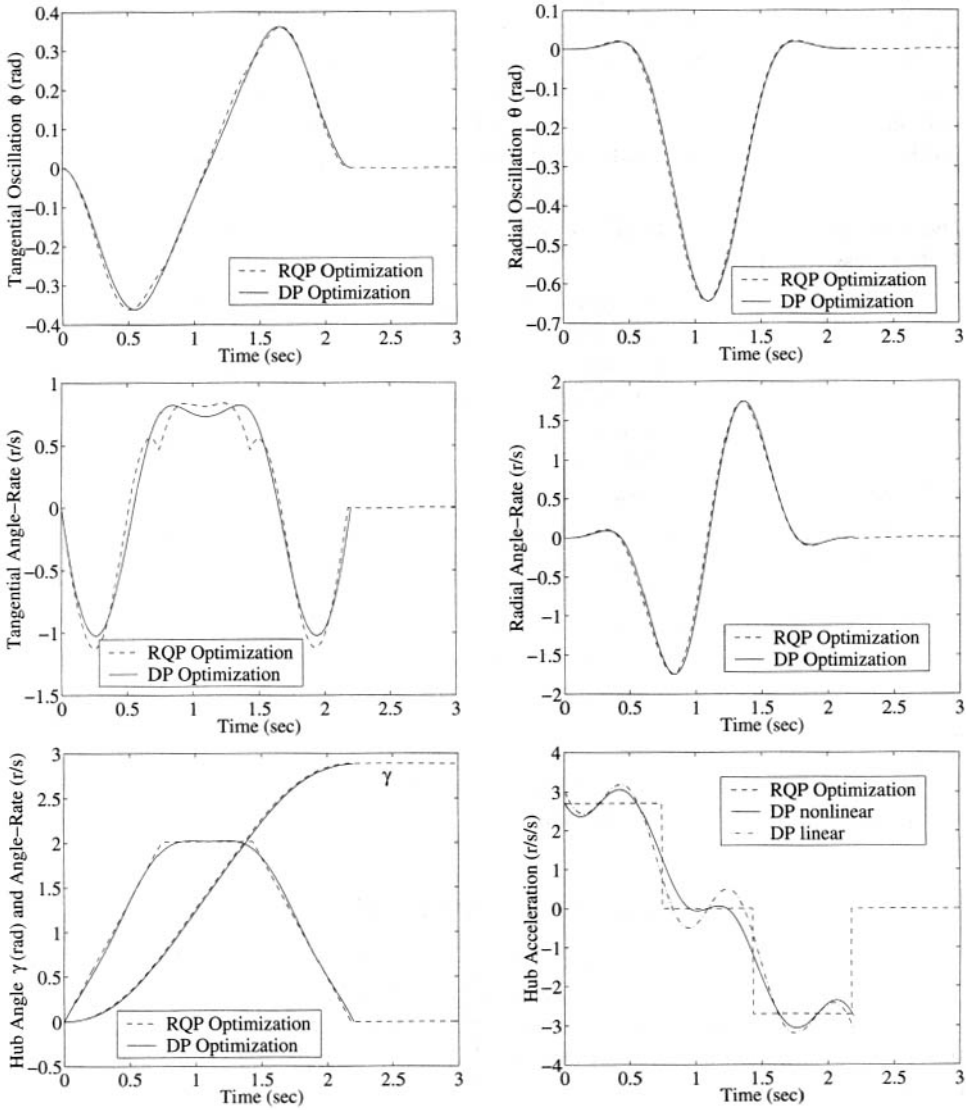


Figure 5.6. *RQP versus DPIP optimization* 165° *trajectory results.*

the optimization diverged. The results are recorded in Table 5.3, run #2a. For the next step, the same initial guess was used but linear dynamics were used. After four iterations, the optimization solution converged (see Table 5.3, run #2b). In the final step, the linear optimization hub acceleration solution was used as the initial guess for the nonlinear dynamic problem. This time, the optimization solution converged in three iterations (see Table 5.3, run #2c). The results for the RQP and DPIP optimization runs are plotted in Fig. 5.6. These results are shown for tangential and radial oscillations, tangential and radial angle rates, hub angle and angle rates, and hub acceleration. In the bottom right plot, the linear and nonlinear hub acceleration results are included along with the RQP solution. Again, the DPIP solution resulted in overall fewer iterations and smoother trajectories.

Dynamic programming/interior point optimization 165° trajectory—Robustness variation

For the final DPIP case, higher derivatives were included in the cost function and smoother trajectories resulted for the lower derivatives. In this case, the third derivative in time or hub jerk profile was selected as the cost function. This choice helped increase the robustness to parameter variations. Again, the first four steps in Section 5.2.4 were used. The first run started with an arbitrary guess of all ones (1s) for nonlinear dynamics. After two iterations, the optimization diverged. The results are recorded in Table 5.3, run #3a. The next step used the same initial guess of all ones, but for linear dynamics. After four iterations, the solution converged (see Table 5.3, run #3b). In the last step, the linear solution to u_i was used to start the nonlinear dynamic problem. The DPIP optimization converged to an acceptable value in two iterations (see Table 5.3, run #3c). The results for the DPIP optimization runs are plotted in Fig. 5.7. These results are shown for tangential and radial oscillations, tangential and radial angle rates, hub angle and angle rates, and hub acceleration. Again, the bottom right plot shows the linear and nonlinear hub acceleration results.

The final results as compared to the DPIP optimization results in Fig. 5.6 are smoother trajectory profiles which are especially noticeable in the hub angle rates and accelerations. In contrast, for the same time duration, the robustness variation does result in larger angle swings and increased angle rate magnitudes.

5.2.6 Case Study 1 Discussion (Summary/Conclusions)

The objective of this case study was to design feedforward-type commands for a rotary jib crane to produce rest-to-rest, residual–oscillation-free maneuvers. This required the development of the dynamic equations of motion, a numerical implementation, optimized trajectory design, verification using numerical simulation, and experimental hardware validation [66] through RQP-correlated runs.

Several successful swing-free maneuvers were designed for both the DP and the RQP approaches over a wide range of operating conditions. For example, during the 90° swing, the pendulum only had peak transient swings of about 10°. This is well within the linear assumptions used to develop the simulation models and optimized input trajectories. The 125° swing produced peak transient swings on the order of 20°. However, with the linear model, a swing-free maneuver was produced. The nonlinear equations were then used for

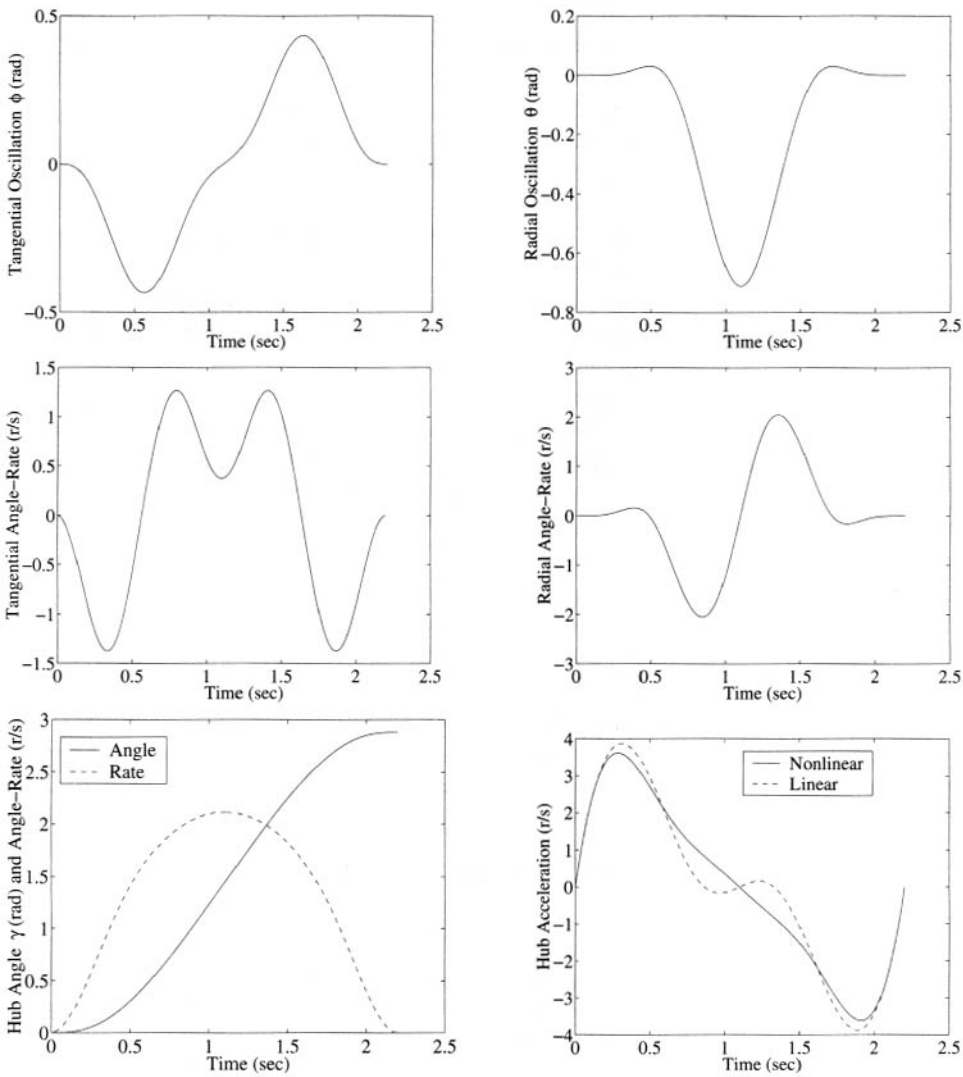


Figure 5.7. *DPIP optimization* 165° *robustness variation trajectory results.*

the 125° swing and tested on the hardware without any visual improvement in the residual oscillations over the linear design. Some of the reasons for this are discussed next. Finally, the large swing, a 165° maneuver, was designed using the nonlinear equations of motion. This maneuver produced peak transient swings on the order of 35°, clearly justifying the use of the nonlinear analysis.

All the trajectories were finished in a little more than 2 sec, at which time all the residuals are shown to be oscillation free. However, using only linear dynamics resulted in residual oscillations.

These trajectories were originally tested on hardware [66]. Although the simulations showed oscillation-free residuals, the hardware runs showed some small amount of residual oscillations. Upon further exploration, some partial conclusions can be drawn: (1) the actuator dynamics of the stepper motor may not be considered negligible, (2) stiction/friction at the joint and its interaction with the motor may not be considered negligible, (3) air friction and air currents in the room may cause unwanted oscillations, and (4) the actual roller at the end of the boom does not allow ideal oscillations of the hanging pendulum. In addition, there are no calibration data to verify that the control computer generated the true trajectory. Fortunately, general trends were demonstrated.

The DP optimization runs for the most part duplicated the RQP optimization runs. In general, the RQP optimization algorithm required extensive computer runs and iterations (much greater than 10) to determine satisfactory optimized trajectory results. In addition, the RQP optimizer was sensitive to the initial guess and required an intuitively close neighboring trajectory that was within a certain error radius in order to converge to the desired optimized trajectory. Otherwise, the optimization procedure would diverge.

In contrast, the DPIP optimization runs showed benefits in overall fewer iterations (less than 10), smooth trajectories, and less sensitivity to initial conditions. Both methods were able to take advantage of linear solutions to help find solutions to the nonlinear problem. A five-step procedure was identified for finding solutions to the nonlinear problem and was applied during the DPIP runs. In summary, the DPIP algorithm was found to be an effective optimization procedure and was verified through numerical simulations for the rotary jib crane, rest-to-rest, residual–oscillation-free maneuvers.

5.3 Case Study 2: Slewing Flexible Link

5.3.1 Introduction

An alternative method for generating optimized open-loop and closed-loop control profiles for slewing flexible structures is investigated. Slewing motion is the rotation of a structure about some point or axis. Real applications include slewing ground-based antenna arrays and pivoting a solar panel about a point on a satellite. During the era of large space structures, investigators in the dynamics and control field reviewed slewing maneuvers and control of flexible structures [70, 71]. Analogous to slewing flexible structures is slewing a flexible single-link manipulator moving in the horizontal plane (see Fig. 5.8). This has also been an area of research interest in the robotics field [72]. A large number of problems can be classified as slewing problems, e.g., attitude maneuvers of rigid satellites, satellites with flexible appendages, high-speed rotating structures such as helicopter blades, and rotating flexible blades associated with alternative wind energy turbine devices. The mechanical flexibil-

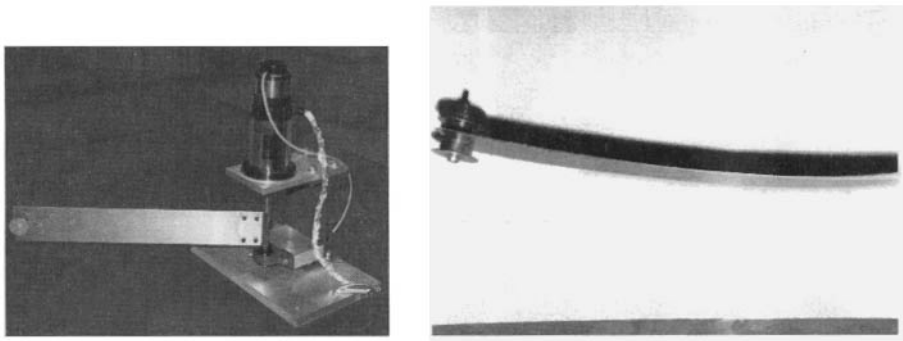


Figure 5.8. *Slewing flexible link hardware testbed setup and corresponding close-up of linkage and tip mass.*

ity of robotic manipulators is a major problem in motion control. To achieve satisfactory performance, the mechanical vibrations need to be suppressed.

In both robotics research and the structural control communities, the simple one-link structure is usually modeled as a flexible beam fixed to a rotating axis. For revolving thin flexible links, it is most common to use Euler–Bernoulli beam theory to help predict the natural frequencies of the beam [73, 74]. In addition, for significantly high slewing rates, the effects of centrifugal stiffening need to be included in the modeling of the flexible structure. An efficient approach that includes the centrifugal stiffening effect has recently been presented [69].

Vibration control of machines that exhibit flexibility becomes more important, especially since the use of lightweight machinery with high operation speed is becoming more prevalent in the production and manufacturing industries. Several fundamental problems for improving a slewing flexible link motion can be considered [69, 75]: First, an accurate experimentally identified dynamic model of the system must be determined. Second, a control system design needs to consider realizable sensors and actuators and robustness to modeling inaccuracies and external disturbances. Third, ideal control efforts need to be understood that reflect the desired and achievable performance. Finally, reference inputs to the controller should include knowledge of the dynamic response of the system. In this case study, the focus is mainly on the last two problems, that is, to determine an optimal open-loop torque response. In addition, for the experimental implementation, an ideal reference trajectory input is considered for the closed-loop controller.

As discussed in the previous case study, the DP method [68] is used for general command shaping of optimal maneuvers. These results are compared with the MATLAB Optimization Toolbox *fmincon* function. Numerical simulations are performed for both methods. In addition, a parameter optimization problem that employs an RQP technique was used to perform a closed-loop trajectory for an experimental system and was documented in detail [76, 77]. Several of the critical features included a calibrated model encompassing the effects of Coulomb friction. Without the Coulomb friction effect, the performance on the hardware was deteriorated. The DPIP algorithm is employed to find an optimal trajectory for the same closed-loop system that also takes into account the Coulomb friction term. The numerical results are compared with the original RQP results.

The objective of this case study is to design a feedforward, open-loop control using numerical optimization techniques for a slewing flexible link. The following steps are included.

Step 1: Derive the equations of motion for a slewing flexible link that includes the first two modes, actuator dynamics, and the centrifugal stiffening effect.

Step 2: Develop a numerical simulation of the slewing flexible link.

Step 3: Solve a trajectory optimization problem using the MATLAB Optimization Toolbox *fmincon* function.

Step 4: Solve a trajectory optimization problem using a DP design.

Step 5: Compare the numerical simulations for the optimization approaches.

Step 6: Verify an optimization design for a real slewing flexible link hardware.

Step 7: Summarize the case study findings.

5.3.2 Model Development

In this section, the dynamic equations of motion for a slewing flexible link are derived based on the quadratic modes formulation [69]. A schematic of the flexible link is shown in Fig. 5.9. In addition, actuator dynamics [4, 78] are included and merged with the flexible link dynamic model. This model is then implemented numerically in the MATLAB environment for each optimization code. A single flexible mode version of this model was used to perform an experimental validation that used the RQP optimization routine [76, 77].

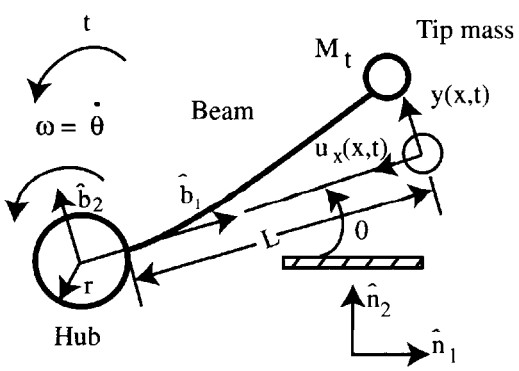


Figure 5.9. *Mathematical description of a slewing flexible structure.*

Table 5.5. *Flexible link physical parameters.*

Description	Symbol	Value	Unit
Length	L	45.72	cm
Beam stiffness	EI	0.024	kg-m^2
Hub radius	r	5.715	cm
Motor inertia	J_m	4.46e−4	kg-m^2
Density/length	ρ	0.1744	kg/m
Viscous damping	B_{vf}	3.67e−3	kg-m^2/sec
Tip mass	M_t	0.104	kg
Coulomb friction	C_{df}	0.07414	N-m

The dynamics of a unidirectional rotating beam in gravity was considered using quadratic modes with a tip mass [76]. The physical parameters are given in Table 5.5. The corresponding hardware setup and geometry of the flexible slewing link are given in Fig. 5.9.

Referring to Fig. 5.9, an expression for the deformation of a point along the link is

$$\mathbf{u}(x,t) = u_x(x,t)\hat{\mathbf{b}}_1 + y(x,t)\hat{\mathbf{b}}_2, \tag{5.10}$$

where $\hat{\mathbf{b}}_i$ are unit vectors associated with a moving coordinate system attached to the hub. The $\hat{\mathbf{n}}_i$ unit vectors are associated with an inertial coordinate system. Using the assumed modes approach and expansion for axial deflection, $u_x(x,t)$ can be defined using quadratic modes. The lateral deflection, $y(x,t)$, uses standard linear mode shapes. This results in

$$\mathbf{u}(x,t) = q^i(t)q^j(t)g^{ij}(x)\hat{\mathbf{b}}_1 + q^i(t)\phi^i(x)\hat{\mathbf{b}}_2, \tag{5.11}$$

where generalized coordinates of deformation are given by the $q^i(t)$ terms. The mode shapes, $\phi^i(x)$, are the standard linear mode shapes for lateral beam vibration. The quadratic modes, $g^{ij}(x)$, represent foreshortening of the beam consistent with the linear mode shapes. The quadratic mode shapes for a beam are given by

$$\frac{\partial g^{ij}}{\partial x} = -\frac{1}{2}\frac{\partial \phi^i}{\partial x}\frac{\partial \phi^j}{\partial x}. \tag{5.12}$$

Taking into account the hub radius offset, an expression for the velocity along the beam is

$$\dot{\mathbf{x}}(x,t) = \frac{d}{dt}\{[r+x]\hat{\mathbf{b}}_1 + \mathbf{u}(x,t)\}. \tag{5.13}$$

Lagrange's equations are used to derive the equations of motion for both the rigid body, θ, and the flexible body, q^i DOFs (degrees of freedom). The kinetic energy, T; potential energy, V; and work from external forces, W_F, are used to form the Lagrangian, $L = T - V + W_F$. The kinetic energy is

$$T = \frac{1}{2}I_{hub}\dot{\theta}^2 + \frac{1}{2}\int_0^L \bar{\rho}\,\dot{\mathbf{x}}^T \cdot \dot{\mathbf{x}}\,dx, \tag{5.14}$$

where $\bar{\rho} = \rho + M_t\delta(x - L)$, ρ is mass per unit length, and I_{hub} is the effective hub inertia. The potential energy is

$$V = \frac{1}{2}(EI)q^i q^j \int_0^L \phi^{i''}(x)\phi^{j''}(x)\, dx + \bar{\rho} g_r Y, \tag{5.15}$$

where I is the mass moment of inertia of the beam, E is the link modulus of elasticity, g_r is the gravitational constant, and Y is the potential height. The generalized work term is

$$W_F = \tau\delta\theta. \tag{5.16}$$

After the expressions for kinetic energy, potential energy, and work are substituted into Lagrange's equations, the following expressions for the beam deflection and rotation are determined. The first equation of motion that results from Lagrange's derivation is the general beam deflection equation

$$\left[\rho\int_o^L \phi_i\phi_j dx\right]\ddot{q}_i + \left\{EI\int_o^L \phi_i''\phi_j'' dx - \left(\rho\int_o^L \phi_i\phi_j dx + 2\rho\int_o^L [r+x]g_{ij}dx\right)\dot{\theta}^2\right\} q_i$$
$$+ \rho g_r\cos\theta\int_o^L \phi_i dx - \rho g_r\sin\theta\int_o^L g_{ij}dx q_i + \left[\rho\int_o^L [r+x]\phi_i dx\right]\ddot{\theta} = 0. \tag{5.17}$$

Polynomial-like expressions are used to perform the mode shape expansions. The first two mode shapes used are

$$\phi_1(x) = x^2(3L - x) = 3Lx^2 - x^3 \tag{5.18}$$

and

$$\phi_2(x) = x^2 L. \tag{5.19}$$

The quadratic modes for a beam are defined [69] as

$$g_{ij} = -\frac{1}{2}\int_o^x \phi_i'\phi_j' d\xi. \tag{5.20}$$

The needed terms are

$$\phi_1'(\xi) = 6L\xi - 3\xi^2 \tag{5.21}$$

and

$$\phi_2'(\xi) = 2\xi L. \tag{5.22}$$

Substituting the necessary components into (5.20) results in

$$g_{11} = -\frac{1}{2}\int_o^x \phi_1'(\xi)^2 d\xi = -\frac{1}{2}\left[12L^2x^3 - 9Lx^4 + \frac{9}{5}x^5\right], \tag{5.23}$$

$$g_{12} = g_{21} = -\frac{1}{2}\int_o^x \phi_1'(\xi)\phi_2'(\xi) d\xi = -\frac{1}{2}\left[4L^2x^3 - \frac{3}{2}Lx^4\right], \tag{5.24}$$

$$g_{22} = -\frac{1}{2}\int_o^x \phi_2'(\xi)^2 d\xi = -\frac{1}{2}\left[\frac{4}{3}L^2x^3\right]. \tag{5.25}$$

Starting with the mass density terms and working through term by term, the necessary integrations yield

$$\rho\int_o^L \phi_i\phi_j dx\ddot{q}_i = \rho L^7 \begin{bmatrix} \frac{33}{35} & \frac{13}{30} \\ \frac{13}{30} & \frac{1}{5} \end{bmatrix} \begin{Bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{Bmatrix},$$

$$\rho\int_o^L \phi_i\phi_j dx\dot{\theta}^2 q_i = \rho L^7\dot{\theta}^2 \begin{bmatrix} \frac{33}{35} & \frac{13}{30} \\ \frac{13}{30} & \frac{1}{5} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$\phi_1''(x) = 6L - 6x,$$

$$\phi_2''(x) = 2L,$$

$$EI\int_o^L \phi_i''\phi_j'' dx q_i = EIL^3 \begin{bmatrix} 12 & 6 \\ 6 & 4 \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$2\rho\int_o^L r g_{ij} dx\dot{\theta}^2 q_i = -\rho r L^6\dot{\theta}^2 \begin{bmatrix} \frac{3}{2} & \frac{7}{10} \\ \frac{7}{10} & \frac{1}{3} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$2\rho\int_o^L x g_{ij} dx\dot{\theta}^2 q_i = -\rho L^7\dot{\theta}^2 \begin{bmatrix} \frac{81}{70} & \frac{11}{20} \\ \frac{11}{20} & \frac{4}{15} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$\rho g_r \cos\theta \int_o^L \phi_i dx = \begin{Bmatrix} \frac{3}{4} \\ \frac{1}{3} \end{Bmatrix} \rho g_r L^4 \cos\theta,$$

$$-\rho g_r \sin\theta \int_o^L g_{ij} dx q_i = \begin{bmatrix} \frac{3}{2} & \frac{7}{10} \\ \frac{7}{10} & \frac{1}{3} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$-\rho r\int_o^L \phi_i dx\ddot{\theta} = -r\rho L^4 \begin{Bmatrix} \frac{3}{4} \\ \frac{1}{3} \end{Bmatrix} \ddot{\theta},$$

$$-\rho\int_o^L x\phi_i dx\ddot{\theta} = -\rho L^5 \begin{Bmatrix} \frac{11}{20} \\ \frac{1}{4} \end{Bmatrix} \ddot{\theta}.$$

Next, investigate the tip mass terms

$$m_T \int_o^L \delta(x-L)\phi_i\phi_j dx\ddot{q}_i = m_T L^6 \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{Bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{Bmatrix},$$

$$m_T \int_o^L \delta(x-L)\phi_i\phi_j dx\dot{\theta}^2 q_i = m_T L^6 \dot{\theta}^2 \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$2m_T \int_o^L \delta(x-L) r g_{ij} dx\dot{\theta}^2 q_i = -m_T \dot{\theta}^2 r L^5 \begin{bmatrix} \frac{24}{5} & \frac{5}{2} \\ \frac{5}{2} & \frac{4}{3} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$2m_T \int_o^L \delta(x-L) x g_{ij} dx\dot{\theta}^2 q_i = -m_T \dot{\theta}^2 L^6 \begin{bmatrix} \frac{24}{5} & \frac{5}{2} \\ \frac{5}{2} & \frac{4}{3} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$m_T g_r \cos\theta \int_o^L \delta(x-L)\phi_i dx = m_T g_r \cos\theta L^3 \begin{Bmatrix} 2 \\ 1 \end{Bmatrix},$$

$$-m_T g_r \sin\theta \int_o^L \delta(x-L) g_{ij} dx q_i = m_T g_r \sin\theta L^5 \begin{bmatrix} \frac{24}{5} & \frac{5}{2} \\ \frac{5}{2} & \frac{4}{3} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix},$$

$$-m_T r \int_o^L \delta(x-L)\phi_i dx\ddot{\theta} = -m_T r L^3 \ddot{\theta} \begin{Bmatrix} 2 \\ 1 \end{Bmatrix},$$

$$-m_T \int_o^L \delta(x-L) x\phi_i dx\ddot{\theta} = -m_T L^4 \ddot{\theta} \begin{Bmatrix} 2 \\ 1 \end{Bmatrix}.$$

The completed derivation results are brought together by assembling all the above terms as

$$\left[\rho L^7 \begin{bmatrix} \frac{33}{35} & \frac{13}{30} \\ \frac{13}{30} & \frac{1}{5} \end{bmatrix} + m_T L^6 \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}\right] \begin{Bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{Bmatrix} + \left[EIL^3 \begin{bmatrix} 12 & 6 \\ 6 & 4 \end{bmatrix} - \rho L^7 \dot{\theta}^2 \begin{bmatrix} \frac{33}{35} & \frac{13}{30} \\ \frac{13}{30} & \frac{1}{5} \end{bmatrix}\right.$$

$$-m_T \dot{\theta}^2 L^6 \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} + \rho\dot{\theta}^2 r L^6 \begin{bmatrix} \frac{3}{2} & \frac{7}{10} \\ \frac{7}{10} & \frac{1}{3} \end{bmatrix} + m_T \dot{\theta}^2 r L^5 \begin{bmatrix} \frac{24}{5} & \frac{5}{2} \\ \frac{5}{2} & \frac{4}{3} \end{bmatrix} + \rho L^7 \dot{\theta}^2 \begin{bmatrix} \frac{81}{70} & \frac{11}{20} \\ \frac{11}{20} & \frac{4}{15} \end{bmatrix}$$

$$\left. + m_T L^6 \dot{\theta}^2 \begin{bmatrix} \frac{24}{5} & \frac{5}{2} \\ \frac{5}{2} & \frac{4}{3} \end{bmatrix} + \rho g_r L^6 \sin\theta \begin{bmatrix} \frac{3}{2} & \frac{7}{10} \\ \frac{7}{10} & \frac{1}{3} \end{bmatrix} + m_T g_r L^5 \sin\theta \begin{bmatrix} \frac{24}{5} & \frac{5}{2} \\ \frac{5}{2} & \frac{4}{3} \end{bmatrix}\right] \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix}$$

$$= -\left\{ r\rho L^4 \begin{Bmatrix} \frac{3}{4} \\ \frac{1}{3} \end{Bmatrix} + m_T r L^3 \begin{Bmatrix} 2 \\ 1 \end{Bmatrix} + \rho L^5 \begin{Bmatrix} \frac{11}{20} \\ \frac{1}{4} \end{Bmatrix} + m_T L^4 \begin{Bmatrix} 2 \\ 1 \end{Bmatrix} \right\} \ddot{\theta}$$

$$-\rho g_r L^4 \cos\theta \begin{Bmatrix} \frac{3}{4} \\ \frac{1}{3} \end{Bmatrix} - m_T g_r \cos\theta L^3 \begin{Bmatrix} 2 \\ 1 \end{Bmatrix}.$$

The second equation of motion that results from Lagrange's derivation is the beam rotation equation

$$\frac{1}{3}\rho L^3\ddot{\theta} + \left[\rho\int_0^L x\phi_i dx\right]\ddot{q}_i - \rho g_r\left[\int_0^L \phi_i dx \sin\theta\right]q_i + \frac{1}{2}\rho g_r L^2\cos\theta = \tau_{link}\ . \quad (5.26)$$

The same steps and integrations as performed for the beam deflection equation are also completed for the rotation equation. The same two mode shapes, (5.18) and (5.19), are used and result in the following rotation equation of motion:

$$\left[\frac{1}{3}\rho L^3 + \rho r L^2 + \rho r^2 L + m_T(r+L)^2\right]\ddot{\theta} + \left[\frac{11}{20}\rho L^5 + \frac{3}{4}\rho r L^4 + 2m_T r L^3 + 2m_T L^4\right]\ddot{q}_1$$

$$+\left[\frac{1}{4}\rho L^5 + \frac{1}{3}\rho r L^4 + m_T r L^3 + m_T L^4\right]\ddot{q}_2 - g_r\sin\theta\left[\frac{3}{4}\rho L^4 + 2m_T L^3\right]q_1$$

$$-g_r\sin\theta\left[\frac{1}{3}\rho L^4 + m_T L^3\right]q_2 + g_r\cos\theta\left[\frac{1}{2}\rho L^2 + \rho r L + m_T(r+L)\right] = \tau_{link}\ . \quad (5.27)$$

The actuator dynamics are given [4, 78] as

$$J\ddot{\theta}_m + b_{vf}\dot{\theta}_m + c_{df}\text{sign}(\dot{\theta}_m) = \tau - n_r\tau_{link}. \quad (5.28)$$

This model includes both a viscous friction and a Coulomb friction term. The inertia is the total rotational inertia seen at the hub. For a direct-drive system, the gear ratio is $n_r = 1$; this also determines that $\theta = \theta_m$.

For the closed-loop system, the motor is given an input command that uses a Proportional-Derivative (PD) control law defined [78] as

$$\tau = K_p(\theta_{ref} - \theta) - K_d\dot{\theta}, \quad (5.29)$$

where K_p and K_d are the proportional and derivative controller gains, respectively.

The final step is to assemble the deformation, rotation, and actuator equations into an overall set of system equations. In addition, for a beam rotating in the horizontal plane, gravity is not considered and a term for light dampening in the beam is added. This results in the matrix equation

$$\begin{bmatrix} M_{00} & M_{01} & M_{02} \\ M_{01} & M_{11} & M_{12} \\ M_{02} & M_{12} & M_{22} \end{bmatrix}\begin{Bmatrix} \ddot{\theta} \\ \ddot{q}_1 \\ \ddot{q}_2 \end{Bmatrix} + \begin{bmatrix} C_0 & 0 & 0 \\ 0 & c_1 & 0 \\ 0 & 0 & c_2 \end{bmatrix}\begin{Bmatrix} \dot{\theta} \\ \dot{q}_1 \\ \dot{q}_2 \end{Bmatrix}$$

$$+\left[\dot{\theta}^2\begin{bmatrix} 0 & 0 & 0 \\ 0 & KC_{11} & KC_{12} \\ 0 & KC_{12} & KC_{22} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & K_{11} & K_{12} \\ 0 & K_{12} & K_{22} \end{bmatrix}\right]\begin{Bmatrix} \theta \\ q_1 \\ q_2 \end{Bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\tau, \quad (5.30)$$

where τ is the open-loop control torque, and for a closed loop the control law is given as (5.29).

The matrix components then become

$$
\begin{aligned}
M_{00} &= J + \frac{1}{3}\rho L^3 + \rho r L^2 + \rho r^2 L + m_T(r+L)^2,\\
M_{01} &= \frac{11}{20}\rho L^5 + \frac{3}{4}\rho r L^4 + 2m_T r L^3 + 2m_T L^4,\\
M_{02} &= \frac{1}{4}\rho L^5 + \frac{1}{3}\rho r L^4 + m_T r L^3 + m_T L^4,\\
M_{11} &= \frac{33}{35}\rho L^7 + 4m_T L^6,\\
M_{12} &= \frac{13}{30}\rho L^7 + 2m_T L^6,\\
M_{22} &= \frac{1}{5}\rho L^7 + m_T L^6,\\
C_0 &= b_{vf}\dot{\theta} + c_{df}\,\mathrm{sign}(\dot{\theta}),\\
KC_{11} &= -\frac{33}{35}\rho L^7 - 4m_T L^6 + \frac{3}{2}\rho r L^6 + \frac{24}{5}m_T r L^5 + \frac{81}{70}\rho L^7 + \frac{24}{5}m_T L^6,\\
KC_{12} &= -\frac{13}{30}\rho L^7 - 2m_T L^6 + \frac{7}{10}\rho r L^6 + \frac{5}{2}m_T r L^5 + \frac{11}{20}\rho L^7 + \frac{5}{2}m_T L^6,\\
KC_{22} &= -\frac{1}{5}\rho L^7 - m_T L^6 + \frac{1}{3}\rho r L^6 + \frac{4}{3}m_T r L^5 + \frac{11}{20}\rho L^7 + \frac{4}{3}m_T L^6,\\
K_{11} &= 12EIL^3,\\
K_{12} &= 6EIL^3,\\
K_{22} &= 4EIL^3.
\end{aligned}
$$

The complete system equations of motion in matrix form are

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + [\dot{\theta}^2\mathbf{K}_c + \mathbf{K}]\mathbf{x} = \mathbf{B}u, \tag{5.31}$$

where the state $\mathbf{x} = \{\theta \quad q_1 \quad q_2\}^T$, $\mathbf{B} = \{1 \quad 0 \quad 0\}^T$, and $u = \tau$.

Develop a numerical simulation for the flexible slewing link

The MathWorks MATLAB software [27] was used to develop a numerical simulation. A user-supplied driver script code *fbeamsim.m* calls the MATLAB RK *ode*23 function, which then calls the user-supplied equations of motion function *fbeam*1.*m*. These codes were integrated into each optimization simulation. The script files for the simulation and equations of motion are provided in Section B.2 of Appendix B.

5.3.3 Optimization Feedforward Command Design

The goal of this section is to design a feedforward command for the slewing flexible link to produce a rest-to-rest, residual–vibration-free maneuver. A single slew maneuver from 0° to 90° was used for all the optimization studies. Two separate optimization techniques were used to develop open-loop minimum effort and minimum effort with bounds trajectories.

This included the MATLAB Optimization Toolbox function *fmincon*, which finds the constrained minimum of a function of several variables, and the current DPIP method approach. An RQP optimization solution was originally generated [76, 77] with a calibrated model to determine a closed-loop optimal trajectory. This was verified experimentally with the single flexible link hardware. The solution required the inclusion of Coulomb friction. For comparison, the same model was simulated with the DPIP software and a similar solution was obtained. The problem formulations for each method are discussed next.

FMINCON optimization approach

The minimum effort trajectory problem was set up within the MATLAB Optimization Toolbox [19] *fmincon* function with the cost function

$$J = \int_0^{t_f} u^2 dt$$

subject to the flexible link dynamics (5.31). Note that the control effort is defined as $u = \tau$. The system is originally at rest, so the initial conditions are

$$\theta(0) = q_1(0) = q_2(0) = \dot{\theta}(0) = \dot{q}_1(0) = \dot{q}_2(0) = 0. \tag{5.32}$$

The equality end condition constraints are given by

$$\theta(t_f) - \theta_{desired} = q_1(t_f) = q_2(t_f) = 0 \tag{5.33}$$

and

$$\dot{\theta}(t_f) = \dot{q}_1(t_f) = \dot{q}_2(t_f) = 0. \tag{5.34}$$

The final time was set to $t_f = 1.0$ sec. The number of discretization points was set to $N = 26$.

In the next run, the minimum control effort is considered with upper and lower bounds. The problem is set up exactly as in the previous case, for minimum control effort, but with the additional inequality constraint

$$u_{min} \leq u \leq u_{max}.$$

For these open-loop torque trajectories, only viscous friction was simulated and the Coulomb friction was not included. The flexible link model included both modes. All of the numerical simulation results are discussed in Section 5.3.5.

Dynamic programming/interior point optimization approach

For both the minimum control effort and the minimum control effort with bounds trajectories, the DP algorithm was set up in the same way as the MATLAB *fmincon* problem. The DPIP approach is to minimize control effort with identical cost functions, subject to the same dynamic equations (5.31) and constraints as the MATLAB *fmincon* function. For the DPIP algorithm, the number of discretization points was set to $N = 201$. The numerical simulation results are reviewed in Section 5.3.5.

For the closed-loop trajectory, the dynamic model only included one mode and the Coulomb friction term was included. The dynamic model is combined with the PD control law (5.29) so that the control effort now becomes the derivative of the reference trajectory input, or

$$J = \int_0^{t_f} \left(\dot{\theta}_{ref}\right)^2 dt,$$

subject to the dynamic equations of motion (includes one mode and Coulomb friction, $c_{df} \neq 0$) and the same initial conditions as (5.32) and end condition constraints as (5.34) with the addition of

$$\theta_{ref}(0) = 0$$

and

$$\theta_{ref}(t_f) - \theta_{ref_{desired}}(t_f) = 0,$$

respectively.

Recursive quadratic programming optimization approach

The slewing flexible link constrained optimization problem was set up with respect to the dynamic model. The trajectory optimization problem was solved using the RQP algorithm `VF02AD` [67]. The problem used 20 discretized temporal control inputs.

To determine an initial guess, the slewing flexible link model was set up with the performance index

$$J_1(\xi) = \int_0^{t_f} [\dot{\theta}(t)^2 + q(t)^2 + \dot{q}(t)^2]dt,$$

subject to the dynamic equations of motion (5.31), that includes only a single flexible mode and has the constraints

$$\psi_1(\xi) = \theta(t_f) - \frac{\pi}{2}$$

and

$$\psi_2(\xi) = \theta_{ref}(t_f) - \frac{\pi}{2}.$$

An approximate convergence was reached. These results were used as an initial guess for the modified performance index

$$J_2(\xi) = 0.5 \int_0^{t_f} u^2 dt + 20(\dot{\theta}(t_f))^2$$

subject to the same dynamic equations (5.31) with modified constraints

$$\begin{aligned}
\psi_1(\xi) &= \theta(t_f) - \frac{\pi}{2}, \\
\psi_2(\xi) &= q(t_f) - 0.0, \\
\psi_3(\xi) &= \dot{q}(t_f) - 0.0, \\
\psi_4(\xi) &= \theta_{ref}(t_f) - \frac{\pi}{2}.
\end{aligned}$$

The control variable was set to the command input rate

$$u = \dot{\theta}_{ref}.$$

The RQP results [77] are reviewed in Section 5.3.5 along with the corresponding DPIP numerical simulation results.

5.3.4 Implementation Issues

During the implementation of the DP algorithm with the Coulomb friction effect, a modification was introduced. Normally, Coulomb friction is a sharp on/off sign function (see Fig. 5.10; $\beta = 100.0$ approaches the sign function). The direct implementation of the sign function is not compatible with the current DPIP algorithm implementation. To circumvent this problem, an approximation was introduced and implemented. The sign function was approximated with a hyperbolic tangent function given as

$$\alpha \, \text{sign}(\dot{\theta}) \approx \alpha \tanh(\beta\dot{\theta}).$$

Several varying slope β parameters are shown in Fig. 5.10. For the closed-loop reference angle DPIP-optimized trajectories, $\beta = 7.0$ was employed in the runs found in Section 5.3.5.

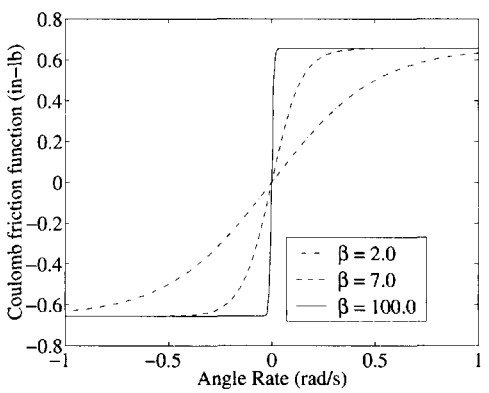


Figure 5.10. *Coulomb friction via smooth hyperbolic tangent function.*

The DPIP code requires four user-supplied problem specific functions as input [68]. These four functions define the nonlinear differential state equations, the cost function to be minimized, the state and control variable equality constraints, and the state and control variable inequality constraints. A driver routine is also included that helps define problem specifics and other iteration specific logistics. For the first DP optimization run (see Table 5.6), these functions are *exflink4a.m*, the flexible link driver script code; *esflink4a.m*, the dynamics script code; *coflink4a.m*, the cost function script code; *eqflink4a.m*, the equality constraints script code; and *inflink4a.m*, the inequality constraints script code. The script files are listed in Section B.2 of Appendix B.

5.3.5 Numerical Simulation Results

Open-loop minimum effort and minimum effort with bounds runs

Two different optimization runs were performed to determine open-loop minimum effort and minimum effort with bounds for the *fmincon* and DPIP designs, respectively. Both runs used a typical slew from an initial point A to a final point B. The slew was configured to move from 0° to $\theta_{desired} = 90°$ in a specified amount of time, t_f. The study was performed for both minimum control effort and minimum control effort with bounds. The summarized results are given in Table 5.6. The rows are defined as (1) run type, (2) initial u_i guess, (3) iteration number, (4) convergence, and (5) final time.

Table 5.6. *DPIP slewing flexible link optimization results.*

Run type	Minimum effort	Minimum effort with bounds
u_i guess	$0.2 \forall i$	$0.2 \forall i$
Iteration #	13	22
Convergence	9.206e−12	1.286e−13
t_f (sec)	1.0	1.0

The goal of these optimization runs was to generate optimized trajectories that demonstrate a minimum effort solution for a rest-to-rest maneuver. The same optimization trajectory is discovered for the minimum effort with bounds set on the control solution for the rest-to-rest maneuver. Numerical simulation results for each optimization trajectory are included for the hub positions and velocities for each design, *fmincon* and DPIP, respectively, in Fig. 5.11.

Both designs meet the original criteria and show different but acceptable solutions. For example, the DPIP design for the minimum control effort solution has a more aggressive solution with overall higher transient hub velocities. The related open-loop torque trajectories for each design and each trajectory type are shown in Fig. 5.11.

Note that, for the bounded solution (right side), the controls saturate at ±2.5 in-lb. The corresponding flexible modes and mode rates for each design and trajectory type are given in Fig. 5.12. The modes and their rates meet the initial and final end conditions associated with the optimized trajectories. The DPIP algorithm produced efficient trajectories that required less computational time to determine than the MATLAB Optimization Toolbox *fmincon* function. However, the DPIP solutions resulted in higher transients for most of the variables (hub velocities, flex modes, and flex mode rates).

Closed-loop reference angle trajectory run

The final run investigated considers a closed-loop configuration where an optimized reference angle trajectory is desired. Originally, an RQP design [77] was used to determine a solution, which was confirmed with an experimental setup. Both the numerical and experimental results are shown in Fig. 5.13 for the hub position and velocity. Although the RQP optimizer stopped short of full convergence after 100 iterations, it was close to the end conditions and provided useful results that visually showed Coulomb friction compensation. This is identified as a small bump at the 1.6-sec to 2.0-sec region for the θ_{ref} plot in Fig. 5.13 (left side).

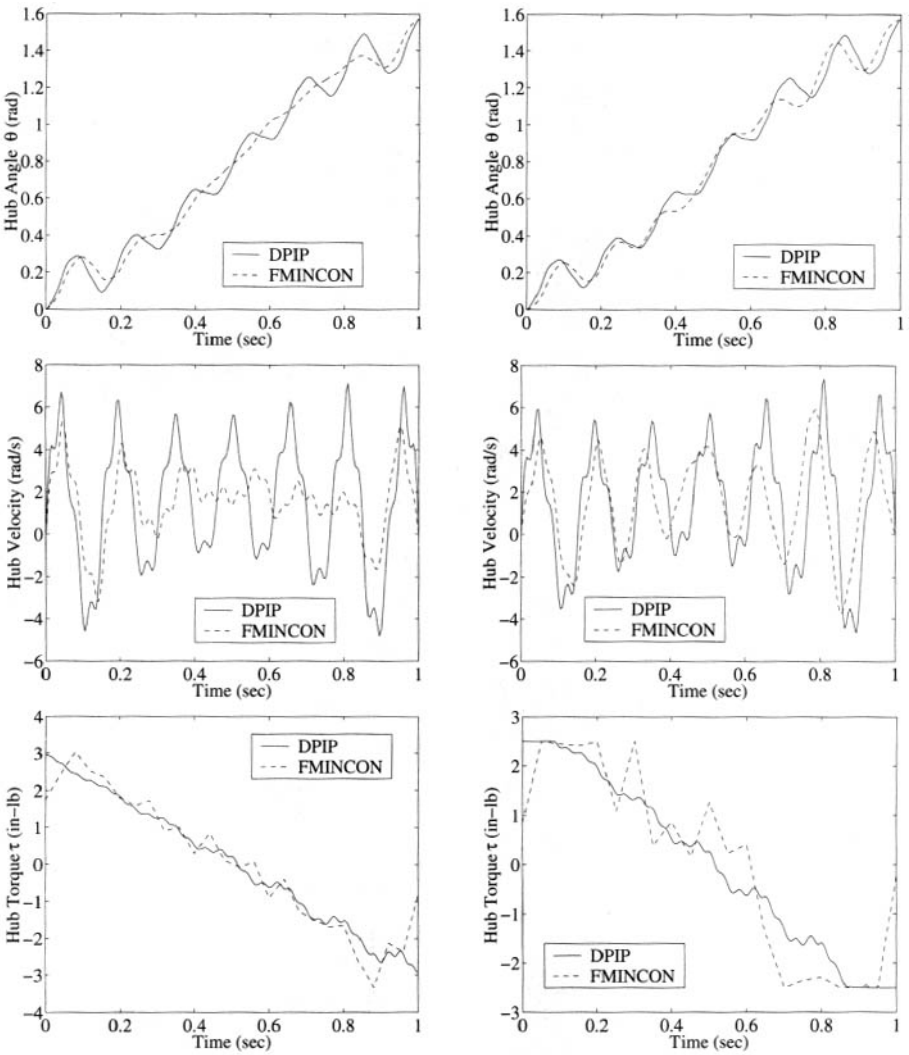


Figure 5.11. *Hub angles, velocities, and torques for both minimum control effort and minimum control effort with bounds.*

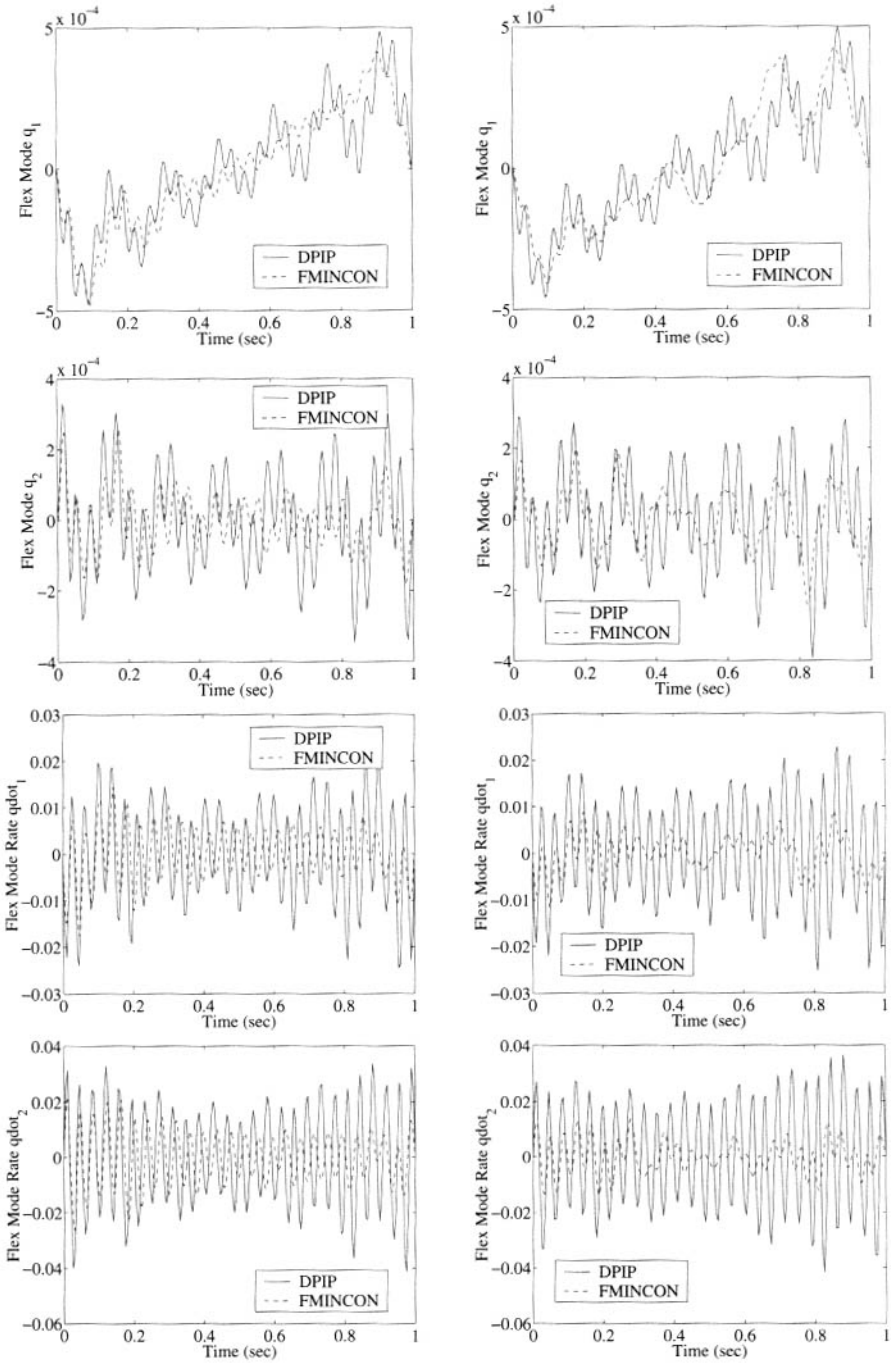


Figure 5.12. *Flex modes and mode rates for both minimum control effort and minimum control effort with bounds.*

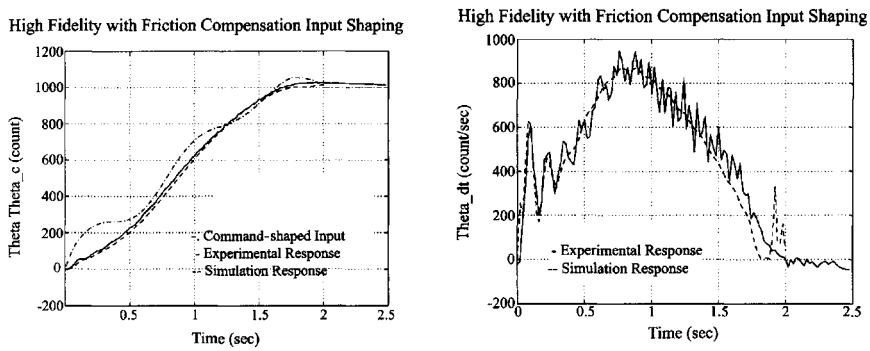


Figure 5.13. *RQP experimental position and velocity responses from* [77].

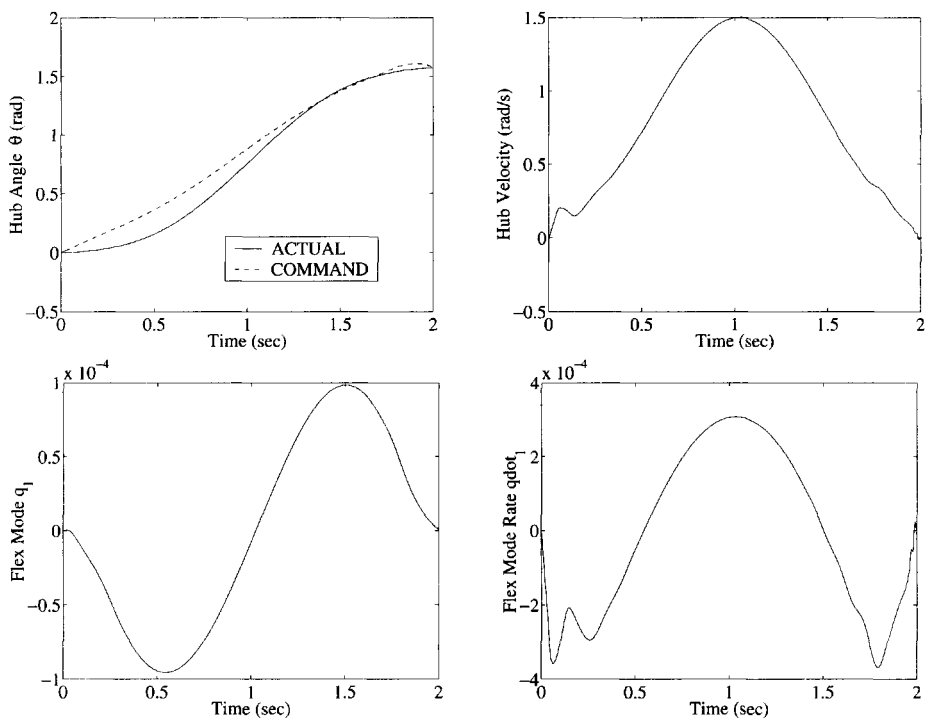


Figure 5.14. *Hub position and velocity; flex mode and rate for DPIP solution.*

In comparison, a DPIP-optimized solution was generated with the same model used for the RQP design. The actual hub position and optimized reference angle trajectory are shown in Fig. 5.14 (upper left side). The corresponding hub velocity is shown on the upper right side of Fig. 5.14. The first mode shape response is plotted in the lower left side of Fig. 5.14. The first mode rate response is plotted in the lower right side of Fig. 5.14. These results meet

the criteria discussed earlier for a rest-to-rest response trajectory. The reference trajectory also shows a similar trend as to the inclusion of the Coulomb friction term discussed for the RQP design. The nonlinear Coulomb friction model was substituted with an analogous function that includes a smoothing parameter for the slope associated with the hyperbolic tangent function. Without this substitution, the DPIP model was unable to converge and oscillated between several upper and lower bounds. Even with the substitution, the final solution was only able to converge for a relatively small bias error. This is not necessarily a disadvantage but rather a numerical implementation anomaly.

5.3.6 Case Study 2 Discussion (Summary/Conclusions)

The objective of this case study was to design feedforward commands for a slewing flexible link to produce rest-to-rest, residual–vibration-free maneuvers. This required the development of the dynamic system equations of motion, numerical implementation, optimized trajectory design, and verification using numerical simulations.

A Lagrangian energy method was used to develop the equations of motion. This development included the centrifugal stiffening effect. Two polynomial-type mode shapes were employed to develop the link equations, along with the inclusion of actuator dynamics and nonlinear friction effects. A fundamental slew from point A to point B in a specified amount of time was used to evaluate all of the optimization trajectories. For the open-loop optimized trajectories, two separate scenarios were investigated: minimum effort and minimum effort with bounds. The first scenario produced optimized trajectories from both the MATLAB *fmincon* function and the DPIP algorithm that represent conservative energy solutions. In the second scenario, upper and lower bounds were applied to the hub torque that represent motor/actuator constraints that are present with any real hardware setup. Both optimization designs produced sufficient trajectories that met the original rest-to-rest maneuver criteria. The MATLAB *fmincon* function required a higher number of iterations to converge to a final solution than the DPIP algorithm. However, the DPIP solution produced higher overall transients between the starting point and the final destination, for the minimum control effort scenario, than the MATLAB *fmincon* function.

In the final optimization study, the DPIP algorithm was compared with an RQP optimization solution that was used with actual hardware [76]. The RQP optimizer produced optimal reference angle trajectories while addressing the Coulomb friction effect. This is typical of actual hardware implementation. Such real-world dynamic effects as friction and closed-loop control become a reality that needs to be overcome between theory and actual implementation.

The RQP-optimized trajectory was tested on the hardware and showed good correlation with the predicted numerical simulation responses and produced minimal residual vibration. The small remaining oscillations were considered to be unobservable and/or unmodeled dynamics such as torsion or higher bending modes. An alternative result was also produced that used the same dynamic model as the RQP design but with the DPIP algorithm. It also included the effect of Coulomb friction that required an implementation modification to produce a successful result.

In summary, the DPIP algorithm was found to be an effective optimization procedure for generating discrete control trajectories that are subject to both equality and inequality

constraints. Several DP optimization designs were generated for both open- and closed-loop controls for the slewing flexible link case study and were verified through numerical simulations.

5.4 Case Study 3: Flexible Planar Arm

5.4.1 Introduction

In this case study, a dynamic system is reviewed that contains both gross and fine motion capability. An ideal representative model is the planar two-link flexible arm. Several key points are addressed:

- Manipulators with lightweight members are subject to flexibility/vibration.
- Point-to-point maneuvers that minimize vibration can reduce delay times.
- DP optimization can be employed to find discretized optimal control solutions while keeping residual vibrations to a minimum.

Numerous application areas can benefit from robotic systems with lightweight flexible members. Space applications need lightweight robotic manipulators to reduce launch costs, power consumption, and storage volume of the robot. Both extravehicular and intravehicular activities use automation. These activities require accurate positioning of payloads while keeping vibrations of the robotic mechanisms to a minimum [79, 80]. To avoid large vibrations, the current space shuttle robotic arm must operate very slowly. A study performed by Newsom et al. [81] estimated that 30% of the operational time of the arm is spent waiting for the arm vibrations to settle out. Reduction of delays during robot operation can greatly reduce costs. Industrial robots have used massive structures to suppress deflection, resulting in slow motion and high power consumption. Current industrial robots typically have a high ratio between mass of the robot and mass of the payload, often ranging from 10:1 to 100:1. Along with improved industrial robot designs with lighter structures come flexibility/vibration issues. Assembly mechanisms and manufacturing facilities can benefit from improved accuracy, hence improved efficiency and reduced maintenance. For example, reducing seek times while maintaining high precision in positional accuracy for computer hard drives is of critical importance. Another application that can benefit is remote operation of robotic systems for hazardous workspace/storage tank servicing (a Department of Energy need). Underground storage tank remediation requires robots slender enough to fit through a small opening, yet able to maneuver in a large workspace once inside [82, 83, 84, 85]. This leads to flexibility/vibration of the robotic manipulator.

Optimized feedforward trajectories can address point-to-point operations with ideally zero residual vibrations. By including these flexibility effects in the dynamical model, DP optimization techniques can be used to find optimized trajectories that minimize vibrations. In the past, other optimization techniques that employ RQP to solve for minimum-time solutions for straight line trajectories were reported by Eisler et al. [5]. In addition, the details of applying RQP optimization to flexible manipulators are reviewed in Robinett et al. [69].

The objective of this case study is to use a discrete dynamic programming (DDP) approach to generate optimal feedforward paths for the point-to-point maneuvering of flexible robots while minimizing vibrations. The following steps will accomplish this.

Step 1: Derive the equations of motion for a two-link flexible planar robot arm.

Step 2: Develop a numerical simulation of the two-link flexible planar robot arm.

Step 3: Solve a trajectory optimization problem using DP design.

Step 4: Verify the design with numerical simulations.

Step 5: Summarize the case study findings.

5.4.2 Model Development

General development

Initially, a general finite element procedure is developed and later applied to the planar flexible robot arm model. The details of this modeling procedure are given in Robinett et al. [69]. The key points of the method are summarized below:

- Links are modeled using specially formulated beam elements.
- Generalized coordinates are nodal rotations of the beam elements.
- Equations of motion are obtained using Lagrange's equations.

The assumed form of the kinematics is given by

$$x(s,t) = x_i(t) + \int_0^s \cos\theta(r,t)dr, \tag{5.35}$$

$$y(s,t) = y_i(t) + \int_0^s \sin\theta(r,t)dr, \tag{5.36}$$

where the meanings of x, y, and θ are shown in Fig. 5.15.

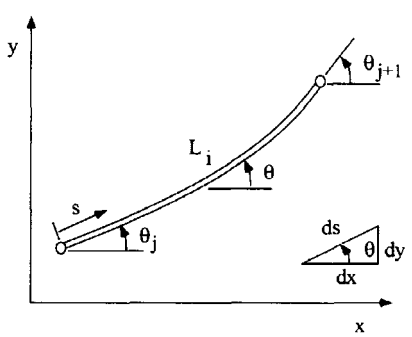


Figure 5.15. *Kinematic definitions for the ith element.*

The integrals in (5.35) and (5.36) are evaluated by assuming each integrand is constant and equal to the average of its values at the endpoints. Thus,

$$x(s,t) = x_i(t) + \frac{1}{2}\int_0^s (\cos\theta_j(t) + \cos\theta_{j+1}(t))dr, \tag{5.37}$$

$$y(s,t) = x_i(t) + \frac{1}{2}\int_0^s (\sin\theta_j(t) + \sin\theta_{j+1}(t))dr. \tag{5.38}$$

The approximation used in (5.37) and (5.38) is justified as long as $(\theta_{j+1} - \theta_j) \ll 1$. Carrying out the integrations in (5.37) and (5.38) yields

$$x(s,t) = x_i + \frac{1}{2}s(\cos\theta_j + \cos\theta_{j+1}), \tag{5.39}$$

$$y(s,t) = y_i + \frac{1}{2}s(\sin\theta_j + \sin\theta_{j+1}). \tag{5.40}$$

Differentiating (5.39) and (5.40) with respect to time, one obtains

$$\dot{x}(s,t) = \dot{x}_i - \frac{1}{2}s(\dot{\theta}_j\sin\theta_j + \dot{\theta}_{j+1}\sin\theta_{j+1}), \tag{5.41}$$

$$\dot{y}(s,t) = \dot{y}_i + \frac{1}{2}s(\dot{\theta}_j\cos\theta_j + \dot{\theta}_{j+1}\cos\theta_{j+1}). \tag{5.42}$$

Setting $s = L_i$ in (5.41) and (5.42) yields

$$\dot{x}_{i+1} = \dot{x}_i - \frac{1}{2}L_i(\dot{\theta}_j\sin\theta_j + \dot{\theta}_{j+1}\sin\theta_{j+1}), \tag{5.43}$$

$$\dot{y}_{i+1} = \dot{y}_i + \frac{1}{2}L_i(\dot{\theta}_j\cos\theta_j + \dot{\theta}_{j+1}\cos\theta_{j+1}). \tag{5.44}$$

Combining (5.41)–(5.44) leads to

$$\dot{x}(s,t) = \left(1 - \frac{s}{L_i}\right)\dot{x}_i + \left(\frac{s}{L_i}\right)\dot{x}_{i+1}, \tag{5.45}$$

$$\dot{y}(s,t) = \left(1 - \frac{s}{L_i}\right)\dot{y}_i + \left(\frac{s}{L_i}\right)\dot{y}_{i+1}. \tag{5.46}$$

The kinetic energy of the ith element is given by

$$T_i = \frac{1}{2}\int_0^{L_i} \rho_i(s)(\dot{x}^2 + \dot{y}^2)ds, \tag{5.47}$$

where $\rho_i(s)$ is the mass density per unit length of the element. Substituting (5.45) and (5.46) into (5.47) and performing the integrations yields

$$T_i = \frac{1}{2}([\dot{x}_i\ \dot{x}_{i+1}]\mathbf{H}_i[\dot{x}_i\ \dot{x}_{i+1}]^T + [\dot{y}_i\ \dot{y}_{i+1}]\mathbf{H}_i[\dot{y}_i\ \dot{y}_{i+1}]^T), \tag{5.48}$$

where

$$\mathbf{H}_i = \begin{bmatrix} \int_0^{L_i}\rho_i(s)\left(1 - \frac{s}{L_i}\right)^2 ds & \int_0^{L_i}\rho_i(s)\left(\frac{s}{L_i}\right)\left(1 - \frac{s}{L_i}\right)ds \\ \int_0^{L_i}\rho_i(s)\left(\frac{s}{L_i}\right)\left(1 - \frac{s}{L_i}\right)ds & \int_0^{L_i}\rho_i(s)\left(\frac{s}{L_i}\right)^2 ds \end{bmatrix}. \tag{5.49}$$

It proves useful to define

$$m_{rs}^{i} = \frac{\partial^2 T_i}{\partial \dot{\theta}_r \partial \dot{\theta}_s}. \tag{5.50}$$

Substituting (5.48) into (5.50), one obtains

$$\begin{aligned} m_{rs}^{i} &= \frac{1}{2}\frac{\partial^2}{\partial \dot{\theta}_r \partial \dot{\theta}_s}\left([\dot{x}_i \ \dot{x}_{i+1}]\mathbf{H}_i[\dot{x}_i \ \dot{x}_{i+1}]^T + [\dot{y}_i \ \dot{y}_{i+1}]\mathbf{H}_i[\dot{y}_i \ \dot{y}_{i+1}]^T\right) \\ &= \frac{\partial}{\partial \dot{\theta}_r}\left(\begin{bmatrix}\frac{\partial \dot{x}_i}{\partial \dot{\theta}_s} & \frac{\partial \dot{x}_{i+1}}{\partial \dot{\theta}_s}\end{bmatrix}\mathbf{H}_i\left[\dot{x}_i \ \dot{x}_{i+1}\right]^T + \begin{bmatrix}\frac{\partial \dot{y}_i}{\partial \dot{\theta}_s} & \frac{\partial \dot{y}_{i+1}}{\partial \dot{\theta}_s}\end{bmatrix}\mathbf{H}_i\left[\dot{y}_i \ \dot{y}_{i+1}\right]^T\right) \\ &= \begin{bmatrix}\frac{\partial \dot{x}_i}{\partial \dot{\theta}_s} & \frac{\partial \dot{x}_{i+1}}{\partial \dot{\theta}_s}\end{bmatrix}\mathbf{H}_i\begin{bmatrix}\frac{\partial \dot{x}_i}{\partial \dot{\theta}_r} & \frac{\partial \dot{x}_{i+1}}{\partial \dot{\theta}_r}\end{bmatrix}^T + \begin{bmatrix}\frac{\partial \dot{y}_i}{\partial \dot{\theta}_s} & \frac{\partial \dot{y}_{i+1}}{\partial \dot{\theta}_s}\end{bmatrix}\mathbf{H}_i\begin{bmatrix}\frac{\partial \dot{y}_i}{\partial \dot{\theta}_r} & \frac{\partial \dot{y}_{i+1}}{\partial \dot{\theta}_r}\end{bmatrix}^T. \end{aligned} \tag{5.51}$$

The kinetic energy of the entire system is the sum of the kinetic energies of each element and any rotational inertias. Thus,

$$T = \sum_{i=1}^{N_e} T_i + \frac{1}{2}\sum_{j=1}^{N_a} J_j \dot{\theta}_j^2, \tag{5.52}$$

where J_j is the rotational inertia associated with θ_j. It is straightforward to show that

$$T = \frac{1}{2}\sum_{r=1}^{N_a}\sum_{s=1}^{N_a} m_{rs}\dot{\theta}_r\dot{\theta}_s. \tag{5.53}$$

Taking the partial derivative of both sides of (5.53) with respect to $\dot{\theta}_r$ and $\dot{\theta}_s$, one obtains

$$\begin{aligned} m_{rs} &= \frac{\partial^2 T}{\partial \dot{\theta}_r \partial \dot{\theta}_s} \\ &\overset{(5.52)}{=} \sum_{i=1}^{N_e}\frac{\partial^2 T_i}{\partial \dot{\theta}_r \partial \dot{\theta}_s} + \sum_{j=1}^{N_a} J_j \delta_{js}\delta_{jr} \\ &\overset{(5.50)}{=} \sum_{i=1}^{N_e} m_{rs}^{i} + \sum_{j=1}^{N_a} J_j \delta_{js}\delta_{jr}, \end{aligned} \tag{5.54}$$

where

$$\delta_{js} = \begin{cases} 1 & \text{if } j = s, \\ 0 & \text{if } j \neq s. \end{cases} \tag{5.55}$$

The strain energy of the ith element is given by

$$U^i = \frac{1}{2}(EI)_i \frac{(\theta_{j+1} - \theta_j)^2}{L_i}, \tag{5.56}$$

where $(EI)_i$ is the flexural rigidity of the ith element. The strain energy of the entire system is the sum of the strain energies of each element. Thus,

$$U = \frac{1}{2}\sum_{i=1}^{N_e}(EI)_i \frac{(\theta_{j+1} - \theta_j)^2}{L_i}. \tag{5.57}$$

Lagrange's equations are expressed as

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\theta}_k}\right) - \frac{\partial T}{\partial \theta_k} + \frac{\partial U}{\partial \theta_k} + \frac{\partial R}{\partial \dot{\theta}_k} = Q_k, \tag{5.58}$$

where R is the Rayleigh dissipation function and Q_k is the kth generalized force.

Starting with (5.53), one obtains

$$\frac{\partial T}{\partial \dot{\theta}_k} = \sum_{r=1}^{N_a} m_{kr}\dot{\theta}_r,$$

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\theta}_k}\right) = \sum_{r=1}^{N_a}\left(\dot{m}_{kr}\dot{\theta}_r + m_{kr}\ddot{\theta}_r\right)$$

$$= \sum_{r=1}^{N_a}\sum_{s=1}^{N_a}\left(\frac{\partial m_{kr}}{\partial \theta_s}\dot{\theta}_s\right)\dot{\theta}_r + \sum_{r=1}^{N_a} m_{kr}\ddot{\theta}_r, \tag{5.59}$$

and

$$\frac{\partial T}{\partial \theta_k} = \frac{1}{2}\sum_{r=1}^{N_a}\sum_{s=1}^{N_a}\frac{\partial m_{rs}}{\partial \theta_k}\dot{\theta}_r\dot{\theta}_s. \tag{5.60}$$

Substituting (5.59) and (5.60) into (5.58), one arrives at

$$\sum_{r=1}^{N_a} m_{kr}\ddot{\theta}_r + \sum_{r=1}^{N_a}\sum_{s=1}^{N_a}\left(\frac{\partial m_{kr}}{\partial \theta_s} - \frac{1}{2}\frac{\partial m_{rs}}{\partial \theta_k}\right)\dot{\theta}_r\dot{\theta}_s + \frac{\partial U}{\partial \theta_k} + \frac{\partial R}{\partial \dot{\theta}_k} = Q_k. \tag{5.61}$$

One can show from (5.43), (5.44), (5.51), and (5.54) that the m_{rs} terms can be expressed as

$$m_{rs} = \bar{m}_{rs}\cos(\theta_r - \theta_s), \tag{5.62}$$

where each $\bar{m}_{rs}$ term is a constant (this point will become evident in the example problem). Substitution of (5.62) into (5.61) leads to the simplification

$$\sum_{r=1}^{N_a}\bar{m}_{kr}\cos(\theta_k - \theta_r)\ddot{\theta}_r + \bar{m}_{kr}\dot{\theta}_r^2\sin(\theta_k - \theta_r) + \frac{\partial U}{\partial \theta_k} + \frac{\partial R}{\partial \dot{\theta}_k} = Q_k. \tag{5.63}$$

In summary, the equations of motion for the system can be derived with the following steps.

Step 1: Obtain expressions for $(\dot{x}_2, \ldots, \dot{x}_{N_e+1})$ and $(\dot{y}_2, \ldots, \dot{y}_{N_e+1})$ by recursively using (5.43) and (5.44), starting with $\dot{x}_1 = 0$ and $\dot{y}_1 = 0$.

Step 2: Obtain expressions for the partial derivatives $\partial \dot{x}_i / \partial \dot{\theta}_r$ $(i = 2, \ldots, N_e + 1; r = 1, \ldots, N_a)$ by differentiating the results of Step 1.

Step 3: Calculate the integrals appearing in (5.49) for all of the elements.

Step 4: Obtain expressions for the m_{rs}^i terms for all of the elements by substituting the results of Steps 2 and 3 into (5.51).

Step 5: Obtain expressions for the elements of the mass matrix m_{rs} by substituting the results of Step 4 into (5.54). Identify the $\bar{m}_{rs}$ terms by inspection.

Step 6: Obtain an expression for the strain energy, U, using (5.57).

Step 7: Determine the generalized forces, Q_k, and specify the Rayleigh dissipation function, R.

Step 8: Assemble the equations of motion per (5.63).

The equations of motion given by (5.63) are in general nonlinear form and are integrated to produce numerical simulations. If an explicit numerical integration scheme is used (e.g., RK), care must be taken when modeling stiff members. The formulation presented above also accounts for the effects of geometric (centrifugal) stiffening. Further discussion can be found in Robinett et al. [69].

Apply the method to a planar flexible two-link arm

A two-link flexible robot arm is modeled using a single element for each link. Both links have torsional rigidities of EI and uniform mass densities of ρ. Other parameters defining the problem are shown in Fig. 5.16, and their numerical values are listed in Table 5.7.

Step 1:

$$\dot{x}_2 = \frac{-L_1(\dot{\theta}_1 \sin\theta_1 + \dot{\theta}_2 \sin\theta_2)}{2},$$

$$\dot{y}_2 = \frac{L_1(\dot{\theta}_1 \cos\theta_1 + \dot{\theta}_2 \cos\theta_2)}{2},$$

$$\dot{x}_3 = \frac{-L_1(\dot{\theta}_1 \sin\theta_1 + \dot{\theta}_2 \sin\theta_2)}{2} - \frac{L_2(\dot{\theta}_3 \sin\theta_3 + \dot{\theta}_4 \sin\theta_4)}{2},$$

$$\dot{y}_3 = \frac{L_1(\dot{\theta}_1 \cos\theta_1 + \dot{\theta}_2 \cos\theta_2)}{2} + \frac{L_2(\dot{\theta}_3 \cos\theta_3 + \dot{\theta}_4 \cos\theta_4)}{2}.$$

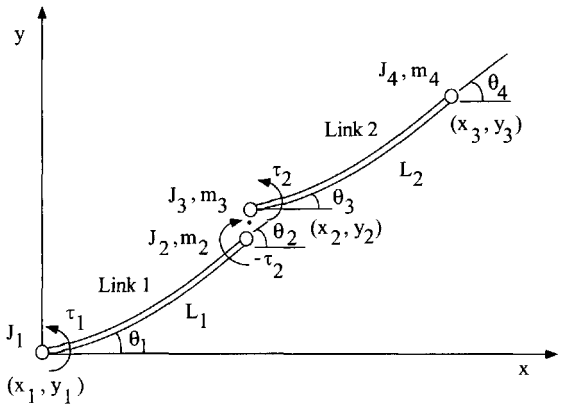


Figure 5.16. *Two-link planar flexible robot with element definitions.*

Table 5.7. *Flexible robot arm physical parameters.*

Description	Symbol	Value	Unit
Moment of inertia at 1st node	J_1	1.0	kg-m^2
Moment of inertia at 2nd node	J_2	1.0	kg-m^2
Moment of inertia at 3rd node	J_3	1.0	kg-m^2
Moment of inertia at 4th node	J_4	0.0	kg-m^2
Mass at 2nd node	m_2	5.415	kg
Mass at 3rd node	m_3	0.830	kg
Mass at 4th node	m_4	0.0	kg
Link 1 length	L_1	0.5969	m
Link 2 length	L_2	0.4842	m
Link 1 thickness	t_1	0.005537	m
Link 2 thickness	t_2	0.001574	m
Link 1 width	w_1	0.1524	m
Link 2 width	w_2	0.0762	m
Viscous friction joint 1	c_1	0.05	N-m/(rad/sec)
Viscous friction joint 2	c_2	0.05	N-m/(rad/sec)
Density for each link	ρ	2700.0	kg/m^3
Modulus of elasticity	E	6.9e10	N/m^2

Step 2:

$$\frac{\partial \dot{x}_2}{\partial \dot{\theta}_1} = \frac{-L_1 \sin\theta_1}{2}, \quad \frac{\partial \dot{x}_2}{\partial \dot{\theta}_2} = \frac{-L_1 \sin\theta_2}{2}, \quad \frac{\partial \dot{x}_2}{\partial \dot{\theta}_3} = 0, \quad \frac{\partial \dot{x}_2}{\partial \dot{\theta}_4} = 0,$$

$$\frac{\partial \dot{y}_2}{\partial \dot{\theta}_1} = \frac{L_1 \cos\theta_1}{2}, \quad \frac{\partial \dot{y}_2}{\partial \dot{\theta}_2} = \frac{L_1 \cos\theta_2}{2}, \quad \frac{\partial \dot{y}_2}{\partial \dot{\theta}_3} = 0, \quad \frac{\partial \dot{y}_2}{\partial \dot{\theta}_4} = 0,$$

$$\frac{\partial \dot{x}_3}{\partial \dot{\theta}_1} = \frac{-L_1 \sin\theta_1}{2}, \quad \frac{\partial \dot{x}_3}{\partial \dot{\theta}_2} = \frac{-L_1 \sin\theta_2}{2}, \quad \frac{\partial \dot{x}_3}{\partial \dot{\theta}_3} = \frac{-L_2 \sin\theta_3}{2}, \quad \frac{\partial \dot{x}_3}{\partial \dot{\theta}_4} = \frac{-L_2 \sin\theta_4}{2},$$

$$\frac{\partial \dot{y}_3}{\partial \dot{\theta}_1} = \frac{L_1 \cos\theta_1}{2}, \quad \frac{\partial \dot{y}_3}{\partial \dot{\theta}_2} = \frac{L_1 \cos\theta_2}{2}, \quad \frac{\partial \dot{y}_3}{\partial \dot{\theta}_3} = \frac{L_2 \cos\theta_3}{2}, \quad \frac{\partial \dot{y}_3}{\partial \dot{\theta}_4} = \frac{L_2 \cos\theta_4}{2}.$$

Step 3: As an example, look at $(h_1)_{22}$:

$$
\begin{aligned}
(h_1)_{22} &\overset{(5.49)}{=} \int_0^{L_1} \rho \left(\frac{s}{L_1}\right)^2 ds \\
&= \rho \left.\frac{s^3}{(3L_1^2)}\right|_0^{L_1} + m_2 \\
&= \rho \frac{L_1}{3} + m_2.
\end{aligned}
$$

Following a similar procedure for the other elements of $\mathbf{H}_1$ and $\mathbf{H}_2$, one obtains

$$
\mathbf{H}_1 = \begin{bmatrix} \frac{1}{3}\rho L_1 & \frac{1}{6}\rho L_1 \\ \frac{1}{6}\rho L_1 & \frac{1}{3}\rho L_1 + m_2 \end{bmatrix},
$$

$$
\mathbf{H}_2 = \begin{bmatrix} \frac{1}{3}\rho L_2 + m_3 & \frac{1}{6}\rho L_2 \\ \frac{1}{6}\rho L_2 & \frac{1}{3}\rho L_2 + m_4 \end{bmatrix}.
$$

Step 4: As an example, look at m_{12}^2:

$$
\begin{aligned}
m_{12}^2 &\overset{(5.51)}{=} \begin{bmatrix} \frac{\partial \dot{x}_2}{\partial \dot{\theta}_2} & \frac{\partial \dot{x}_3}{\partial \dot{\theta}_2} \end{bmatrix} \mathbf{H}_2 \begin{bmatrix} \frac{\partial \dot{x}_2}{\partial \dot{\theta}_1} & \frac{\partial \dot{x}_3}{\partial \dot{\theta}_1} \end{bmatrix}^T + \begin{bmatrix} \frac{\partial \dot{y}_2}{\partial \dot{\theta}_2} & \frac{\partial \dot{y}_3}{\partial \dot{\theta}_2} \end{bmatrix} \mathbf{H}_i \begin{bmatrix} \frac{\partial \dot{y}_2}{\partial \dot{\theta}_1} & \frac{\partial \dot{y}_3}{\partial \dot{\theta}_1} \end{bmatrix}^T \\
&= \frac{1}{4} L_1^2 [\sin\theta_2 \ \ \sin\theta_2] \begin{bmatrix} \frac{1}{3}\rho L_2 + m_3 & \frac{1}{6}\rho L_2 \\ \frac{1}{6}\rho L_2 & \frac{1}{3}\rho L_2 + m_4 \end{bmatrix} \begin{bmatrix} \sin\theta_1 \\ \sin\theta_1 \end{bmatrix} \\
&\quad + \frac{1}{4} L_1^2 [\cos\theta_2 \ \ \cos\theta_2] \begin{bmatrix} \frac{1}{3}\rho L_2 + m_3 & \frac{1}{6}\rho L_2 \\ \frac{1}{6}\rho L_2 & \frac{1}{3}\rho L_2 + m_4 \end{bmatrix} \begin{bmatrix} \cos\theta_1 \\ \cos\theta_1 \end{bmatrix} \\
&= \frac{1}{4} L_1^2 (\rho L_2 + m_3 + m_4) \cos(\theta_1 - \theta_2).
\end{aligned}
$$

All the values of m_{rs}^i can be calculated in a similar manner and are given below for $s \geq r$. Notice that by definition $m_{rs}^i = m_{sr}^i$. Only nonzero values are reported:

$$
m_{11}^1 = \left[L_1^2 \left(\frac{\rho L_1}{12} + \frac{m_2}{4}\right)\right], \qquad m_{12}^1 = \left[L_1^2 \left(\frac{\rho L_1}{12} + \frac{m_2}{4}\right)\right] \cos(\theta_1 - \theta_2),
$$

$$
m_{22}^1 = \left[L_1^2 \left(\frac{\rho L_1}{12} + \frac{m_2}{4}\right)\right], \qquad m_{12}^2 = \left[L_1^2 \frac{(\rho L_2 + m_3 + m_4)}{4}\right] \cos(\theta_1 - \theta_2),
$$

$$m_{11}^2 = \left[L_1^2 \frac{(\rho L_2 + m_3 + m_4)}{4}\right], \qquad m_{13}^2 = \left[L_1 L_2 \left(\frac{\rho L_2}{8} + \frac{m_4}{4}\right)\right] \cos(\theta_1 - \theta_3),$$

$$m_{22}^2 = \left[L_1^2 \frac{(\rho L_2 + m_3 + m_4)}{4}\right], \qquad m_{14}^2 = \left[L_1 L_2 \left(\frac{\rho L_2}{8} + \frac{m_4}{4}\right)\right] \cos(\theta_1 - \theta_4),$$

$$m_{33}^2 = \left[L_2^2 \left(\frac{\rho L_2}{12} + \frac{m_4}{4}\right)\right], \qquad m_{24}^2 = \left[L_1 L_2 \left(\frac{\rho L_2}{8} + \frac{m_4}{4}\right)\right] \cos(\theta_2 - \theta_4),$$

$$m_{44}^2 = \left[L_2^2 \left(\frac{\rho L_2}{12} + \frac{m_4}{4}\right)\right], \qquad m_{34}^2 = \left[L_2^2 \left(\frac{\rho L_2}{12} + \frac{m_4}{4}\right)\right] \cos(\theta_3 - \theta_4),$$

$$m_{23}^2 = \left[L_1 L_2 \left(\frac{\rho L_2}{8} + \frac{m_4}{4}\right)\right] \cos(\theta_2 - \theta_3).$$

Step 5: The elements of the mass matrix are obtained by substituting the results of the previous step into (5.54). It is clear from the results of Step 4 that one can express the elements of the mass matrix in the simple form $m_{rs} = \bar{m}_{rs} \cos(\theta_r - \theta_s)$, where $\bar{m}_{rs}$ are constants (see (5.62)). The values of $\bar{m}_{rs}$ are given below for $s \geq r$. Note that $\bar{m}_{rs} = \bar{m}_{sr}$:

$$\bar{m}_{11} = J_1 + L_1^2 \left(\frac{\rho L_1}{12} + \frac{\rho L_2}{4} + \frac{(m_2 + m_3 + m_4)}{4}\right), \qquad \bar{m}_{23} = L_1 L_2 \left(\frac{\rho L_2}{8} + \frac{m_4}{4}\right),$$

$$\bar{m}_{12} = L_1^2 \left(\frac{\rho L_1}{12} + \frac{\rho L_2}{4} + \frac{(m_2 + m_3 + m_4)}{4}\right), \qquad \bar{m}_{24} = L_1 L_2 \left(\frac{\rho L_2}{8} + \frac{m_4}{4}\right),$$

$$\bar{m}_{13} = L_1 L_2 \left(\frac{\rho L_2}{8} + \frac{m_4}{4}\right), \qquad \bar{m}_{33} = J_3 + L_2^2 \left(\frac{\rho L_2}{12} + \frac{m_4}{4}\right),$$

$$\bar{m}_{14} = L_1 L_2 \left(\frac{\rho L_2}{8} + \frac{m_4}{4}\right), \qquad \bar{m}_{34} = L_2^2 \left(\frac{\rho L_2}{12} + \frac{m_4}{4}\right),$$

$$\bar{m}_{22} = J_2 + L_1^2 \left(\frac{\rho L_1}{12} + \frac{\rho L_2}{4} + \frac{(m_2 + m_3 + m_4)}{4}\right), \qquad \bar{m}_{44} = J_4 + L_2^2 \left(\frac{\rho L_2}{12} + \frac{m_4}{4}\right).$$

Step 6: The strain energy is given as

$$U = \frac{1}{2} \sum_{i=1}^{N_e} (EI)_i \frac{(\theta_{j+1} - \theta_j)^2}{L_i}$$

$$= \frac{1}{2} \left[(EI)_1 \frac{(\theta_2 - \theta_1)^2}{L_1} + (EI)_2 \frac{(\theta_4 - \theta_3)^2}{L_2}\right].$$

Compute the partial derivatives as

$$\frac{\partial U}{\partial \theta_1} = -\frac{(EI)_1}{L_1}(\theta_2 - \theta_1),$$

$$\frac{\partial U}{\partial \theta_2} = \frac{(EI)_1}{L_1}(\theta_2 - \theta_1),$$

$$\frac{\partial U}{\partial \theta_3} = -\frac{(EI)_2}{L_2}(\theta_4 - \theta_3),$$

$$\frac{\partial U}{\partial \theta_4} = \frac{(EI)_2}{L_2}(\theta_4 - \theta_3).$$

Step 7: The joint torques are defined as open loop, for which the generalized forces are

$$Q_1 = \tau_1,$$

$$Q_2 = -\tau_2,$$

$$Q_3 = \tau_2,$$

$$Q_4 = 0.$$

The Rayleigh dissipation function is of the form

$$R = \frac{1}{2}(c_1\dot{\theta}_1^2 + c_2(\dot{\theta}_3 - \dot{\theta}_2)^2).$$

Compute the partial derivatives as

$$\frac{\partial R}{\partial \dot{\theta}_1} = c_1\dot{\theta}_1,$$

$$\frac{\partial R}{\partial \dot{\theta}_2} = -c_2(\dot{\theta}_3 - \dot{\theta}_2),$$

$$\frac{\partial R}{\partial \dot{\theta}_3} = c_2(\dot{\theta}_3 - \dot{\theta}_2),$$

$$\frac{\partial R}{\partial \dot{\theta}_4} = 0.$$

Step 8: The general form of the dynamic equations of motion is

$$\sum_{r=1}^{N_a} \bar{m}_{kr}\cos(\theta_k - \theta_r)\ddot{\theta}_r + \bar{m}_{kr}\dot{\theta}_r^2 \sin(\theta_k - \theta_r) + \frac{\partial U}{\partial \theta_k} + \frac{\partial R}{\partial \dot{\theta}_k} = Q_k.$$

For $k = 1$, this general equation gives

$$\bar{m}_{11}\ddot{\theta}_1 + \bar{m}_{12}\cos(\theta_1 - \theta_2)\ddot{\theta}_2 + \bar{m}_{13}\cos(\theta_1 - \theta_3)\ddot{\theta}_3 + \bar{m}_{14}\cos(\theta_1 - \theta_4)\ddot{\theta}_4 + \bar{m}_{12}\dot{\theta}_2^2\sin(\theta_1 - \theta_2) + \bar{m}_{13}\dot{\theta}_3^2\sin(\theta_1 - \theta_3) + \bar{m}_{14}\dot{\theta}_4^2\sin(\theta_1 - \theta_4) + \frac{\partial U}{\partial \theta_1} + \frac{\partial R}{\partial \dot{\theta}_1} = Q_1.$$

Similar equations exist for $k = 2, 3, 4$.

Assembling Lagrange's equations of motion yields

$$\begin{Bmatrix} \tau_1 \\ -\tau_2 \\ \tau_2 \\ 0 \end{Bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{12} & M_{22} & M_{23} & M_{24} \\ M_{13} & M_{23} & M_{33} & M_{34} \\ M_{14} & M_{24} & M_{34} & M_{44} \end{bmatrix} \begin{Bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \\ \ddot{\theta}_4 \end{Bmatrix} + \begin{Bmatrix} C_{12}^{(1)}\dot{\theta}_2^2 + C_{13}^{(1)}\dot{\theta}_3^2 + C_{14}^{(1)}\dot{\theta}_4^2 \\ -C_{12}^{(1)}\dot{\theta}_1^2 + C_{23}^{(2)}\dot{\theta}_3^2 + C_{24}^{(2)}\dot{\theta}_4^2 \\ -C_{13}^{(1)}\dot{\theta}_1^2 - C_{23}^{(2)}\dot{\theta}_2^2 + C_{34}^{(3)}\dot{\theta}_4^2 \\ -C_{14}^{(1)}\dot{\theta}_1^2 - C_{24}^{(2)}\dot{\theta}_2^2 - C_{34}^{(3)}\dot{\theta}_3^2 \end{Bmatrix}$$

$$+ \begin{bmatrix} c_1 & 0 & 0 & 0 \\ 0 & c_2 & -c_2 & 0 \\ 0 & -c_2 & c_2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \end{Bmatrix} + \begin{bmatrix} \frac{(EI)_1}{L_1} & -\frac{(EI)_1}{L_1} & 0 & 0 \\ -\frac{(EI)_1}{L_1} & \frac{(EI)_1}{L_1} & 0 & 0 \\ 0 & 0 & \frac{(EI)_2}{L_2} & -\frac{(EI)_2}{L_2} \\ 0 & 0 & -\frac{(EI)_2}{L_2} & \frac{(EI)_2}{L_2} \end{bmatrix} \begin{Bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{Bmatrix}, \tag{5.64}$$

where

$$M_{11} = \bar{m}_{11}, \qquad M_{24} = \bar{m}_{24}\cos(\theta_2 - \theta_4), \qquad C_{14}^{(1)} = \bar{m}_{14}\sin(\theta_1 - \theta_4),$$

$$M_{12} = \bar{m}_{12}\cos(\theta_1 - \theta_2), \qquad M_{33} = \bar{m}_{33}, \qquad C_{23}^{(2)} = \bar{m}_{23}\sin(\theta_2 - \theta_3),$$

$$M_{13} = \bar{m}_{13}\cos(\theta_1 - \theta_3), \qquad M_{34} = \bar{m}_{34}\cos(\theta_3 - \theta_4), \qquad C_{24}^{(2)} = \bar{m}_{24}\sin(\theta_2 - \theta_4),$$

$$M_{14} = \bar{m}_{14}\cos(\theta_1 - \theta_4), \qquad M_{44} = \bar{m}_{44}, \qquad C_{34}^{(3)} = \bar{m}_{34}\sin(\theta_3 - \theta_4),$$

$$M_{22} = \bar{m}_{22}, \qquad C_{12}^{(1)} = \bar{m}_{12}\sin(\theta_1 - \theta_2),$$

$$M_{23} = \bar{m}_{23}\cos(\theta_2 - \theta_3), \qquad C_{13}^{(1)} = \bar{m}_{13}\sin(\theta_1 - \theta_3).$$

By inspection, (5.64) is more compactly defined as

$$\boldsymbol{\tau} = \mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \mathbf{C}(\boldsymbol{\theta})(\dot{\boldsymbol{\theta}})^2 + \mathbf{F}\dot{\boldsymbol{\theta}} + \mathbf{K}\boldsymbol{\theta}, \tag{5.65}$$

where $\boldsymbol{\tau}$ is the vector of input control torques, $\mathbf{M}(\boldsymbol{\theta})$ is the mass matrix, $\mathbf{C}(\boldsymbol{\theta})$ is the nonlinear velocity matrix, $\mathbf{F}$ is the damping matrix, and $\mathbf{K}$ is the stiffness matrix.

Develop a numerical simulation of the flexible planar arm

Numerical integration is set up with first-order differential equations. Given specific values for the parameters, one would solve for the highest derivative of (5.65) as

$$\ddot{\boldsymbol{\theta}} = \mathbf{M}(\boldsymbol{\theta})^{-1}[\boldsymbol{\tau} - \mathbf{C}(\boldsymbol{\theta})(\dot{\boldsymbol{\theta}})^2 - \mathbf{F}\dot{\boldsymbol{\theta}} - \mathbf{K}\boldsymbol{\theta}].$$

To simulate the motion of the linkages, which are subjected to optimized input torques, the RK numerical integration technique was employed. The equations of motion, as described by (5.64), are used in the following DP optimization study.

A numerical simulation was developed for the flexible planar arm. The dynamic equations of motion were written in the MATLAB software. The modules were then incorporated directly into the DP driver and support routines. For example, the driver script code for the minimum effort run is *exarmf1.m*. The support modules for the DPIP code [68] are the dynamical equations of motion script code *esarmf1.m*, the cost function script code *coarmf1.m*, the equality constraints script code *eqarmf1.m*, and the inequality constraints script code *inarmf1.m*. These script files are listed in Section B.3 of Appendix B.

5.4.3 Dynamic Programming Optimization Design

The goal of this section is to design optimized feedforward trajectories for a two-link planar flexible manipulator that produce rest-to-rest minimal vibration-free maneuvers. Four separate runs are considered: minimum effort, minimum effort with bounds, minimum-time, and minimum torque-rate effort. These DP results are from [86].

In the first run, the DP algorithm was set up to minimize control effort with the cost function given as

$$J = \int_0^{t_f} \left(u_1^2 + u_2^2\right) dt$$

subject to planar manipulator dynamics (5.64). Note that $u_1 = \tau_1$ and $u_2 = \tau_2$. The system was initially at rest, so the initial conditions are

$$\theta_1(0) = \theta_2(0) = \dot{\theta}_1(0) = \dot{\theta}_2(0) = \dot{\theta}_3(0) = \dot{\theta}_4(0) = 0 \tag{5.66}$$

and

$$\theta_3(0) = \theta_4(0) = 180^\circ, \tag{5.67}$$

with the equality constraints given by

$$\theta_1(t_f) - \theta_{1_{desired}} = \theta_2(t_f) - \theta_{2_{desired}} = \theta_3(t_f) - \theta_{3_{desired}} = \theta_4(t_f) - \theta_{4_{desired}} = 0 \tag{5.68}$$

and

$$\dot{\theta}_1(t_f) = \dot{\theta}_2(t_f) = \dot{\theta}_3(t_f) = \dot{\theta}_4(t_f) = 0. \tag{5.69}$$

The final time was set to $t_f = 2.0$ sec. The number of discretization points was set to $N = 201$. The numerical simulation results are reviewed in Section 5.4.5.

In the second run, minimum control effort was again considered and also included bounds on the control signals. The problem is set up exactly as in the previous case, for minimum control effort, but with the additional inequality constraints

$$u_{min_i} \leq u_i \leq u_{max_i}, \qquad i = 1, 2, \tag{5.70}$$

for each joint control input. The third run considers minimum time as the cost function

$$J = t_f$$

subject to planar manipulator dynamics (5.64). The initial conditions (5.66) and (5.67), equality constraints (5.68) and (5.69), and inequality constraints (5.70) are again applied.

The final run considers the derivative of torque to minimize in the cost function, or

$$J = \int_0^{t_f} \left(\dot{\tau}_1^2 + \dot{\tau}_2^2\right) dt$$

subject to the same dynamics and initial and final conditions as the minimum control effort case. Note that the torque terms, $\tau_i(0) = 0$ and $\tau_i(t_f) = 0$, for $i = 1, 2$ were incorporated into the initial and final conditions, respectively.

5.4.4 Implementation Issues

The third test scenario in this case study investigates minimum-time solutions. In particular, minimum time is introduced as a new optimization setup for the DP algorithm. To cast the minimum-time problem so that it fits within the framework of DP, a new independent variable τ is introduced [68], running from zero to one and related to time, t, as $t = a^2\tau$. The constant a is a new state variable that represents the minimum-time value. The differential equations are modified to be a function of the new independent variable, τ. This is a form of nondimensionalized time. This format is used to solve for the minimum-time solution in Section 5.4.5.

Frequently, during optimization, a smaller step size is needed to approach the final solution from an initial guess. Several types of homotopy approaches [53, 54] have been successfully used with other optimization techniques. One specific approach is to use a simple linear approximation defined as

$$u_i = u_i + \alpha u_{i+1},$$

where α is used to incrementally apply the latest solution (u_{i+1}) to the current solution (u_i). α is a constant value normally set between zero and one. This approach was used to help find the solution in the last scenario in Section 5.4.5.

5.4.5 Numerical Simulation Results

Four different optimization runs were performed for the DP design. All runs used a joint angle trajectory minimum to maximum to minimum inertia configuration maneuver. Both link angles were configured such that the initial conditions are given as (5.66) and (5.67). For the final end conditions, the desired link angles were set to

$$\theta_{1_{desired}} = \theta_{2_{desired}} = 180^\circ$$

and

$$\theta_{3_{desired}} = \theta_{4_{desired}} = 0°.$$

Initially, the arm is folded or tucked in at the minimum inertia configuration and then slewed to a full extension maximum inertia and returned to a minimum inertia configuration. The following sequence of runs was performed: (1) minimum effort, (2) minimum effort with bounds, (3) minimum time, and (4) minimum torque rate. The summarized results for each run are given in Table 5.8, whose entries for the DPIP optimization results include run number, initial $\mathbf{u}_i$ guess, number of iterations performed, final convergence value, final time (t_f), and the α iteration constant. The numerical results are reviewed in detail next.

Table 5.8. *DPIP flexible robot arm optimization results.*

Run #	1	2	3	4
$\mathbf{u}_i$ guess	$1.0 \; \forall \; i$	$\mathbf{u}_{1_{result}}$	$\mathbf{u}^*_{2_{result}}$	$0.0 \; \forall \; i$
Iteration #	2	2	3	2
Convergence	$4.99241e^{-7}$	6.42775e−7	3.51394e−4	6.231e−2
t_f (sec)	2.0 (fixed)	2.0 (fixed)	1.72434	2.0 (fixed)
α	N/A	N/A	N/A	0.9

Minimum effort run

The goal of this run was to generate DP-optimized trajectories that perform a minimum effort solution for a rest-to-rest maneuver. Each finite element angle is defined in Fig. 5.15 above, and the positions are shown for the complete maneuver in the upper two plots of Fig. 5.17. The corresponding velocities are shown in the middle two plots of Fig. 5.17. The minimum effort torque profiles are shown in the bottom two plots of Fig. 5.17. During the maneuver, the arm is allowed to move freely, but at the final orientation all positions and velocities are returned to rest. The DPIP algorithm torques for each link were set to all ones and converged in two iterations to a minimum tolerance given in Table 5.8, run #1. The DPIP algorithm quickly converged to an optimized solution. This cost function is typical of systems that need to conserve energy during a particular operation.

Minimum effort with bounds run

The goal of this run was to generate DP-optimized trajectories that perform a minimum effort with bounds for a rest-to-rest maneuver. Oftentimes, real systems have actuator constraints that limit the amount of control effort that can be applied. This control bound option is accommodated in the DPIP algorithm by adding inequality constraints on the discretized control parameters. For this particular case study, the constrained maximum and minimum were defined for each link input to $u_{max/min_1} = \pm 13.5$ N-m and $u_{max/min_2} = \pm 4.0$ N-m, respectively. The DPIP optimizer was initially started with the torque optimization profiles from run #1 and converged quickly in two iterations to a minimum tolerance given in Table 5.8, run #2. The results are shown for positions, velocities, and torques in Fig. 5.18. Again, the system is also constrained to rest at the beginning and end of the maneuver. Note that, for the torque profiles (see the bottom plots of Fig. 5.18), each actuator saturates at the minimum and maximum bounds at the beginning and end of the maneuver, respectively. In contrast, the minimum effort solution torques are not bounded.

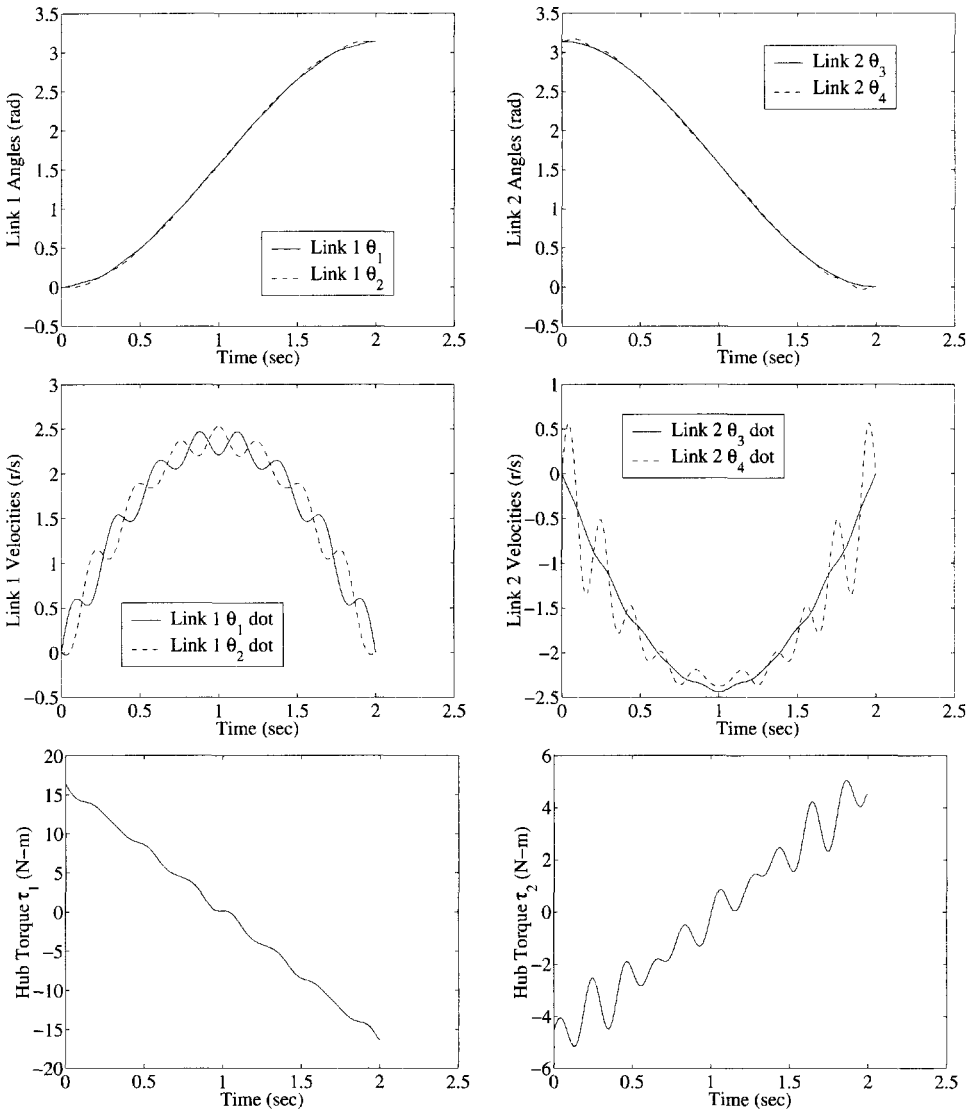


Figure 5.17. *Minimum effort optimized feedforward numerical simulation trajectory results. (From Wilson, Robinett, and Eisler* [86]. *© Institute of Electrical and Electronics Engineers, Inc.)*

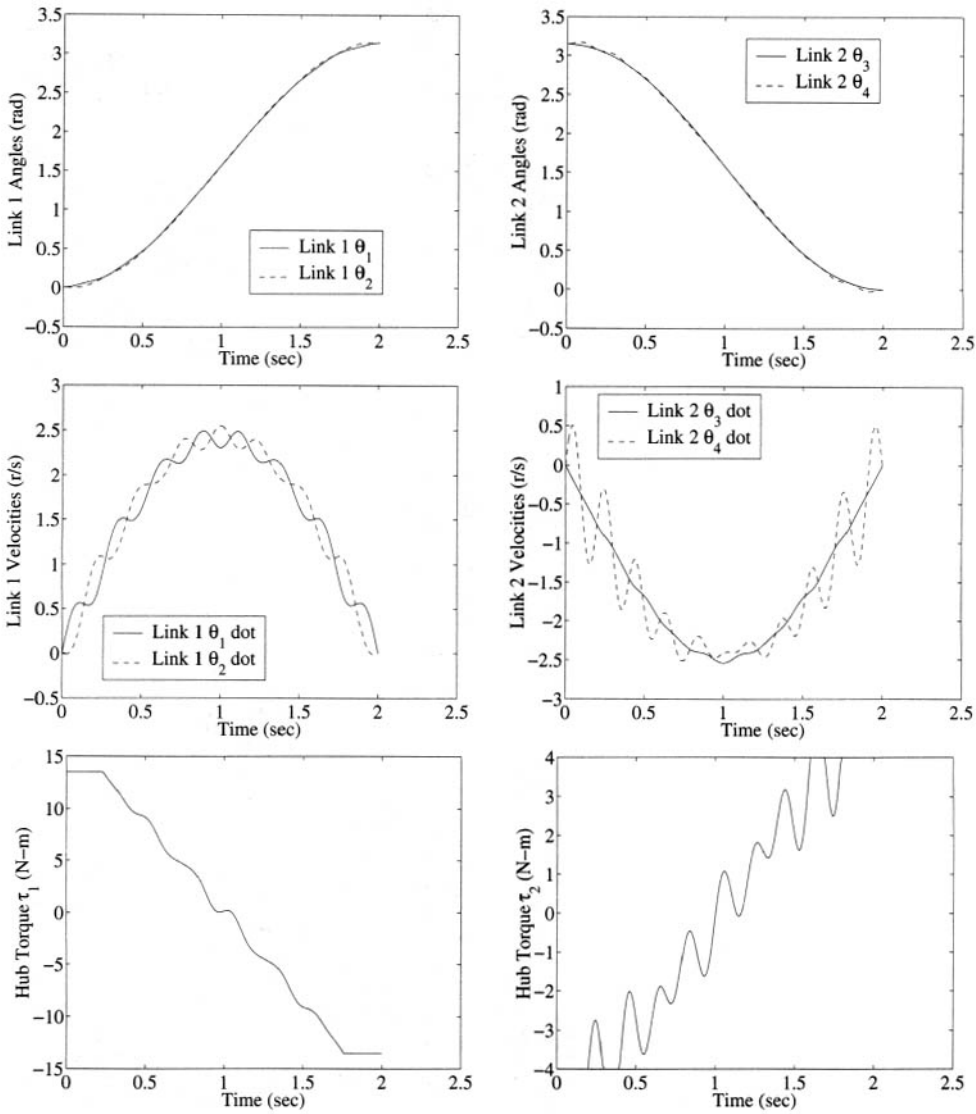


Figure 5.18. *Minimum effort with bounds optimized feedforward numerical simulation trajectory results. (From Wilson, Robinett, and Eisler* [86]. © *Institute of Electrical and Electronics Engineers, Inc.)*

Minimum-time run

The goal of this run was to generate DP-optimized trajectories that perform a rest-to-rest maneuver in minimum time. This solution represents the maximum theoretical performance capable under the constraints that are given. The DPIP algorithm uses a nondimensionalized time parameter (as discussed in Section 5.4.4). Again, the same equality and inequality constraints are used as in the previous run. The DPIP optimizer was initially started with the torque optimization profiles from run #2. In addition, the initial minimum-time parameter was set to 1.7 sec. Within three iterations, the solution converged to a minimum tolerance given in Table 5.8, run #3. The minimum time was determined to be equal to 1.72434 sec. Further improvements in tolerance were investigated, but stable solutions were not achievable. This may be due to the low number of discretized points ($N = 201$).

To obtain a higher resolution in tolerance, further solution variations would be required. However, for a simple (two finite element) model, a lot of information is extracted. As the requirements for a specific system are determined, further details can be added to a more complex model [5]. The numerical simulation results for positions, velocities, and torques are given in Fig. 5.19. The torque profiles shown in the bottom plots of Fig. 5.19 display classical bang-bang type profiles. In addition, the velocity profiles shown in the middle plots of Fig. 5.19 display a sharp peak at the midpoint of the trajectory, also characteristic of bang-bang type solutions. Although this solution provides the minimum time for the maneuver, the vibration levels in the links are higher than in previous runs.

Torque rate cost function run

The goal of the final run was to generate DP-optimized trajectories that perform a minimum torque rate rest-to-rest maneuver. This solution represents the need to generate smoother trajectory profiles by taking into account additional constraint conditions on the hub torques (i.e., torque constraints on the initial and final conditions). The DPIP optimizer was initially set to all zeros and converged within two iterations to a minimum tolerance given in Table 5.8, run #4. Further improvements in tolerance were investigated, but the next iteration would diverge. Some of the reasons stated in the previous run may also apply. To generate a final iteration and determine the direction (in the sense of the optimization direction), the α parameter was set to 0.9 (see Section 5.4.4 for discussion) to generate the current solution. The numerical simulation results for positions, velocities, and torques are given in Fig. 5.20. Overall, the trajectories show smoother results than in all previous runs. This result represents the lowest vibration throughout the maneuver while only using beginning and ending constraints. An alternative would be to either apply additional constraints on specific states (i.e., velocities) or investigate strain rate component constraints (i.e., minimize strain rate energy). Note that the bottom plots in Fig. 5.20 show torque (left) and parabolic torque rate profiles (right) for each link.

5.4.6 Case Study 3 Discussion (Summary/Conclusions)

The objective of this case study was to provide an optimized feedforward trajectory planner for a planar flexible two-link arm to produce rest-to-rest, minimal–vibration-free maneuvers. This required the development of the dynamic equations of motion, numerical implementation, optimized DP trajectory design, and verification using numerical simulations.

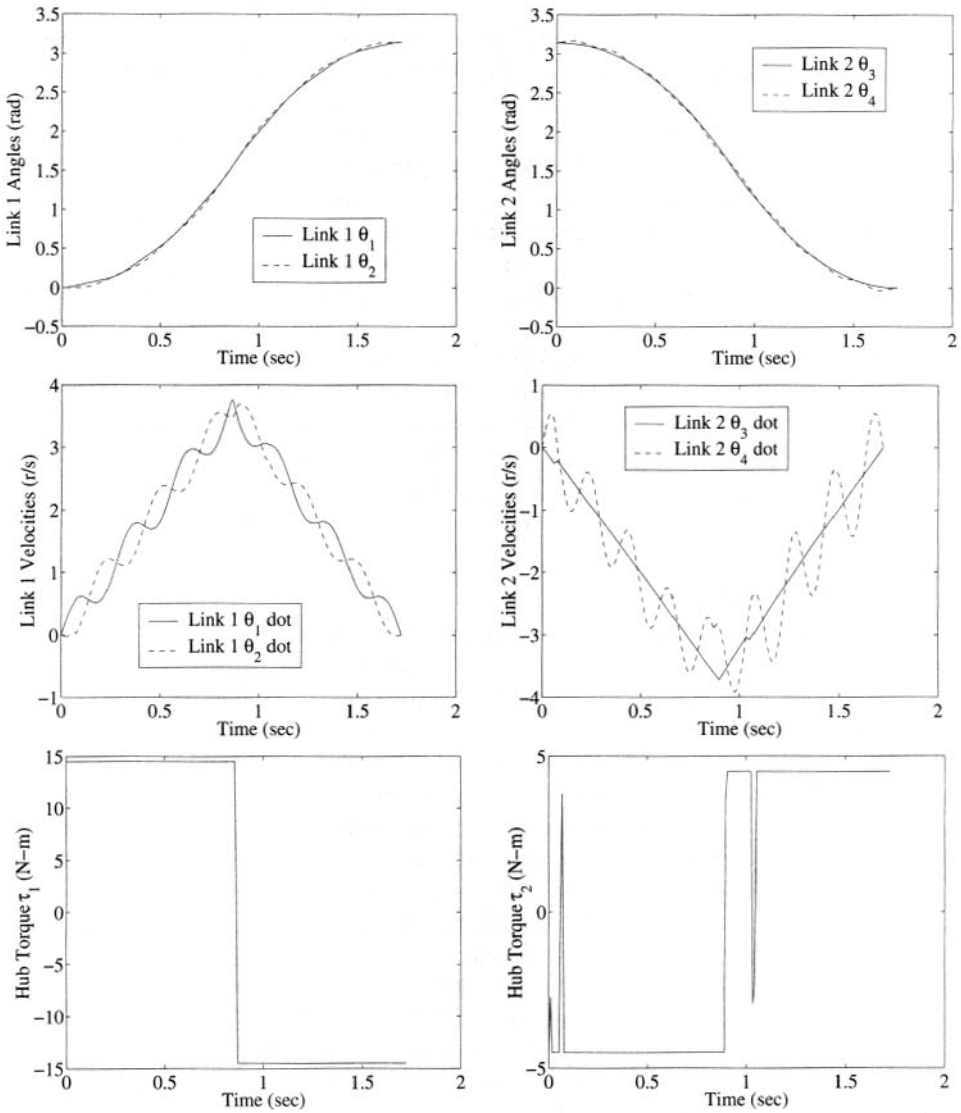


Figure 5.19. *Minimum-time optimized feedforward numerical simulation trajectory results. (From Wilson, Robinett, and Eisler* [86]. *© Institute of Electrical and Electronics Engineers, Inc.)*

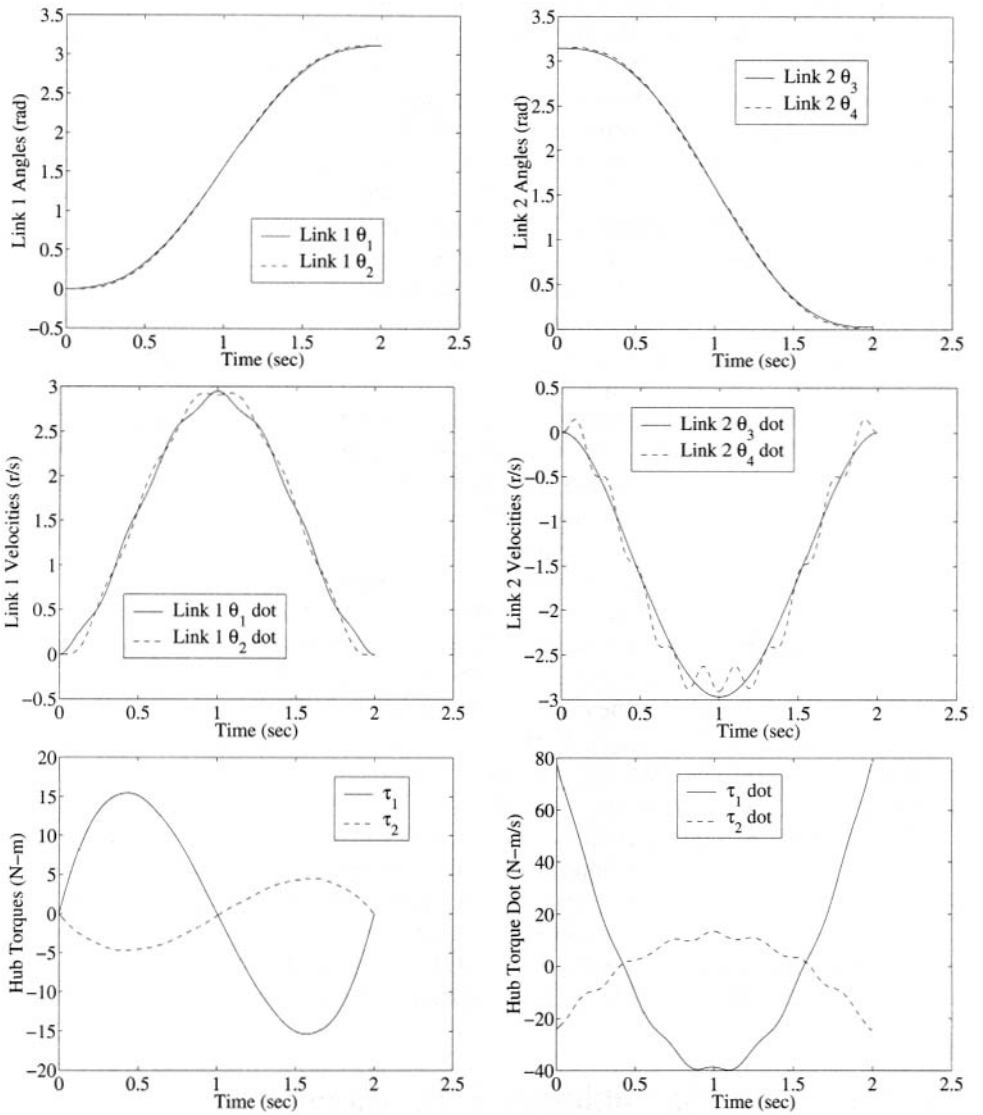


Figure 5.20. *Torque rate cost function optimized feedforward numerical simulation trajectory results. (From Wilson, Robinett, and Eisler* [86]. © *Institute of Electrical and Electronics Engineers, Inc.)*

A unique finite element technique was employed to develop the equations of motion. In the interest of simplicity, only a single element per link was selected. This allowed the model to be simple enough to actually focus on the DP trajectory approach and its variations. A joint angle maneuver that starts with the arm in a minimum inertia orientation and goes to a maximum inertia operation and ends in a final minimum inertia orientation was used for all scenarios. Four separate scenarios were investigated: (1) minimum effort, (2) minimum effort with bounds, (3) minimum time, and (4) minimum torque rate or power effort. The first scenario produced a set of optimized feedforward trajectories characteristic of systems that need to conserve energy from point to point. In the second scenario, bounds were applied to the hub torques that represent realistic actuator constraints. This modified the optimized feedforward trajectories by allowing the torques to saturate during the maneuver. Although the trajectory profiles were altered, the end result was still achieved. In the third scenario, the flexible arm was still subject to the equality and inequality constraints of the previous scenario, but the cost function was changed to minimize the amount of time required to complete the maneuver. The results shown are very similar to the popular bang-bang type results of other similar dynamic systems. This optimization result is typical when determining the theoretically maximum performance of the flexible arm within the minimum time to complete the maneuver. This trajectory resulted in the highest vibration levels in transit for the flexible arm. In the final scenario, the torque rate was minimized in the cost function. This resulted in the smoothest trajectory profiles for positions, velocities, and torques as compared to the previous scenarios. This solution is equivalent to the minimum amount of vibration while in transit as the rest-to-rest maneuver.

The DPIP algorithm performed efficiently for the number of points ($N = 201$) used during the analysis. To further improve the accuracy, a larger number of discretized control nodes could be investigated. To help generate further detailed results, a larger number of finite elements could be used in the model. However, the main goal was to keep the model simple enough that the reader could follow along and confirm the steps used to perform the DP optimization. In the final scenario, an additional homotopy-like parameter, α, was introduced to help microstep into the final solution.

In summary, the DPIP algorithm was found to be an effective optimization procedure to generate discrete-time optimal control feedforward trajectories that are subject to both equality and inequality constraints. Several DP optimization designs were verified through numerical simulations for the planar flexible two-link arm. Typically, the optimized feedforward trajectories would be combined with a closed-loop control algorithm to help compensate for unmodeled dynamics and external disturbances. (Refer to Fig. 1.1.)

5.5 Case Study 4: Minimum-Time Maneuvers of Rigid Systems

5.5.1 Introduction

The objective of this case study is to present some findings regarding the time-optimal maneuvers of a rigid symmetric spacecraft. This subject was investigated by Bilimoria and Wie [87], who presented new results for rest-to-rest maneuvers. They found that, in general, the eigenaxis rotation maneuver is not time optimal and that the number of control switches

depended on the reorientation angle. Their approach was based on the calculus of variations and they were able to identify the controls for a time-optimal 180° maneuver by employing a continuation approach. Seywald, Kumar, and Deshpande [88] presented a three-part method to obtain highly accurate solutions to time-optimal maneuver problems. In the first part, a genetic algorithm was used to determine controls at discrete time points. This was done without any a priori knowledge of the optimal control switching structure. In the second part, the controls at the discrete time points from the genetic algorithm solution were treated as a parameter vector and used as an initial guess in a nonlinear programming (NP) method. This essentially produced the optimal controls. In the third part, a more refined solution was obtained by feeding the information from the NP solution into an indirect method involving costate variables. The initial conditions on costate variables and the values of constant Lagrange multipliers were obtained using a multiple shooting method.

In this work, the DPIP approach [29, 31] was used to determine the controls that result in time-optimal maneuvers. Recall that this approach is able to solve generally nonlinear equality- and inequality-constrained discrete-time optimal control problems. The algorithm is based on a combined SQP and interior point method. The interior point method is used to accommodate the inequality constraints of the states and controls should they be present.

Initially, a time-optimal double integrator problem is investigated. This is a classic minimum-time problem with linear dynamics. One purpose of studying this problem is to gain an understanding of when the DP approach is fruitful and when it is not. For comparison, the time-optimal double integrator problem is solved using the MATLAB function *fmincon*, which is part of the Optimization Toolbox. The time-optimal double integrator also happens to represent an eigenaxis rotation maneuver of the problem studied next, which is the time-optimal maneuver of a rigid symmetric spacecraft.

5.5.2 The Classic Double Integrator Example

A popular version of the time-optimal double integrator problem is to find u, bounded between plus and minus one, that moves a dynamical system governed by

$$\dot{x}^1 = x^2, \quad \dot{x}^2 = u \tag{5.71}$$

from one state of rest to another in the least amount of time. The performance index is $J = t_f$, the rest-to-rest conditions may be written as

$$x^1(t_f) = \text{specified}, \quad x^1(0) = x^2(0) = x^2(t_f) = 0, \tag{5.72}$$

and the bounds on the control may be written as $-1 \leq u \leq 1$. Note that the superscript on the state variable denotes the element of the state vector.

Solution via the dynamic programming/interior-point method

A discrete-time equivalent to the double integrator problem is to find u_i, also bounded, that moves the dynamical system governed by

$$\dot{x}^1_{i+1} = x^1_i + x^2_i x^3_i + \frac{1}{2} x^3_i x^3_i u_i = g^1_i(x_i, u_i), \tag{5.73}$$

$$\dot{x}^2_{i+1} = x^2_i + x^3_i u_i = g^2_i(x_i, u_i), \tag{5.74}$$

$$\dot{x}^3_{i+1} = x^3_i = g^3_i(x_i, u_i) \tag{5.75}$$

from one state of rest to another such that $J = x_1^3(N-1)$ is minimized. The index i has range $i = 1, \ldots, N-1$, where N is the total number of discrete time points. Notice that x_i^α is the αth state at the ith time point. Also notice that x^3 is a new state that represents the continuous differential dt, and by the introduction of this state the system dynamics are now nonlinear. The rest-to-rest conditions are written as the four equality constraints

$$x_N^1 = \text{specified}, \quad x_1^1 = x_1^2 = x_N^2 = 0, \tag{5.76}$$

and the bounds on the control are written as the $2(N-1)$ inequality constraints

$$-1 - u_i \leq 0, \quad u_i - 1 \leq 0, \quad i = 1, \ldots, N-1. \tag{5.77}$$

In particular, the time-optimal maneuver described by $x_N^1 = \pi$ was considered. The exact closed-form solution for this specific maneuver gives a final time equal to 3.545 sec. Findings from several simulations are listed in Table 5.9, where the number of iterations means the number of quadratic subproblems that were solved. Generally speaking, the DPIP method [68] worked well.

1. Tests 1 to 4 (See Table 5.9) indicate that the method produced a solution when the initial guess for the optimal control history was set to 0.5 for all nodes. This is one half of the allowed amount. The method worked even with a poor initial guess of the final time. Notice that for $N = 31$, an odd value, the exact minimum time was found, whereas for $N = 30$, an even value, the minimum time was slightly in error. This was expected for the following reason: The optimal trajectory of one state of the system has a sharp corner at the midpoint. Consequently, for N odd, a node is placed at the midpoint and this captures the sharp corner of the optimal trajectory, whereas, for N even, this does not occur.

2. Test 5 (See Table 5.9) shows that the method worked when the initial guess for the optimal control history was set to equal 0.8 for approximately the first half of the nodes, but set to zero for the remaining nodes. Essentially, the initial control guess pulses the dynamical system, after which the system is left to coast. (This initial control guess was considered because coupled nonlinear dynamical systems that are subject to multiple controls may be driven far from a possible optimal solution if the controls are set to constant values for an entire time history, i.e., all nodes.)

3. The method was unable to converge to a solution when the initial guess for the optimal control history was set to a value less than 0.2 for all nodes.

This simple study illustrates that the DPIP method generally works well, but it can fail to produce solutions to (generally) nonlinear discrete-time optimal control problems. This is not surprising because the statement is true of any NP solution method. To widen the range of convergence, one can incorporate a homotopy (or continuation) parameter as part of the method [68].

Table 5.9. *Summary of results for the double integrator example.*

Test	N	u_i guess	t_f guess	Comment
1	31	$0.5\ \forall\ i$	5.1 sec	4 iterations, $t_f = 3.545$ sec
2	30	$0.5\ \forall\ i$	4.93 sec	5 iterations, $t_f = 3.547$ sec
3	31	$0.5\ \forall\ i$	10.2 sec	5 iterations, $t_f = 3.545$ sec
4	31	$0.5\ \forall\ i$	20.4 sec	5 iterations, $t_f = 3.545$ sec
5	31	$0.8, i = 1, \ldots, 12$	10.2 sec	5 iterations, $t_f = 3.545$ sec

In order to gauge the efficiency of the DPIP method, in the next subsection the time-optimal double integrator problem is solved using a routine from the MATLAB Optimization Toolbox [89].

Solution using MATLAB function *fmincon*

The time-optimal double integrator problem can be solved using the MATLAB Optimization Toolbox function *fmincon*. This function is used to solve constrained nonlinear parameter optimization problems, so it is suitable for solving the discrete-time optimal control problems described by (5.71)–(5.75). A complete description of this function can be found in the Optimization Toolbox user guide [89]; consequently, only some differences between the method employed by *fmincon* and the DPIP method are identified.

It is important to note that although the function *fmincon* accepts many option settings, only the default settings were selected. The DPIP method and *fmincon* are both based on the SQP method. The function *fmincon*, however, uses a quasi-Newton method to approximate the Hessian of the Lagrangian function, whereas the currently implemented version of the DPIP method computes the Hessian using finite difference calculations. Another main difference in the approaches is that the DPIP method uses, as the name suggests, an interior point method to determine the binding inequality constraints, whereas the function *fmincon* uses an active set method. A discussion of active set methods can be found in the texts by Gill, Murray, and Wright [90] and Luenberger [91].

The function *fmincon* solved the time-optimal double integrator problem for the cases presented in Table 5.9. The number of (quadratic subproblem) iterations required to solve the problem ranged from 18 to 25 and hence were more than those needed by the DPIP method. The function *fmincon* was, however, able to determine correct solutions for a slightly larger range of initial guess control histories.

5.5.3 Three-Axis Maneuvers of a Symmetric Spacecraft

Next, a time-optimal three-axis reorientation of a rigid symmetric spacecraft is considered. This is the problem that was investigated by Bilimoria and Wie [87] and discussed by Seywald et al. [88]. To some extent, the results can be contrasted against those presented by Seywald et al. [88]. One recalls that the method of Seywald et al. [88] is a three-part method. The first two parts (a genetic algorithm followed by an NP method) find the optimal controls at discrete time points, that is, precisely what the DPIP method produces. Consequently, the results can be contrasted against the first two parts of their method.

The problem is to determine the controls u^1, u^2, u^3 that transfer the system

$$\dot{q}^1 = \frac{1}{2}(\omega^1 q^0 - \omega^2 q^3 + \omega^3 q^2), \tag{5.78}$$

$$\dot{q}^2 = \frac{1}{2}(\omega^1 q^3 + \omega^2 q^4 - \omega^3 q^1), \tag{5.79}$$

$$\dot{q}^3 = \frac{1}{2}(-\omega^1 q^2 + \omega^2 q^1 + \omega^3 q^4), \tag{5.80}$$

$$\dot{q}^4 = \frac{1}{2}(-\omega^1 q^1 - \omega^2 q^2 - \omega^3 q^3), \tag{5.81}$$

$$\dot{\omega}^1 = u^1, \tag{5.82}$$

$$\dot{\omega}^2 = u^2, \tag{5.83}$$

$$\dot{\omega}^3 = u^3 \tag{5.84}$$

from the initial conditions

$$q^1(0) = q^2(0) = q^3(0) = 0, \quad q^4(0) = 1, \tag{5.85}$$

$$\omega^1(0) = \omega^2(0) = \omega^3(0) = 0 \tag{5.86}$$

to the final conditions

$$q^1(t_f) = q^2(t_f) = q^4(t_f) = 0, \quad q^3(t_f) = 1, \tag{5.87}$$

$$\omega^1(t_f) = \omega^2(t_f) = \omega^3(t_f) = 0 \tag{5.88}$$

in the least time. These initial and final conditions specify a 180° reorientation about a body-fixed axis $\mathbf{b}_3$. The controls are bounded according to $|u^\alpha| \leq 1$. The variables q_i, for $i = 1, \ldots, 4$, are the Euler parameters. These four parameters are used to represent the rotational attitude of the spacecraft. The parameters never encounter a singularity in their description of rotational attitude, but the parameters are not independent and are constrained according to $q_1^2 + q_2^2 + q_3^3 + q_4^2 = 1$. The variables ω_i, for $i = 1, \ldots, 3$, are the body axis components of the angular velocity vector of the spacecraft. The spacecraft is assumed to be symmetric with inertia scaled to equal one.

One suboptimal solution to this reorientation problem is an eigenaxis rotation maneuver (the eigenaxis is the body-fixed axis $\mathbf{b}_3$). Indeed, Bilimoria and Wie [87] essentially start with the eigenaxis maneuver and use a continuation approach to identify the time-optimal controls. Similarly, the genetic algorithm solutions of Seywald et al. [88], which are used to start the NP method, closely resemble an eigenaxis maneuver.

The DPIP method was started with initial control guesses like those presented in Test 5 of the double integrator problem (see Table 5.9). That is, the initial controls were set to a constant level for some initial nodes and then set to zero for the remaining nodes. There were eight possible permutations to consider once a level and duration of pulse were decided upon. For example, one permutation is that all three controls are initially set to positive levels.

Table 5.10. *Summary of results for the three-axis reorientation example.*

Sequence	Comment	Maneuver
$+++$	17 iterations, $t_f = 3.2438$ sec	1
$++-$	16 iterations, $t_f = 3.2473$ sec	3
$+-+$	29 iterations, $t_f = 3.2435$ sec	3
$+--$	19 iterations, $t_f = 3.2443$ sec	4
$-++^*$	25 iterations, $t_f = 3.2467$ sec	2
$-+-^*$	53 iterations, $t_f = 3.2439$ sec	4
$--+$	23 iterations, $t_f = 3.2436$ sec	2
$---^*$	27 iterations, $t_f = 3.2468$ sec	2

Typical results are presented in Table 5.10, where the magnitude of the pulse level for each initial control guess was set to 0.8 and the duration was set to 6 of 30 nodes. The initial guess for the final time was 5 sec, which is the guess that Seywald et al. [88] used. Convergence was recognized when the terminal constraints were satisfied and the controls were no longer updating. For all runs, except those marked with an asterisk, the parameter that governs the step length along the search direction was set to unity. A merit function [92, 93] to determine an optimal step length was not defined or employed. The three runs marked with an asterisk failed to converge for a step length parameter equal to one, but did converge for a step length parameter equal to 0.9. The column labeled "Maneuver" specifies which optimal maneuver was converged upon. Because of the symmetry of the problem, there are four time-optimal maneuvers, all with the same cost. (Incidentally, the minimum time as reported by Bilimoria and Wie [87] was $t_f = 3.243$ sec.) Bilimoria and Wie [87] describe the control profiles of the four maneuvers, although their continuation process only converged to one of the four. Similarly, Seywald et al. [88] found two of the four maneuvers using their approach and mention that they expected that all symmetric time-optimal solutions could be found by beginning the genetic algorithm with different seeds in a random number generator. The DPIP method was able to find all four time-optimal maneuvers by a simple change in the initial settings (plus or minus) of the three controls. The four time-optimal maneuvers are shown in Fig. 5.21.

There seemed to be nothing special about the final maneuver found by a particular initial control guess. For example, for the $+++$ control guess, the DPIP method converged to Maneuver 1 for $N = 40$ (12 iterations, $t_f = 3.2467$ sec) but to Maneuver 2 for $N = 50$ (34 iterations, $t_f = 3.2416$ sec).

fmincon was also used to solve this problem and gave mixed results. For example, for the same initial parameters associated with the $+++$ control guess of Table 5.10, *fmincon* was unable to converge and terminated after 94 iterations. When the guess for the final time was set to 3 sec, *fmincon* converged to a suboptimal solution (with $t_f = 3.2848$ sec) after 64 iterations. For a guess of nearly zero controls and a final time guess of 5 sec, *fmincon* essentially converged to Maneuver 2 (with $t_f = 3.2504$ sec) after 77 iterations. The angular velocities associated with the *fmincon* suboptimal solution and with time-optimal Maneuver 2 are shown in Fig. 5.22.

The DPIP code [68] performed this case study by using a DP driver file and support routines. For this example, the driver script code is *exmt.m*. The support modules are the dynamical equations of motion script code *esmt.m*, the cost function script code *comt.m*, the

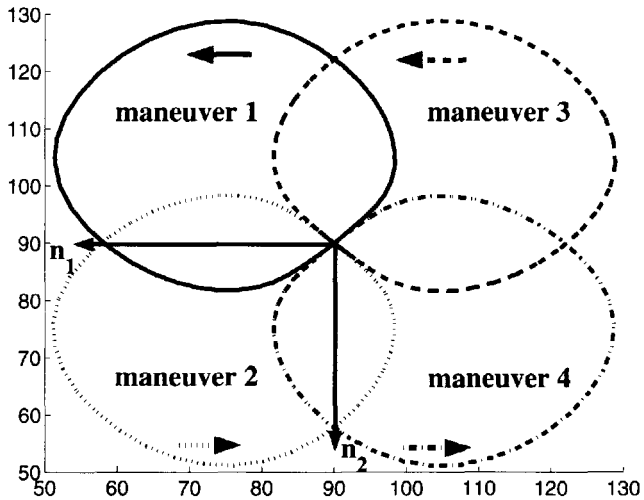


Figure 5.21. *Nutation of the body axes* $\mathbf{b}_3$ *with respect to the inertial axes for the four time-optimal maneuvers. For each maneuver, the tip of the* $\mathbf{b}_3$ *axis nutates in a counterclockwise direction, tracing out the curve. This view is the projection onto the inertial* $\mathbf{n}_1, \mathbf{n}_2$ *plane. The maneuver begins and ends with* $\mathbf{b}_3$ *at the intersection of the* $\mathbf{n}_1$ *and* $\mathbf{n}_2$ *axes.*

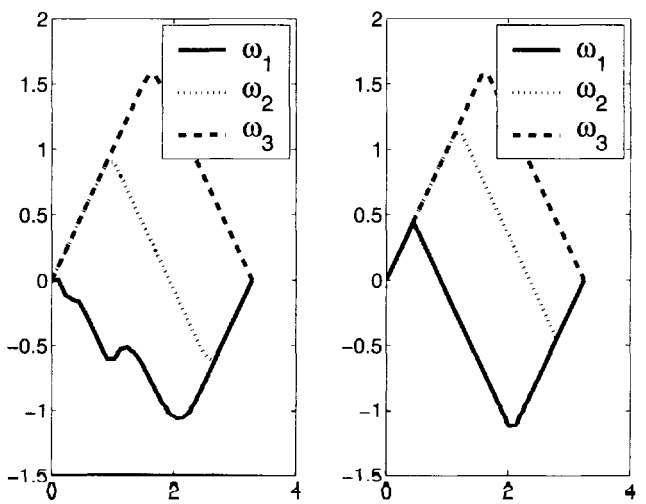


Figure 5.22. *Suboptimal solution from* fmincon *(left plot) and time-optimal Maneuver 2 (right plot).*

equality constraints script code *eqmt.m*, and the inequality constraints script code *inmt.m*. These script files are listed in Appendix B.4.

5.5.4 Case Study 4 Discussion (Summary/Conclusions)

An investigation of time-optimal maneuvers of rigid symmetric spacecraft was conducted using the DPIP method.

The method generally worked well as it was applied to the classic double integrator problem and a time-optimal three-axis 180° reorientation maneuver. Due to the symmetry of the three-axis problem, there were four time-optimal solutions with the same cost. The DPIP method was able to find all four time-optimal solutions as different initial control settings (plus or minus) were used. To gauge the efficiency of the DPIP method, the double integrator and three-axis 180° reorientation problems were also solved using the MATLAB Optimization Toolbox function *fmincon*. Only the default settings of this function were utilized. At times, *fmincon* worked well, but the number of (quadratic subproblem) iterations that were required was always significantly greater than those needed by the DPIP method.

5.6 Case Study 5: Maximum-Radius Orbit Transfer

5.6.1 Introduction

The following classic problem was first introduced by Lindorfer and Moyer [94]. The problem statement is as follows [95]: Given a constant-thrust rocket engine with thrust, T, operating for a given length of time, t_f, find the thrust direction history, $\phi(t)$, to transfer a rocket vehicle from a given initial circular orbit to the largest possible circular orbit. A standard diagram of this problem is shown in Fig. 5.23. One main point of this case study is to show that very reasonable solutions to generally nonlinear problems may be obtained using a low number of nodes, but because of the low number of nodes, small values of the step length parameter α may be necessary.

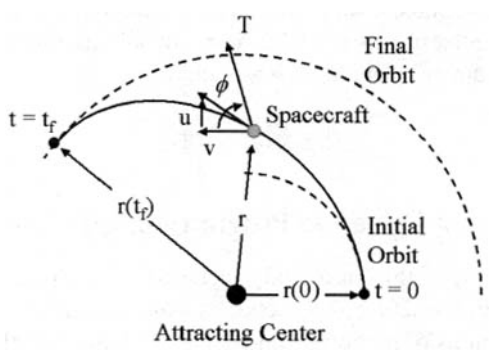


Figure 5.23. *Maximum-radius orbit transfer diagram.*

5.6.2 Dynamical Model

The traditional nomenclature for this problem is given below:

r = radial position of the rocket from the attracting center,
u = radial component of the velocity vector,
v = tangential component of the velocity vector,
m = rocket mass, with constant mass fuel consumption rate given by $\dot{m}$,
ϕ = thrust direction angle measured from the local horizontal,
μ = gravitational constant of the attracting center.

The governing differential equations of motion are

$$\dot{r} = u, \tag{5.89}$$

$$\dot{u} = -\frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T \sin\phi}{m_0 - \dot{m}t}, \tag{5.90}$$

$$\dot{v} = -\frac{uv}{r} + \frac{T \cos\phi}{m_0 - \dot{m}t}. \tag{5.91}$$

Because the initial and final orbits are prescribed to be circular, the initial and final state equality constraints are as follows:

$$\begin{aligned} &\text{Initial:} \quad r(0) = \text{specified}, \quad u(0) = 0, \quad v(0) = \sqrt{\mu/r(0)}; \\ &\text{Final:} \quad u(t_f) = 0, \quad v(t_f) = \sqrt{\mu/r(t_f)}. \end{aligned}$$

The performance index is $J = -r(t_f)$.

A numerical solution is determined for the following normalized problem parameters, which represent an idealized Earth to Mars orbit transfer [96]:

$$r(0) = 1.0, \quad \mu = 1.0, \quad m_0 = 1.0, \quad \dot{m} = 0.07487, \quad T = 0.1405.$$

For these normalized constants, a time unit equals 58.18 days. The final time, t_f, is set to 3.3207 units, which corresponds to 193.2 days. In solving this problem, an inequality constraint on the thrust direction angle, ϕ, was included:

$$0 \leq \phi(t) \leq 2\pi.$$

5.6.3 Solution Using Dynamic Programming/Interior Point

The DPIP code [68] used in this case study included a DP driver file and support routines. For this example, the driver script code is *exmaxradius.m*. The support modules are the dynamical equations of motion script code *esmaxradius.m*, the cost function script code *comaxradius.m*, the equality constraints script code *eqmaxradius.m*, and the inequality constraints script code *inmaxradius.m*. These script files are listed in Appendix B.5.

5.6.4 Case Study 5 Discussion (Summary/Conclusions)

The initial thrust direction history was selected as that given by Moyer and Pinkham [96]. The angle was set to the constant value $\pi/3$ for the first half of the maneuver and directed inward along the local vertical $3\pi/2$ for the remaining half of the transit time. For this example, the number of nodes, N, was set to 11. The initial thrust direction history and resulting state histories of r, u, and v are shown in Fig. 5.24. In using the DPIP method [68], the updates to the control profiles may be adjusted with the parameter $\alpha \in (0, 1]$. For this particular example, $\alpha = 0.08$. For higher values of α, difficulties were encountered. Convergence was recognized in 25 iterations for both thrust direction history and resulting state histories (r, u, and v) and is shown in Fig. 5.25. These results match well with the previously published results by Moyer and Pinkham [96] that give a final radius of $r(t_f) = 1.525$, whereas the DPIP method with 11 nodes gives $r(t_f) = 1.4963$, which is within 2% of the true value.

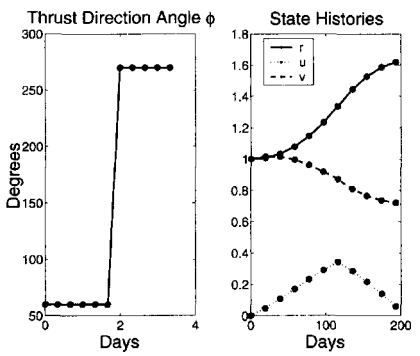


Figure 5.24. *Initial thrust direction history for the orbit transfer and the resulting time history profiles of r, u, and v.*

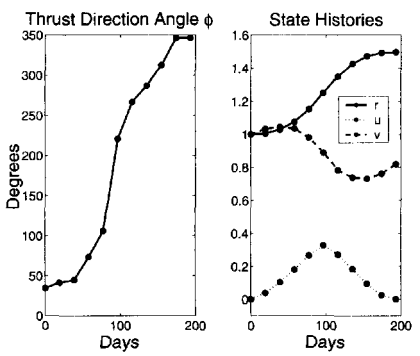


Figure 5.25. *Converged thrust direction history for the orbit transfer and the resulting time history profiles of r, u, and v. The thrust is directed outward for the first half of the transit time and inward for the second half of the transit time.*

5.7 Case Study 6: PUMA 560 Industrial Manipulator

5.7.1 Introduction

A given path in the presence of physical and path constraints is tracked in robotic manipulation tasks such as inspecting, welding, painting, assembling, and moving objects. Physical constraints usually consist of joint torque limits, due to joint motor voltage saturation; joint velocity and acceleration limits; and limits on joint positions associated with mechanical construction [97]. Path constraints may include jerk-free and tracking-error constraints (usually due to manufacturing tolerances) [97]. These constraints can be taken into account in robot motion planning. The motion planning problem in its most fundamental form is to determine an optimal trajectory from specified starting and desired completion configurations [98]. The type of performance index selected, subject to the possible constraints, to develop the optimization of robotic manipulator motions has already been reviewed (see earlier case studies).

Often, trajectory planning is studied off-line with consideration given to moving the end effector between two goal points. Many variations of optimized path planning techniques have been studied in the literature. Some of the highlights and results of several varying approaches by different researchers are given in [97, 98, 99, 100, 101, 102, 103, 104]. Previous performance indexes (see Case Studies 1 to 3) could be applied once again to this case study.

In this case study, the objective is to emphasize in-transit constraints such as the individual joint maximum speed of operation. The DPIP approach will be used to generate optimized feedforward trajectories. The steps to achieve this are as follows.

Step 1: Define a mathematical model that takes into account the PUMA 560 manipulator nonlinear coupled equations of motion.

Step 2: Develop a numerical simulation of the PUMA 560 manipulator.

Step 3: Solve a trajectory optimization problem using DP design that accommodates in-transit constraints.

Step 4: Verify the design with numerical simulations.

Step 5: Summarize the case study findings.

This problem is further defined and analyzed in the following sections.

5.7.2 Model Development

PUMA 560 model definition

For this case study, the first three DOFs (q_1 to q_3) of the PUMA 560 robot manipulator arm are modeled. The schematic is shown in Fig. 5.26. The wrist DOFs are considered only as additional payload mass and inertia.

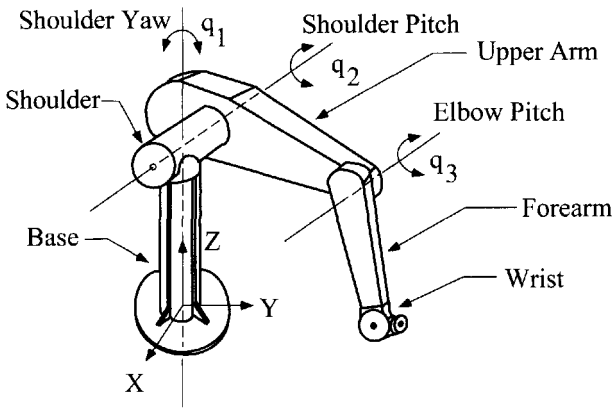


Figure 5.26. *PUMA 560 robot with first three gross positioning DOFs.*

The PUMA 560 physical parameters have been well documented in the literature, and a complete description and discussion can be found in [105, 106]. All the numerical simulation model physical parameters used in this case study are from [105]. The maximum joint parameters, such as velocities, accelerations, and torques, are given in Table 5.11, where the link side values are listed for all the DOFs. For this study, only the first three DOFs associated with the end effector XYZ location are considered (for complete end-effector position and orientation, the wrist DOFs would also need to be included). The in-transit constraints for the PUMA 560 shoulder yaw/pitch and elbow pitch are composed of the maximum link velocities given in the first three rows of column two of Table 5.11.

Table 5.11. *Summary of PUMA 560 robot performance limits from* [109].

Joint	$\dot{\mathbf{q}}_{link}$ (deg/sec)	$\ddot{\mathbf{q}}_{link}$ (deg/sec/sec)	τ_{link} (N-m)
1	110.0	744.845	56.0
2	86.59	1718.87	97.0
3	137.5	3151.26	52.0
4	306.0	2807.49	10.0
5	291.6	3151.27	10.0
6	328.9	2750.20	10.0

Forward manipulator dynamics: Recursive Newton–Euler algorithm

The robot manipulator dynamics are well known and documented in several resources [4, 106, 107, 108, 109]. The recursive Newton–Euler (RNE) algorithm [108] is a popular method in the robotics field for the efficient derivation of manipulator dynamics. The robot manipulator dynamic model [106, 107, 108, 109] is given by

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}), \tag{5.92}$$

where

$\mathbf{q}$ = an n-vector of joint angle coordinates,

$\dot{\mathbf{q}}$ = an n-vector of joint velocities,

$\ddot{\mathbf{q}}$ = an n-vector of joint accelerations,

$\mathbf{M}$ = the positive definite joint-space configuration-dependent mass matrix,

$\mathbf{C}$ = the vector of Coriolis and centripetal terms,

$\mathbf{G}$ = the gravitational vector,

$\boldsymbol{\tau}$ = the vector of generalized torques applied at the joints.

Historically, two types of manipulator dynamic problems have been investigated: the inverse dynamics problem and the forward dynamics problem. The inverse dynamics problem solves for the torques given all the terms on the right-hand side of (5.92). This is typically used for robot model-based control law computation. The reverse problem uses these equations to solve the forward dynamics problem. This solves for the accelerations from positions, velocities, and torques and is commonly used in numerical simulations.

To solve the forward dynamics problem using the RNE algorithm [108], several additional steps are required. The first step is to calculate the right-hand side terms of (5.92) given $\mathbf{q}$ and $\dot{\mathbf{q}}$ or

$$\mathbf{b}_{vec} = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}).$$

Note that the acceleration is set to zero and the torque becomes the right-hand term, $\boldsymbol{\tau} = \mathbf{b}_{vec}$. The second step is to calculate the inertia matrix without the velocity or gravitational terms:

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}.$$

The inertia matrix is computed column by column n times until all the terms are known for $\mathbf{M}(\mathbf{q})$. This is computed n times while recursively cycling the single unity nonzero term through the acceleration vector $\ddot{\mathbf{q}}$ n times until all the terms are known for $\mathbf{M}(\mathbf{q})$. For each nonzero term, the corresponding inertia matrix column is determined in the $\boldsymbol{\tau}$ vector.

In the final step, the joint accelerations can be computed. The inverse is taken for the inertia matrix calculated in step two and multiplied by the torque inputs minus the right-hand side vector calculated in step one. This is given as

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\left[\boldsymbol{\tau} - \mathbf{b}_{vec}\right] = \mathbf{M}^{-1}(\mathbf{q})\left[\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q})\right].$$

The details of the RNE algorithm are given next for reference. The RNE algorithm, as defined by [108], propagates the kinematic information through an outward recursion, and then propagates the forces/moments exerted at each link through an inward recursion. The outward recursion (from $1 \leq i \leq n$) for the angular/linear velocities/accelerations and

COM accelerations, forces, and moments is given (for rotational joints) as

$$
\begin{aligned}
{}^{i+1}\boldsymbol{\omega}_{i+1} &= {}^{i+1}\mathbf{R}_i({}^{i}\boldsymbol{\omega}_i + \mathbf{z}_0\dot{\mathbf{q}}_{i+1}),\\
{}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} &= {}^{i+1}\mathbf{R}_i\left\{{}^{i}\dot{\boldsymbol{\omega}}_i + \mathbf{z}_0\ddot{\mathbf{q}}_{i+1} + {}^{i}\boldsymbol{\omega}_i \times (\mathbf{z}_0\dot{\mathbf{q}}_{i+1})\right\},\\
{}^{i+1}\mathbf{v}_{i+1} &= {}^{i+1}\boldsymbol{\omega}_{i+1} \times {}^{i+1}\mathbf{p}^*_{i+1} + {}^{i+1}\mathbf{R}_i\,{}^{i}\mathbf{v}_i,\\
{}^{i+1}\dot{\mathbf{v}}_{i+1} &= {}^{i+1}\dot{\boldsymbol{\omega}}_{i+1} \times {}^{i+1}\mathbf{p}^*_{i+1} + {}^{i+1}\boldsymbol{\omega}_{i+1} \times \left\{{}^{i+1}\boldsymbol{\omega}_{i+1} \times {}^{i+1}\mathbf{p}^*_{i+1}\right\} + {}^{i+1}\mathbf{R}_i\,{}^{i}\dot{\mathbf{v}}_i,\\
{}^{i}\dot{\bar{\mathbf{v}}}_i &= {}^{i}\dot{\boldsymbol{\omega}}_i \times \mathbf{s}_i + {}^{i}\boldsymbol{\omega}_i \times \left\{{}^{i}\boldsymbol{\omega}_i \times \mathbf{s}_i\right\} + {}^{i}\dot{\mathbf{v}}_i,\\
{}^{i}\mathbf{F}_i &= m_i\,{}^{i}\dot{\bar{\mathbf{v}}}_i,\\
{}^{i}\mathbf{N}_i &= \mathbf{J}_i\,{}^{i}\dot{\boldsymbol{\omega}}_i + {}^{i}\boldsymbol{\omega}_i \times \left(\mathbf{J}_i\,{}^{i}\boldsymbol{\omega}_i\right).
\end{aligned}
$$

Next, the inward recursion propagates (from $n \geq i \geq 1$) for the forces and moments exerted on each link from the end effector to the base reference frame as

$$
\begin{aligned}
{}^{i}\mathbf{f}_i &= {}^{i}\mathbf{R}_{i+1}\,{}^{i+1}\mathbf{f}_{i+1} + {}^{i}\mathbf{F}_i,\\
{}^{i}\mathbf{n}_i &= {}^{i}\mathbf{R}_{i+1}\left\{{}^{i+1}\mathbf{n}_{i+1} + \left({}^{i+1}\mathbf{R}_i\,{}^{i}\mathbf{p}^*_i\right) \times {}^{i+1}\mathbf{f}_{i+1}\right\} + \left({}^{i}\mathbf{p}^*_i + \mathbf{s}_i\right) \times {}^{i}\mathbf{F}_i + {}^{i}\mathbf{N}_i,\\
\boldsymbol{\tau}_i &= \left({}^{i}\mathbf{n}_i\right)^T \left({}^{i}\mathbf{R}_{i+1}\mathbf{z}_0\right),
\end{aligned}
$$

where

$${}^{i-1}\mathbf{R}_i =$$ an orthonormal rotation matrix,

$\mathbf{z}_0 =$ a unit vector in the Z direction $= [0\ 0\ 1]$,

${}^{i}\mathbf{p}^*_i =$ the displacement from the origin of frame $i-1$ to frame i with respect to frame i,

$\mathbf{s}_i =$ the position vector of the COM of link i with respect to frame i,

$m_i =$ the mass of link i,

$\mathbf{J}_i =$ the moment of inertia of link i about its COM,

$\boldsymbol{\omega}_i =$ the angular velocity of link i,

$\dot{\boldsymbol{\omega}}_i =$ the angular acceleration of link i,

$\mathbf{v}_i =$ the linear velocity of frame i,

$\dot{\mathbf{v}}_i =$ the linear acceleration of frame i,

$\dot{\bar{\mathbf{v}}}_i =$ the linear acceleration of the COM of link i,

$\mathbf{n}_i =$ the moment exerted on link i by link $i-1$,

$\mathbf{f}_i =$ the force exerted on link i by link $i-1$,

$\mathbf{N}_i =$ the total moment at the COM of link i,

$\mathbf{F}_i =$ the total force at the COM of link i,

$\boldsymbol{\tau}_i =$ the torque exerted by the actuator at joint i.

The base velocities and angular acceleration are generally initialized to $\mathbf{v}_0 = \boldsymbol{\omega}_0 = \dot{\boldsymbol{\omega}}_0 = 0$. In addition, to introduce the effect of gravity, the base link acceleration is set to $\dot{\mathbf{v}}_0 = -\mathbf{g}$, where $\mathbf{g}$ is the gravity vector in the base coordinate system. This formulation was used for the

PUMA 560 manipulator arm forward dynamics with rotational joints. For complete details and further explanations, please refer to [108, 109, 110]. The next section discusses how to implement the forward dynamic model, which employs the RNE algorithm to perform optimization and numerical simulations.

PUMA 560 numerical simulation model

A numerical simulation model was developed as prescribed in [109, 110, 107]. A symbolic derivation for the dynamics of the first three DOFs of the PUMA 560 manipulator was constructed in MAPLE [55]. The output was computer-generated C-code. The MATLAB CMEX function was used to wrap the C-code into the MATLAB environment. The numerical simulation for the manipulator dynamics and control implementation followed the developments in Chapters 6, 9, and 10 in [107]. Simple Euler integration was used to simulate an independent joint control (IJC) law [107] defined as

$$\tau = \tau_{IJC} = \mathbf{M}_{max}\left[\ddot{\mathbf{q}}_{REF} + \mathbf{K}_p\left(\mathbf{q}_{REF} - \mathbf{q}\right) + \mathbf{K}_v\left(\dot{\mathbf{q}}_{REF} - \dot{\mathbf{q}}\right)\right], \tag{5.93}$$

where $\mathbf{M}_{max}$, $\mathbf{K}_p$, and $\mathbf{K}_v$ are diagonal positive definite controller gains. In addition, $\mathbf{q}_{REF}$, $\dot{\mathbf{q}}_{REF}$, and $\ddot{\mathbf{q}}_{REF}$ are reference input position, velocity, and acceleration trajectories generated from the MATLAB Robotic Toolbox [106]. The function *jtraj.m* computes joint space trajectories between two goal points. The trajectory is based on a seventh-order polynomial. The numerical values for the controller are given in Table 5.12.

Table 5.12. *PUMA 560 independent joint control gain parameters.*

Joint	$\mathbf{M}_{max}$ (kg-m^2)	$\mathbf{K}_p$ (rad/sec)^2	$\mathbf{K}_v$ (rad/sec)
1	2.0	256.0	16.0
2	2.0	256.0	16.0
3	2.0	256.0	16.0

Closed-loop responses were generated by applying (5.93) to (5.92) with reference inputs provided by *jtraj.m*. The controller time histories served as the initial starting point for the DPIP algorithm.

The PUMA 560 manipulator dynamic C-code and reference time histories were incorporated directly into the DP driver and support routines. These included the driver script code *expuma2.m*, the cost function *copuma2.m*, the equality constraint function *eqpuma2.m*, and the inequality constraint function *inpuma2.m*; the embedded equations of motion were called from *espuma2.m*. The MAPLE-generated C-code includes the dynamic equations *yprimetest3.c* and the dynamic equations with zero gravity *ypnograv3.c*. These script files are given in Appendix B.6.

5.7.3 Dynamic Programming Optimization Design

The goal of this section is to design feedforward-optimized trajectories for an industrial robot that are subject to both transient actuator constraints and velocity constraints. Traditionally,

input trajectories are designed based on conventional cubic splines or polynomial trajectories. A popular higher-order polynomial that includes specifications for position, velocity, and acceleration is the seventh-order polynomial [106, 107]. Initially, this polynomial was used as a reference trajectory for a linear feedback IJC system to generate a controlled dynamic response to the reference input. An alternative to this traditional approach is to generate an optimized set of trajectories with DPIP. In this design, the closed-loop IJC responses were used as a nominal set of input trajectories to the DPIP algorithm. The DPIP algorithm was set up to minimize the control effort with the cost function given as

$$J = \sum_{0}^{t_f} \left(u_1^2 + u_2^2 + u_3^2\right) dt$$

subject to the PUMA 560 manipulator dynamics (5.92). Note that $u_i = \tau_i$ for $i = 1, 2, 3$. In addition, the PUMA 560 was subject to the equality constraints

$$q_1(t_f) - q_{1_{desired}} = q_2(t_f) - q_{2_{desired}} = q_3(t_f) - q_{3_{desired}} = 0$$

and

$$\dot{q}_1(t_f) = \dot{q}_2(t_f) = \dot{q}_3(t_f) = 0.$$

First, the actuator saturation inequality constraints were given as

$$u_{min_i} \leq u_i \leq u_{max_i}, \qquad i = 1, 2, 3.$$

Second, the joint velocity saturation inequality constraints were given as

$$\dot{q}_{min_i} \leq \dot{q}_i \leq \dot{q}_{max_i}, \quad i = 1, 2, 3.$$

The PUMA 560 was configured initially as

$$q_1(0) = q_{1_0}, \qquad q_2(0) = q_{2_0}, \qquad q_3(0) = q_{3_0}$$

and was initially at rest:

$$\dot{q}_1(0) = \dot{q}_2(0) = \dot{q}_3(0) = 0.$$

The final time was set to $t_f = 1.5$ sec. The number of discretization points was set to $N = 1501$. These numerical simulation results are reviewed in Section 5.7.5.

5.7.4 Implementation Issues

To generate the DPIP solution, the IJC system control effort time histories helped to provide a starting point in the neighborhood of the DPIP-optimized solution. The four-step process consisted of (1) generating minimum jerk trajectories between two goal points, (2) generating an IJC closed-loop response, (3) initiating the DPIP control effort with the IJC controller time histories, and (4) performing a sufficient number of iterations to minimize

the given cost function subject to the constraints. Although a minimum effort with bounds approach was investigated, other approaches based on minimum time, minimum energy, or minimum power could be investigated.

The overall DPIP optimization framework was established, and the manipulator model consisted of several simplifications. For further investigations and other physical effects, one could include (1) the full six-DOF dynamic model, (2) viscous/Coulomb friction terms, (3) nonlinear actuator/gear transmission models, and (4) other nonlinear terms (e.g., servo, amplifier, encoder, discretization effects, etc.). These types of details about the PUMA 560 manipulator servo control system can be found in [111]. The same procedure outlined in this case study could be modified to help determine many different cases of optimized feedforward trajectories for the robot manipulator dynamic and control design performance/limitation specifications.

5.7.5 Numerical Simulation Results

A minimum effort with bounds DPIP optimization run was performed for the PUMA 560. The four-step procedure outlined in the previous section was realized. A large-angle joint motion was selected for the analysis and is given as

$$\begin{Bmatrix} q_{1_0} \\ q_{2_0} \\ q_{3_0} \end{Bmatrix} = \begin{Bmatrix} 50^\circ \\ -135^\circ \\ 135^\circ \end{Bmatrix} \quad \text{to} \quad \begin{Bmatrix} q_{1_{desired}} \\ q_{2_{desired}} \\ q_{3_{desired}} \end{Bmatrix} = \begin{Bmatrix} -50^\circ \\ -85^\circ \\ 30^\circ \end{Bmatrix}.$$

These two goal points resulted in a trajectory that reduced gravitational effects. Normally, the PUMA 560 manipulator maneuvers are dominated by gravity forces [112, 113]. The DPIP optimization results are summarized in Table 5.13.

Table 5.13. *PUMA 560 DPIP optimization results.*

Iterations	$\mathbf{u}_{start}$	Convergence	t_{final} (sec)
12	τ_{IJC}	4.1817e−9	1.5

The goal of this run was to generate DPIP-optimized trajectories that perform a minimum-effort solution with in-transit constraints between two goal points. The position and velocity responses for each DOF are given in Fig. 5.27. The joints 1 to 3 correspond to the PUMA 560 shoulder yaw, shoulder pitch, and elbow pitch DOFs. In each plot, *ref* is the reference input polynomial trajectory, *closed* is the PUMA 560 IJC closed-loop response, and *dpip* is the DPIP-optimized trajectory response. The DPIP position responses are characteristically different from the others. The same observation is noted for the DPIP velocity responses. Note that the velocity responses are limited due to the in-transit velocity constraints. The limit or saturation values correspond to those listed in Table 5.11, column two (first three entries). This accounts for the different responses. The DPIP framework takes into account the in-transit velocity constraints during the optimization process. The acceleration and torque responses for each DOF are given in Fig. 5.28. The acceleration responses are recorded for the reference, closed-loop, and DPIP runs. The peak accelerations were all below the maximum allowed value (see Table 5.11, column three, first three entries).

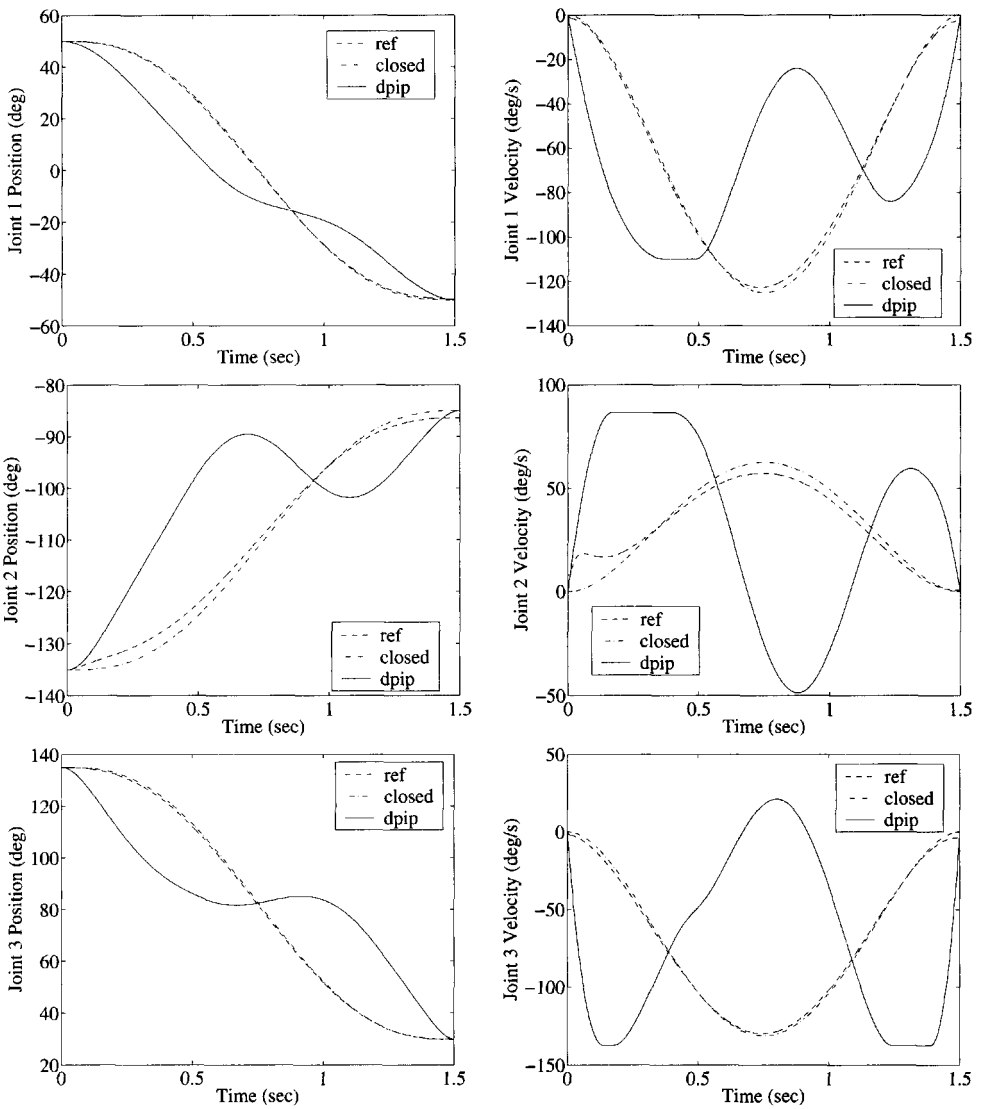


Figure 5.27. *PUMA 560 reference trajectories, closed-loop feedback responses, and feedforward minimum effort with velocity bounds numerical simulation trajectory results—positions (left column) and velocities (right column) for shoulder yaw, shoulder pitch, and elbow pitch.*

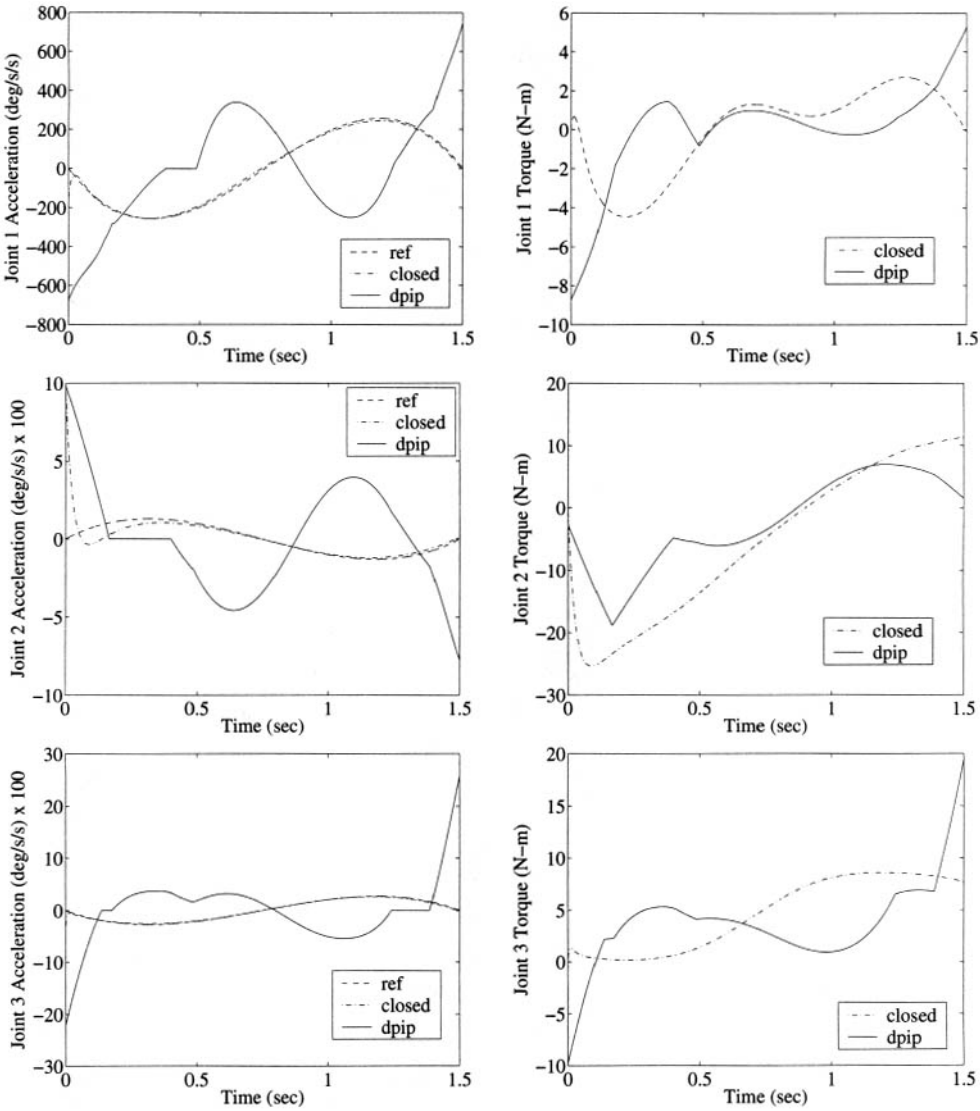


Figure 5.28. *PUMA 560 reference trajectories, closed-loop feedback responses, and feedforward minimum effort with velocity bounds numerical simulation trajectory results—accelerations (left column) and torques (right column) for shoulder yaw, shoulder pitch, and elbow pitch.*

The torque responses are recorded for both the closed-loop and DPIP runs. The torque response profiles show flat spots and mild cusps. These again are representative of the in-transit velocity constraints. However, the torques did not come close to saturation (see Table 5.11, column four, first three entries) during the maneuver. The numerical simulation results presented in Figs. 5.27–5.28 are representative of trajectory tracking evaluations [112, 113] for the first three links of the PUMA 560. These results demonstrate the impact of including in-transit constraints during feedforward-optimized trajectory designs.

5.7.6 Case Study 6 Discussion (Summary/Conclusions)

The objective of this case study was to design feedforward-optimized trajectories subject to in-transit constraints for a PUMA 560 manipulator. This required the development of the dynamic equations of motion, numerical implementation, optimized trajectory design, and verification through numerical simulations.

The RNE algorithm was employed to solve the forward dynamics problem. Only the first three DOFs of the PUMA 560 manipulator were considered. This allowed for the evaluation of gross motion and end-effector positions. A four-step process was identified to initiate the DPIP optimization design process. In-transit velocity and torque constraints influenced the actual realizable responses. All three joint velocities saturated during the maneuver. However, the DPIP algorithm was able to accommodate this scenario during the design and analysis process. The DPIP algorithm performed efficiently during the analysis. The framework for the PUMA 560 optimized trajectory problem was set up so that it would be capable of solving many other types of cost function variations. Future investigations could include minimum time, minimum energy, or minimum power type scenarios. Exploring more design variations will result in further understanding of the manipulator performance and limitations to execute the various maneuvers. This could also aid in providing dynamic and control system parameterized performance curves relevant to specific slewing maneuvers.

5.8 Summary

This chapter explored a series of case studies that demonstrated the application of the DPIP method algorithm. A congruent set of steps was applied to each case study. Every step helped to highlight the key features of the dynamical system model, the feedforward trajectory optimization designs, the implementation issues, and the review of the numerical simulation results. Both robotic and aerospace applications were included.

Case Study 1, the rotary jib crane, demonstrated optimized feedforward commands that produced rest-to-rest maneuvers with minimal residual oscillation. The DPIP algorithm was compared with RQP and showed equivalent results. However, DPIP required fewer iterations to converge to an optimized trajectory. Large-angle maneuvers with significant nonlinear dynamics effects were also considered and validated.

In Case Study 2, the slewing flexible link, both open-loop and closed-loop feedforward-optimized trajectories were investigated. The DPIP algorithm accommodated real-world dynamical effects such as friction during the design procedure. Comparisons were made with an RQP optimization approach. The results were validated through numerical simulations.

For the open-loop scenario, both minimum effort and minimum effort with bounds optimization designs were reviewed.

In Case Study 3, the flexible planar arm, open-loop feedforward-optimized trajectories were investigated. DPIP was able to demonstrate several important optimization scenarios representing actuator limitations and specific constraints. Four specific scenarios were investigated: (1) minimum effort, characteristic of systems that need to conserve energy from point to point; (2) minimum effort with bounds, which are representative of realistic actuator constraints; (3) minimum time, representative of the theoretically maximum performance possible to complete a maneuver; and (4) minimum torque rate effort, which corresponds to minimizing the vibration levels during the in-transit portion of the maneuver. This case study can assist the system design engineer in maximizing specific performance criteria, subject to certain types of boundary constraints.

In Case Study 4, minimum-time maneuvers of rigid systems, time-optimal maneuvers were investigated for rigid symmetric spacecraft. Initially, the DPIP was configured to solve the double integrator problem. Finally, the DPIP was able to determine all four time-optimal solutions with corresponding initial control settings. Comparisons were made with an alternative optimization program that required significantly more iterations.

In Case Study 5, maximum-radius orbit transfers, DPIP was employed for the transfer of a rocket vehicle from some given initial circular orbit to the largest possible circular orbit. Results were obtained and reviewed for the initial thrust direction history for the orbit transfers and corresponding time histories. The DPIP results compared favorably with previously published material.

Finally, Case Study 6, PUMA 560 industrial manipulator, determined optimized DPIP trajectories for in-transit constraints, including both torque and velocity. It was illustrated through numerical simulations that the transient joint velocities saturated, a characteristic of many servo motor-voltage limitations. An automated dynamic model development system helped to realize the nonlinear dynamic equations of motion for the PUMA 560 manipulator. The model, along with the control system, was utilized by the DPIP algorithm. The optimized results reflected the in-transit constraints in the DPIP solution. Although only the optimization of a minimum-effort cost function with in-transit constraints was investigated, many other cost function variations could be considered with the accommodating design and analysis framework that was established.

In conclusion, this series of case studies demonstrated the efficiency and effectiveness of the DPIP algorithm to solve for solutions to many different conditions that may be needed to generate feedforward-type optimized trajectories, both for minimum-time and other energy-related cost functions. The DPIP direct method approach often reduced the overall number of iterations to convergence in comparison with other popular indirect methods such as SQP/RQP. These case studies were intentionally presented so that each step was explicitly developed to include sufficient detail for the benefit of the reader.

The DPIP algorithm proved to be an effective off-line dynamics and control design tool to support performance trade studies. In the future, it could also be implemented to support *critical* flight control system optimized real-time trajectory generation/modification. Although the DPIP focus was primarily on dynamical or physical systems, a much wider class of problems (e.g., econometric) could be addressed that lie outside the intended purpose and scope of this material.

Appendix A
Mathematical Supplement

A.1 Determination of the State Transition Matrices

Given the existence and uniqueness of the solution to the initial value problem given by (4.3), adjacent states in time can be related as

$$\mathbf{x}_{i+1} = \mathbf{g}_k(\mathbf{x}_k, \mathbf{u}_k). \tag{A.1}$$

The state transition matrices introduced in (4.24) are defined as

$$\mathbf{A}_k = \frac{\partial \mathbf{g}_k}{\partial \mathbf{x}_k}, \tag{A.2}$$

$$\mathbf{B}_k = \frac{\partial \mathbf{g}_k}{\partial \mathbf{u}_k}, \tag{A.3}$$

where the partial derivatives in (A.2) and (A.3) are evaluated at the nominal trajectory. As was mentioned previously, a fixed-step, fourth-order Runge–Kutta (RK) method is used to numerically integrate (4.3). The formula used by this method is

$$\mathbf{g}_k = \mathbf{x}_k + \frac{(\mathbf{k}_1 + \mathbf{k}_2 + \mathbf{k}_3 + \mathbf{k}_4)}{6}, \tag{A.4}$$

where

$$\mathbf{k}_1 = h\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, t_k), \tag{A.5}$$

$$\mathbf{k}_2 = h\mathbf{f}\left(\mathbf{x}_k + \frac{\mathbf{k}_1}{2}, \mathbf{u}_k, t_k + \frac{h}{2}\right), \tag{A.6}$$

$$\mathbf{k}_3 = h\mathbf{f}\left(\mathbf{x}_k + \frac{\mathbf{k}_2}{2}, \mathbf{u}_k, t_k + \frac{h}{2}\right), \tag{A.7}$$

$$\mathbf{k}_4 = h\mathbf{f}(\mathbf{x}_k + \mathbf{k}_3, \mathbf{u}_k, t_k + h). \tag{A.8}$$

Using the chain rule for differentiation, one obtains

$$\mathbf{A}_k = \mathbf{I} + \frac{1}{6}\left(\frac{\partial \mathbf{k}_1}{\partial \mathbf{x}_k} + 2\frac{\partial \mathbf{k}_2}{\partial \mathbf{x}_k} + 2\frac{\partial \mathbf{k}_3}{\partial \mathbf{x}_k} + \frac{\partial \mathbf{k}_4}{\partial \mathbf{x}_k}\right), \tag{A.9}$$

$$\mathbf{B}_k = \mathbf{I} + \frac{1}{6}\left(\frac{\partial \mathbf{k}_1}{\partial \mathbf{u}_k} + 2\frac{\partial \mathbf{k}_2}{\partial \mathbf{u}_k} + 2\frac{\partial \mathbf{k}_3}{\partial \mathbf{u}_k} + \frac{\partial \mathbf{k}_4}{\partial \mathbf{u}_k}\right), \tag{A.10}$$

where $\mathbf{I}$ is the identity matrix and

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{x}_k} = h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_k}\right|_{\mathbf{x}_k, \mathbf{u}_k, t_k}, \tag{A.11}$$

$$\frac{\partial \mathbf{k}_1}{\partial \mathbf{u}_k} = h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}_k}\right|_{\mathbf{x}_k, \mathbf{u}_k, t_k}, \tag{A.12}$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{x}_k} = h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_k}\right|_{\mathbf{x}_k + \frac{\mathbf{k}_1}{2}, \mathbf{u}_k, t_k + \frac{h}{2}}\left(\mathbf{I} + \frac{1}{2}\frac{\partial \mathbf{k}_1}{\partial \mathbf{x}_k}\right), \tag{A.13}$$

$$\frac{\partial \mathbf{k}_2}{\partial \mathbf{u}_k} = h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}_k}\right|_{\mathbf{x}_k + \frac{\mathbf{k}_1}{2}, \mathbf{u}_k, t_k + \frac{h}{2}}\left(\mathbf{I} + \frac{1}{2}\frac{\partial \mathbf{k}_1}{\partial \mathbf{u}_k}\right), \tag{A.14}$$

$$\frac{\partial \mathbf{k}_3}{\partial \mathbf{x}_k} = h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_k}\right|_{\mathbf{x}_k + \frac{\mathbf{k}_2}{2}, \mathbf{u}_k, t_k + \frac{h}{2}}\left(\mathbf{I} + \frac{1}{2}\frac{\partial \mathbf{k}_2}{\partial \mathbf{x}_k}\right), \tag{A.15}$$

$$\frac{\partial \mathbf{k}_3}{\partial \mathbf{u}_k} = h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}_k}\right|_{\mathbf{x}_k + \frac{\mathbf{k}_2}{2}, \mathbf{u}_k, t_k + \frac{h}{2}}\left(\mathbf{I} + \frac{1}{2}\frac{\partial \mathbf{k}_2}{\partial \mathbf{u}_k}\right), \tag{A.16}$$

$$\frac{\partial \mathbf{k}_4}{\partial \mathbf{x}_k} = h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}_k}\right|_{\mathbf{x}_k + \mathbf{k}_3, \mathbf{u}_k, t_k + h}\left(\mathbf{I} + \frac{\partial \mathbf{k}_3}{\partial \mathbf{x}_k}\right), \tag{A.17}$$

$$\frac{\partial \mathbf{k}_4}{\partial \mathbf{u}_k} = h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}_k}\right|_{\mathbf{x}_k + \mathbf{k}_3, \mathbf{u}_k, t_k + h}\left(\mathbf{I} + \frac{\partial \mathbf{k}_3}{\partial \mathbf{u}_k}\right). \tag{A.18}$$

A.2 Example 4.3 Numerical Approach

For a fourth-order RK method, the right-hand side of (4.36) is given by

$$\mathbf{g}_i = \mathbf{x}_i + \frac{(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)}{6}, \tag{A.19}$$

where

$$\mathbf{k}_j = h\mathbf{f}(\bar{\mathbf{x}}_j), \tag{A.20}$$

$$(\bar{\mathbf{x}}_j) = (\mathbf{x}_i + \sigma_j \mathbf{k}_{j-1}, \mathbf{u}_i, t_i + \sigma_j h), \tag{A.21}$$

$$[\sigma_1 \ \ \sigma_2 \ \ \sigma_3 \ \ \sigma_4] = \left[0 \ \ \frac{1}{2} \ \ \frac{1}{2} \ \ 1\right], \tag{A.22}$$

$$h = t_{i+1} - t_i. \tag{A.23}$$

Given three vectors, **a**, **b**, and **c**, define

$$a_b^{\alpha\beta j} = \left.\frac{\partial a^\alpha}{\partial b^\beta}\right|_{\bar{\mathbf{x}}_j}, \tag{A.24}$$

$$a_{bc}^{\alpha\beta\gamma j} = \left.\frac{\partial^2 a^\alpha}{\partial b^\beta \partial c^\gamma}\right|_{\bar{\mathbf{x}}_j}. \tag{A.25}$$

The right-hand sides of (4.45)–(4.49) are obtained from differentiation of (A.19). The result is

$$a_i^{\alpha\beta} = \delta^{\alpha\beta} + \frac{1}{6}(k_{1x_i}^{\alpha\beta} + 2k_{2x_i}^{\alpha\beta} + 2k_{3x_i}^{\alpha\beta} + k_{4x_i}^{\alpha\beta}), \tag{A.26}$$

$$b_i^{\alpha\beta} = \frac{1}{6}(k_{1u_i}^{\alpha\beta} + 2k_{2u_i}^{\alpha\beta} + 2k_{3u_i}^{\alpha\beta} + k_{4u_i}^{\alpha\beta}), \tag{A.27}$$

$$c_i^{\alpha\beta\gamma} = \frac{1}{6}(k_{1x_ix_i}^{\alpha\beta\gamma} + 2k_{2x_ix_i}^{\alpha\beta\gamma} + 2k_{3x_ix_i}^{\alpha\beta\gamma} + k_{4x_ix_i}^{\alpha\beta\gamma}), \tag{A.28}$$

$$d_i^{\alpha\beta\gamma} = \frac{1}{6}(k_{1x_iu_i}^{\alpha\beta\gamma} + 2k_{2x_iu_i}^{\alpha\beta\gamma} + 2k_{3x_iu_i}^{\alpha\beta\gamma} + k_{4x_iu_i}^{\alpha\beta\gamma}), \tag{A.29}$$

$$e_i^{\alpha\beta\gamma} = \frac{1}{6}(k_{1u_iu_i}^{\alpha\beta\gamma} + 2k_{2u_iu_i}^{\alpha\beta\gamma} + 2k_{3u_iu_i}^{\alpha\beta\gamma} + k_{4u_iu_i}^{\alpha\beta\gamma}), \tag{A.30}$$

where $\delta^{\alpha\beta}$ is the Kronecker delta symbol and

$$k_{jx_i}^{\alpha\beta} = h(f_{x_i}^{\alpha\beta j} + \sigma_j f_{x_i}^{\alpha\eta j} k_{j-1,x_i}^{\eta\beta}), \tag{A.31}$$

$$k_{ju_i}^{\alpha\beta} = h(f_{u_i}^{\alpha\beta j} + \sigma_j f_{x_i}^{\alpha\eta j} k_{j-1,u_i}^{\eta\beta}), \tag{A.32}$$

$$k_{jx_ix_i}^{\alpha\beta\gamma} = h[(f_{x_ix_i}^{\alpha\eta\gamma j} + \sigma_j f_{x_ix_i}^{\alpha\eta\epsilon j} k_{j-1,x_i}^{\epsilon\gamma})(\delta^{\eta\beta} + \sigma_j k_{j-1,x_i}^{\eta\beta}) + \sigma_j f_{x_i}^{\alpha\eta j} k_{j-1,x_ix_i}^{\eta\beta\gamma}], \tag{A.33}$$

$$k_{jx_iu_i}^{\alpha\beta\gamma} = h[(f_{x_iu_i}^{\alpha\eta\gamma j} + \sigma_j f_{x_ix_i}^{\alpha\eta\epsilon j} k_{j-1,u_i}^{\epsilon\gamma})(\delta^{\eta\beta} + \sigma_j k_{j-1,x_i}^{\eta\beta}) + \sigma_j f_{x_i}^{\alpha\eta j} k_{j-1,x_iu_i}^{\eta\beta\gamma}], \tag{A.34}$$

$$\begin{aligned} k_{ju_iu_i}^{\alpha\beta\gamma} = h[&(f_{x_iu_i}^{\alpha\eta\gamma j} + \sigma_j f_{x_ix_i}^{\alpha\eta\epsilon j} k_{j-1,u_i}^{\epsilon\gamma})\sigma_j k_{j-1,u_i}^{\eta\beta} + (f_{u_iu_i}^{\alpha\beta\gamma j} + \sigma_j f_{x_iu_i}^{\alpha\epsilon\eta j} k_{j-1,u_i}^{\epsilon\gamma}) \\ &+ \sigma_j f_{x_i}^{\alpha\eta j} k_{j-1,u_iu_i}^{\eta\beta\gamma}]. \end{aligned} \tag{A.35}$$

A.3 Modified Dynamic Programming Algorithm

The algorithm presented below is used to solve the equivalent problem of Section 4.4.4. Let

$$\mathbf{H}_{1i} = \bar{\mathbf{Q}}_i + \mathbf{A}_i^T \mathbf{W}_{i+1} \mathbf{A}_i, \tag{A.36}$$

$$\mathbf{H}_{2i} = \bar{\mathbf{R}}_i + \mathbf{A}_i^T \mathbf{W}_{i+1} \mathbf{B}_i, \tag{A.37}$$

$$\mathbf{H}_{3i} = \bar{\mathbf{S}}_i + \mathbf{B}_i^T \mathbf{W}_{i+1} \mathbf{B}_i, \tag{A.38}$$

$$\mathbf{h}_{4i} = \bar{\mathbf{y}}_i + \mathbf{A}_i^T \boldsymbol{\nu}_{i+1}, \tag{A.39}$$

$$\mathbf{h}_{5i} = \bar{\mathbf{z}}_i + \mathbf{B}_i^T \boldsymbol{\nu}_{i+1}, \tag{A.40}$$

$$\mathbf{F}_{1i} = \begin{bmatrix} \mathbf{H}_{3i} & \bar{\mathbf{B}}_i^T & \mathbf{B}_i^T \mathbf{A}_{i+1,n}^T \\ \bar{\mathbf{B}}_i & -\bar{\mathbf{G}}_i & \mathbf{0} \\ \mathbf{A}_{i+1,n} \mathbf{B}_i & \mathbf{0} & \mathbf{B}_{i+1,n} \end{bmatrix}, \tag{A.41}$$

$$\mathbf{F}_{2i} = \begin{bmatrix} \mathbf{H}_{2i}^T \\ \bar{\mathbf{A}}_i \\ \mathbf{A}_{i+1,n} \mathbf{A}_i \end{bmatrix}, \tag{A.42}$$

$$\mathbf{f}_{3i} = \begin{bmatrix} \mathbf{h}_{5i} \\ \bar{\mathbf{c}}_i \\ \mathbf{c}_{i+1,n} \end{bmatrix}, \tag{A.43}$$

where the terms $\mathbf{A}_{in}$, $\mathbf{B}_{in}$, $\mathbf{c}_{in}$, $\boldsymbol{\nu}_i$, and $\mathbf{W}_i$ are calculated from recursive equations presented subsequently. The only modification to the algorithm [29] is the presence of the matrix $\bar{\mathbf{G}}_i$ in (A.41).

Two options that involve different factorizations of $\mathbf{F}_{1i}$ are considered in [29] for solving the equivalent problem. The first is based on the eigenvalues and eigenvectors of $\mathbf{F}_{1i}$, while the second is based on the LDL^T decomposition. For simplicity of exposition, one considers the first option, in which $\mathbf{F}_{1i}$ is factored as

$$\mathbf{F}_{1i} = \boldsymbol{\Phi}_i \boldsymbol{\Sigma}_i \boldsymbol{\Phi}_i^T, \tag{A.44}$$

where $\boldsymbol{\Phi}_i$ is an orthogonal matrix of eigenvectors and $\boldsymbol{\Sigma}_i = \mathrm{diag}(\sigma_1, \ldots, \sigma_{nf})$. For convenience, the eigenvalues σ_j are assumed to be sorted such that $|\sigma_j| \geq |\sigma_{j+1}|$ for $j = 1, \ldots, n_{if-1}$. Define

$$n_{ir} = \max\left(\max\left\{j \;\middle|\; \left|\frac{\sigma_j}{\sigma_1}\right| \geq \frac{1}{c_{max}}\right\}, n_{if} - n\right), \tag{A.45}$$

$$n_{in} = n_{if} - n_{ir}, \tag{A.46}$$

and

$$\boldsymbol{\Sigma}_{ir} = \mathrm{diag}(\sigma_1, \ldots, \sigma_{n_{ir}}), \tag{A.47}$$

$$\boldsymbol{\Sigma}_{in} = \mathrm{diag}(\sigma_{n_{ir}+1}, \ldots, \sigma_{n_{if}}), \tag{A.48}$$

$$[\boldsymbol{\Phi}_{ir} \quad \boldsymbol{\Phi}_{in}] = \boldsymbol{\Phi}_i, \tag{A.49}$$

where $\boldsymbol{\Phi}_{ir} \in R^{n_{if} \times n_{ir}}$, $\boldsymbol{\Phi}_{in} \in R^{n_{if} \times n_{in}}$, and c_{max} is a positive scalar.

The algorithm to solve the equivalent problem is summarized below.

Step 1: Calculate

$$\mathbf{A}_{Nn} = \bar{\mathbf{A}}_N, \tag{A.50}$$

$$\mathbf{B}_{Nn} = -\bar{\mathbf{G}}_N, \tag{A.51}$$

$$\mathbf{c}_{Nn} = \bar{\mathbf{c}}_N, \tag{A.52}$$

$$\boldsymbol{\nu}_N = \bar{\mathbf{y}}_N, \tag{A.53}$$

$$\mathbf{W}_N = \bar{\mathbf{Q}}_N. \tag{A.54}$$

Step 2: Calculate

$$\mathbf{A}_{in} = \boldsymbol{\Phi}_{in}^T \mathbf{F}_{2i}, \tag{A.55}$$

$$\mathbf{B}_{in} = \boldsymbol{\Sigma}_{in}, \tag{A.56}$$

$$\mathbf{c}_{in} = \boldsymbol{\Phi}_{in}^T \mathbf{f}_{3i}, \tag{A.57}$$

$$\mathbf{W}_i = \mathbf{H}_{1i} - \mathbf{F}_{2i}^T \boldsymbol{\Phi}_{ir}^T \mathbf{E}_{1i}, \tag{A.58}$$

$$\boldsymbol{\nu}_i = \mathbf{h}_{4i} - \mathbf{F}_{2i}^T \boldsymbol{\Phi}_{ir}^T \mathbf{e}_{2i}, \tag{A.59}$$

where

$$\mathbf{E}_{1i} = \boldsymbol{\Sigma}_{ir}^{-1} \boldsymbol{\Phi}_{ir}^T \mathbf{F}_{2i}, \tag{A.60}$$

$$\mathbf{e}_{2i} = \boldsymbol{\Sigma}_{ir}^{-1} \boldsymbol{\Phi}_{ir}^T \boldsymbol{f}_{3i} \tag{A.61}$$

for $i = N-1, \ldots, 1$, storing the terms $\mathbf{E}_{1i}$, $\mathbf{e}_{2i}$, and $\boldsymbol{\Phi}_i$ in the process.

Step 3: Solve the linear system

$$\begin{bmatrix} \mathbf{W}_1 & \mathbf{A}_{1n}^T \\ \mathbf{A}_{1n} & \mathbf{B}_{1n} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_1 \\ \tilde{\mathbf{a}}_{1n} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{\nu}_1 \\ \mathbf{c}_{1n} \end{bmatrix}. \tag{A.62}$$

Step 4: Calculate

$$\tilde{\mathbf{a}}_{ir} = -\left(\mathbf{E}_{1i}\tilde{\mathbf{x}}_i + \mathbf{e}_{2i}\right), \tag{A.63}$$

$$\tilde{\mathbf{a}}_{it} = \boldsymbol{\Phi}_i [\tilde{\mathbf{a}}_{ir}^T \ \tilde{\mathbf{a}}_{in}^T]^T, \tag{A.64}$$

$$[\tilde{\mathbf{u}}_i^T \ \tilde{\boldsymbol{\lambda}}_i^T \ \tilde{\mathbf{a}}_{i+1,n}^T] = \tilde{\mathbf{a}}_{it}^T, \tag{A.65}$$

$$\tilde{\mathbf{x}}_{i+1} = \mathbf{A}_i \tilde{\mathbf{x}}_i + \mathbf{B}_i \tilde{\mathbf{u}}_i \tag{A.66}$$

for $1, \ldots, N-1$ and set $\tilde{\boldsymbol{\lambda}}_N = \tilde{\mathbf{a}}_{Nn}$.

The residual terms in (4.269) are given by

$$(\mathbf{r}_1^k)^2 = \sum_{i=1}^{N-1} \left(\frac{\partial L_q^k}{\partial \tilde{\mathbf{u}}_i}\right)^T \left(\frac{\partial L_q^k}{\partial \tilde{\mathbf{u}}_i}\right), \tag{A.67}$$

$$(\mathbf{r}_2^k)^2 = \sum_{i=1}^{N} \left(\mathbf{A}_{iE}\mathbf{x}_i^k + \mathbf{B}_{iE}\mathbf{u}_i^k + \hat{\mathbf{c}}_{iE}\right)^T \left(\mathbf{A}_{iE}\mathbf{x}_i^k + \mathbf{B}_{iE}\mathbf{u}_i^k + \hat{\mathbf{c}}_{iE}\right), \tag{A.68}$$

$$(\mathbf{r}_3^k)^2 = \sum_{i=1}^{N} \left(\mathbf{A}_{iI}\mathbf{x}_i^k + \mathbf{B}_{iI}\mathbf{u}_i^k + \hat{\mathbf{c}}_{iI} + \mathbf{s}_{iI}^k\right)^T \left(\mathbf{A}_{iI}\mathbf{x}_i^k + \mathbf{B}_{iI}\mathbf{u}_i^k + \hat{\mathbf{c}}_{iI} + \mathbf{s}_{iI}^k\right), \tag{A.69}$$

$$(\mathbf{r}_4^k)^2 = \sum_{i=1}^{N} [\min(\boldsymbol{\lambda}_{iI}^k, \mathbf{s}_{iI}^k)]^T [\min(\boldsymbol{\lambda}_{iI}^k, \mathbf{s}_{iI}^k)], \tag{A.70}$$

where

$$\frac{\partial L_q^k}{\partial \tilde{\mathbf{u}}_i} = \bar{\mathbf{z}}_i + \mathbf{B}_i^T \mathbf{c}_{i+1} \tag{A.71}$$

and $\mathbf{c}_i$ is obtained from the recursive equation

$$\mathbf{c}_i = \bar{\mathbf{y}}_i + \mathbf{A}_i^T \mathbf{c}_{i+1} \tag{A.72}$$

starting with

$$\mathbf{c}_N = \bar{\mathbf{y}}_N. \tag{A.73}$$

Appendix B

Applied Case Studies—MATLAB Software Addendum

B.1 Case Study 1 Driver Functions and Script Files

This appendix includes driver functions and MATLAB script files used to solve the DPIP problem in Case Study 1.

B.1.1 Rotary Jib Crane Numerical Simulations Code

The simulation driver script file *jibsim.m*:

```
    test
```

```
%
% This file simulates linear 2 DOF jib crane using ode23
%
t0=0.0;
tfinal=5.0;
phi_init = 0.0;
theta_init = 0.0;
y0 = [ theta_init phi_init 0.0 0.0 0.0 0.0]';
%
% Perform numerical simulation
%
%     dyn1  - linear simulation
%     dyn2  - nonlinear simulation
%
[t,y]=ode23('dyn1',t0,tfinal,y0);
%[t,y]=ode23('dyn2',t0,tfinal,y0);
%
% recreate torque profiles
%
[m,n]=size(y);
%accel=2.0;
%t_a=0.5;
%t_c=1.0;
```

```
params
gam_init = 0.0;
gamdt_init=0.0;
for i = 1:m,
if t(i) < t_a
gamddt(i) = accel;
elseif (( t(i) >= t_a) & (t(i) <= t_a+t_c))
gamddt(i) = 0.0;
elseif ( ( t(i) > t_c) & ( t(i) <= 2*t_a+t_c))
gamddt(i) = -accel;
elseif ( t(i) > 2*t_a+t_c)
gamddt(i) = 0.0;
end
end
```

The linear equations of motion script file *dyn1.m*:

```
function yprime = dyn1(t,y);
%
% linear dynamic  equations of motion for 2 DOF jib crane
% which is set up as a script file in jibsim.m
%
% Pendulum mass, m; Pendulum length, d; Boom length, l
%
m = 0.05;     % kg
d = 0.251;    % m
l = 0.659;    % m
g = 9.81;       % gravitational constant  m/sec^2
gam_init = 0.0;
gamdt_init = 0.0;
%fric = 0.0001;
fric = 0.0;
%accel = 1.0;  % constant acceleration
%t_a = 1.0;
%t_c = 0.5;
params
%
% set up state variables
%
x(1,1)=y(1);
x(2,1)=y(2);
xdt(1,1)=y(3);
xdt(2,1)=y(4);
gam=y(5);
gamdt=y(6);
%
% Prescribed Input
%
if t < t_a
gamddt=accel;
elseif (( t>= t_a) & (t <= t_c+t_a))
```

```
gamddt=0.0;
elseif ( ( t > t_c) & ( t <= 2*t_a+t_c) )
gamddt= -accel;
elseif ( t > 2*t_a+t_c)
gamddt=0.0;
end
%
% Define z parameters
%
z(1)=m*d^2;
z(2)=2.0*z(1)*gamdt;
z(3)=z(1)*gamddt;
z(4)=z(1)*gamdt^2;
z(5)=m*d;
z(6)=g*z(5);
z(7)=z(6)-z(4);
z(8)=l*z(5);
z(9)=z(8)*gamdt^2;
z(10)=z(8)*gamddt;
%
% Mass matrix (2 x 2)
%
mass=[z(1) 0.0; 0.0 z(1)];
%
% Damping Matrix (2 x 2)
%
damp(1,1) =0.0+ fric;
damp(1,2)=z(2);
damp(2,1)=-z(2);
damp(2,2) = 0.0+ fric;
%
% Stiffening Matrix (2 x 2)
%
stiff(1,1) = z(7);
stiff(1,2) = z(3);
stiff(2,1) = -z(3);
stiff(2,2) = z(7);
%
% prescribed input vector
%
tau(1) = -z(9);
tau(2) = -z(10);
%
% Solve for accelerations
%
xddt = inv(mass)*(tau' -damp*xdt - stiff*x);
%
% send y prime back to main simulation file jibsim.m
%
yprime = [ xdt(1); xdt(2); xddt(1); xddt(2); gamdt; gamddt];
```

The nonlinear equations of motion script file *dyn2.m*:

```
function yprime = dyn2(t,y);
%
% nonlinear dynamic  equations of motion for 2 DOF jib crane
% which is set up as a script file in jibsim.m
%
% Pendulum mass, m; Pendulum length, d; Boom length, l
%
m = 0.05;     % kg
d = 0.251;    % m
l = 0.659;    % m
g = 9.81;     % gravitational constant  m/sec^2
gam_init = 0.0;
gamdt_init = 0.0;
%fric = 0.0001;
fric = 0.0;
params
%
% set up state variables
%
x(1,1)=y(1);
x(2,1)=y(2);
xdt(1,1)=y(3);
xdt(2,1)=y(4);
gam=y(5);
gamdt=y(6);
%
% Prescribed Input
%
if t < t_a
gamddt=accel;
elseif (( t>= t_a) & (t <= t_c+t_a))
gamddt=0.0;
elseif ( ( t > t_c) & ( t <= 2*t_a+t_c) )
gamddt= -accel;
elseif ( t > 2*t_a+t_c)
gamddt=0.0;
end
%
% Define z parameters
%
z(1)=sin(x(1,1));
z(2)=cos(x(1,1));
z(3)=sin(x(2,1));
z(4)=cos(x(2,1));
z(5)=g/d;
z(6)=l/d;
z(7)=2.0*(z(3)/z(4));
z(8)=2.0*z(2);
```

```
z(9)=z(2)*z(3)/z(4);
z(10)=z(2)*(z(1)-z(6)/z(4));
z(11)=z(5)*z(1)/z(4);
z(12)=z(3)*z(4);
z(13)=z(8)*z(4)^2;
z(14)=z(4)*z(6)-z(1);
z(15)=z(3)*(z(2)^2*z(4)+z(6)*z(1));
z(16)=z(5)*z(2)*z(3);
%
% Solve for accelerations
%
xddt(1,1) = z(7)*xdt(1,1)*xdt(2,1)-z(8)*gamdt*xdt(2,1)
           -z(9)*gamddt+z(10)*gamdt^2-z(11);
xddt(2,1) = -z(12)*xdt(1,1)^2+z(13)*gamdt*xdt(1,1)
            -z(14)*gamddt+z(15)*gamdt^2-z(16);
%
% send y prime back to main simulation file jibsim.m
%
yprime = [ xdt(1); xdt(2); xddt(1); xddt(2); gamdt; gamddt];
```

The output plot script file *jibplt.m*:

```
% This file plots all jib crane data of interest
%
plot(t,y(:,1)),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Theta (rad)')
pause
plot(t,y(:,2)),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Phi (rad)')
pause
plot(t,y(:,3)),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Theta_dt (rad/sec)')
pause
plot(t,y(:,4)),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Phi_dt (rad/sec)')
pause
plot(t,gamddt),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Gamma_ddt (rad/sec/sec)')
pause
plot(t,y(:,6)),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Gamma_dt (rad/sec)')
pause
plot(t,y(:,5)),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Gamma (rad)')
pause
plot(t,y(:,1),t,y(:,2)),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Theta  Phi (rad)')
pause
plot(t,y(:,3),t,y(:,4)),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Theta_dt  Phi_dt (rad/sec)')
```

```
pause
plot(t,y(:,1),t,y(:,2),t,y(:,3),t,y(:,4)),grid,title('Jib Crane Sim'),
xlabel('Time(sec)'),ylabel('Theta Phi Theta_dt  Phi_dt')
pause
plot(t,y(:,5),t,y(:,6),t,gamddt),grid,title('Jib Crane Simulation'),
xlabel('Time(sec)'),ylabel('Input accel vel pos')
```

B.1.2 Dynamic Programming/Interior-Point Driver, Cost, Equality, Inequality, and Dynamics Code

The DPIP driver code script file *exjcrane*1*.m*:

```
% Solves rotary jib crane problem - 90 deg trajectory
% number of states
n=6;
% number of controls
m=1;
% number of steps
N=201;

% problem parameters
% initial and final conditions
wi = [0 0 0 0 0 0];
wf = [0 0 0 0 pi/2 0];
ti=0;
tf=2.2;

x1h = wi'; % initial conditions of states
xNh=wf';

    dt=(tf-ti)/(N-1);
    t=ti:dt:tf;
    t=t';

uh=ones(N,1);

lamEh=ones(12,1)*0; % six initial/final specifications of states

lamIh=[]

controlbound=1;
param=[ti tf wi wf controlbound];

% view trajectory from initial guess
    [t,xh]=drungeu('esjcrane1',[param(1) param(2)],N-1,x1h,uh);
    figure(1),
    subplot(121), plot(t,xh,'.-'); title('x'); grid; shg
    subplot(122), plot(t,uh,'.-'); grid; title('u'); shg
OIG=input('Optimize initial guess? (y=1/n=0)   ');
if (OIG==1),
```

```
iterateQ=1;
counter=1;
alpha=1;
maxit=8;
while iterateQ==1,
%while iterateQ<=maxit,
    %  dpeq returns changes to nominal values
    %  ub : change in controls
    %  x1b : change in initial state
    %  lamEb : change in Lagrange multipliers for equality constraints

    [ub,x1b,lamEb,lamIb]=dpip(uh,x1h,lamEh,lamIh,N,n,m,'esjcrane1',...
                          'cojcrane1','eqjcrane1','injcrane1',param);
    uh = uh + alpha*ub;
    uh(N,:) = uh(N-1,:);
    [t,xh]=drungeu('esjcrane1',[param(1) param(2)],N-1,x1h,uh);

        fcerror=eqjcrane1(xh,uh,N,N,n,m,param);
        fcnorm=norm(fcerror);
        fprintf(1,'fcnorm = %g\n',fcnorm)
        fprintf(1,'\n');
        figure(1),
        subplot(121), plot(t,xh,'.-'); title('x'); grid; shg
        subplot(122), plot(t,uh,'.-'); grid; title('u'); shg

    iterateQ=input('Continue to iterate? (y=1/n=0)   ');
%    iterateQ=iterateQ+1;
    counter=counter+1;
end
end
shg
```

The cost function script file *cojcrane1.m*:

```
function [y,z,Q,R,S]=cojcrane1(xh,uh,i,N,n,m,param);
%
% Defines the cost function and some partial derivatives
% f is cost function
% yy is state
% uu is control
% xx is independent variable
%
% y is dfdy
% z is dfdu
% Q is dfdyy (second derivative)
% R is dfdydu
% S is dfduu

t1=param(1);
t2=param(2);
```

```
dt=(t2-t1)/(N-1);

tt=t1+dt*(i-1);  % current x (time, or independent variable); x_i
xx=xh(i,:);  % current y (state); y_i
uu=uh(i,:);  % current u (control); u_i

dfdx = zeros(n,1);
dfdxx = zeros(n,n);
dfdu = dt*[uu(1)]';
dfduu = dt;
y=dfdx;
z=dfdu;
Q=dfdxx;
R=zeros(n,m); % dfdydu matrix of partials that couples states/controls
S=dfduu;
if (i==1)+(i==N) % the plus sign is an "or"
  y=y/2; % the first and second partials of the cost wrt
         % the state are different if at first or last node
  Q=Q/2;
end
```

The equality constraint script file *eqjcrane1.m*:

```
function [cE,AE,BE,cExx,cExu,cEuu]=conejcrane1(xh,uh,i,N,n,m,param);
%  cone   name of function to calculate c_{iE}, A_{iE}, B_{iE},
%           c_{iE}_{xx}, c_{iE}_{xu} and c_{iE}_{uu}

% param=[ti tf xini' xfin'];

% initial and final equality constraints

nc=0;
[cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);

if i==1
  nc=6;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
ic=param(3:8);
  cE=xh(i,:)'-ic';
  AE=eye(6);
end

if i==N
  nc=6;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  fc=param(9:14);
  cE=xh(i,:)'-fc';
  AE=eye(6);
end
```

The inequality constraint script file *injcrane1.m*:

```
function [cI,AI,BI,cIxx,cIxu,cIuu]=conijcrane1(xh,uh,i,N,n,m,param);

% No inequality constraints
  nc=0;
  [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
```

The linear crane dynamics script file *esjcrane1.m*:

```
function xdot=esjcrane1(t,x,u);

% Linear rotary jib crane dynamics

[nrows, ncols] = size(x);

xdot = zeros(nrows,1);
%
d=0.251;    % Pendulum bob length
g=9.81;     % gravity coefficient
L=0.659;    % Boom length
%
temp1=L/d;
temp2=g/d;
xdot(1) = x(2);
xdot(2) = -temp1*x(6)^2-2*x(6)*x(4)-(temp2-x(6)^2)*x(1)-u(1)*x(3);
xdot(3) = x(4);
xdot(4) = 2*x(6)*x(2)+u(1)*x(1)-(temp2-x(6)^2)*x(3)-temp1*u(1);
xdot(5) = x(6);
xdot(6) = u(1);
```

B.2 Case Study 2 Driver Functions and Script Files

This appendix includes driver functions and MATLAB script files used to solve the DPIP problem in Case Study 2.

B.2.1 Slewing Flexible Link Numerical Simulations Code

The simulation driver script file *fbeamsim.m*:

```
% This file simulates rotating cantilever beam using ode23
%
t0=0.0;
tfinal=1.5;
y0 = [ 0.0 0.0 0.0 0.0 0.0 0.0]';
%
tol=1.e-4;
%
global MINV STIFF STIFFC K_P K_D B_vf C_df C_beam
```

```
global THETA_REF THETADT_REF BVEC
%
% reference input
%
THETA_REF = pi/2;
THETADT_REF = 0.0;
%
% Beam parameters
%
DENSITY=3.8447e-3;        % oz-sec^2/in^4
AREA = 96e-3;             % in^2
RHO = DENSITY*AREA;       % mass/unit length   oz-sec^2/in^2
L = 18.0;                 % effective length of beam  in
R = 2.25;                 % hub radius   in
MT = 9.5427e-3;           % tip mass   oz-sec^2/in
EI = 0.86*(1.31072e+3);         % oz-in^2
C_beam(1) = 0.01;         % dampening in beam mode 1
C_beam(2) = 0.01;         % dampening in beam mode 2
%
% Motor parameters
%
J_inertia = 0.06298;      % oz-in-sec^2
B_vf = (1.0)*0.51941;     % oz-in-sec
C_df = 10.5;              % oz-in
K_t = 16.7435;            % oz-in/amp
K_a = 2.5;                % amp/volt
K_f = 636.6198;           % counts/rad
%
%  Control system gains
%
omega = 45.0;
zeta = 0.5;
K_P = J_inertia*omega^2;
K_D = 2*zeta*omega*J_inertia - B_vf;
%
BVEC=[1;0;0];             % input transmission vector
%
% mass matrix elements
%
m00=RHO*L*(L^2/3+R*L+R^2)+MT*(R+L)^2;
m00=m00+J_inertia;
m01=RHO*L^4*((11/20)*L+(3/4)*R)+2*MT*L^3*(R+L);
m02=RHO*L^4*(L/4+R/3)+MT*L^3*(R+L);
m11=(33/35)*RHO*L^7+4*MT*L^6;
m12=(13/30)*RHO*L^7+2*MT*L^6;
m22=(1/5)*RHO*L^7+MT*L^6;
%
% Mass matrix
%
mass=[m00 m01 m02;m01 m11 m12; m02 m12 m22];
```

```
%
% mass inverse
%
MINV=inv(mass);
%
% stiffness matrix elements
%
k11=12*EI*L^3;
k12=6*EI*L^3;
k22=4*EI*L^3;
%
% Stiffness matrix
%
stiff=[(K_P) 0 0;0 k11 k12; 0 k12 k22];
STIFF=[0 0 0;0 k11 k12; 0 k12 k22];
%
% centripetal stiffening matrix elements
%
kc11=((81/70)-(33/35))*RHO*L^7+1.5*RHO*R*L^6+ ...
((24/5)-4)*MT*L^6+(24/5)*MT*R*L^5;
kc12=((11/20)-(13/20))*RHO*L^7+(7/10)*RHO*R*L^6+((5/2)-2)*MT*L^6+ ...
(5/2)*MT*R*L^5;
kc22=((11/20)-(1/5))*RHO*L^7+(1/3)*RHO*R*L^6+((4/3)-1)*MT*L^6+ ...
(4/3)*MT*R*L^5;
%
% Centripetal Stiffness matrix
%
STIFFC = [0 0 0; 0 kc11 kc12; 0 kc12 kc22];
stiffc=(3.799)^2*STIFFC;
%
% Perform numerical simulation
%
[t,y]=ode23('fbeam1',t0,tfinal,y0,tol);
%[t,y]=ode45('fbeam1',t0,tfinal,y0,tol);
%
% recreate torque profiles
%
xx=L;
%
% determine tip deflection from assumed modes equation
%
phi(1) = xx^2*(3*L-xx);
phi(2) = xx^2*L;
%
psi_l=-0.5*L^5*((9/5)-9+12);
%
[m,n]=size(y);
for i = 1:m,
yxt(i)=y(i,2)*phi(1)+y(i,3)*phi(2);
uxt(i) = y(i,2)^2*psi_l;
```

```
u_all(i) = R+L+uxt(i);
tau(i) = K_P*(THETA_REF -y(i,1))-K_D*y(i,4);
if tau(i) <= -126.0
tau(i) = -126.0;
elseif tau(i) >= 126.0
tau(i) = 126.0;
else
end
damp1(i)=B_vf*y(i,4);
damp2(i)=C_df*sign(y(i,4));
vectemp=[y(i,1); y(i,2); y(i,3)];
stiff22(i)=STIFF(2,:)*vectemp;
stiff33(i)=STIFF(3,:)*vectemp;
%stiffc11(i)=(y(i,4))^2*STIFFC(1,:)*vectemp;
stiffc22(i)=(y(i,4))^2*STIFFC(2,:)*vectemp;
stiffc33(i)=(y(i,4))^2*STIFFC(3,:)*vectemp;
end
```

The equations of motion script file *fbeam*1.*m*:

```
function yprime = fbeam1(t,y);
%
%  dynamic  equations of motion for torque driven rotating
%  cantilever beam with two modes which is set up as a
%  script file in fbeamsim.m
%
global MINV STIFF STIFFC K_P K_D B_vf C_df C_beam
global THETA_REF THETADT_REF BVEC
%
% set up state variables
%
x(1,1)=y(1);
x(2,1)=y(2);
x(3,1)=y(3);
xdt(1,1)=y(4);
xdt(2,1)=y(5);
xdt(3,1)=y(6);
%
% system error
%
e=THETA_REF - x(1);
edt=THETADT_REF - xdt(1);
%
% control law
%
tau = K_P*e+K_D*edt;
%
if tau <= -126.0
tau = -126.0;
elseif tau >= 126.0
```

```
tau = 126.0;
else
end
%
u=BVEC*tau;
%
% dampening matrix
%
damp1=B_vf;
damp2=BVEC*C_df*sign(xdt(1));

%
cdamp=[damp1 0 0; 0 C_beam(1) 0; 0 0 C_beam(2)];
%
stifftot=STIFF+(xdt(1)^2)*STIFFC;
%stifftot=STIFF;
%
% Solve for accelerations
%
xddt = MINV*(u - cdamp*xdt - damp2 - stifftot*x);
%
% send y prime back to main simulation file fbeamsim.m
%
yprime = [ xdt(1); xdt(2); xdt(3); xddt(1); xddt(2); xddt(3)];
```

B.2.2 Dynamic Programming/Interior-Point Driver, Cost, Equality, Inequality, and Dynamics Code

The DPIP driver code script file *exflink4a.m*:

```
global MINV STIFF STIFFC K_P K_D B_vf C_df C_beam
%THETA_REF THETADT_REF BVEC beta
%
% reference input
%
THETA_REF = pi/2;
THETADT_REF = 0.0;
%
% Beam parameters
%
DENSITY=3.8447e-3/16.0;        % lb-sec^2/in^4
AREA = 96e-3;                  % in^2
RHO = DENSITY*AREA;            % mass/unit length lb-sec^2/in^2
L = 18.0;                      % effective length of beam  in
R = 2.25;                      % hub radius   in
MT = 9.5427e-3/16.0;           % tip mass   lb-sec^2/in
EI = 0.86*(1.31072e+3)/16.0;   % lb-in^2
C_beam(1) = 0.0001;            % dampening in beam mode 1
C_beam(2) = 0.0001;            % dampening in beam mode 2
%
```

```
% Motor parameters
%
J_inertia = 0.06298/16.0;      % lb-in-sec^2
B_vf = (1.0)*0.51941/16.0;     % lb-in-sec
C_df = 0.0*10.5/16.0;          % lb-in
beta = 7.0;                    % note:sign(thetadt) = tanh(beta*thetadt)
%                                -reduces sharpness
K_t = 16.7435;                 % oz-in/amp
K_a = 2.5;                     % amp/volt
K_f = 636.6198;                % counts/rad
%
%  Control system gains
%
%
omega = 45.0;
zeta = 0.5;
K_P = J_inertia*omega^2;                    % lb-in
K_D = 2*zeta*omega*J_inertia - B_vf;        % lb-in-sec
%
BVEC=[1;0;0];              % input transmission vector
%
% mass matrix elements
%
m00=RHO*L*(L^2/3+R*L+R^2)+MT*(R+L)^2;
m00=m00+J_inertia;
m01=RHO*L^4*((11/20)*L+(3/4)*R)+2*MT*L^3*(R+L);
m02=RHO*L^4*(L/4+R/3)+MT*L^3*(R+L);
m11=(33/35)*RHO*L^7+4*MT*L^6;
m12=(13/30)*RHO*L^7+2*MT*L^6;
m22=(1/5)*RHO*L^7+MT*L^6;
%
% Mass matrix
%
mass=[m00 m01 m02;m01 m11 m12; m02 m12 m22];
%
% mass inverse
%
MINV=inv(mass);
%
% stiffness matrix elements
%
k11=12*EI*L^3;
k12=6*EI*L^3;
k22=4*EI*L^3;
%
% Stiffness matrix
%
%stiff=[(K_P) 0 0;0 k11 k12; 0 k12 k22];
STIFF=[0 0 0;0 k11 k12; 0 k12 k22];
%
```

```
% centripetal stiffening matrix elements
%
kc11=((81/70)-(33/35))*RHO*L^7+1.5*RHO*R*L^6+((24/5)-4)*MT*L^6+ ...
(24/5)*MT*R*L^5;
kc12=((11/20)-(13/20))*RHO*L^7+(7/10)*RHO*R*L^6+((5/2)-2)*MT*L^6+ ...
(5/2)*MT*R*L^5;
kc22=((11/20)-(1/5))*RHO*L^7+(1/3)*RHO*R*L^6+((4/3)-1)*MT*L^6+ ...
(4/3)*MT*R*L^5;
%
% Centripetal Stiffness matrix
%
STIFFC = [0 0 0; 0 kc11 kc12; 0 kc12 kc22];
%
% number of states
n=6;
% number of controls
m=1;
% number of steps
N=201;
%N=2001;  % control servo sample at 1000 Hz
% problem parameters
% initial and final conditions
wi = [0 0 0 0 0 0];
wf = [pi/2 0 0 0 0 0];
ti=0.0;
tf=1.0;

x1h = wi'; % initial conditions of states
xNh=wf';
    dt=(tf-ti)/(N-1);
    t=ti:dt:tf;
    t=t';
uh=0.2*ones(N,1);

lamEh=ones(12,1)*0; % six initial/final specifications of states

lamIh=[]       % minimum effort problem
%lamIh=zeros(2*(N-1),1);   % bounded mininum effort problem

controlbound=1;
param=[ti tf wi wf controlbound];

% view trajectory from initial guess
    [t,xh]=drungeu('esflink4a',[param(1) param(2)],N-1,x1h,uh);
    figure(1),
    subplot(121), plot(t,xh,'.-'); title('x'); grid; shg
    subplot(122), plot(t,uh,'.-'); grid; title('u'); shg
OIG=input('Optimize initial guess? (y=1/n=0)   ');
if (OIG==1),
```

```
iterateQ=1;
counter=1;
alpha=1;
maxit=8;
while iterateQ==1,
%while iterateQ<=maxit,
    %  dpeq returns changes to nominal values
    %  ub : change in controls
    %  x1b : change in initial state
    %  lamEb : change in Lagrange multipliers for equality constraints

    [ub,x1b,lamEb,lamIb]=dpip(uh,x1h,lamEh,lamIh,N,n,m,'esflink4a',...
                          'coflink4a','eqflink4a','inflink4a',param);
    uh = uh + alpha*ub;

    uh(N,:) = uh(N-1,:);
    [t,xh]=drungeu('esflink4a',[param(1) param(2)],N-1,x1h,uh);

        fcerror=eqflink4a(xh,uh,N,N,n,m,param);
        fcnorm=norm(fcerror);
        fprintf(1,'fcnorm = %g\n',fcnorm)
        fprintf(1,'\n');
        figure(1),
        subplot(121), plot(t,xh,'.-'); title('x'); grid; shg
        subplot(122), plot(t,uh,'.-'); grid; title('u'); shg

    iterateQ=input('Continue to iterate? (y=1/n=0)   ');
%    iterateQ=iterateQ+1;
    counter=counter+1;
end
end
shg
```

The cost function script file *coflink4a.m*:

```
function [y,z,Q,R,S]=coflink4a(xh,uh,i,N,n,m,param);
%
% Defines the cost function and some partial derivatives
% f is cost function
% yy is state
% uu is control
% xx is independent variable
%
% y is dfdy
% z is dfdu
% Q is dfdyy (second derivative)
% R is dfdydu
% S is dfduu

t1=param(1);
```

```
t2=param(2);
dt=(t2-t1)/(N-1);

tt=t1+dt*(i-1);  % current x (time, or independent variable); x_i
xx=xh(i,:);  % current y (state); y_i
uu=uh(i,:);  % current u (control); u_i

dfdx = zeros(n,1);
dfdxx = zeros(n,n);
dfdu = dt*[uu(1)]';
dfduu = dt;
y=dfdx;
z=dfdu;
Q=dfdxx;
R=zeros(n,m); % dfdydu matrix of partials that couples states/controls
S=dfduu;
if (i==1)+(i==N) % the plus sign is an "or"
  y=y/2; % the first and second partials of the cost wrt the state are
%          different if at first or last node
  Q=Q/2;
end
```

The equality constraint script file *eqflink4a.m*:

```
function [cE,AE,BE,cExx,cExu,cEuu]=coneflink4a(xh,uh,i,N,n,m,param);
%  cone   name of function to calculate c_{iE}, A_{iE}, B_{iE},
%           c_{iE}_{xx}, c_{iE}_{xu} and c_{iE}_{uu}

% param=[ti tf xini' xfin'];

nc=0;
[cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);

if i==1
  nc=6;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  ic=param(3:8);
  cE=xh(i,:)'-ic';
  AE=eye(6);
end

if i==N
  nc=6;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  fc=param(9:14);
  cE=xh(i,:)'-fc';
  AE=eye(6);
end
```

The inequality constraint script file *inflink4a.m*:

```
function [cI,AI,BI,cIxx,cIxu,cIuu]=coniflink4a(xh,uh,i,N,n,m,param);

  nc=0;
  [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);

% These inequalities are used for bounded version:
%  if (i==N)
%  nc=0;
%  [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
%else
%    nc=2;
%    [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);

%b1=2.5;

%cI=[uh(i,1)-b1;
%    -uh(i,1)-b1];
%BI=[1; -1];
%end
```

The dynamic equations of motion script file *esflink4a.m*:

```
function xdot=esflink4a(t,x,u);
[nrows, ncols] = size(x);

%  function yprime = fbeam1(t,y);
%
%  dynamic equations of motion for torque driven rotating
%  cantilever beam  with two modes which is set up as a
%  script file in fbeamsim.m
%
global MINV STIFF STIFFC K_P K_D B_vf C_df C_beam
global THETA_REF THETADT_REF BVEC beta
%
% set up state variables
%
xx(1,1)=x(1);
xx(2,1)=x(2);
xx(3,1)=x(3);
xxdt(1,1)=x(4);
xxdt(2,1)=x(5);
xxdt(3,1)=x(6);
%
%
tau=u(1);
%tau=x(7);
uu=BVEC*tau;
%
```

```
% dampening matrix
%
damp1=B_vf;
%damp2=BVEC*C_df*sign(xxdt(1));
damp2=BVEC*C_df*tanh(beta*xxdt(1));
%
cdamp=[damp1 0 0; 0 C_beam(1) 0; 0 0 C_beam(2)];
%
stifftot=STIFF+(xxdt(1)^2)*STIFFC;
%
% Solve for accelerations
%
xxddt = MINV*(uu - cdamp*xxdt - damp2 - stifftot*xx);
%
% send y prime back to main simulation file fbeamsim.m
%
%yprime = [ xdt(1); xdt(2); xdt(3); xddt(1); xddt(2); xddt(3)];

xdot = zeros(nrows,1);
%
xdot(1)=xxdt(1);
xdot(2)=xxdt(2);
xdot(3)=xxdt(3);
xdot(4)=xxddt(1);
xdot(5)=xxddt(2);
xdot(6)=xxddt(3);
%xdot(7)=u(1);
%xdot(7)=x(8);
%xdot(8)=u(1);
```

B.2.3 *fmincon* Driver, Cost, Constraints, and Dynamics Code

The *fmincon* driver code script file *oneflexlinkdriver.m*:

```
% One flexible link driver routine oneflexLinkDriver.m

global MINV STIFF STIFFC CDAMP BVEC QMFLAG
global C_df beta
global tarray;
global xo;
global xdesired;
global tfinal;

% SET all physical parameters for flexible link dynamic model
%
QMFLAG = 1;     % quadratic modes flag =1 on/ =0 off
% Beam parameters
DENSITY=3.8447e-3/16.0;          % lb-sec^2/in^4
AREA = 96e-3;                    % in^2
RHO = DENSITY*AREA;              % mass/unit length lb-sec^2/in^4
```

```
L = 18.0;                          % effective length of beam  in
R = 2.25;                          % hub radius   in
MT = (9.5427e-3/16.0);             % tip mass   lb-sec^2/in
EI = 0.86*(1.31072e+3)/16.0;       % lb-in^2
C_beam(1) = 0.0001;                % dampening in beam mode 1
C_beam(2) = 0.0001;                % dampening in beam mode 2
%
% Motor parameters
%
J_inertia = 0.06298/16.0;          % lb-in-sec^2
B_vf = 0.51941/16.0;               % lb-in-sec
C_df = 0.0*10.5/16.0;              % lb-in
beta = 2.0;
%
damp1=B_vf;
%
BVEC=[1;0;0];                      % input transmission vector
%
% mass matrix elements
%
m00=RHO*L*(L^2/3+R*L+R^2)+MT*(R+L)^2;
m00=m00+J_inertia;
m01=RHO*L^4*((11/20)*L+(3/4)*R)+2*MT*L^3*(R+L);
m02=RHO*L^4*(L/4+R/3)+MT*L^3*(R+L);
m11=(33/35)*RHO*L^7+4*MT*L^6;
m12=(13/30)*RHO*L^7+2*MT*L^6;
m22=(1/5)*RHO*L^7+MT*L^6;
%
% Mass matrix
%
mass=[m00 m01 m02;m01 m11 m12; m02 m12 m22];
%
% mass inverse
%
MINV=inv(mass);
%
% stiffness matrix elements
%
k11=12*EI*L^3;
k12=6*EI*L^3;
k22=4*EI*L^3;
%
% Stiffness matrix
%
STIFF=[0 0 0;0 k11 k12; 0 k12 k22];
%
% centripetal stiffening matrix elements
%
kc11=((81/70)-(33/35))*RHO*L^7+1.5*RHO*R*L^6+((24/5)-4)*MT*L^6 ...
+(24/5)*MT*R*L^5;
```

```
kc12=((11/20)-(13/20))*RHO*L^7+(7/10)*RHO*R*L^6+((5/2)-2)*MT*L^6 ...
+(5/2)*MT*R*L^5;
kc22=((11/20)-(1/5))*RHO*L^7+(1/3)*RHO*R*L^6+((4/3)-1)*MT*L^6+ ...
(4/3)*MT*R*L^5;
%
% Centripetal Stiffness matrix
%
STIFFC = [0 0 0; 0 kc11 kc12; 0 kc12 kc22];
%
% Damping matrix
%
CDAMP=[damp1 0 0; 0 C_beam(1) 0; 0 0 C_beam(2)];
%
tfinal=1.0;
deltat=0.05;  % discretized number of control parameters 20 pts -bounded
%deltat=0.04; % discretized number of control parameters 40 pts
tarray=[0:deltat:tfinal];
xo=[0 0 0 0 0 0 0]; % theta, q1, q2, thetdot, q1dot, q2dot, u^2
xdesired=[pi/2 0 0 0 0 0];  % rotate link through 90 degress w/ q's
%                             and qdot's zero at end
pvalue=.2;
params0=pvalue*ones(1,length(tarray));
A=[];b=[];Aeq=[];beq=[];
%lowerBndValue=-10.  % use for unbounded control problem
%upperBndValue=10.   % use for unbounded control problem
lowerBndValue=-2.5   % use for bounded control problem
upperBndValue=2.5    %  use for bounded control problem
%
lowerBnds=lowerBndValue*ones(1,length(tarray));
upperBnds=upperBndValue*ones(1,length(tarray));
%
params=fmincon('perfIndex_f1',params0,A,b,Aeq,beq,lowerBnds,...
upperBnds,'constraints_f1')
```

The cost function script file *perfindex_f1.m*:

```
% fun function handle:
%Performance Index matlab M-file - perfIndex_f1.m

function f = perfIndex_f1(params)
global xo;
global p;
global tfinal;
p=params;
[t,x]=ode45('xprime_f1',[0:tfinal],xo(:));
[rows,cols]=size(x);
%f=x(rows,7)+x(rows,4)*x(rows,4);
f=x(rows,7);
return;
```

The constraints script file *constraints_f1.m*:

```
% nonlcon function handle:
%constraint matlab M-file - constraints_f1.m
function [c,coneq,GC,GCeq]=constraints_f1(params)
global xo;
global xdesired;
global p;
global tfinal;
p=params;
[t,x]=ode45('xprime_f1',[0:tfinal],xo(:));
[rows,cols]=size(x);
coneq(1)=x(rows,1)-xdesired(1);      % at end of slew 90 degrees
coneq(2)=x(rows,2)-0.0;              % zero
coneq(3)=x(rows,3)-0.0;              % zero
coneq(4)=x(rows,4)-0.0;              % zero
coneq(5)=x(rows,5)-0.0;              % zero
coneq(6)=x(rows,6)-0.0;              % zero
c=[];GC=[];GCeq=[];coneq
return;
```

The dynamic equations of motion script file *xprime_f1.m*:

```
% first order odes for flexible horizontal link:
%matlab file xprime_f1.m
function xdot=xprime_f1(t,x)

global MINV STIFF STIFFC CDAMP BVEC QMFLAG
global C_df beta
global p;
global tarray;

% Determined interpolated control effort (hub torque)
u=interp1(tarray,p,t);
%
xx(1,1)=x(1);
xx(2,1)=x(2);
xx(3,1)=x(3);
xxdt(1,1)=x(4);
xxdt(2,1)=x(5);
xxdt(3,1)=x(6);
%
% nonlinear damping term
%damp2=BVEC*C_df*sign(xxdt(1));
damp2=BVEC*C_df*tanh(beta*xxdt(1));
% define total stiffness matrix that
%includes centrifugal stiffening matrix
stifftot=STIFF+(xxdt(1)^2)*STIFFC*QMFLAG;
% Solve for accelerations
xxddt = MINV*(BVEC*u - CDAMP*xxdt -damp2 - stifftot*xx);
% Define state equations
```

```
xdot(1) = x(4);
xdot(2) = x(5);
xdot(3) = x(6);
xdot(4) = xxddt(1);
xdot(5) = xxddt(2);
xdot(6) = xxddt(3);
% squared control effort
xdot(7)=u*u;
%
% send y prime back
xdot=xdot(:);
return;
```

The plot script file *postplt_f1.m*:

```
% This file generates
%plot for horizontal flexible link - postplt_f1.m
[t0,x0]=ode45('xprime_f1',[0:tfinal],xo(:));
tar0=tarray;
par0=params;
save hslew0 t0 x0 tar0 par0
plot(t0,x0(:,1),'-',t0,x0(:,4),'--',tar0,par0,'-.'),grid
legend('-','Angle','--','Angle Rate','-.','Torque')
xlabel('Time (sec)')
ylabel('Minimum-Effort Trajectories')
title('Horizontal Flexible Link Slew')
```

B.3 Case Study 3 Driver Functions and Script Files

This appendix includes driver functions and MATLAB script files used to solve the DPIP problem in Case Study 3.

B.3.1 Dynamic Programming/Interior-Point Driver, Cost, Equality, Inequality, and Dynamics Code

The DPIP driver code script file *exarmf1.m*:

```
% 2link flexible planar arm - minimum effort DPIP driver code
% number of states
n=8;
% number of controls
m=2;
% number of steps
N=201;
% problem parameters
% initial and final conditions
%wi = [0 0 0 0 0 0 0 0];
%wf = [pi/2 pi/2 -pi/2 -pi/2 0 0 0 0];
wi = [0 0 pi pi 0 0 0 0];
```

```
wf = [pi pi 0 0 0 0 0 0];
ti=0;
tf=2.0;

x1h = wi'; % initial conditions of states
xNh=wf';

    dt=(tf-ti)/(N-1);
    t=ti:dt:tf;
    t=t';

uh=[zeros(N,1) zeros(N,1)];  % used for mininum effort start-up

lamEh=ones(16,1)*0; % eight initial/final specifications of states

lamIh=[]   % set for minimum effort only

controlbound=1;
param=[ti tf wi wf controlbound];

% view trajectory from initial guess
    [t,xh]=drungeu('esarmf1',[param(1) param(2)],N-1,x1h,uh);
    figure(1),
    subplot(121), plot(t,xh,'.-'); title('x'); grid; shg
    subplot(122), plot(t,uh,'.-'); grid; title('u'); shg
OIG=input('Optimize initial guess? (y=1/n=0)   ');
if (OIG==1),

iterateQ=1;
counter=1;
alpha=1;
maxit=8;
while iterateQ==1,
%while iterateQ<=maxit,
    %  dpeq returns changes to nominal values
    %  ub : change in controls
    %  x1b : change in initial state
    %  lamEb : change in Lagrange multipliers for equality constraints

    [ub,x1b,lamEb,lamIb]=dpip(uh,x1h,lamEh,lamIh,N,n,m,'esarmf1',...
                          'coarmf1','eqarmf1','inarmf1',param);

    uh = uh + alpha*ub;

    uh(N,:) = uh(N-1,:);
    [t,xh]=drungeu('esarmf1',[param(1) param(2)],N-1,x1h,uh);

        fcerror=eqarmf1(xh,uh,N,N,n,m,param);
        fcnorm=norm(fcerror);
        fprintf(1,'fcnorm = %g\n',fcnorm)
```

```
        fprintf(1,'\n');
        figure(1),
        subplot(121), plot(t,xh,'.-'); title('x'); grid; shg
        subplot(122), plot(t,uh,'.-'); grid; title('u'); shg

    iterateQ=input('Continue to iterate? (y=1/n=0)    ');
%    iterateQ=iterateQ+1;
    counter=counter+1;
end
end
shg
```

The cost function script file *coarmf1.m*:

```
function [y,z,Q,R,S]=coarmf1(xh,uh,i,N,n,m,param);
%
% Defines the cost function and some partial derivatives
% f is cost function
% yy is state
% uu is control
% xx is independent variable
%
% y is dfdy
% z is dfdu
% Q is dfdyy (second derivative)
% R is dfdydu
% S is dfduu

t1=param(1);
t2=param(2);
dt=(t2-t1)/(N-1);

tt=t1+dt*(i-1);  % current x (time, or independent variable); x_i
xx=xh(i,:);  % current y (state); y_i
uu=uh(i,:);  % current u (control); u_i

dfdx = zeros(n,1);
dfdxx = zeros(n,n);
dfdu = dt*[uu(1) uu(2)]';
dfduu = dt*eye(2);
y=dfdx;
z=dfdu;
Q=dfdxx;
R=zeros(n,m); % dfdydu matrix of partials that couples states/controls
S=dfduu;
if (i==1)+(i==N) % the plus sign is an "or"
  y=y/2; % the first and second partials of the cost wrt the state are
%          different if at first or last node
  Q=Q/2;
end
```

The equality constraint script file *eqarmf*1.*m*:

```
function [cE,AE,BE,cExx,cExu,cEuu]=conearmf1(xh,uh,i,N,n,m,param);
%  cone   name of function to calculate c_{iE}, A_{iE}, B_{iE},
%           c_{iE}_{xx}, c_{iE}_{xu} and c_{iE}_{uu}
nc=0;
[cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);

if i==1
  nc=8;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  ic=param(3:10);
  cE=xh(i,:)'-ic';
  AE=eye(8);
end

if i==N
  nc=8;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  fc=param(11:18);
  cE=xh(i,:)'-fc';
  AE=eye(8);
end
```

The inequality constraint script file *inarmf*1.*m*:

```
function [cI,AI,BI,cIxx,cIxu,cIuu]=coniarmf1(xh,uh,i,N,n,m,param);

  nc=0;
  [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
```

The equations of motion for the two-link planar flexible arm script file *esarmf*1.*m*:

```
function xdot=esarmf1(t,x,u);

[nrows, ncols] = size(x);

% linear dynamic  equations of motion for 2 flexible link planar
% robot arm which is set up as a script file in 2lnksim.m
% - from original standalone simulation
%
% Flexible links; mass and inertia parameters
% (generic test arm - used for initial testing)
%  Uses only masses and inertias from generic test arm
j1 = 1.0;
j2 = 1.0;
m2 = 5.415;  % kg
j3 = 1.0;
m3 = 0.830;  % kg
j4 = 0.0;
m4 = 0.01;    % mass payload
```

```
%
% Other arm parameters based on Sandia and hybrid 2-link arms -
%
c1 = 0.05;      % N.m/(rad/sec)  viscous friction joint 1
c2 = 0.05;      % N.m/(rad/sec)  viscous friction joint 2
%
% Flexible links; mass and inertia parameters
%(representative Sandia 2-link)
%
l1 = 0.5969;                % m link 1 length only sandia
l2 = 0.4842;                % m link 2 length only sandia
th1 = 0.005537;             % m thickness link 1 sandia
th2 = 0.001574;             % m thickness link 2 sandia
width1 = 0.1524;            % m width link 1 sandia
width2 = 0.0762;            % m width link 2 sandia
a1 = th1*width1;            % m^2 link 1 cross-sectional area
a2 = th2*width2;            % m^2 link 2 cross-sectional area
rho = 2700.0;               % Aluminum density  kg/m^3
rho1 = rho*a1;
rho2 = rho*a2;
e=6.9e10;                   % N/m^2     modulus of elasticity
inertia1 = width1*(th1)^3/12;
inertia2 = width2*(th2)^3/12;
ei1=e*inertia1;             % N.m^2 bending stiffness
ei2=e*inertia2;             % N.m^2 bending stiffness
%
% set up state variables
%
xx(1,1) = x(1);
xx(2,1) = x(2);
xx(3,1) = x(3);
xx(4,1) = x(4);
xxdt(1,1) = x(5);
xxdt(2,1) = x(6);
xxdt(3,1) = x(7);
xxdt(4,1) = x(8);
%
% Define z parameters
%
z(1) = rho1*l1/12;
z(2) = rho2*l2/4;
z(3) = (m2+m3+m4)/4;
z(4) = rho2*l2/8;
z(5) = l1*l2;
z(6) = l1^2;
z(7) = l2^2;
z(8) = m4/4;
z(9) = z(6)*(z(1)+z(2)+z(3));
z(10) = z(5)*(z(4)+z(8));
z(11) = rho2*l2/12;
```

```
z(12) = z(7)*(z(11)+z(8));
z(13) = ei1/l1;
z(14) = ei2/l2;
%
m11hat = j1+z(9);
m12hat = z(9);
m13hat = z(10);
m14hat = z(10);
m22hat = j2+z(9);
m23hat = z(10);
m24hat = z(10);
m33hat = j3+z(12);
m34hat = z(12);
m44hat = j4+z(12);
%
% Mass matrix (4 x 4)
%
mass(1,1) = m11hat;
mass(1,2) = m12hat*cos(xx(1,1)-xx(2,1));
mass(1,3) = m13hat*cos(xx(1,1)-xx(3,1));
mass(1,4) = m14hat*cos(xx(1,1)-xx(4,1));
mass(2,1) = mass(1,2);
mass(2,2) = m22hat;
mass(2,3) = m23hat*cos(xx(2,1)-xx(3,1));
mass(2,4) = m24hat*cos(xx(2,1)-xx(4,1));
mass(3,1) = mass(1,3);
mass(3,2) = mass(2,3);
mass(3,3) = m33hat;
mass(3,4) = m34hat*cos(xx(3,1)-xx(4,1));
mass(4,1) = mass(1,4);
mass(4,2) = mass(2,4);
mass(4,3) = mass(3,4);
mass(4,4) = m44hat;
%
% Centripetal coefficients
%
c121 = m12hat*sin(xx(1,1) - xx(2,1));
c131 = m13hat*sin(xx(1,1) - xx(3,1));
c141 = m14hat*sin(xx(1,1) - xx(4,1));
c232 = m23hat*sin(xx(2,1) - xx(3,1));
c242 = m24hat*sin(xx(2,1) - xx(4,1));
c343 = m34hat*sin(xx(3,1) - xx(4,1));
%
cent(1,1) = c121*xxdt(2,1)^2 + c131*xxdt(3,1)^2+c141*xxdt(4,1)^2;
cent(2,1) = -c121*xxdt(1,1)^2 + c232*xxdt(3,1)^2+c242*xxdt(4,1)^2;
cent(3,1) = -c131*xxdt(1,1)^2 - c232*xxdt(2,1)^2+c343*xxdt(4,1)^2;
cent(4,1) = -c141*xxdt(1,1)^2 - c242*xxdt(2,1)^2-c343*xxdt(3,1)^2;
%
% Damping Matrix (4 x 4)
%
```

```
damp=[c1 0 0 0; 0 c2 -c2 0; 0 -c2 c2 0; 0 0 0 0];
%
% Stiffening Matirx (4 x 4)
%
stiff=[z(13) -z(13) 0 0; -z(13) z(13) 0 0; 0 0 z(14) -z(14); ...
0 0 -z(14) z(14)];
%
%qgen = [tau1; -tau2; tau2; 0];
qgen = [u(1); -u(2); u(2); 0];
%
% Solve for accelerations
%
xddt = inv(mass)*(qgen -cent -damp*xxdt - stiff*xx);
%
% send y prime back to main simulation file armsim.m
% - from original standalone simulation
%
%yprime = [xdt(1); xdt(2); xdt(3); xdt(4); xddt(1); xddt(2); ...
% xddt(3); xddt(4); gamdt; gamddt];
%
xdot(1) = x(5);
xdot(2) = x(6);
xdot(3) = x(7);
xdot(4) = x(8);
xdot(5) = xddt(1);
xdot(6) = xddt(2);
xdot(7) = xddt(3);
xdot(8) = xddt(4);
```

B.4 Case Study 4 Driver Functions and Script Files

This appendix includes driver functions and MATLAB script files used to solve the DPIP problem in Case Study 4.

B.4.1 Dynamic Programming/Interior-Point Driver, Cost, Equality, Inequality, and Dynamics Code

The DPIP driver code script file *exmt.m*:

```
% number of states
n=7+1;
% number of controls
m=3;
% number of steps
N=30;

ti=0;
tf=1;
```

```
% problem parameters
% initial and final conditions
wi = [0 0 0 1 0 0 0 sqrt(5)];
phi=pi;
wf = [0 0 sin(phi/2) cos(phi/2) 0 0 0];

x1h = wi'; % initial conditions of states
xNh=wf';

uh1 = zeros(N,1); uh1(1:floor(.2*N))=.8;
uh2 = zeros(N,1); uh2(1:floor(.2*N))=.8;
uh3 = zeros(N,1); uh3(1:floor(.2*N))=-.8;

uh=[uh1 uh2 uh3];
lamEh=zeros(7+6,1); % three initial/final specifications of states
lamIh=zeros(6*(N-1),1);
% 2 inequality constraints at every point in trajectory for the
% (single) control except last point

param=[ti tf wi(1:7) wf];

% view trajectory from initial guess
  [t,xh]=drungeu('esmt',[ti tf],N-1,x1h,uh);
  fcerror=eqmt(xh,uh,N,N,n,m,param);
  fcnorm=norm(fcerror);
  fprintf(1,'fcnorm = %g\n',fcnorm)
  fprintf(1,'Min Time = %g\n',xh(N,n)^2)
  figure(1),
  t = t*xh(N,n)^2;
  subplot(221), plot(t,xh(:,1:4)); title('Euler Parameters'); grid;
  tname = 'Min Time = jj';
  Tname=strrep(tname,'jj',num2str(xh(N,n)^2)); title(Tname);
  subplot(222), plot(t,xh(:,5:7)); grid;
  title('Angular Velocities');
  subplot(223), plot(t,uh(:,1:3)); grid; title('Controls');

OIG=input('Optimize initial guess? (y=1/n=0)   ');
if (OIG==1),

  iterateQ=1;
  counter=1;
  alpha=1; % alpha=.75;
  maxit=5;
  while iterateQ==1,
%  while iterateQ<=maxit,
    % dpip returns changes to nominal values
    % ub : change in controls
    % x1b : change in initial state
    % lamEb : change in Lagrange multipliers for equality constraints
    % lamIb : change in Lagrange multipliers for inequality constraints
```

```
    [ub,x1b,lamEb,lamIb]=dpip(uh,x1h,lamEh,lamIh,N,n,m,'esmt',...
                          'comt','eqmt','inmt',param);
    uh = uh + alpha*real(ub);
    x1h = x1h + alpha*real(x1b);
    uh(N,:) = uh(N-1,:);
    [t,xh]=drungeu('esmt',[ti tf],N-1,x1h,uh);
    fcerror=eqmt(xh,uh,N,N,n,m,param);
    fcnorm=norm(fcerror);
    fprintf(1,'fcnorm = %g\n',fcnorm)
    mintime=xh(N,n)^2;
    fprintf(1,'Min Time = %g\n',mintime);
    fprintf(1,'Change in Controls = %g\n',norm(real(ub),'fro'))
    fprintf(1,'End Iteration = %g;\n',counter);
    fprintf(1,'\n');
    figure(1),
    t = t*xh(N,n)^2;
    subplot(221), plot(t,xh(:,1:4)); title('Euler Parameters'); grid;
    tname = 'Min Time = jj';
    Tname=strrep(tname,'jj',num2str(xh(N,n)^2));
    title(Tname);
    subplot(222), plot(t,xh(:,5:7)); grid;
    title('Angular Velocities');
    subplot(223), plot(t,uh(:,1:3)); grid; title('Controls');
    shg

    iterateQ=input('Continue to iterate? (y=1/n=0)    ');
    if fcnorm<5e-6, iterateQ=0; end
    if iterateQ==1, save results; end
    counter=counter+1;
    pause(1)
  end
end
```

The cost function script file *comt.m*:

```
function [y,z,Q,R,S]=comt(xh,uh,i,N,n,m,param);
%
% Defines the cost function and some partial derivatives
% f is cost function
% yy is state
% uu is control
% xx is independent variable
%
% y is dfdy
% z is dfdu
% Q is dfdyy (second derivative)
% R is dfdydu
% S is dfduu

x=xh(i,:);  % current y (state); y_i
u=uh(i,:);  % current u (control); u_i
```

```
y=zeros(n,1);
z=zeros(m,1);
Q=zeros(n,n);
R=zeros(n,m);
S=zeros(m,m);
if (i==1)
  y(n,1)=1;
end
```

The equality constraint script file *eqmt.m*:

```
function [cE,AE,BE,cExx,cExu,cEuu]=eqmt(xh,uh,i,N,n,m,param);
nc=0;
[cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
if i==1
  nc=7;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  ic=param(3:9);
  cE=xh(i,1:7)'-ic';
  AE=[eye(7) zeros(7,1)];
end
if i==N
  nc=6;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  % state 4 final conditions (orig)
  fc=[param(10:12) param(14:16)];
  cE=[xh(i,1:3)'-fc(1:3)';
      xh(i,5:7)'-fc(4:6)'];
  AE=[eye(3) zeros(3,5);
      zeros(3,4) eye(3) zeros(3,1)];
end
```

The inequality constraint script file *inmt.m*:

```
function [cI,AI,BI,cIxx,cIxu,cIuu]=inmt(xh,uh,i,N,n,m,param);

if (i==N)
%  nc=1;
%  [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
%  cI=[-xh(i,n)+.001];
%  AI=[zeros(1,7) -1];
  nc=0;
  [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
else
        nc=6;
        [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
    cI=[uh(i,1)-1;
      -uh(i,1)-1;
      uh(i,2)-1;
      -uh(i,2)-1;
```

```
      uh(i,3)-1;
      -uh(i,3)-1];
    BI=[1 0 0;
      -1 0 0;
      0 1 0;
      0 -1 0;
      0 0 1;
      0 0 -1];
end
```

The equations of motion for Case Study 4 are contained in script file *esmt.m*:

```
function xdot=esmt(t,x,u);

[nrows, ncols] = size(x);

xdot = zeros(nrows,1);

xdot(1) = .5*(x(5)*x(4)-x(6)*x(3)+x(7)*x(2));
xdot(2) = .5*(x(5)*x(3)+x(6)*x(4)-x(7)*x(1));
xdot(3) = .5*(-x(5)*x(2)+x(6)*x(1)+x(7)*x(4));
xdot(4) = .5*(-x(5)*x(1)-x(6)*x(2)-x(7)*x(3));

i1 = 1;
i2 = 1;
i3 = 1;

xdot(5)=((i2-i3)*x(6)*x(7) + u(1))/i1;
xdot(6)=((i3-i1)*x(5)*x(7) + u(2))/i2;
xdot(7)=((i1-i2)*x(5)*x(6) + u(3))/i3;
xdot(8) = 0;
for indx=1:nrows
    xdot(indx) = xdot(indx)*x(nrows)^2;
end
```

B.5 Case Study 5 Driver Functions and Script Files

This appendix includes driver functions and MATLAB script files used to solve the DPIP problem in Case Study 5.

B.5.1 Dynamic Programming/Interior-Point Driver, Cost, Equality, Inequality, and Dynamics Code

The DPIP driver code script file *exmaxradius.m*:

```
% number of states
n=3;
% number of controls
m=1;
```

```
% number of steps
N=11;
% problem parameters
% initial and final conditions
wi = [1 0 1];
wf = [0 0 0];
ti=0;
tf=3.321;

x1h = wi'; % initial conditions of states
xNh=wf';

    dt=(tf-ti)/(N-1);
    t=ti:dt:tf;
    t=t';

uh = [ones(ceil(N/2),1)*(60)*pi/180; ones(floor(N/2),1)*(270)*pi/180];

lamEh=ones(5,1)*1; % three initial/final specifications of states

lamIh=ones(2*(N-1),1)*1;

param=[ti tf wi wf];

% view trajectory from initial guess
    [t,xh]=drungeu('esmaxradius',[param(1) param(2)],N-1,x1h,uh);
    figure(1),
    subplot(122), plot(t,xh,'.-'); title('x'); grid; shg
    subplot(121), plot(t,uh*180/pi,'.-'); grid; title('u'); shg
        fcerror=eqmaxradius(xh,uh,N,N,n,m,param);
        fcnorm=norm(fcerror);
        fprintf(1,'fcerror = %g %g\n',fcerror)
        fprintf(1,'x(N) = %g %g %g \n',xh(N,:))

OIG=input('Optimize initial guess? (y=1/n=0)   ');
if (OIG==1),

iterateQ=1;
alpha=.08;
maxit=25;
while iterateQ<=maxit,
    %  dpeq returns changes to nominal values
    %  ub : change in controls
    %  x1b : change in initial state
    %  lamEb : change in Lagrange multipliers for equality constraints

    [ub,x1b,lamEb,lamIb]=dpip(uh,x1h,lamEh,lamIh,N,n,m,...
'esmaxradius','comaxradius','eqmaxradius','inmaxradius',param);
    uh = uh + alpha*ub;
    uh(N,:) = uh(N-1,:);
```

```
    [t,xh]=drungeu('esmaxradius',[param(1) param(2)],N-1,x1h,uh);

        fcerror=eqmaxradius(xh,uh,N,N,n,m,param);
        fcnorm=norm(fcerror);
        fprintf(1,'fcerror = %g %g\n',fcerror)
        fprintf(1,'x(N) = %g %g %g \n',xh(N,:))
        fprintf(1,'\n');
        figure(1),
        subplot(122), plot(t*58.18,xh,'.-'); title('x'); grid; shg
        subplot(121), plot(t*58.18,uh*180/pi,'.-'); grid; title('u');
        shg
    iterateQ=iterateQ+1;
    pause(.1)
end
end
shg
```

The cost function script file *comaxradius.m*:

```
function [y,z,Q,R,S]=comaxradius(xh,uh,i,N,n,m,param);
%
% Defines the cost function and some partial derivatives
% f is cost function
% yy is state
% uu is control
% xx is independent variable
%
% y is dfdy
% z is dfdu
% Q is dfdyy (second derivative)
% R is dfdydu
% S is dfduu

t1=param(1);
t2=param(2);
dt=(t2-t1)/(N-1);

tt=t1+dt*(i-1);  % current x (time, or independent variable); x_i
xx=xh(i,:);  % current y (state); y_i
uu=uh(i,:);  % current u (control); u_i

dfdx = zeros(n,1);
dfdxx = zeros(n,n);
dfdu = zeros(m,1);
dfduu = zeros(m,m);
y=dfdx;
z=dfdu;
Q=dfdxx;
% dfdydu matrix of partials that couples states and controls
R=zeros(n,m);
```

```
S=dfduu;
if i==N, y(1)=-1; end
```

The equality constraint script file *eqmaxradius.m*:

```
function [cE,AE,BE,cExx,cExu,cEuu]=eqmaxradius(xh,uh,i,N,n,m,param);

nc=0;
[cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);

if i==1
  nc=3;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  ic=param(3:5);
  cE=xh(i,:)'-ic';
  AE=eye(3);
end

if i==N
    mu=1;
  nc=2;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  cE=[xh(i,2)-0;
      xh(i,3)-sqrt(mu)/xh(i,1)^(3/2);
];
  AE=[0 1 0;
      3/2*xh(i,1)^(-5/2) 0 1;
];
  cExx = [zeros(3,3);
      -15/4*xh(i,1)^(-7/2) 0 0;
      0 0 0;
      0 0 0
];
end
```

The inequality constraint script file *inmaxradius.m*:

```
function [cI,AI,BI,cIxx,cIxu,cIuu]=inmaxradius(xh,uh,i,N,n,m,param);

if (i==N)
  nc=0;
  [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
else
        nc=2;
        [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);

  cI=[uh(i,1)-2*pi;
      -uh(i,1)];
  BI=[1; -1];
end
```

The equations of motion for Case Study 5 in script file *esmaxradius.m*:

```
function xdot=esmaxradius(t,x,u);

[nrows, ncols] = size(x);

mu = 1;
T = .1405;
m0 = 1;
mdot = 0.07487;

xdot = zeros(nrows,1);
xdot(1) = x(2);
xdot(2) = x(3)^2/x(1) - mu/x(1)^2 + T*sin(u(1))/(m0-mdot*t);
xdot(3) = -x(2)*x(3)/x(1) + T*cos(u(1))/(m0-mdot*t);
```

B.6 Case Study 6 Driver Functions and Script Files

This appendix includes driver functions and MATLAB script files used to solve the DPIP problem in Case Study 6.

B.6.1 Dynamic Programming/Interior-Point Driver, Cost, Equality, Inequality, and Dynamics Code

The DPIP driver code script file *expuma2.m*:

```
%
% PUMA560 DPIP driver code
%
D2R=pi/180;
% number of states
n=6;
% number of controls
m=3;
% number of steps
N=1501;
% problem parameters
% initial and final conditions
wi = [50*D2R -135*D2R 135*D2R 0 0 0];
wf = [-50*D2R -85*D2R 30*D2R 0 0 0];
tf=1.5;
x1h = wi'; % initial conditions of states
xNh=wf';

    dt=(tf-ti)/(N-1);
    t=ti:dt:tf;
    t=t';
```

```
uh=[temp1u temp2u temp3u]; % initially start with closed-loop torque
%                          responses to open-loop jerk trajectories

lamEh=ones(12,1)*0; % twelve initial/final specifications of states

lamIh = zeros(12*(N-1),1);     %  minimum effort with bounds

controlbound=1;
param=[ti tf wi wf controlbound];

% view trajectory from initial guess
    [t,xh]=drungeu('espuma2',[param(1) param(2)],N-1,x1h,uh);
    figure(1),
    subplot(121), plot(t,xh,'.-'); title('x'); grid; shg
    subplot(122), plot(t,uh,'.-'); grid; title('u'); shg
OIG=input('Optimize initial guess? (y=1/n=0)   ');
if (OIG==1),

iterateQ=1;
counter=1;
alpha=1.0;
maxit=8;
while iterateQ==1,
%while iterateQ<=maxit,
    %  dpeq returns changes to nominal values
    %  ub : change in controls
    %  x1b : change in initial state
    %  lamEb : change in Lagrange multipliers for equality constraints

    [ub,x1b,lamEb,lamIb]=dpip(uh,x1h,lamEh,lamIh,N,n,m,'espuma2',...
                        'copuma2','eqpuma2','inpuma2',param);
    uh = uh+ alpha*ub;
    uh(N,:) = uh(N-1,:);
    [t,xh]=drungeu('espuma2',[param(1) param(2)],N-1,x1h,uh);

        fcerror=eqpuma2(xh,uh,N,N,n,m,param);
        fcnorm=norm(fcerror);
        fprintf(1,'fcnorm = %g\n',fcnorm)
        fprintf(1,'\n');
        figure(1),
        subplot(121), plot(t,xh,'.-'); title('x'); grid; shg
        subplot(122), plot(t,uh,'.-'); grid; title('u'); shg

    iterateQ=input('Continue to iterate? (y=1/n=0)   ');
%    iterateQ=iterateQ+1;
    counter=counter+1;
end
end
shg
```

The cost function script file *copuma2.m*:

```
function [y,z,Q,R,S]=copuma2(xh,uh,i,N,n,m,param);
%
% Defines the cost function and some partial derivatives
% f is cost function
% yy is state
% uu is control
% xx is independent variable
%
% y is dfdy
% z is dfdu
% Q is dfdyy (second derivative)
% R is dfdydu
% S is dfduu

t1=param(1);
t2=param(2);
dt=(t2-t1)/(N-1);

tt=t1+dt*(i-1);  % current x (time, or independent variable); x_i
xx=xh(i,:);  % current y (state); y_i
uu=uh(i,:);  % current u (control); u_i

dfdx = zeros(n,1);
dfdxx = zeros(n,n);
dfdu = dt*[uu(1) uu(2) uu(3)]';
dfduu = dt*eye(3);
y=dfdx;
z=dfdu;
Q=dfdxx;
R=zeros(n,m); %dfdydu matrix of partials-couples states and controls
S=dfduu;
if (i==1)+(i==N) % the plus sign is an "or"
  y=y/2; % first & second partials of cost wrt state are different
  %        if at first or last node
  Q=Q/2;
end
```

The equality constraint script file *eqpuma2.m*:

```
function [cE,AE,BE,cExx,cExu,cEuu]=conepuma2(xh,uh,i,N,n,m,param);
%function [cE,AE,BE,cExx,cExu,cEuu]=conepuma2(xh,uh,i,N,n,m,param);
%  cone   name of function to calculate c_{iE}, A_{iE}, B_{iE},
%           c_{iE}_{xx}, c_{iE}_{xu} and c_{iE}_{uu}

nc=0;
[cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);

if i==1
```

```
  nc=6;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  ic=param(3:8);
  cE=xh(i,:)'-ic';
  AE=eye(6);
end

if i==N
  nc=6;
  [cE,AE,BE,cExx,cExu,cEuu]=zeroa(n,m,nc);
  fc=param(9:14);
  cE=xh(i,:)'-fc';
  AE=eye(6);
end
```

The inequality constraint script file *inpuma2.m*:

```
function [cI,AI,BI,cIxx,cIxu,cIuu]=conipuma2(xh,uh,i,N,n,m,param);

if (i==N)
  nc=0;
  [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
else
    nc=12;
    [cI,AI,BI,cIxx,cIxu,cIuu]=zeroa(n,m,nc);
    %
    a1=1.92;      % joint 1 velocity limit (rad/sec)
    a2=1.51;      % joint 2 velocity limit (rad/sec)
    a3=2.4;       % joint 3 velocity limit (rad/sec)
    b1=56.0;      % joint 1 torque limit (N-m)
    b2=97.0;      % joint 2 torque limit (N-m)
    b3=52.0;      % joint 3 torque limit (N-m)

    cI =[xh(i,4)-a1;
        -xh(i,4)-a1;
        xh(i,5)-a2;
        -xh(i,5)-a2;
        xh(i,6)-a3;
        -xh(i,6)-a3;
        uh(i,1) - b1;
        -uh(i,1) - b1;
        uh(i,2) - b2;
        -uh(i,2) - b2;
        uh(i,3) - b3;
        -uh(i,3) - b3];
    AI =[0 0 0 1 0 0;
        0 0 0 -1 0 0;
        0 0 0 0 1 0;
        0 0 0 0 -1 0;
        0 0 0 0 0 1;
```

```
        0 0 0 0 0 -1;
        0 0 0 0 0 0;
        0 0 0 0 0 0;
        0 0 0 0 0 0;
        0 0 0 0 0 0;
        0 0 0 0 0 0;
        0 0 0 0 0 0];
    BI = [0 0 0;
          0 0 0;
          0 0 0;
          0 0 0;
          0 0 0;
          0 0 0;
          1 0 0;
          -1 0 0;
          0 1 0;
          0 -1 0;
          0 0 1;
          0 0 -1];
end
```

The equations of motion for Case Study 6 in script file *espuma2.m*:

```
function xdot=espuma2(t,x,u);
[nrows, ncols] = size(x);
% Dynamic equations of motion for 3DOF PUMA560 industrial robot
% Only first 3 DOFs are used.  Wrist DOFs incorporated into last link
% The dynamics use RNE formulation and iteration, forward dynamics
% Calls a special routine that calls symbolically generated
% equations of motion
% This file is test file to use RNE to calculate the forward
% dynamics for PUMA560.
%
% Torques from incoming calculations.
%
tau(1)=u(1);
tau(2)=u(2);
tau(3)=u(3);
%
% Solve for bvec = G(q)+C(q_dot,q)
%
% set qq(i) and qqd(i) to the current values in the integration
% set qqdd(i) = 0   - all accelerations to zero
%
%
% set up state variables
%
qq(1) = x(1);
qq(2) = x(2);
qq(3) = x(3);
```

```
qqd(1) = x(4);
qqd(2) = x(5);
qqd(3) = x(6);
%
qqdd(1)=0.0;
qqdd(2)=0.0;
qqdd(3)=0.0;
%
% calculate bvec:
%
bvec=yprimetest3(qq,qqd,qqdd);
%
% Calculate mass matrix components  column by column
%
% set velocities to zero and use
% call to RNE that has gravity equal to zero
%
qqd(1)=0.0;
qqd(2)=0.0;
qqd(3)=0.0;
%
for i=1:3,
qqdd(1)=0.0;
qqdd(2)=0.0;
qqdd(3)=0.0;
qqdd(i)=1.0;
mass(:,i) = ypnograv3(qq,qqd,qqdd)';
end
imass=inv(mass);
%
% In calculation TAU = M qqdd + C(qq,qqd)+G(qq)
%
% Solve for the accelerations as
%
% qqdd = M^{-1}( TAU-C(qq,qqd)-G(qq) )
%
% OUTPUT: solve for accelerations
%
accel=imass*(tau'-bvec');
%
% yprime definitions
%
xdot(1) = x(4);
xdot(2) = x(5);
xdot(3) = x(6);
xdot(4) = accel(1);
xdot(5) = accel(2);
xdot(6) = accel(3);
```

B.6.2 PUMA560 Dynamic Equations of Motion C-Code

The MATLAB CMEX function provided the interface to existing C-code and has been modified to implement automated C-code developed with MAPLE software. This is contained in **yprimetest3.c** and **ypnograv3.c** (for no gravity).

yprimetest3.c:

```
/*=================================================================
 *
 * YPRIME.C Sample .MEX file corresponding to YPRIME.M
 *         Solves simple 3 body orbit problem
 *
 * The calling syntax is:
 *
 * [yp] = yprime(t, y)
 *
 *  You may also want to look at the corresponding M-code, yprime.m.
 *
 * This is a MEX-file for MATLAB.
 * Copyright 1984-2000 The MathWorks, Inc.
 *
 *=================================================================*/
/* $Revision: 1.10 $ */
#include <math.h>
#include "mex.h"

/* Input Arguments */

#define Q_IN prhs[0]
#define QD_IN prhs[1]
#define QDD_IN  prhs[2]

/* Output Arguments */

#define TAU_OUT plhs[0]

#if !defined(MAX)
#define MAX(A, B) ((A) > (B) ? (A) : (B))
#endif

#if !defined(MIN)
#define MIN(A, B) ((A) < (B) ? (A) : (B))
#endif

#define PI 3.14159265

/*
 * Main body of the run-time manipulator inverse dynamics function.
 * Requires 3 intermediate files produced by the MAPLE script genCcode.
 *
```

```
 * Copyright (c) 1996 Peter I. Corke
 */

static void
yprimetest3(double Tau[], double q[], double qd[], double qdd[])
{
#include        "TEMP-DECL3"
      double    C[N], Si[N], Co[N] ;
      int       i;

/* rne(*tautemp,*qtemp,*qdtemp,*qddtemp); */

for (i=0; i<N; i++)
{ Si[i] = sin(q[i]);
   C[i]= cos(q[i]);
   Co[i]= cos(q[i]); }

/*        for (i=0; i<N; i++)
                S[i] = sin(q[i]); C[i]= cos(q[i]);  */

#include        "TEMPS3"
#include        "TORQUES3"

    return;
}


void mexFunction( int nlhs, mxArray *plhs[],
  int nrhs, const mxArray*prhs[] )

{
    double *Tau;
    double *q,*qd,*qdd;
    unsigned int m,n;

    /* Check for proper number of arguments */

    if (nrhs != 3) {
mexErrMsgTxt("Three input arguments required.");
    } else if (nlhs > 1) {
mexErrMsgTxt("Too many output arguments.");
    }

    /* Check the dimensions of Q.  Q can be 3 X 1 or 1 X 3. */

    m = mxGetM(Q_IN);
    n = mxGetN(Q_IN);
    if (!mxIsDouble(Q_IN) || mxIsComplex(Q_IN) ||
(MAX(m,n) != 3) || (MIN(m,n) != 1)) {
mexErrMsgTxt("YPRIME requires that Q be a 3 x 1 vector.");
```

```
    }

    /* Create a matrix for the return argument */
    TAU_OUT = mxCreateDoubleMatrix(m, n, mxREAL);

    /* Assign pointers to the various parameters */
    Tau = mxGetPr(TAU_OUT);

    q  = mxGetPr(Q_IN);
    qd = mxGetPr(QD_IN);
    qdd = mxGetPr(QDD_IN);

    /* Do the actual computations in a subroutine */
    yprimetest3(Tau,q,qd,qdd);
    return;

}
```

ypnograv3.c:

```
/*=========================================================================
 *
 * YPRIME.C Sample .MEX file corresponding to YPRIME.M
 *         Solves simple 3 body orbit problem
 *
 * The calling syntax is:
 *
 * [yp] = yprime(t, y)
 *
 *  You may also want to look at the corresponding M-code, yprime.m.
 *
 * This is a MEX-file for MATLAB.
 * Copyright 1984-2000 The MathWorks, Inc.
 *
 *=======================================================================*/
/* $Revision: 1.10 $ */
#include <math.h>
#include "mex.h"

/* Input Arguments */

#define Q_IN1 prhs[0]
#define QD_IN1 prhs[1]
#define QDD_IN1  prhs[2]

/* Output Arguments */

#define TAU_OUT1 plhs[0]

#if !defined(MAX)
```

```
#define MAX(A, B) ((A) > (B) ? (A) : (B))
#endif

#if !defined(MIN)
#define MIN(A, B) ((A) < (B) ? (A) : (B))
#endif

#define PI 3.14159265

/*
 * Main body of the run-time manipulator inverse dynamics function.
 * Requires 3 intermediate files produced by the MAPLE script genCcod
 *
 * Copyright (c) 1996 Peter I. Corke
 */

static void
ypnograv3(double Tau[], double q[], double qd[], double qdd[])
{
#include        "TEMP-DECL3"
      double    C[N], Si[N], Co[N] ;
      int       i;

/* rne(*tautemp,*qtemp,*qdtemp,*qddtemp); */

for (i=0; i<N; i++)
{ Si[i] = sin(q[i]);
   C[i]= cos(q[i]);
   Co[i]= cos(q[i]); }

/*        for (i=0; i<N; i++)
                S[i] = sin(q[i]); C[i]= cos(q[i]);  */

#include        "TEMPSNOGRAV3"
#include        "TORQUES3"

    return;
}

void mexFunction( int nlhs, mxArray *plhs[],
  int nrhs, const mxArray*prhs[] )

{
    double *Tau;
    double *q,*qd,*qdd;
    unsigned int m,n;

    /* Check for proper number of arguments */

    if (nrhs != 3) {
```

```
mexErrMsgTxt("Three input arguments required.");
    } else if (nlhs > 1) {
mexErrMsgTxt("Too many output arguments.");
    }

    /* Check the dimensions of Q.  Q can be 3 X 1 or 1 X 3. */

    m = mxGetM(Q_IN1);
    n = mxGetN(Q_IN1);
    if (!mxIsDouble(Q_IN1) || mxIsComplex(Q_IN1) ||
(MAX(m,n) != 3) || (MIN(m,n) != 1)) {
mexErrMsgTxt("YPRIME requires that Q be a 3 x 1 vector.");
    }

    /* Create a matrix for the return argument */
    TAU_OUT1 = mxCreateDoubleMatrix(m, n, mxREAL);

    /* Assign pointers to the various parameters */
    Tau = mxGetPr(TAU_OUT1);

    q  = mxGetPr(Q_IN1);
    qd = mxGetPr(QD_IN1);
    qdd = mxGetPr(QDD_IN1);

    /* Do the actual computations in a subroutine */

    ypnograv3(Tau,q,qd,qdd);
    return;

}
```

The automated C-code is given in the following files:

1. **TEMP-DECL3** (declarations),
2. **TEMPS3** (three-DOF PUMA560 dynamics),
3. **TEMPSnograv3** (three-DOF PUMA560 dynamics without gravity), and
4. **TORQUES3** (torque definitions).

TEMP-DECL3:

```
#define N    3
        double
           T0   ,T1   ,T2   ,T3   ,T4   ,T5   ,
           T6   ,T7   ,T8   ,T9   ,T10   ,T11   ,
           T12   ,T13   ,T14   ,T15   ,T16   ,
           T17   ,T18   ,T19   ,T20   ,T21   ,
           T22   ,T23   ,T24   ,T25   ,T26   ,
           T27   ,T28   ,T29   ,T30   ,T31   ,
```

```
            T32   ,T33   ,T34   ,T35   ,T36   ,
            T37   ,T38   ;
```

TEMPS3:

```
/* MAPLE symbolically derived equations of motion for 3 DOF PUMA560 */

      T0 = Co[1]*qd[0]*qd[1]+Si[1]*qdd[0];
      T1 = -Si[1]*qd[0]*qd[1]+Co[1]*qdd[0];
      T2 = -0.4318*Co[1]*Co[1]*qd[0]*qd[0]-0.4318*qd[1]*qd[1]
+0.981E1*Si[1];
      T3 = 0.4318*qdd[1]+0.4318*Co[1]*qd[0]*qd[0]*Si[1]+0.981E1*Co[1];
      T4 = -0.4318*T1+0.4318*Si[1]*qd[0]*qd[1];
      T5 = 0.774E-1*T1-0.6E-2*qdd[1]+Co[1]*qd[0]*(0.6E-2*Si[1]*qd[0]
+0.3638*Co[1]*qd[0])-qd[1]*(-0.3638*qd[1]-0.774E-1*Si[1]*qd[0])+T2;
      T6 = -0.3638*qdd[1]-0.774E-1*T0+qd[1]*(0.774E-1*Co[1]*qd[0]
-0.6E-2*qd[1])-Si[1]*qd[0]*(0.6E-2*Si[1]*qd[0]+0.3638*Co[1]*qd[0])+T3;
      T7 = 0.6E-2*T0+0.3638*T1+Si[1]*qd[0]*(-0.3638*qd[1]
-0.774E-1*Si[1]*qd[0])-Co[1]*qd[0]*(0.774E-1*Co[1]*qd[0]
           -0.6E-2*qd[1])+T4;
      T8 = Co[2]*Si[1]*qd[0]+Si[2]*Co[1]*qd[0];
      T9 = -qd[1]-qd[2];
      T10 = -Si[2]*Si[1]*qd[0]+Co[2]*Co[1]*qd[0];
      T11 = Co[2]*(T0+Co[1]*qd[0]*qd[2])+Si[2]*(T1-Si[1]*qd[0]*qd[2]);
      T12 = -qdd[1]-qdd[2];
      T13 = -Si[2]*(T0+Co[1]*qd[0]*qd[2])+Co[2]*(T1-Si[1]*qd[0]*qd[2])
      T14 = 0.1254*T13+T9*(-0.1254*T8-0.2032E-1*T9)-0.2032E-1*T10*T10
+Co[2]*T2+Si[2]*T3;
      T15 = 0.2032E-1*T13+0.1254*T10*T10
            -T8*(-0.1254*T8-0.2032E-1*T9)-T4;
      T16 = -0.1254*T11-0.2032E-1*T12+0.2032E-1*T8*T10-0.1254*T9*T10
-Si[2]*T2+Co[2]*T3;
      T17 = 0.143*T12+0.14E-1*T13-0.14E-1*T9*T8+0.143*T8*T10+T14;
      T18 = -0.143*T11+T10*(0.143*T9+0.14E-1*T10)+0.14E-1*T8*T8+T15;
      T19 = -0.14E-1*T11-0.143*T8*T8-T9*(0.143*T9+0.14E-1*T10)+T16;
      T20 = 0.192*T11-0.1966*T9*T10;
      T21 = 0.212*T12+0.1766*T8*T10;
      T22 = 0.154E-1*T13+0.2E-1*T9*T8;
      T23 = -0.841976*T19-0.86372*T18+T20;
      T24 = 0.86372*T17-0.1227328*T19+T21;
      T25 = 0.1227328*T18+0.841976*T17+T22;
      T26 = 0.604E1*Co[2]*T17-0.604E1*Si[2]*T19+0.174E2*T5;
      T27 = 0.604E1*Si[2]*T17+0.604E1*Co[2]*T19+0.174E2*T6;
      T28 = -0.604E1*T18+0.174E2*T7;
      T29 = 0.13*T0+0.15E-1*Co[1]*qd[0]*qd[1];
      T30 = 0.524*T1-0.409*Si[1]*qd[0]*qd[1];
      T31 = 0.539*qdd[1]+0.394*Co[1]*qd[0]*qd[0]*Si[1];
      T32 = Co[2]*(T23+0.2608072E1*Si[2]*T18)-Si[2]*(T25+0.2608072E1
*Co[2]*T18)+0.1044*T7-0.134676E1*T6+T29;
      T33 = Si[2]*(T23+0.2608072E1*Si[2]*T18)+Co[2]*(T25+0.2608072E1
```

```
*Co[2]*T18)+0.134676E1*T5-0.11832E1*T7+T30;
      T34 = -T24+0.2608072E1*Si[2]*T17+0.2608072E1*Co[2]*T19
+0.11832E1*T6-0.1044*T5+T31;
      T35 = Co[1]*T26-Si[1]*T27;
      T36 = Si[1]*T26+Co[1]*T27;
      T37 = T32*Co[1]-T33*Si[1];
      T38 = T32*Si[1]+T33*Co[1]+0.35*qdd[0];
```

TEMPSnograv3:

```
/* MAPLE symbolically derived equations of motion for 3-DOF PUMA560 */

      T0 = Co[1]*qd[0]*qd[1]+Si[1]*qdd[0];
      T1 = -Si[1]*qd[0]*qd[1]+Co[1]*qdd[0];
      T2 = -0.4318*Co[1]*Co[1]*qd[0]*qd[0]-0.4318*qd[1]*qd[1];
      T3 = 0.4318*qdd[1]+0.4318*Co[1]*qd[0]*qd[0]*Si[1];
      T4 = -0.4318*T1+0.4318*Si[1]*qd[0]*qd[1];
      T5 = 0.774E-1*T1-0.6E-2*qdd[1]+Co[1]*qd[0]*(0.6E-2*Si[1]*qd[0]
+0.3638*Co[1]*qd[0])-qd[1]*(-0.3638*qd[1]-0.774E-1*Si[1]*qd[0])+T2;
      T6 = -0.3638*qdd[1]-0.774E-1*T0+qd[1]*(0.774E-1*Co[1]*qd[0]
-0.6E-2*qd[1])-Si[1]*qd[0]*(0.6E-2*Si[1]*qd[0]+0.3638*Co[1]*qd[0])+T3;
      T7 = 0.6E-2*T0+0.3638*T1+Si[1]*qd[0]*(-0.3638*qd[1]
-0.774E-1*Si[1]*qd[0])-Co[1]*qd[0]*(0.774E-1*Co[1]*qd[0]
          -0.6E-2*qd[1])+T4;
      T8 = Co[2]*Si[1]*qd[0]+Si[2]*Co[1]*qd[0];
      T9 = -qd[1]-qd[2];
      T10 = -Si[2]*Si[1]*qd[0]+Co[2]*Co[1]*qd[0];
      T11 = Co[2]*(T0+Co[1]*qd[0]*qd[2])+Si[2]*(T1-Si[1]*qd[0]*qd[2]);
      T12 = -qdd[1]-qdd[2];
      T13 = -Si[2]*(T0+Co[1]*qd[0]*qd[2])+Co[2]*(T1-Si[1]*qd[0]*qd[2]);
      T14 = 0.1254*T13+T9*(-0.1254*T8-0.2032E-1*T9)-0.2032E-1*T10*T10
+Co[2]*T2+Si[2]*T3;
      T15 = 0.2032E-1*T13+0.1254*T10*T10
          -T8*(-0.1254*T8-0.2032E-1*T9)-T4;
      T16 = -0.1254*T11-0.2032E-1*T12+0.2032E-1*T8*T10-0.1254*T9*T10
-Si[2]*T2+Co[2]*T3;
      T17 = 0.143*T12+0.14E-1*T13-0.14E-1*T9*T8+0.143*T8*T10+T14;
      T18 = -0.143*T11+T10*(0.143*T9+0.14E-1*T10)+0.14E-1*T8*T8+T15;
      T19 = -0.14E-1*T11-0.143*T8*T8-T9*(0.143*T9+0.14E-1*T10)+T16;
      T20 = 0.192*T11-0.1966*T9*T10;
      T21 = 0.212*T12+0.1766*T8*T10;
      T22 = 0.154E-1*T13+0.2E-1*T9*T8;
      T23 = -0.841976*T19-0.86372*T18+T20;
      T24 = 0.86372*T17-0.1227328*T19+T21;
      T25 = 0.1227328*T18+0.841976*T17+T22;
      T26 = 0.604E1*Co[2]*T17-0.604E1*Si[2]*T19+0.174E2*T5;
      T27 = 0.604E1*Si[2]*T17+0.604E1*Co[2]*T19+0.174E2*T6;
      T28 = -0.604E1*T18+0.174E2*T7;
      T29 = 0.13*T0+0.15E-1*Co[1]*qd[0]*qd[1];
      T30 = 0.524*T1-0.409*Si[1]*qd[0]*qd[1];
```

```
      T31 = 0.539*qdd[1]+0.394*Co[1]*qd[0]*qd[0]*Si[1];
      T32 = Co[2]*(T23+0.2608072E1*Si[2]*T18)-Si[2]*(T25+0.2608072E1
*Co[2]*T18)+0.1044*T7-0.134676E1*T6+T29;
      T33 = Si[2]*(T23+0.2608072E1*Si[2]*T18)+Co[2]*(T25+0.2608072E1
*Co[2]*T18)+0.134676E1*T5-0.11832E1*T7+T30;
      T34 = -T24+0.2608072E1*Si[2]*T17+0.2608072E1*Co[2]*T19
+0.11832E1*T6-0.1044*T5+T31;
      T35 = Co[1]*T26-Si[1]*T27;
      T36 = Si[1]*T26+Co[1]*T27;
      T37 = T32*Co[1]-T33*Si[1];
      T38 = T32*Si[1]+T33*Co[1]+0.35*qdd[0];
```

TORQUES3:

```
      Tau[0] = T38;
      Tau[1] = T34;
      Tau[2] = -T24;
```

B.7 Discrete Dynamic Programming Standalone Code—Discrete-Time Optimal Control Problem

This appendix includes the driver function that calls discrete dynamic programming (DDP) discrete-time optimal control solutions for Examples 3.4 and 3.5 in Chapter 3. The dynamic system is the single rigid slewing link.

Driver and plot file:

```
%  dp_driver_ex3p45.m    - Examples 3.4 and 3.5 in DP book
%  This file is the driver file for both dp_ex_3p4.m and
%  dp_ex_3p5.m.  This file plots data in document-ready
%  form for the DP rigid link slew problem.
%  NOTE:  runs with MATLAB 6.1
%
dp_ex_3p4    % Run DP code for Example 3.4
tout=time;
h=plot(tout,xout1),grid;
ha=get(h,'Parent');
h=plot(tout,xout1,tout,xout2);
set(ha,'FontName','Times');
set(ha,'FontSize',22);
xlabel('Time (sec)');
ylabel('States 1 and 2');
pause % Hit return
clf
h=plot(tout,xout1),grid;
ha=get(h,'Parent');
h=plot(tout,uout);
set(ha,'FontName','Times');
set(ha,'FontSize',22);
xlabel('Time (sec)');
```

```
ylabel('Torque');
pause  % Hit return

clear all

dp_ex_3p5     % Run DP code for Example 3.5
tout=time;
h=plot(tout,xout1),grid;
ha=get(h,'Parent');
h=plot(tout,xout1,tout,xout2);
set(ha,'FontName','Times');
set(ha,'FontSize',22);
xlabel('Time (sec)');
ylabel('States 1 and 2');
pause    % Hit return
clf
h=plot(tout,xout1),grid;
ha=get(h,'Parent');
h=plot(tout,xout3);
set(ha,'FontName','Times');
set(ha,'FontSize',22);
xlabel('Time (sec)');
ylabel('Torque');
```

Example 3.4 DDP solution:

```
% dp_ex_3p4.m
% This file generates DP solution
% to rigid robot plots - Example 3.4
% Store main matrices in structure format:
%    hstuff(k).h3invh5=h3invh5;
%    hstuff(k).h3invh2t=h3invh2t;
%
n=201;
tf=1.2;
j=1.5;
s=1.0;
p=1.e+6;
h=tf/(n-1);
theta_f=pi/2;
xf=[theta_f;0];
v=-p*xf;
w=p*eye(2);
a=[1 h; 0 1]
b=(1/j)*[h^2/2;h]
for k1=1:n-1,
   k=n-k1;  % step backward from final---> initial
   h1=a'*w*a;
   h2=a'*w*b;
   h3=s+b'*w*b;
```

```
   h4=a'*v;
   h5=b'*v;
   h3invh5=(1/h3)*h5;
   h3invh2t=(1/h3)*h2';
   v=h4-h2*h3invh5;
   w=h1-h2*h3invh2t;
% save matrix data:
   hstuff(k).h3invh5=h3invh5;
   hstuff(k).h3invh2t=h3invh2t;
end
x=[0;0];
for k=1:n-1,
%  step forward in time as normal  initial ---> final
%  x=a*x+b*u;
   u=-hstuff(k).h3invh2t*x-hstuff(k).h3invh5;
   x=a*x+b*u;
   time(k)=(k-1)*h;
   uout(k)=u;
   xout1(k)=x(1,1);
   xout2(k)=x(2,1);
end
```

Example 3.5 DDP solution:

```
% dp_ex_3p5.m
%  This file creates rigid slewing link DP
%  solution with M_dot as a third state and constraint on
%  torque (acceleration)
n=201;
tf=1.2;
j=1.5;
p=1.e+8;
h=tf/(n-1);
theta_f=pi/2;
a=[1 h h^2/(2*j); 0 1 h/j; 0 0 1];
b=[h^3/(6*j);h^2/(2*j); h];
s=1;
xf=[theta_f;0;0];
v=-p*xf;
w=p*eye(3);
r=1*[0; 0; 1];
for k1=1:n-1,
   k=n-k1;   % step backward from final --> initial
   h1=a'*w*a;
   h2=r+a'*w*b;
   h3=s+b'*w*b;
   h4=a'*v;
   h5=b'*v;
   h3invh5=(1/h3)*h5;
   h3invh2t=(1/h3)*h2';
```

```
   v=h4-h2*h3invh5;
   w=h1-h2*h3invh2t;
   % save matrix data:
   hstuff(k).h3invh5=h3invh5;
   hstuff(k).h3invh2t=h3invh2t;
end
x=[0;0;0];
for k=1:n-1,
%  step forward in time as normal  initial --> final
   u=-hstuff(k).h3invh2t*x-hstuff(k).h3invh5;
   x=a*x+b*u;
   time(k)=(k-1)*h;
   uout(k)=u;
   xout1(k)=x(1,1);
   xout2(k)=x(2,1);
   xout3(k)=x(3,1);
end
```

This page intentionally left blank

Bibliography

[1] R. M. MURRAY, ED., *Control in an Information Rich World: Report of the Panel on Future Directions in Control, Dynamics, and Systems*, SIAM, Philadelphia, 2003.

[2] M. B. MILAM, K. MUSHAMBI, AND R. M. MURRAY, *A New Computational Approach to Real-Time Trajectory Generation for Constrained Mechanical Systems*, paper presented at the IEEE Conference on Decision and Control, Sydney, Australia, 2000.

[3] C. R. DOHRMANN AND R. D. ROBINETT, III, *Robot Trajectory Planning via Dynamic Programming*, Proceedings of the Fifth International Symposium on Robotics and Manufacturing, and Applications, Maui, HI, pp. 75–81, 1994.

[4] M. W. SPONG AND M. VIDYASAGAR, *Robot Dynamics and Control*, John Wiley and Sons, New York, 1989.

[5] G. R. EISLER, R. D. ROBINETT, III, D. J. SEGALMAN, AND J. T. FEDDEMA, *Approximate Optimal Trajectories for Flexible-Link Manipulator Slewing Using Recursive Quadratic Programming*, Journal of Dynamic Systems, Measurement, and Control, Vol. 115, pp. 405–410, September, 1993.

[6] B. J. PETTERSON AND R. D. ROBINETT, III, *Model-Based Damping of Coupled Horizontal and Vertical Oscillations in a Flexible Rod*, Journal of Intelligent and Robotic Systems, Vol. 4, pp. 285–299, 1991.

[7] J. BEN-ASHER, J. A. BURNS, AND E. M. CLIFF, *Time-Optimal Slewing of Flexible Spacecraft*, Journal of Guidance, Control, and Dynamics, Vol. 15, No. 2, pp. 360–367, March–April, 1992.

[8] M. J. D. POWELL, *A Fast Algorithm for Nonlinearly Constrained Optimization*, Springer-Verlag Lecture Notes in Mathematics, Vol. 630, pp. 144–157, 1977.

[9] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, New York, 1981.

[10] G. N. VANDERPLAATS, *Numerical Optimization Techniques for Engineering Design with Applications*, McGraw–Hill, New York, 1984.

[11] H. W. KUHN AND A. W. TUCKER, *Nonlinear Programming*, Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, J. Neyman (Ed.), pp. 481–492, 1950.

[12] W. KARUSH, *Minima of Functions of Several Variables with Inequalities as Side Conditions*, Master's Thesis, University of Chicago, 1939.

[13] C. G. BROYDEN, *The Convergence of a Class of Double-Rank Minimization Algorithms*, Journal of the Institute of Mathematics and Its Applications, Vol. 6, pp. 76–90, 1970.

[14] R. FLETCHER, *A New Approach to Variable Metric Algorithms*, Computer Journal, Vol. 13, pp. 317–322, 1970.

[15] D. GOLDFARB, *A Family of Variable Metric Updates Derived by Variational Means*, Mathematics of Computing, Vol. 24, pp. 23–26, 1970.

[16] D. F. SHANNO, *Conditioning of Quasi-Newton Methods for Function Minimization*, Mathematics of Computing, Vol. 24, pp. 647–656, 1970.

[17] J. E. DENNIS, JR., AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1983.

[18] P. S. MAYBECK, *Stochastic Models, Estimation, and Control*, Vol. 1, Academic Press, New York, 1979.

[19] T. COLEMAN, M. A. BRANCH, AND A. GRACE, *Optimization Toolbox—User's Guide Version* 2, The MathWorks Inc., Natick, MA, 1999.

[20] The Optimization Technology Center of Northwestern University at http://www.ece.northwestern.edu/OTC.

[21] P. W. FUERSCHBACH, *Measurement and Prediction of Energy Transfer Efficiency in Laser Beam Welding*, Welding Journal, Vol. 75, pp. 24s–34s, 1994.

[22] N. N. RYKALIN, *Calculation of Heat Flow in Welding*, The Welding Institute, Moscow, 1951.

[23] G. R. EISLER AND P. W. FUERSCHBACH, *A Computational Method for Developing Laser Weld Schedules*, Sandia National Laboratories, Albuquerque, NM, 1994.

[24] G. R. EISLER AND D. G. HULL, *A Guidance Law for Hypersonic Descent to a Point*, Journal of Guidance, Control, and Dynamics, pp. 649–654, July–August, 1994.

[25] W. H. PRESS, B. P. FLANNERY, S. A. TEUKOLSKY, AND W. T. VETTERLING, *Numerical Recipes: The Art of Scientific Computing,* Cambridge University Press, Cambridge, U.K., 1986.

[26] R. D. ROBINETT, III, *Angular Equations of Motion for a Prototype, Two-Gimbal, Satellite Pointing System*, unpublished notes, Sandia National Laboratories, Albuquerque, NM, September, 1996.

[27] MATLAB, The MathWorks, Inc., Natick, MA, January 1999.

[28] J. C. DUNN AND D. P. BERTSEKAS, *Efficient Dynamic Programming Implementations of Newton's Method for Unconstrained Optimal Control Problems*, Journal of Optimization Theory and Applications, Vol. 63, No. 1, 1989, pp. 23–38.

[29] C. R. DOHRMANN AND R. D. ROBINETT, III, *Efficient Sequential Quadratic Programming Implementations for Equality-Constrained Discrete-Time Optimal Control*, Journal of Optimization Theory and Applications, Vol. 95, No. 2, 1997, pp. 323–346.

[30] C. R. DOHRMANN AND R. D. ROBINETT, III, *Efficient Sequential Quadratic Programming Implementations for Equality-Constrained Discrete-Time Optimal Control*, Sandia Report SAND95-2574J, Sandia National Laboratories, October, 1995.

[31] C. R. DOHRMANN AND R. D. ROBINETT, III, *Dynamic Programming Method for Discrete-Time Optimal Control*, Journal of Optimization Theory and Applications, Vol. 101, No. 2, 1999, pp. 259–283.

[32] R. E. BELLMAN, *An Introduction to the Theory of Dynamic Programming*, Rand Corporation, Santa Monica, CA, 1953.

[33] R. E. BELLMAN, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[34] R. E. BELLMAN, *Applied Dynamic Programming*, Princeton University Press, Princeton, NJ, 1962.

[35] S. E. DREYFUS, *Dynamic Programming and the Calculus of Variations*, Academic Press, New York, 1965.

[36] C. R. DOHRMANN AND R. D. ROBINETT, III, *Input Shaping for Three-Dimensional Slew Maneuvers of a Precision Pointing Flexible Spacecraft*, Proceedings of the 1994 American Control Conference, pp. 2543–2547, June 1994.

[37] J. R. CANAVIN AND P. W. LIKINS, *Floating Reference Frames for Flexible Spacecraft*, Journal of Spacecraft and Rockets, Vol. 14, No. 12, pp. 724–732, 1977.

[38] T. R. KANE, P. W. LIKINS, AND D. A. LEVINSON, *Spacecraft Dynamics*, McGraw–Hill, New York, 1983.

[39] C. R. DOHRMANN, G. R. EISLER, AND R. D. ROBINETT, III, *Dynamic Programming Approach for Burnout-to-Apogee Guidance of Precision Munitions*, AIAA Journal of Guidance, Control, and Dynamics, Vol. 19, No. 2, pp. 340–346, March–April, 1996.

[40] C. R. DOHRMANN, G. R. EISLER, AND R. D. ROBINETT, III, *Dynamic Programming Approach for Burnout-to-Apogee Guidance of Precision Munitions*, Sandia Report SAND95-1333J, Sandia National Laboratories, 1995.

[41] S. SNYDER, L. TELLMAN, P. TORREY, AND S. KOHLI, *INS/GPS Operational Concept Demonstration (OCD) High Gear Program*, IEEE Aerospace and Electronic Systems Magazine, Vol. 9, No. 9, pp. 38–44, 1994.

[42] G. P. Frost and B. P. Schweitzer, *Operational Issues of GPS Aided Precision Missiles*, Proceedings of the 1993 National Technical Meeting of the Institute of Navigation, San Francisco, CA, pp. 325–338.

[43] K. M. Kessler, F. G. Karkalik, J. Sun, and J. D. Brading, *Performance Analysis of Integrated GPS/INS Guidance for Air-to-Ground Weapons in a Jamming Environment*, Proceedings of the Fourth International Technical Meeting of the Satellite Division of the Institute of Navigation, Albuquerque, NM, pp. 139–148, 1991.

[44] M. Singh, J. Fraysee, and S. Kohli, *Silicon Inertial and GPS Guidance for Gun-Launched Munitions*, Proceedings of the Seventh International Technical Meeting of the Satellite Division of the Institute of Navigation, Salt Lake City, UT, 1994, pp. 477–482.

[45] W. H. T. Loh, *Dynamics and Thermodynamics of Planetary Entry*, Prentice–Hall, Englewood Cliffs, NJ, 1963.

[46] R. D. Robinett, III, B. Rainwater, and S. A. Kerr, *Moving Mass Trim Control for Aerospace Vehicles*, Proceedings of the Annual AIAA Missile Sciences Conference, Monterey, CA, 1994.

[47] K. V. Chavez, *MLRS Aerodynamic Coefficients*, Sandia National Laboratories Internal Memorandum, February, 1993.

[48] S. E. Dreyfus and M. L. Averill, *The Art and Theory of Dynamic Programming*, Academic Press, New York, 1977.

[49] F. E. Stengel, *Stochastic Optimal Control: Theory and Application*, Wiley, New York, 1986.

[50] A. Ralston and P. Rabinowitz, *A First Course in Numerical Analysis*, 2nd Ed., McGraw–Hill, New York, 1978.

[51] B. Friedland, *Control System Design: An Introduction to State-Space Methods*, McGraw–Hill, New York, 1986.

[52] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The John Hopkins University Press, Baltimore, MD, 1989.

[53] D. F. Davidenko, *On a New Method of Numerical Solution of Systems of Non-Linear Equations*, Doklady Akademii Nauk USSR, Vol. 88, pp. 601–602, 1953.

[54] J. P. Dunyak, J. L. Junkins, and L. T. Watson, *Robust Nonlinear Least Squares Estimation Using the Chow-York Homotopy Method*, Journal of Guidance, Control, and Dynamics, Vol. 7, No. 7, pp. 752–754, 1984.

[55] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt, *First Leaves: A Tutorial Introduction to Maple* V, Springer-Verlag, New York, 1992.

[56] J. L. JUNKINS AND J. D. TURNER, *Optimal Continuous Torque Attitude Maneuvers*, Journal of Guidance, Control, and Dynamics, Vol. 3, No. 3, pp. 210–217, 1980.

[57] H. GOLDSTEIN, *Classical Mechanics*, Addison–Wesley, Reading, MA, 1980.

[58] M. J. FORRAY, *Variational Calculus in Science and Engineering*, McGraw–Hill, New York, 1968.

[59] S. J. WRIGHT, *Interior Point Methods for Optimal Control of Discrete Time Systems*, Journal of Optimization Theory and Application, Vol. 77, No. 1, pp. 161–187, 1993.

[60] S. J. WRIGHT, *Applying New Optimization Algorithms to Model Predictive Control*, Preprint MCS-P561-O 197, Argonne National Laboratory, Argonne, IL, 1996.

[61] S. J. WRIGHT, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, 1997.

[62] J. J. MORÉ AND S. J. WRIGHT, *Optimization Software Guide*, SIAM, Philadelphia, 1993.

[63] R. FLETCHER, *Practical Methods of Optimization*, 2nd Ed., John Wiley & Sons, New York, 1985.

[64] J. S. LONGSDON, *Efficient Determination of Optimal Control Profiles for Differential Algebraic Systems*, PhD Thesis, Carnegie-Mellon University, 1990.

[65] G. G. PARKER, B. PETTERSON, C. R. DOHRMANN, AND R. D. ROBINETT, III, *Command Shaping for Residual Vibration Free Crane Maneuvers*, Proceedings of the American Control Conference, Seattle, WA, June, 1995.

[66] D. STOKES AND D. G. WILSON, *Optimized Input Shaped Open-Loop Control for a Jib Crane*, Midterm Project, University of New Mexico, Manufacturing and Engineering Department, March, 1995.

[67] M. J. HOPPER, *Harwell Subroutine Library* AERE-R9185, UJAEA, Harwell, Oxon, UK, 1978.

[68] J. E. HURTADO, C. R. DOHRMANN, G. R. EISLER, AND R. D. ROBINETT, III, *MATLAB Code and User's Guide for DPIP*, Sandia National Laboratories, February, 2005.

[69] R. D. ROBINETT, III, C. R. DOHRMANN, G. R. EISLER, J. T. FEDDEMA, G. G. PARKER, D. G. WILSON, AND D. STOKES, *Flexible Robot Dynamics and Controls*, Kluwer Academic/Plenum Publishers, New York, 2002.

[70] E. GARCIA AND D. J. INMAN, *Modeling of the Slewing Control of a Flexible Structure*, Journal of Guidance, Control and Dynamics, Vol. 4, No. 4, pp. 736–742, July–August, 1991.

[71] J.-N. JUANG, L. G. HORTA, AND H. H. ROBERTSHAW, *A Slewing Control Experiment for Flexible Structures*, Journal of Guidance, Control and Dynamics, Vol. 9, No. 5, pp. 599–607, September–October, 1986.

[72] R. H. CANNON, JR., AND E. SCHMITZ, *Initial Experiments on the End-Point Control of a Flexible One-Link Robot*, International Journal of Robotics Research, Vol. 3, No. 3, pp. 62–75, 1984.

[73] L. MEIROVITCH, *Computational Methods in Structural Dynamics*, Sijhoff & Noordhoff, Rockville, MD, 1980.

[74] W. C. HURTY AND M. F. RUBENSTEIN, *Dynamics of Structures*, Prentice–Hall, Englewood Cliffs, NJ, 1964.

[75] J. F.-C. KUO AND S.-C. LIN, *An Entire Strategy for Precise System Tracking Control of a Revolving Thin Flexural Link, Part* I: *Finite Element Modeling and Direct Tuning Controller Design*, Mathematical and Computer Modeling, Vol. 29, pp. 99–113, 1999.

[76] D. G. WILSON AND D. STOKES, *Optimized Input Shaping Control for a Single Flexible Robot Arm*, Final Project, University of New Mexico, Manufacturing and Engineering Department, May, 1995.

[77] D. G. WILSON, D. STOKES, G. P. STARR, AND R. D. ROBINETT, III, *Optimized Input Shaping for a Single Flexible Link*, SPACE 96: Fifth International Conference and Exposition on ECOS, Albuquerque, NM, June 1996.

[78] F. L. LEWIS, C. T. ABDALLAH, AND D. M. DAWSON, *Control of Robot Manipulators*, MacMillan, New York, 1993.

[79] M. E. STRIEBLER, C. P. TRUDEL, AND D. G. HUNTER, *Robotic Systems for the International Space Station*, Proceedings of the IEEE International Conference on Robotics and Automation, Albuquerque, NM, pp. 3068–3073, 1997.

[80] H.-P. XIE, S. KALACIOGLU, AND R. V. PATEL, *Control of Residual Vibrations in the Space Shuttle Remote Manipulator System*, Proceedings of the IEEE International Conference on Robotics and Automation, Albuquerque, NM, pp. 2759–2764, 1997.

[81] J. R. NEWSOM, W. E. LAYMAN, H. B. WAITES, AND R. J. HAYDUK, *The NASA Controls-Structure Interaction Technology Program*, Proceedings of the 41st Congress of the International Astronautical Federation, Dresedent, NM, October, 1990.

[82] R. KRESS, L. LOVE, R. DUBEY, AND A. GIZELAR, *A Waste Tank Cleanup Manipulator: Modeling and Control*, Proceedings of the IEEE International Conference on Robotics and Automation, Albuquerque, NM, pp. 662–668, 1997.

[83] L. LOVE, R. KRESS, AND J. JANSEN, *Modeling and Control of a Hydraulically Actuated Flexible-Prismatic Link Robot*, Proceedings of the IEEE International Conference on Robotics and Automation, Albuquerque, NM, pp. 669–675, 1997.

[84] R. E. GRAHAM, *Demonstration and Testing of a Waste Conveyance System Based on Robotic Manipulator EMMA*, Robotics 98, Proceedings of the Third ASCE Specialty Conference on Robotics for Challenging Environments, pp. 99–105, 1998.

[85] B. HATCHELL, J. YOUNT, AND E. BERGLIN, *Long Reach Manipulator Demonstrations for Tank Waste Retrieval*, Robotics 98, Proceedings of the Third ASCE Specialty Conference on Robotics for Challenging Environments, pp. 106–112, 1998.

[86] D. G. WILSON, R. D. ROBINETT, III, AND G. R. EISLER, *Discrete Dynamic Programming for Optimized Path Planning of Flexible Robots*, in Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, pp. 2918–2923, 2004.

[87] K. D. BILIMORIA AND B. WIE, *Time-Optimal Three-Axis Reorientation of a Rigid Spacecraft*, Journal of Guidance, Control, and Dynamics, Vol. 16, No. 3, pp. 446–452, May–June, 1993.

[88] H. SEYWALD, R. R. KUMAR, AND S. M. DESHPANDE, *Genetic Algorithm Approach for Optimal Control Problems with Linearly Appearing Controls*, Journal of Guidance, Control, and Dynamics, Vol. 18, No. 1, pp. 177–182, January–February, 1995.

[89] *Optimization Toolbox for Use with MATLAB*, User's Guide, Version 2, The MathWorks, Inc., pp. 4-32–4-44, 1999–2000.

[90] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, San Diego, CA, Chapter 5, Section 2, pp. 167–168, 1981.

[91] D. G. LUENBERGER, *Linear and Nonlinear Programming*, 2nd Ed., Addison–Wesley, Reading, MA, Chapter 11, Section 3, pp. 326–330, 1989.

[92] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, San Diego, CA, Chapter 6, Section 1.1, p. 206, 1981.

[93] D. G. LUENBERGER, *Linear and Nonlinear Programming*, 2nd Ed., Addison–Wesley, Reading, MA, Chapter 14, Section 2, pp. 427–429, 1989.

[94] W. LINDORFER AND H. G. MOYER, *Application of a Low Thrust Trajectory Optimization Scheme to Planar Earth-Mars Transfer*, ARS Journal, Vol. 32, pp. 260–262, February, 1962.

[95] A. E. BRYSON AND Y.-C. HO, *Applied Optimal Control: Optimization, Estimation, and Control*, Hemisphere Publishing Corporation, Washington, Chapter 2, Section 2.5, pp. 66–69, 1975.

[96] H. G. MOYER AND G. PINKHAM, *Several Trajectory Optimization Techniques, Part* 2: *Application, Computing Methods in Optimization Problems*, A.V. Balakrishnan and L.W. Neustadt (Eds.), Academic Press, New York, 1964.

[97] A. BEMPORAD, T. J. TARN, AND N. XI, *Predictive Path Parameterization for Constrained Robot Control,* IEEE Transactions on Control Systems Technology, Vol. 7, No. 6, pp. 648–656, November, 1999.

[98] T. BALKAN, *A Dynamic Programming Approach to Optimal Control of Robotic Manipulators*, Mechanics Research Communications, Vol. 25, No. 2, pp. 225–230, 1998.

[99] J. BOBROW, S. DUBOWSKY, AND J. GIBSON, *Time-Optimal Control of Robotic Manipulators Along Specified Paths,* International Journal of Robotics Research, Vol. 4, No. 3, pp. 3–17, 1985.

[100] S. F. P. Saramago and V. Steffen, Jr., *Optimization of the Trajectory Planning of Robot Manipulators Taking into Account the Dynamics of the System*, Mechanism and Machine Theory, Vol. 33, No. 7, pp. 883–894, 1998.

[101] G. Bessonnet and J. P. Lallemand, *On the Optimization of Robotic Manipulator Trajectories with Bounded Joint Actuators or Joint Kinetic Loads Considered as Control Variables*, Journal of Dynamic Systems, Measurement, and Control, Vol. 116, pp. 819–826, December, 1994.

[102] A. Piazzi and A. Visioli, *Global Minimum-Jerk Trajectory Planning of Robot Manipulators*, IEEE Transactions on Industrial Electronics, Vol. 47, No. 1, pp. 140–149, February, 2000.

[103] O. Von Stryk and M. Schlemmer, *Optimal Control of the Industrial Robot Manutec R3*, in Computational Optimal Control, R. Bulirsch and D. Kraft (Eds.), Vol. 115 of the International Series of Numerical Mathematics, Birkhauser-Verlag, Basel, Switzerland, pp. 367–382, 1994.

[104] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, Philadelphia, 2001.

[105] P. I. Corke and B. Armstrong-Helouvry, *A Search for Consensus Among Model Parameters Reported for the PUMA 560 Robot*, Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1608–1613, San Diego, CA, May, 1994.

[106] P. I. Corke, *A Robotics Toolbox for MATLAB*, IEEE Robotics and Automation Magazine, Vol. 3, No. 1, pp. 24–32, 1996.

[107] J. J. Craig, *Introduction to Robotics*, Addison–Wesley, Reading, MA, 1989.

[108] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, *On-Line Computational Scheme for Mechanical Manipulators*, ASME Journal of Dynamic Systems, Measurement and Control, Vol. 102, pp. 69–76, 1980.

[109] P. I. Corke, *An Automated Symbolic and Numeric Procedure for Manipulator Rigid-Body Dynamic Significance Analysis and Simplification*, Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, MN, pp. 1018–1023, 1996.

[110] P. I. Corke, *Symbolic Algebra for Manipulator Dynamics*, CSIRO Division of Manufacturing Technology, Pinjarra Hills, Australia, pp. 1–35, August, 1996.

[111] P. I. Corke, *The Unimation Puma Servo System*, MTM-226, CSIRO Division of Manufacturing Technology, Locked bag 9, Preston, Australia, pp. 1–54, July, 1994.

[112] M. B. Leahy, Jr., K. P. Valavanis, and G. N. Saridis, *Evaluation of Dynamic Models for PUMA Robot Control*, IEEE Transactions on Robotics and Automation, Vol. 5, No. 2, pp. 242–245, April, 1989.

[113] M. B. Leahy, Jr., and G. N. Saridis, *Compensation of Industrial Manipulator Dynamics*, The International Journal of Robotics Research, Vol. 8, No. 4, pp. 73–84, August, 1989.

Index