Theory and Implementation of Numerical Methods Based on Runge-Kutta Integration for Solving Optimal Control Problems

by

Adam Lowell Schwartz

S.B. (Massachusetts Institute of Technology) 1989
S.M. (Massachusetts Institute of Technology) 1989

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering—
Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Elijah Polak, Chair
Professor James W. Demmel
Professor Shankar Sastry
Professor Andrew K. Packard

1996

Theory and Implementation of Numerical Methods Based on Runge-Kutta Integration for Solving Optimal Control Problems

Abstract

THEORY AND IMPLEMENTATION OF NUMERICAL METHODS BASED ON RUNGE-KUTTA INTEGRATION FOR SOLVING OPTIMAL CONTROL PROBLEMS

by

Adam Lowell Schwartz

Doctor of Philosophy in Electrical Engineering
University of California at Berkeley
Professor Elijah Polak, Chair

This dissertation presents theory and implementations of numerical methods for accurately and efficiently solving optimal control problems. The methods we consider are based on solving a sequence of discrete-time optimal control problems obtained using explicit, fixed step-size Runge-Kutta integration and finite-dimensional B-spline control parameterizations to discretize the optimal control problem under consideration. Other discretization methods such as Euler's method, collocation techniques, or numerical implementations, using variable step-size numerical integration, of specialized optimal control algorithms are less accurate and efficient than discretization by explicit, fixed step-size Runge-Kutta for many problems. This work presents the first theoretical foundation for Runge-Kutta discretization. The theory provides conditions on the Runge-Kutta parameters that ensure that the discrete-time optimal control problems are consistent approximations to the original problem.

Additionally, we derive a number of results which help in the efficient numerical implementation of this theory. These include methods for refining the discretization mesh, formulas for computing estimates of integration errors and errors of numerical solutions obtained for optimal control problems, and a method for dealing with oscillations that arise in the numerical solution of singular optimal control problems. These results are of great practical importance in solving optimal control problems.

We also present, and prove convergence results for, a family of numerical optimization algorithms for solving a class of optimization problems that arise from the discretization of optimal control problems with control bounds. This family of algorithms is based upon a projection operator and a decomposition of search directions into two parts: one part for the unconstrained subspace and another for the constrained subspace. This decomposition allows the correct active

constraint set to be rapidly identified and the rate of convergence properties associated with an appropriate unconstrained search direction, such as those produced by a limited memory quasi-Newton or conjugate-gradient method, to be realized for the constrained problem. The algorithm is extremely efficient and can readily solve problems involving thousands of decision variables.

The theory we have developed provides the foundation for our software package RIOTS. This is a group of programs and utilities, written mostly in C and designed as a toolbox for Matlab, that provides an interactive environment for solving a very broad class of optimal control problems. A manual describing the use and operation of RIOTS is included in this dissertation. We believe RIOTS to be one of the most accurate and efficient programs currently available for solving optimal control problems.

Professor Elijah Polak
Dissertation Committee Chair

For Mom and Dad

We are generally the better persuaded by the reasons we discover ourselves than by those given to us by others.
— Marcel Proust

You never work so hard as when you're not being paid for it.
— George Burns

There are three types of people in this world:
Those that are good at math—and those that aren't.

Acknowledgments

The work in this thesis would not have been possible without the invaluable discussions I have had with several individuals. These individuals, all of whom were very generous with their time, include Prof. Dimitri Bertsekas, Dr. John Betts, Prof. Larry Biegler, Prof. Carl de Boor, Prof. Asen Dontchev, Prof. Joseph Dunn, Prof. Roger Fletcher, Prof. William Hager, Dr. Craig Lawrence, Prof. Roger Sargent, Prof. Michael Saunders, Dr. Oskar Von Stryk, Prof. André Tits, Dr. Stephen Wright and the helpful engineers at the Mathworks. Also, for sharing with me their programming expertise, I wish to thank my fellow graduate students Steve Burgett and Raja Kadiyala. Two other fellow graduate students, Neil Getz and Shahram Shahruz, deserve mention for the enjoyable time I spent with them discussing and formulating ideas. Thanks also go to Prof. Ron Fearing for keeping me employed as an instructor for Signals and Systems.

For the gritty details of administration, the Cory Hall staff, particularly Dianna Bolt, Mary Byrnes, Chris Colbert, Tito Gatchalian, Heather Levien, Flora Oviedo, and Mary Stewart enormously simplified my life at Berkeley. There is no overstating the importance of their help.

I would like to reserve special acknowledgment for: Carlos Kirjner (my officemate with whom I spent most of my hours) for answering questions on topics ranging from functional analysis to topology to optimization; Prof. Shankar Sastry who provided access to the computer equipment I used for developing my software as well as encouragement and a willingness to become involved in a subject that is removed from his usual area of interest; Prof. James Demmel who sparked my interest in numerical integration and is responsible for my understanding of numerical integration methods; Prof. Andrew Packard, a colleague whose approach to academia is refreshing and stimulating—I have thouroughly enjoyed knowing and working with Andy; and most importantly, my advisor and mentor, Prof. Polak. The work described in this thesis is the result of my collaboration with Prof. Polak and any signs of excellence that may be contained herein are due to the high level of quality that he demanded of me. His insistence on perfection was relentless and often painful. But his commitment to quality will serve as a guide for the rest of my life. I am grateful for his deep involvement in my work.

Finally, I am glad to mention the people in my personal life that made the endless hours of work on this dissertation tolerable. These are my parents Stan and Helene, my brother John and his wife Carrie (and their brand-new daughter Rebecca), my sister Melissa, my grandparents Benjamin, Francis, Nathan, Pauline and Lillian, my pseudo-aunt Joann Lombardo, my various housemates over the years Mitch Berkson, Michael Cohn, John and Tomoko Ferguson, Scott Shenk, Dan Vassilovski, Colin Weeks, and my good friends Lawrence Candell, John Georges, Gary and Laura Grunbaum, Ealon Joelson, Alan Sbarra, and Jeff Steinhauer. I make special mention of my beautiful girlfriend Jessica Daniels who has been very patient, encouraging and loving. The support, in every form, provided to me by these people was indispensable.

Notation

Spaces and Elements

$\mathbb{R}^n$	Euclidean n-space
$\times_r \mathbb{R}^m$	Cartesian product of r copies of $\mathbb{R}^m$
$L^m_{\infty,2}[0,1]$	$(L^m_\infty[0,1], \langle\cdot,\cdot\rangle_{L^m_2[0,1]}, \|\cdot\|_{L^m_2[0,1]})$
L^i_N	finite dimensional subspace of $L^m_{\infty,2}[0,1]$, $i = 1,2$
$\bar{L}^i_N$	time samples of elements in L^i_N, $i = 1,2$
$L^{(\rho)}_N$	$L^{(\rho)}_N \subset L^1_N$, ρ-th order spline subspace
$\bar{L}^{(\rho)}_N$	spline coefficients of elements in $L^{(\rho)}_N$.
H_2	$\mathbb{R}^n \times L^m_2[0,1]$
$H_{\infty,2}$	$\mathbb{R}^n \times L^m_{\infty,2}[0,1] \subset H_2$
H_N	$\mathbb{R}^n \times L^1_N$ or $\mathbb{R}^n \times L^2_N$, $H_N \subset H_{\infty,2}$
$\bar{H}_N$	$\mathbb{R}^n \times \bar{L}^1_N$ or $\mathbb{R}^n \times \bar{L}^2_N$
$\bar{u}_k$	$(\bar{u}_{k,1}, \ldots, \bar{u}_{k,r}) \in \mathbb{R}^m \times \cdots \times \mathbb{R}^m$
$\bar{u}$	$(\bar{u}_0, \ldots, \bar{u}_{N-1}) \in \bar{L}_N$
η	$\eta = (\xi, u) \in H_{\infty,2}$
η_N	$\eta_N = (\xi, u_N) \in H_N$
$\bar{\eta}$	$\bar{\eta} = (\xi, \bar{u}) \in \bar{H}_N$
α	$(\alpha_1, \ldots, \alpha_{N+\rho-1}) \in \bar{L}^{(\rho)}_N$.

Functions

$\langle\cdot,\cdot\rangle_{\mathcal{H}}$	inner product in Hilbert space $\mathcal{H}$
$\|\cdot\|_{\mathcal{H}}$	norm in Hilbert space $\mathcal{H}$
$V_{\mathbf{A},N}$	$V_{\mathbf{A},N} : L^i_N \to \bar{L}^i_N$, $i = 1,2$
$W_{\mathbf{A},N}$	$W_{\mathbf{A},N} : H_N \to \bar{H}_N$, $W_{\mathbf{A},N}((\xi,u)) = (\xi, V_{\mathbf{A},N}(u))$
$S_{N,\rho}$	$S_{N,\rho} : L^{(\rho)}_N \to \bar{L}^{(\rho)}_N$
$\tau_{k,i}$	$t_k + c_i \Delta$
$u[\tau_{k,i}]$	value of control sample at $\tau_{k,i}$
$D\psi(\eta\,;h)$	directional derivative
$d_{\bar{u}}\bar{f}(\eta)$	derivative of $\bar{f}(\bar{\eta})$ with respect to the components of $\bar{u} = V_{\mathbf{A},n}(u)$
$d_\alpha \bar{f}(u)$	derivative of $\bar{f}(u)$ with respect to the components of $\alpha = S_{N,\rho}(u)$
F(x , w)	Right hand side of difference equation produced by RK discretization

Sets

$\mathbf{B}$	$\mathbf{B} \subset L^m_{\infty,2}[0,1]$ is the set on which all differential operators are defined.
$\mathbb{N}$	$\{\, 0, 1, 2, \ldots \,\}$
$\mathbf{N}$	$\{\, d^n \,\}_{n=1}^{\infty}$
$\mathcal{N}$	$\{\, 0, 1, 2, \ldots, N-1 \,\}$
$\mathbb{1}$	column vector of ones.
$B(v,\delta)$	$\{\, v' \in \mathbb{R}^m \mid \|v' - v\|_2 \le \delta \,\}$
$\mathbf{q}$	$\{\, 1, \ldots, q \,\}$
$\mathbf{t}_N$	$\mathbf{t}_N = \{\, t_k \,\}_{k=0}^{N-1}$ is the discretization mesh, or ...
$\mathbf{t}_N$	$\mathbf{t}_N = \{\, t_k \,\}_{k=-\rho+1}^{N+\rho-1}$ is a spline knot sequence
$\mathbf{A}$	Runge-Kutta parameters $\mathbf{A} = [c, A, b]$
I	$I = \{\, i_1, i_2, \ldots, i_r \,\} = \{\, i \mid c_j \ne c_i, j < i \,\}$
I_j	$I_j = \{\, i \mid c_i = c_{i_j}, i_j \in I \,\}$

Constraint Sets

$U \subset \mathbb{R}^m$	pointwise control constraint set
$\mathbf{U} \subset \mathbf{B}$	set of feasible controls
$\bar{\mathbf{U}}_N \subset \bar{\mathbf{L}}_N$	$\bar{u} \in \bar{\mathbf{U}}_N \implies \bar{u}^j_k \in U$
$\mathbf{U}_N$	$\mathbf{U}_N = V^{-1}_{\mathbf{A},N}(\bar{\mathbf{U}}_N) \subset L_{\infty,2}$
$\mathbf{H}$	$\mathbb{R}^n \times \mathbf{U} \subset H_{\infty,2}$
$\bar{\mathbf{H}}_N$	$\mathbb{R}^n \times \bar{\mathbf{U}}_N \subset \bar{H}_N$
$\mathbf{H}_N$	$\mathbb{R}^n \times V^{-1}_{\mathbf{A},N}(\bar{\mathbf{U}}_N)$
$\mathbf{U}^{(\rho)}_N$	$\mathbf{U}^{(\rho)}_N = \{\, u \in L^{(\rho)}_N \mid \alpha_k \in U \,\}$

Differential and Difference Equations

$x^\eta(t)$	solution at time t of differential equation given $\eta = (\xi, u)$: initial condition ξ and control input u
$\bar{x}^{\bar{\eta}}_k$	solution at time step k of difference equation, resulting from RK discretization, for $\bar{\eta} = (\xi, \bar{u})$: initial condition ξ and control samples $\bar{u}$
$\bar{x}^{\eta_N}_k$	$\bar{x}^{\eta_N}_k = \bar{x}^{\bar{\eta}}_k$ with $\bar{\eta} = W_{\mathbf{A},N}(\eta_N)$

Table of Contents

Chapter 1

INTRODUCTION

1.1 NUMERICAL METHODS FOR SOLVING OPTIMAL CONTROL PROBLEMS

Numerical methods for solving optimal control problems have evolved significantly over the past thirty-four years since Pontryagin and his students presented their celebrated maximum principle [1]. Most early methods were based on finding a solution that satisfied the maximum principle, or related necessary conditions, rather than attempting a direct minimization of the objective function (subject to constraints) of the optimal control problem. For this reason, methods using this approach are called indirect methods. Explanations of the indirect approach can be found in [2-6].

The main drawback to indirect methods is their extreme lack of robustness: the iterations of an indirect method must start close, sometimes very close, to a local solution in order to solve the two-point boundary value subproblems. Additionally, since first order optimality conditions are satisfied by maximizers and saddle points as well as minimizers, there is no reason, in general, to expect solutions obtained by indirect methods to be minimizers.

Both of these drawbacks of indirect methods are overcome by so-called direct methods. Direct methods obtain solutions through the direct minimization of the objective function (subject to constraints) of the optimal control problem. In this way the optimal control problem is treated as an infinite dimensional mathematical programming problem. There are two distinct approaches for dealing with the infinite dimensional aspect of these problems. The first approach develops specialized *conceptual* algorithms, and numerical implementations of these algorithms, for solving the mathematical programs. A conceptual algorithm is either a function space analog of a finite dimensional optimization algorithm or a finite dimensional algorithm (obtained by restricting the controls to a finite dimensional subspace of the control space) that requires infinite dimensional operations such as the solution of differential equations and integrals. An implementation of a conceptual algorithm accounts for errors that result when representing elements of an infinite dimensional functions space with finite dimensional approximations and the errors produced by the numerical methods used to perform infinite dimensional operations. There are many examples of conceptual algorithm for solving optimal control problem, some with and some without implementations [7-31].

The conceptual algorithm approach for solving optimal control problems has serious drawbacks. First, customized software for controlling the errors produced in the numerical approximations of infinite dimensional functions and operations must be incorporated into the implementation of a conceptual algorithm. More seriously, because function evaluations are performed only approximately the function gradients used by mathematical programming software will not be coordinated with those same functions. That is, the gradients will only be approximations to the derivatives of the functions. This mean, for example, that it is possible that the negative of a function gradient may not be a direction of descent for the approximation of that function. This problem is exacerbated as a stationary point is approached. A related problem is that a certain amount of precision in the function evaluations is required to ensure successful line searches. Together, these facts mean that, in practice, high precision in numerical operations such as integration is required even in early iterations of the optimization procedure. Since high precision in early iterations does not contribute to the accuracy of the final solution, this requirement makes the implementation of conceptual algorithm inefficient for most problems.

An alternate direct method approach is one which we term consistent approximations. In the consistent approximations approach, the optimal control is obtained by solving a sequence of finite dimensional, discrete-time optimal control problems† that are increasingly accurate representations of the original, continuous-time problem. The solutions of the approximating, discrete-time optimal control problems can be obtained using standard, finite dimensional mathematical programming techniques. Under suitable conditions, solutions of the approximating problems converge to a solution of the original problem. In this sense, such discrete-time optimal control problems are called *consistent approximations* to the original problem.

The first rigorous developments of algorithms based on solving finite dimensional approximating problems used Euler's method and piecewise constant control representations (which results in a finite dimensional control parameterization) to discretize the original problem (see the introduction to Chapter 2 for references). From a numerical analyst's point of view, the choice of Euler's method may seem strange since Euler's method is an extremely inefficient method for solving differential equations. But there are reasons for choosing Euler's method as a

†Speaking more accurately, the discretized problems need not be a discrete-time optimal control problems. For instance, if the controls are represented as finite dimensional B-splines, the decision variables of the discretized problems are spline coefficients, not control values at discrete times.

discretization procedure for optimal control problems. First, up until this work, there has been no theory supporting the use of iterative higher-order integration methods in the construction of consistent approximations. Second, only recently has it been demonstrated that there can be an advantage to using higher-order discretization methods for solving optimal control problems. The use of higher-order discretization methods for solving optimal control problems remains an active area of research. It is difficult to demonstrate a theoretical advantage to using higher order methods rather than Euler's methods when solving general, constrained optimal control problems. However, many optimal control problems that arise in practice are, in fact, solved much more efficiently with higher-order methods.

Within the category of direct methods based on the idea of consistent approximations, there is a further sub-classification that helps to establish where our work stands in relation to other methods. This sub-classification specifies how the discretization of an optimal control problem into a finite dimensional approximating problem is accomplished: via collocation (or more generally, a Galerkin approximation) or via iterative integration. Currently, the most popular discretization scheme is based on collocation and methods similar in spirit to collocation [16-18,32-41]. In collocation methods, the system of differential equations describing the dynamic system is replaced by a system of equations that represent collocation conditions to be satisfied at a finite number of time points. The resulting mathematical program involves not only the control parameters as decision variables but also a large number of additional variables that represents the value of state variables at mesh points. Collocation schemes offer several advantages over iterative integration schemes:

1. It is easier to prove convergence and order of convergence results.
2. Some results for the order of error, as a function of the discretization level, between solutions of the approximating problems and solutions of the original problem (namely, for unconstrained optimal control problems) are superior to other schemes [36].
3. Certain difficulties inherent to some optimal control problems, such as stiff differential equations and highly unstable dynamics, are greatly mitigated in collocation schemes.
4. Simple bounds on state variables translate into simple bounds on the decision variables of the mathematical program.
5. Function gradients are easier to compute since they do not require the derivative of the state with respect to the controls.

However, relative to iterative integration, collocation schemes have serious drawbacks as well:

1. The approximating problems are significantly larger at a given discretization level due to the inclusion of state variables as decision parameters.

2. The approximating problems are significantly harder to solve because of the addition of a large number of (nonlinear) equality constraints that represent the collocation conditions.
3. The accuracy of solutions obtained by solving the approximating problems can be somewhat inaccurate due to the presence of the collocation constraints.
4. If the numerical algorithm for solving the approximating problems is terminated prematurely the solution may not be useful since the collocation conditions will not be satisfied.

Because of these disadvantages, solutions obtained using a collocation scheme often have to be subsequently refined using an indirect solution method [4].

The work in this thesis is based on discretizing optimal control problems using explicit, fixed step-size Runge-Kutta integration techniques. The advantage of this scheme over collocation schemes is that the approximating problems that result can be solved very efficiently and accurately. On the other hand, some of the features listed above as advantages associated with collocation are sacrificed. Specifically, convergence results are more difficult to prove for the Runge-Kutta method and, in the case of unconstrained problems, the order of error for solution of the approximating problems is lower (see [42] and Proposition 4.6.2). Also, it is quite convenient from a programming point of view that state variable bounds become bounds on the decision variables of the mathematical program (advantage 4). However, this advantage is more than offset by the addition of the system of equality constraints representing the collocation conditions. Finally, the difficulties of solving problems with highly unstable dynamics can also be handled when using explicit Runge-Kutta integration. A method for doing so is discussed in the Chapter 6.

As far as we know, the work reported in this thesis represents the only work on consistent approximation schemes using Runge-Kutta integration. Thus, at the very least, our work complements the work of other authors that deal with collocation schemes. But further, we believe that our approach has significant theoretical and practical advantages that will make it, with sufficient development, a leading approach to solving optimal control problems.

1.2 CONTRIBUTIONS TO THE STATE-OF-THE-ART

The original goal of this research was simply to develop a fast and accurate software package for solving optimal control problems using explicit Runge-Kutta integration. In the process of writing this software we have, by necessity, developed a strong theoretical foundation for our discretization approach as well constructing several new algorithms for various types of computation. The following is a concise summary of the contributions provided by this work to the state-of-the-art in numerical methods for solving optimal control problems:

- Provides the first convergence analysis and implementation theory for discretization methods based on Runge-Kutta integration. Specifically, conditions on the parameters of the Runge-Kutta method are presented that ensure, for instance, that stationary points of the discretized problems can only converge to stationary points of the original problem.
- Derives a non-Euclidean metric needed for the finite-dimensional optimization of the approximating problems and presents a coordinate transformation which allows a Euclidean metric to be used. Without this metric, serious ill-conditioning can be introduced into the discretized problem.
- Improves upon the previously known bound for the error in the solution of the approximating problems as a function of the discretization level for RK4 (the most common fourth-order Runge-Kutta integration method) when solving unconstrained optimal control problems. This result, along with the already known bounds for a first, second and third order Runge-Kutta method are extended to the case where the finite dimensional controls are represented by splines.
- Presents a new, very efficient and robust numerical algorithm, based on the projected Newton method of Bertsekas, for solving a class of mathematical programming problems with simple bounds on the decision variables.
- Develops a new method for computing accurate estimates of the error between the solutions computed for the approximating problems and solutions of the original problem. This estimate does not require *a priori* knowledge of error bounds and works for problems with state and control constraints.
- Develops a completely new method for numerically solving singular optimal control problems. This method is designed to eliminate undesirable oscillations that occur in numerical solutions of singular control problems.
- Presents our software package called RIOTS, based on the theory in contained in this thesis, for solving optimal control problems. Although there are many improvements that can be made to RIOTS, it is already one of the fastest, most accurate and easiest to use programs available for solving optimal control problems.

1.3 DISSERTATION OUTLINE

The organization of this dissertation follows a progression leading from basic theoretical foundations of discretizing optimal control problems to the implementation of a software package for solving a large class of optimal control problems. The theoretical foundation is presented in Chapter 2. Chapter 2 begins with a discussion of the concept of consistent approximations as defined by Polak [43]. Polak's definition of consistent approximations extends earlier definitions, namely that of Daniels [44], that were concerned only with convergence of global solutions of the approximating problems to global solutions of the original problem. The earlier definitions were therefore of limited use since optimization algorithms compute stationary points, not global solutions. Polak's definition of consistency deals with stationary points and local minima as well as global solutions. The theory of consistent approximations is used to develop a framework for discretizing optimal control problems with Runge-Kutta integration. The main results in Chapter 2 show that the approximating problems are consistent approximations to the original optimal control problem if the Runge-Kutta method satisfies certain conditions in addition to the standard conditions needed for consistent integration of differential equations. Once the consistency result is established, the convergence results provided by the theory of consistent approximations can be invoked. In the process of constructing consistent approximations based on Runge-Kutta discretization, we show that a non-Euclidean inner-product and norm, depending on the basis used for the finite dimensional control subspaces, must be used for the space of control coefficients upon which the finite dimensional mathematical programs that results from the discretization are defined. Without this non-Euclidean metric, serious ill-conditioning can result. We also show how a coordinate transformation can be used to eliminate the need for the non-Euclidean inner-product and norm. The results are then extended to control representations based on splines.

In Chapter 3, we present a very efficient and robust optimization algorithm for solving finite dimensional mathematical programming problems that include simple bounds on the decision variables. Such problems arise from the discretization of optimal control problems with control bounds. In Chapter 4, other important numerical issues are addressed. These issues include *(i)* obtaining bounds on the error of solutions to the approximating problems based on spline controls, *(ii)* developing heuristics for selecting the integration order and control representation order, *(iii)* providing methods for refining the discretization mesh, *(iv)* providing a computable error estimate for solutions of the approximating problems and *(v)* dealing with the numerical difficulties that arise when solving singular optimal control problems. We also present numerical data to support our claim that implementations of conceptual algorithms are inefficient compared to the consistent approximations approach to solving optimal control problems.

The next chapter, Chapter 5, contains the user's manual for RIOTS. RIOTS is our software package, developed as a toolbox for Matlab[†], for solving a very broad class of optimal control problems. This class includes problems with multiple objective functions, fixed or free final time problems, problems with variable initial conditions and problems with control bounds, endpoint equality and inequality constraints, and trajectory constraints. The user's manual includes a mathematical description of the class of problems that can be handled, a series of sample sessions with RIOTS, a complete reference guide for the programs in RIOTS, explanations of important implementation details, and instructions for installing RIOTS. Chapter 6, presents our conclusions and ideas for future research. Finally, there are two appendices. The first contains the proofs of some of the results in Chapter 2 and the second describes some example optimal control problems that we use, primarily in Chapter 4, for numerical experiments.

[†] Matlab is a scientific computation and visualization program designed by The MathWorks, Inc.

Chapter 2

CONSISTENT APPROXIMATIONS FOR OPTIMAL CONTROL PROBLEMS BASED ON RUNGE-KUTTA INTEGRATION

2.1 INTRODUCTION

In this Chapter, we establish the theoretical foundation of our method for numerically solving optimal control problems. Specifically, we consider approximations to constrained optimal control problems that result from numerical solving the differential equations describing the system dynamics using Runge-Kutta integration. We show that there is a class of higher order, explicit Runge-Kutta (RK) methods that provide *consistent approximations* to the original problem, with consistency defined according to [43]. Consequently, we are assured that stationary points of the approximating problems converge to stationary points of the original problem, and that global solutions (or strict local solutions with a non-vanishing radius of attraction) of the approximating problems converge to global (or local) solutions of the original problem, as the step-size of the RK method is decreased.

The theory of consistent approximations introduced in [43] requires that the approximating problems be defined on finite dimensional subspaces of the control space to which RK methods can be extended. The selection of the control subspaces affects both the accuracy of numerical integration and the accuracy with which solutions of the original problem are approximated. Once the approximating problems are defined, their numerical solution is carried out by means of standard mathematical programming algorithms in the space of coefficients associated with the bases defining the control subspaces. We construct two such families of control subspaces. The "natural" basis functions for one family are piecewise polynomial functions, and for the other, piecewise constant functions. Also, B-splines provide a basis for a subspace of piecewise polynomial functions. None of these sets of basis functions is orthonormal. Hence, to preserve the L_2 inner product and norm used in the control subspace, a non-Euclidean inner product and norm must be used in the associated space of coefficients. Failing to do so introduces a "changed metric'' effect that can adversely affect the performance of algorithms. The possible severity of this phenomenon is demonstrated by our computational results in Section 6. To remove the need to modify nonlinear programming software written for problems defined on a Euclidean space, we introduce coordinate transformations that change our original bases in the control space to an orthonormal set and change the associated coefficient space to a Euclidean space.

Daniel [44] presents one of the first attempts to characterize, in a general framework, consistency of approximations to an optimization problem as well as an application of this framework to approximations of optimal control problems obtained using the Euler integration formula. It can be shown that Daniel's conditions for consistency imply epiconvergence [45,46], *i.e.*, the convergence, in the Kuratowski sense [47], of the constrained epigraphs of the approximating problems to the constrained epigraph of the original problem. Epiconvergence ensures convergence of the global minimizers (or strict local minimizers with a non-vanishing radius of attraction) of the approximating problems to global (or local minimizers) of the original problem.

Polak, in [43], characterizes first order optimality conditions in terms of zeros of *optimality functions*. To define consistency of approximations, he augments the requirement of epiconvergence of the approximating problems with a related requirement for their optimality functions. As a result, consistency, in the Polak sense, ensures convergence of global (local) solutions, and stationary points, of the approximating problems to global (local) solutions, and stationary points, of the original problem. Furthermore, the Polak definition of consistency indirectly imposes the requirement that the mathematical characterization of the constraints of the approximating problems satisfy certain congruence conditions, and that derivatives of the approximating problem functions converge to those of the original problem. In addition to a definition of consistency, we find in [43] diagonalization strategies, in the form of master algorithms, that call nonlinear programming algorithms as subroutines. These algorithms enable one to efficiently obtain an approximate, numerical ``solution'' to an original infinite dimensional problem.

With the exception of [44] and [43], the analysis of the approximating properties of numerical integration techniques (see, *e.g.*, [43,48-56]) in optimal control is not carried in the framework of a general theory†. Convergence of global solutions, or in some cases, of stationary points, of approximating problems obtained using Euler integration to those of the original problem was established in [43,44,48-50,53-55]. Of these, perhaps the most extensive treatment can be found in [54]. The rate of convergence of stationary points of approximating problems, obtained from discretization of unconstrained optimal control problems using a class of RK methods, to those of the original problem was explored in [42].

† This is also true for collocation techniques (see, *e.g.*, [4,18,33,35,36]).

Organization. This chapter is organized as follows. Section 2 summarizes the theory of consistent approximations. Section 3 defines the optimal control problem and develops an optimality function for it. In section 4 the approximating problems are constructed and epiconvergence of the approximating problems is proved. In section 5, optimality functions for the approximating problems are derived and are shown to hypoconverge to the optimality function for the original problem. This completes the proof that the approximating problems are consistent approximations to the original problem. Section 6 introduces a transformation which defines orthonormal bases for the control subspaces and presents a rate of convergence result for the most commonly used RK method, RK4. Some numerical results are also included. Finally, in Section 7 the results are extended to control subspaces based on splines.

2.2 THEORY OF CONSISTENT APPROXIMATIONS

Let $\mathcal{H}$ be a normed linear space and $\mathbf{B} \subset \mathcal{H}$ a convex set and consider the problem

$$\mathbf{P} \qquad \min_{\eta \in \mathbf{F}} \psi(\eta) \tag{2.1a}$$

where $\psi : \mathbf{B} \to \mathbb{R}$ is (at least) lower semi-continuous, and $\mathbf{F} \subset \mathbf{B}$ is the feasible set. Next, let $\mathbb{N} \doteq \{ 1, 2, 3, \cdots \}$, let $\mathbf{N}$ be an infinite subset of $\mathbb{N}$, and let $\{ \mathcal{H}_N \}_{N \in \mathbf{N}}$ be a family of finite dimensional subspaces of $\mathcal{H}$ such that $\mathcal{H}_{N_1} \subset \mathcal{H}_{N_2}$, for all $N_1, N_2 \in \mathbf{N}$ such that $N_1 < N_2$. Now consider a family of approximating problems

$$\mathbf{P}_N \qquad \min_{\eta \in \mathbf{F}_N} \psi_N(\eta)\,, \quad N \in \mathbf{N}\,, \tag{2.1b}$$

where $\psi_N : \mathcal{H}_N \to \mathbb{R}$ is (at least) lower semi-continuous, and $\mathbf{F}_N \subset \mathcal{H}_N \cap \mathbf{B}$.

In [43] we find a characterization of the consistency of the approximating problems $\mathbf{P}_N$, in terms of two concepts. The first is epiconvergence of the $\mathbf{P}_N$ to $\mathbf{P}$ [45] which can be shown to be equivalent to Kuratowski convergence [47] of the restricted epigraphs of the cost functions of the approximating problems to the restricted epigraph of the original problem. Epiconvergence does not involve derivatives of the cost function nor the specific description of the constraint sets, hence it is a kind of ‘‘zero-order’’ property. The second concept consists of the characterization of stationary points as zeros of an ‘‘optimality function’’ and a kind of upper semi-continuity property of the optimality functions of the approximating problems. Optimality functions do depend on derivatives and the specific description of the constraint set, hence they add important first-order and structural information.

Definition 2.1. We will say that the problems in the family $\{ \mathbf{P}_N \}_{N \in \mathbf{N}}$ *converge epigraphically* (or *epiconverge*) to $\mathbf{P}$ ($\mathbf{P}_N \overset{\text{Epi}}{\to} \mathbf{P}$) if

(a) for every $\eta \in \mathbf{F}$, there exists a sequence $\{ \eta_N \}_{N \in \mathbf{N}}$, with $\eta_N \in \mathbf{F}_N$, such that $\eta_N \to \eta$ and $\overline{\lim}\, \psi_N(\eta_N) \le \psi(\eta)$;

(b) for every infinite sequence $\{ \eta_N \}_{N \in K}$, $K \subset \mathbf{N}$, satisfying $\eta_N \in \mathbf{F}_N$ for all $N \in K$ and $\eta_N \to^K \eta$, we have that $\eta \in \mathbf{F}$ and $\underline{\lim}_{N \in K}\, \psi_N(\eta_N) \ge \psi(\eta)$. □

There are two subsets involved in our formulation of this definition. The subset $\mathbf{N}$ is used to provide nesting of the finite dimensional subspaces $\mathcal{H}_N$. The subset $K \subset \mathbf{N}$ is required so that Definition 2.1 is equivalent to Kuratowski convergence. This is because, not only is the sequence $\{ \eta_N \}$ parameterized by N, but so are the problems in the sequence $\{ \mathbf{P}_N \}$.

In [43,45,46] we find the following result:

Theorem 2.2. Suppose that $\mathbf{P}_N \overset{\text{Epi}}{\to} \mathbf{P}$. *(a)* If, for $N \in \mathbf{N}$, $\hat{\eta}_N$ is a global minimizer of $\mathbf{P}_N$, and $\hat{\eta}$ is any accumulation point of the sequence $\{ \hat{\eta}_N \}_{N \in \mathbf{N}}$, then $\hat{\eta}$ is a global minimizer of $\mathbf{P}$; *(b)* if, for $N \in \mathbf{N}$, $\hat{\eta}_N$ is a strict local minimizer of $\mathbf{P}_N$ whose radius of attraction is bounded away from zero, and $\hat{\eta}$ is any accumulation point of the sequence $\{ \hat{\eta}_N \}_{N \in \mathbf{N}}$, then $\hat{\eta}$ is a local minimizer of $\mathbf{P}$. □

Epigraphical convergence does not eliminate the possibility of stationary points of $\mathbf{P}_N$ converging to a non-stationary point of $\mathbf{P}$: a most inconvenient outcome from a numerical optimization point of view. For example, let $\mathcal{H} = \mathbb{R}^2$ with $\eta = (x, y)$, and let $f(\eta) = f_N(\eta) = (x-2)^2$, $N \in \mathbb{N}$. Choose

$$\mathbf{F} \doteq \{ (x, y) \in \mathbb{R}^2 \mid x^2 + y^2 - 2 \le 0 \}, \tag{2.2a}$$

$$\mathbf{F}_N \doteq \{ (x, y) \in \mathbb{R}^2 \mid (x-y)^2(x^2 + y^2 - 2) \le 0,\ x^2 + y^2 \le 2 + 1/N \},\ N \in \mathbb{N}. \tag{2.2b}$$

Then we see that $\mathbf{P}_N \overset{\text{Epi}}{\to} \mathbf{P}$. Nevertheless, the point $(1, 1)$ is feasible and satisfies the F. John optimality condition for all $\mathbf{P}_N$, but it is not a stationary point for the problem $\mathbf{P}$ (see Figure 2.1). The reason for this is an incompatibility of the constraint sets $\mathbf{F}_N$ with the constraint set $\mathbf{F}$, which shows up only at the level of optimality conditions. Hypotheses precluding this pathology, at least for first order non-stationary points, were introduced in [43] using optimality functions as a tool for ensuring a kind of "first order" approximation result that implicitly enforces convergence of derivatives and restricts the forms chosen for the description of the sets $\mathbf{F}$ and $\mathbf{F}_N$.

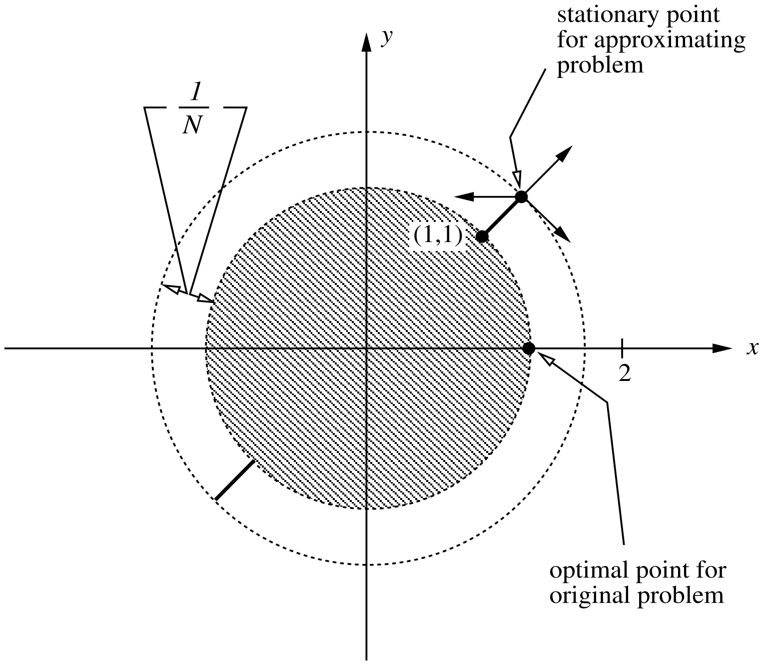


Fig. 2.1: Graph of the feasible regions for the approximating problems showing that the stationary points for the approximating problems converge to the point (1,1) which is a non-stationary point for the original problem. The arrows around (1,1) indicate the gradients (translated from the origin) for the two constraint functions and the objective function.

Definition 2.3. We will say that a function $\theta : \mathbf{B} \to \mathbb{R}$ is an *optimality function* for $\mathbf{P}$ if *(i)* $\theta(\cdot)$ is (at least) upper semi-continuous, *(ii)* $\theta(\eta) \le 0$ for all $\eta \in \mathbf{B}$, and *(iii)* for $\hat{\eta} \in \mathbf{F}$, $\theta(\hat{\eta}) = 0$ if $\hat{\eta}$ is a local minimizer for $\mathbf{P}$. Similarly, we will say that a function $\theta_N : H_N \to \mathbb{R}$ is an *optimality function* for $\mathbf{P}_N$ if *(i)* $\theta_N(\cdot)$ is (at least) upper semi-continuous, *(ii)* $\theta_N(\eta_N) \le 0$ for all $\eta_N \in H_N$, and *(iii)* if $\hat{\eta}_N \in \mathbf{F}_N$ is a local minimizer for $\mathbf{P}_N$ then $\theta_N(\hat{\eta}_N) = 0$ □

Definition 2.4. Consider the problems $\mathbf{P}$, $\mathbf{P}_N$, defined in (2.1a,b). Let $\theta(\cdot)$, $\theta_N(\cdot)$, $N \in \mathbf{N}$, be optimality functions for $\mathbf{P}$, $\mathbf{P}_N$, respectively. We will say that the pairs $(\mathbf{P}_N, \theta_N)$, in the sequence $\{ (\mathbf{P}_N, \theta_N) \}_{N \in \mathbf{N}}$ are *consistent approximations* to the pair $(\mathbf{P}, \theta)$, if *(i)* $\mathbf{P}_N \xrightarrow{\text{Epi}} \mathbf{P}$, and *(ii)* for any sequence $\{ \eta_N \}_{N \in K}$, $K \subset \mathbf{N}$, with $\eta_N \in \mathbf{F}_N$ for all $N \in K$, such that $\eta_N \to^K \eta$, the optimality functions of the approximating problems satisfy the condition

$$\overline{\lim}\, \theta_N(\eta_N) \le \theta(\eta) \,. \tag{2.3}$$

□

Note that part *(ii)* of Definition 2.4 rules out the possibility of stationary points (points such that $\theta_N(\eta_N) = 0$) for the approximating problems converging to non-stationary points of the original problem. In the sequel, we will prove a stronger condition than is required by Definition 2.4, namely, Kuratowski convergence of the hypographs of $\theta_N(\cdot)$ to the hypograph of $\theta(\cdot)$ (that is,

$-\theta_N \xrightarrow{\text{Epi}} -\theta$).

In addition to the characterization of consistency, the theory of consistent approximations in [43] includes various master algorithm models for efficiently solving problems such as **P**. Given a level of discretization defined by N, the master algorithms construct an approximating problem $\mathbf{P}_N$, execute a nonlinear programming or discrete-time optimal control algorithm as a subroutine for a certain number of iterations on $\mathbf{P}_N$, and then increase N. Then the process is repeated. For specific examples, see [55,57].

2.2.1. Overview of the construction of consistent approximations for optimal control problems.

In the remaining sections of this chapter we proceed to construct approximating problems, based on Runge-Kutta integration, to a general class of optimal control problems and show that they are consistent approximations. To guide the reader, we provide here an outline of the development, in a slightly re-arranged order, for a simple class of unconstrained optimal control problems with a smooth objective function.

We start by defining the optimal control problem. In this overview we will just consider unconstrained problems with fixed initial conditions of the form

P $$\min_{u \in \mathbf{U}} f(u)$$

where $f(u) \in \mathbb{R}$ is the objective function defined by

$$f(u) \doteq \zeta(x^u(1)) \tag{2.5a}$$

and $x^u(t) \in \mathbb{R}^n$ is the solution of the system of differential equation

$$\dot{x} = h(x,u)\ ,\ \ t \in [0,1]\ ;\ \ \ x(0) = \xi\ . \tag{2.5b}$$

Hence, the objective is a function of the final state $x^u(1)$ which depends on the control $u \in \mathbf{U}$, $u(t) \in \mathbb{R}^m$. Note that other forms of optimal control problems such as the Bolza and Lagrange forms can be converted into this form.

Problem **P** is defined over the feasible set **U** of controls. In the sequel, we will allow **U** to include control constraints but here we will assume that it does not. The choice of **U** is complicated by the fact that while standard optimality conditions for **P** are expressed in the L_2-norm, $f(\cdot)$ is differentiable in $L_\infty[0,1]$ but not in $L_2[0,1]$. To overcome this difficulty, we define the pre-Hilbert space

$$L^m_{\infty,2}[0,1] \doteq (\ L^m_\infty[0,1]\ \ ,\ \ \langle \cdot , \cdot \rangle_2\ \ ,\ \ \| \cdot \|_2\ ) \tag{2.6}$$

which consists of elements of $L^m_\infty[0,1]$ but is endowed with the L_2 inner-product and norm. Then we allow $\mathbf{U} \subset L^m_{\infty,2}[0,1]$. Since $\mathbf{P}$ is an unconstrained problem, we can choose for its optimality function

$$\theta(u) = -\|\nabla f(u)\|^2 \tag{2.7}$$

because $-\|\nabla f(u)\|^2$ is continuous, negative valued and zero at $\hat{u}$ if $\hat{u}$ is a local minimizer of $\mathbf{P}$ since a first order necessary condition for optimality of $\hat{u}$ is $\nabla f(\hat{u}) = 0$ (that is, $\nabla f(\hat{u})(t) = 0$ for $t \in [0,1]$ $a.e.$). The gradient $\nabla f(u)$ can be computed according to standard formulas which are presented in Section 3.

The next step is to construct the approximating problems through some discretization procedure. Our method involves *(i)* integrating the differential equation numerical using Runge-Kutta (RK) integration and *(ii)* replacing the infinite dimensional control set $\mathbf{U}$ with a finite dimensional approximating set $\mathbf{U}_N$. A RK integration method is specified by a set of parameters collectively called the Butcher array. The Butcher array, $\mathbf{A} = [c, A, b^T]$, consists of three sets of parameters. The c parameters relate to sampling instances where the RK integration evaluates the right-hand side of (2.5b) and the b parameters are relative weights assigned to each of these evaluations. The A parameters affect the order of convergence of the RK method but do not play a role in the first-order convergence analysis. The integration proceeds to compute approximations of the state $\bar{x}_k \approx x^u(t_k)$ at the discrete time points $\{ t_k \}_{k=0}^{N}$ according to

$$\bar{x}_{k+1} = F(\bar{x}_k, \bar{u}_k) \doteq \bar{x}_k + \Delta \sum_{i=1}^{s} b_i K_i \ ; \quad x_0 = \xi \tag{2.8a}$$

with

$$K_i = h(\bar{x}_i + \Delta \sum_{j=1}^{i-1} a_{ij} K_j, \bar{u}_{k,i}) \tag{2.8b}$$

$$\bar{u}_{k,i} = u(t_k + c_i \Delta) \tag{2.8c}$$

where s is the number of stages in the RK method, $\Delta = t_{k+1} - t_k = 1/N$ (we assume a uniform mesh in this Chapter for simplicity) and $c = (c_1, \ldots, c_s)$ and $b = (b_1, \ldots, b_s)$ are parameters from the Butcher array. Equation (2.8c) is a simplification of how we later define $\bar{u}_{k,i}$ to take care of the possibility that $u(\cdot)$ is discontinuous at $t_k + c_i\Delta$. The quantities $\bar{u}_{k,i} \in \mathbb{R}^m$ are called *control samples* and relate to functions $u \in \mathbf{U}_N$ through a map $V_{\mathbf{A},N}$ which depends on the Butcher array $\mathbf{A}$ and the discretization level N. This map defines how the RK method integrates over controls in $\mathbf{U}_N$.

From (2.8b), we see that the RK integration depends on the control samples $\bar{u}_{k,i}$ for $k = 0, \ldots, N-1$, $i = 1, \ldots, s$. We must take some care if control samples occur at identical sampling times, but we will ignore this possibility for now. The control samples are organized as follows:

$$\bar{u}_k = (\bar{u}_{k,1}, \ldots, \bar{u}_{k,s}) \in \bigtimes_{s} \mathbb{R}^m \tag{2.9a}$$

$$\bar{u} = (\bar{u}_0, \ldots, \bar{u}_{N-1}) \in \bigtimes_{N} \bigtimes_{s} \mathbb{R}^m \,. \tag{2.9b}$$

In other words, the collection of control samples used by the RK method is denoted by $\bar{u}$ which has N components, $\bar{u}_k$, each of which contains the control samples used by the RK method over one step. When we use $\bar{u}$ in algebraic expressions, we will be treating it as the $m \times Ns$ matrix

$$\bar{u} = [\ \bar{u}_{0,1} \cdots \bar{u}_{0,s} \ \cdots \ \bar{u}_{N-1,1} \cdots \bar{u}_{N-1,s}\ ]\,. \tag{2.9c}$$

If $m = 1$ then $\bar{u}$ is just a row vector (not a column vector). The space of control samples is

$$\bar{L}_N = \Big(\bigtimes_{N} \bigtimes_{s} \mathbb{R}^m \ ,\ \langle \cdot, \cdot \rangle_{\bar{L}_N} \ ,\ \| \cdot \|_{\bar{L}_N} \Big)\,. \tag{2.9c}$$

We will specify inner-product and norm on $\bar{L}_N$ in a moment. To indicate its dependence on u, we will write the solution to (2.8a) as $\bar{x}_k^{\bar{u}}$.

The next step is to choose finite dimensional subspaces $L_N \subset L_{\infty,2}^m[0,1]$. These subspaces can be defined in many ways. In Section 4 we define two representations for L_N, one based on piecewise polynomials and the other based on piecewise constants. We define a third representation based on splines in Section 7. Given a definition for L_N, we relate functions $u \in L_N$ to their control samples $\bar{u} \in \bar{L}_N$ via the bijective map

$$V_{\mathbf{A},N} : L_N \to \bar{L}_{N.} \tag{2.9d}$$

Essentially, this map is defined in the following way: for each $u \in L_N$, $\bar{u} = V_{\mathbf{A},N}(u)$ is given by $\bar{u}_{k,i} = u(t_k + c_i)$. This is somewhat different when dealing with splines. However, in the sequel we will account for the possibilities mentioned above that *(i)* $u(\cdot)$ is discontinuous and *(ii)* some of the control sample occur at the same sampling times.

Now we can define the approximating problems:

$\mathbf{P}_N$
$$\min_{u \in \mathbf{U}_N} f_N(u)$$

$$f_N(u) \doteq \zeta\big(\bar{x}_N^{V_{\mathbf{A},N}(u)}\big) \tag{2.10}$$

where $\mathbf{U}_N \subset L_N \cap \mathbf{U}$. The reason we do not write $\mathbf{U}_N = L_N \cap \mathbf{U}$ is because we might have to add additional constraint on $\mathbf{U}_N$, depending on how L_N is defined, in order to prove consistency. In order to compute solutions of $\mathbf{P}_N$ using a computer we will actually solve a mathematical program involving the control samples $\bar{u} \in \bar{L}_N$. In order for an optimization program working in the space $\bar{L}_N$ to be equivalent to an optimization program working in the function space L_N, we need to define the inner-product and norm on $\bar{L}_N$ so that, with $\bar{u}, \bar{v} \in \bar{L}_N$ and $u = V_{\mathbf{A},N}^{-1}(\bar{u})$, $v = V_{\mathbf{A},N}^{-1}(\bar{v})$,

$$\langle \bar{u}, \bar{v} \rangle_{\bar{L}_N} = \langle u, v \rangle_2 \ \text{ and } \ \|\bar{u}\|_{\bar{L}_N} = \|u\|_2 \ . \tag{2.11}$$

Equation (2.11) is enough to define $\langle \cdot, \cdot \rangle_{\bar{L}_N}$ and $\| \cdot \|_{\bar{L}_N}$; we give specific formulas in Section 4 and Section 7. With the metric defined in this way, $V_{\mathbf{A},N}$ becomes an isometric isomorphism between L_N and $\bar{L}_N$. Hence, operations in one of the space are equivalent to the same operations in the other. Besides establishing this correspondence of operations, the definition of the metric on $\bar{L}_N$ is important from a purely computational point of view because it can prevent ill-conditioning in the mathematical program used to solve $\mathbf{P}_N$. It is important to note that the metric on $\bar{L}_N$ depends only on the basis for L_N, not on the optimal control problem to be solved.

The final step in the construction of the approximating problems is to define their optimality functions. Following the form of the optimality function $\theta(\cdot)$ for $\mathbf{P}$ we choose

$$\theta_N(u) = -\|\nabla f_N(u)\|_2^2 \ . \tag{2.12a}$$

The computation of $\nabla f_N(u)$ is non-standard because the gradient is defined relative to the space L_N which we define. We will show that, for $u \in L_N$,

$$\nabla f_N(u) = \left[\frac{d}{d\bar{u}} \bar{f}_N(V_{\mathbf{A},N}(u)) \right] \mathbf{M}_N^{-1} \tag{2.12b}$$

where $d\bar{f}_N(\bar{u}) / d\bar{u}$ is the standard discrete-time derivative of $\bar{f}_N(\bar{u})$, which can be computed using formulas similar to those for Euler's method, and $\mathbf{M}_N$ is a positive-definite matrix that depends only on the definition of L_N. This formula is slightly different for splines.

2.3 DEFINITION OF OPTIMAL CONTROL PROBLEM

We will consider optimal control problems with dynamics described by ordinary differential equations of the form:

$$\dot{x}(t) = h(x(t), u(t)) \ , \ \text{a.e. for } \ t \in [0, 1] \ , \ x(0) = \xi \ , \tag{3.1}$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, and hence $h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. Non-autonomous dynamics can be handled by defining time as an extra state variable with $\dot{t} = 1$, $t(0) = 0$.

To establish continuity and differentiability of solutions of (3.1) with respect to controls, one must assume that the controls are bounded in $L_\infty^m[0, 1]$. However, the finite dimensional approximating control subspaces that we will introduce must be treated as Hilbert spaces. This can cause complications in establishing the required approximation properties of the optimality functions for the approximating problems that we will construct. To circumvent this difficulty, we will, as in [43] assume that the controls are elements of the pre-Hilbert space

$$L^m_{\infty,2}[0,1] \doteq (\ L^m_\infty[0,1]\ \ ,\ \ \langle\cdot,\cdot\rangle_2\ \ ,\ \ \|\cdot\|_2\ ), \tag{3.2a}$$

which consists of the elements of $L^m_\infty[0,1]$, but is endowed with the $L^m_2[0,1]$ inner product and norm. Note that $L^m_{\infty,2}[0,1]$ is dense in $L^m_2[0,1]$.

We will define our optimal control problems on the pre-Hilbert space

$$H_{\infty,2} \doteq \mathbb{R}^n \times L^m_{\infty,2}[0,1] \doteq (\ \mathbb{R}^n \times L^m_\infty[0,1]\ \ ,\ \ \langle\cdot,\cdot\rangle_H\ \ ,\ \ \|\cdot\|_H\ ), \tag{3.2b}$$

whose elements η consist of pairs of initial states and controls, *i.e.*, $\eta = (\xi, u)$. Note that $H_{\infty,2}$ is a dense subspace of the Hilbert space

$$H_2 = \mathbb{R}^n \times L^m_2[0,1]\ . \tag{3.2c}$$

The inner product $\langle\cdot,\cdot\rangle_H$ and norm $\|\cdot\|_H$, on H_2, and hence also on $H_{\infty,2}$, are defined as follows. For any $\eta = (\xi, u) \in H_2$ and $\eta' = (\xi', u') \in H_2$,

$$\langle \eta, \eta' \rangle_H \doteq \langle \xi, \xi' \rangle + \langle u, u' \rangle_2, \tag{3.2d}$$

where $\langle \xi, \xi' \rangle$ denotes the Euclidean inner product, and the L_2 inner product $\langle u, u' \rangle_2$ is defined by $\langle u, u' \rangle_2 \doteq \int_0^1 \langle u(t), u'(t) \rangle\, dt$. Consequently, for any $\eta = (\xi, u) \in H_2$,

$$\|\eta\|_H^2 \doteq \langle \eta, \eta \rangle_H = \|\xi\|^2 + \|u\|_2^2\ . \tag{3.2e}$$

Next, we introduce a compact, convex control constraint set $U \subset B(0, \rho_{\max}) \doteq \{\ u \in \mathbb{R}^m \mid \|u\| \le \rho_{\max}\ \}$, where $\rho_{\max}$ is assumed to be sufficiently large to ensure that all the controls $u(\cdot)$ which we expect to deal with take values in the interior of $B(0, \rho_{\max})$. We then define the set of admissible controls by

$$\mathbf{U} \doteq \{\ u \in L^m_{\infty,2}[0,1] \mid u(t) \in U,\ \text{ a.e. for }\ \ t \in [0,1]\ \} \tag{3.3a}$$

and the set of admissible initial state-control pairs by

$$\mathbf{H} \doteq \mathbb{R}^n \times \mathbf{U} \subset H_{\infty,2}\ . \tag{3.3b}$$

The set $\mathbf{H}$ is contained in the larger set

$$\mathbf{B} \doteq \mathbb{R}^n \times \{\ u \in L^m_{\infty,2}[0,1] \mid u(t) \in B(0, \rho_{\max})\ ,\ \text{ a.e. on }\ [0,1]\ \} \subset H_{\infty,2} \tag{3.3c}$$

inside which all of our results concerning differential equations are valid. Finally, solutions of (3.1) corresponding to a particular $\eta \in \mathbf{B}$ will be denoted by $x^\eta(\cdot)$.

We will consider the following canonical constrained minimax optimal control problem:

CP $$\min_{\eta \in \mathbf{H}} \{ \psi_o(\eta) \mid \psi_c(\eta) \le 0 \} , \tag{3.4a}$$

where the objective function, $\psi_o : \mathbf{B} \to \mathbb{R}$, and the state endpoint constraint function, $\psi_c : \mathbf{B} \to \mathbb{R}$ are defined by

$$\psi_o(\eta) \doteq \max_{\nu \in \mathbf{q}_o} f^\nu(\eta) , \quad \psi_c(\eta) \doteq \max_{\nu \in \mathbf{q}_c + q_o} f^\nu(\eta) , \tag{3.4b}$$

where the ν-th function $f^\nu : \mathbf{H} \to \mathbb{R}$ is defined by

$$f^\nu(\eta) \doteq \zeta^\nu(\xi, x^\eta(1)) , \tag{3.4c}$$

with $\zeta^\nu : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, and $\mathbf{q}_o \doteq \{ 1, 2, \ldots, q_o \}$, $\mathbf{q}_c \doteq \{ 1, 2, \ldots, q_c \}$ (with q_o and q_c positive integers). The set $\mathbf{q}_c + q_o \doteq \{ 1 + q_o, \ldots, q_c + q_o \}$. In what follows, we will let $\mathbf{q} \doteq \{ 1, 2, \ldots, q \}$ with $q = q_o + q_c$. By defining the feasible set $\mathbf{F} \doteq \{ \eta \in \mathbf{H} \mid \psi_c(\eta) \le 0 \}$, we can write **CP** in the equivalent form of problem **P** in (2.1a).

Various optimal control problems, such as non-autonomous, integral cost, and free-time problems, can be transcribed into this canonical form. Also, the endpoint constraint in (3.4a) can be discarded by setting $\psi_c(\eta) \equiv -\infty$, and control unconstrained problems can be included by choosing $\rho_{\max}$ and U sufficiently large to ensure that the solutions $u^*(\cdot)$ of **CP** take values in the interior of U.

Properties of the Defining Functions. We will require the following assumptions:

Assumption 3.1.

(a) The function $h(\cdot,\cdot)$ in (3.1) is continuously differentiable, and there exists a Lipschitz constant $\kappa < \infty$ such that for all $x', x'' \in \mathbb{R}^n$, and $v', v'' \in B(0, \rho_{\max})$ the following relations hold:

$$\| h(x', v') - h(x'', v'') \| \le \kappa [\| x' - x'' \| + \| v' - v'' \|] , \tag{3.5a}$$

$$\| h_x(x', v') - h_x(x'', v'') \| \le \kappa [\| x' - x'' \| + \| v' - v'' \|] , \tag{3.5b}$$

$$\| h_u(x', v') - h_u(x'', v'') \| \le \kappa [\| x' - x'' \| + \| v' - v'' \|] , \tag{3.5c}$$

(b) The functions $\zeta^\nu(\cdot,\cdot)$, $\zeta^\nu_\xi(\cdot,\cdot)$ and $\zeta^\nu_x(\cdot,\cdot)$, with $\nu \in \mathbf{q}$, are Lipschitz continuous on bounded sets. □

The following results can be found in [58]

Theorem 3.2. If Assumption 3.1 is satisfied then

(i) there exists an $\kappa < \infty$ such that for all $\eta', \eta'' \in \mathbf{B}$ and for all $t \in [0,1]$

$$\| x^{\eta'}(t) - x^{\eta''}(t) \| \le \kappa \| \eta' - \eta'' \|_H ;$$

(ii) there exists a $L < \infty$ such that for all $\eta \in \mathbf{B}$ and all $t \in [0,1]$

$$\| x^{\eta}(t) \| \le L(1 + \| \xi \|) ;$$

(iii) the functions $\psi_o : \mathbf{B} \to \mathbb{R}$ and $\psi_c : \mathbf{B} \to \mathbb{R}$ are Lipschitz continuous on bounded sets;

(iv) the functions $f^{\nu}(\cdot)$, $\nu \in \mathbf{q}$, have continuous Gâteaux differentials $Df^{\nu} : \mathbf{B} \times H_{\infty,2} \to \mathbb{R}$ that have the form $Df^{\nu}(\eta ; \delta\eta) = \langle \nabla f^{\nu}(\eta), \delta\eta \rangle_H$;

(v) the gradients $\nabla f^{\nu} : \mathbf{B} \to H_{\infty,2}$, $\nabla f^{\nu}(\eta) = (\nabla_{\xi} f^{\nu}(\eta), \nabla_u f^{\nu}(\eta))$, $\nu \in \mathbf{q}$, are given by

$$\nabla_{\xi} f^{\nu}(\eta) = \nabla_{\xi} \zeta^{\nu}(\xi, x^{\eta}(1)) + p^{\nu,\eta}(0) , \tag{3.6a}$$

$$\nabla_u f^{\nu}(\eta)(t) = h_u(x(t), u(t))^T p^{\nu,\eta}(t) \quad , \quad \forall\, t \in [0,1], \tag{3.6b}$$

where $p^{\nu,\eta}(t) \in \mathbb{R}^n$ is the solution to the adjoint equation

$$\dot{p}^{\nu} = - h_x(x^{\eta}, u)^T p^{\nu} \quad , \quad p^{\nu}(1) = \nabla_x \zeta^{\nu}(\xi, x^{\eta}(1)) \quad , \quad t \in [0,1] , \tag{3.6c}$$

and are Lipschitz continuous on bounded sets in **B**. □

An Optimality Function. Referring to [59] the following result holds because of Theorem 3.2:

Theorem 3.3 For any $\eta \in \mathbf{B}$, let

$$\psi_c(\eta)_+ \doteq \max \{ 0, \psi_c(\eta) \} \ , \tag{3.7a}$$

and for any $\eta, \eta' \in \mathbf{B}$ and $\sigma > 0$, let

$$\Psi(\eta, \eta') \doteq \max \{ \psi_o(\eta) - \psi_o(\eta') - \sigma \psi_c(\eta')_+ \, , \psi_c(\eta) - \psi_c(\eta')_+ \} . \tag{3.7b}$$

If Assumption 3.1 is satisfied and $\hat{\eta} \in \mathbf{H}$ is a local minimizer of the problem **CP**, then

$$D_2 \Psi \, (\hat{\eta}, \hat{\eta} \, ; \eta - \hat{\eta}) \ge 0 \ , \quad \forall\, \eta \in \mathbf{H} \, , \tag{3.8}$$

where $D_2\Psi$ indicates the directional derivative of $\Psi(\cdot,\cdot)$ with respect to its second argument. □

Next we define an optimality function $\theta : \mathbf{B} \to \mathbb{R}$ for **CP**. For any $\eta, \eta' \in \mathbf{B}$ and $\nu \in \mathbf{q}$, we define a first-order quadratic approximation to $f^{\nu}(\cdot)$ at η by

$$\tilde{f}^{\nu}(\eta, \eta') \doteq f^{\nu}(\eta) + \langle \nabla f^{\nu}(\eta), \eta' - \eta \rangle_H + \tfrac{1}{2} \| \eta' - \eta \|_H^2 \ . \tag{3.9a}$$

We define the optimality function, with the same fixed $\sigma > 0$ used in (3.7b), by

$$\theta(\eta) \doteq \min_{\eta' \in \mathbf{H}} \max \left\{ \max_{\nu \in \mathbf{q}_o} \tilde{f}^{\nu}(\eta, \eta') - \psi_o(\eta) - \sigma \psi_c(\eta)_+ \,,\ \max_{\nu \in \mathbf{q}_c + q_o} \tilde{f}^{\nu}(\eta, \eta') - \psi_c(\eta)_+ \right\} . \tag{3.9b}$$

The existence of the minimum in (3.9b) follows from the convexity of the constraint set **H** and of the max functions in (3.9b) with respect to η', and the fact that $\tilde{f}^{\nu}(\eta, \eta') \to \infty$ as $\|\eta'\| \to \infty$ [60, Corollary III.20 (p. 46)]. Note that if $f^{\nu}(\eta) \equiv -\infty$ for all $\nu \in \mathbf{q}_c + q_o$, so that $\psi_c(\eta) \equiv -\infty$, then (3.9b) reduces to

$$\theta(\eta) \doteq \min_{\eta' \in \mathbf{H}} \max_{\nu \in \mathbf{q}_o} f_{\nu}(\eta) + \langle \nabla f^{\nu}(\eta), \eta' - \eta \rangle_H + \tfrac{1}{2} \|\eta' - \eta\|_H^2 - \psi_o(\eta) . \tag{3.9c}$$

Referring once again to [58] we find the following result:

Theorem 3.5. Let $\theta : \mathbf{B} \to \mathbb{R}$ be defined by (3.9b). If Assumption 3.1 holds then, *(i)* $\theta(\cdot)$ is negative valued and continuous; *(ii)* the relation (3.8) holds if and only if $\theta(\hat{\eta}) = 0$. □

2.4 APPROXIMATING PROBLEMS

The construction of a family of approximating problems for our problem **CP**, in (3.4a), satisfying the axioms of the theory of consistent approximation requires the construction of nested families of finite-dimensional subspaces of the initial state-control space $H_{\infty,2}$, approximating cost functions, and approximating constraint sets. Our selection of these approximations is largely determined by our intention to use explicit, fixed step-size Runge-Kutta (RK) methods [61,62] for integrating the dynamic equations (3.1). Throughout this chapter, we assume, without loss of generality, that the integration proceeds with a uniform step-size. We will relax this assumption in Chapter 4.

2.4.1 Finite Dimensional Initial-State-Control Subspaces

We begin by defining families of finite dimensional subspaces H_N, with $H_N = \mathbb{R}^n \times L_N \subset H_{\infty,2}$, where the L_N are finite-dimensional subspaces of $L^m_{\infty,2}[0,1]$, spanned by piecewise-continuous functions to which RK methods can be extended. Hence, given an explicit, fixed step-size RK integration method, using step-size $\Delta = 1/N$, we impose the following conditions on the subspaces L_N:

(i) For any bounded subset S of **B**, there exists a $\kappa < \infty$ such that for any $\eta \in S \cap H_N$, the RK method results in an integration error no greater than κ / N in solving the differential equation (3.1).

(ii) The data used by the RK integration method is an initial state and a set of control

samples[†]. We will require that each set of control samples corresponds to a unique element $u \in L_N$.

Condition *(i)* will be needed to prove that our approximating problems epiconverge to the original problem. For the subspaces L_N that we will present, we will actually be able to prove more than first order accuracy. Condition *(ii)* facilitates the definition of the approximating problems and makes it possible to define gradients for the approximating cost and constraint functions.

We will now show how the choice of an RK integration method affects the selection of the subspaces L_N. The generic, explicit fixed step-size, s-stage RK method computes an approximate solution to a differential equation of the form

$$\dot{x}(t) = \tilde{h}(t, x(t)) \ , \quad x(0) = \xi \ , \quad t \in [0, 1] \ , \tag{4.1a}$$

where $\tilde{h} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is continuous in t and Lipschitz continuous in x. It does so by solving the difference equation

$$\bar{x}_{k+1} = \bar{x}_k + \Delta \sum_{i=1}^{s} b_i K_{k,i} \ , \quad \bar{x}_0 = x(0) = \xi \ , \quad k \in \mathcal{N} \doteq \{ 0, 1, \ldots, N-1 \} \ , \tag{4.1b}$$

with $\Delta = 1/N$, $t_k \doteq k\Delta$, and $K_{k,i}$ defined by the recursion

$$K_{k,1} = \tilde{h}(\tau_{k,1}, \bar{x}_k) \ , \quad K_{k,i} = \tilde{h}(\tau_{k,i}, \bar{x}_k + \Delta \sum_{j=1}^{i-1} a_{i,j} K_{k,j}) \ , \ i = 2, \ldots, s \ , \tag{4.1c}$$

where, for convenience, we have defined

$$\tau_{k,i} \doteq t_k + c_i \Delta \ , \ \Delta = 1 / N \ . \tag{4.1d}$$

The variable $\bar{x}_k$ is the computed estimate of $x(t_k)$. The time points $\{ t_k \}_{k=0}^{N}$ define the *integration mesh*, also referred to as the *discretization mesh*. These time points will also be referred to as *breakpoints* in the context of piecewise control representations.

The parameters $a_{i,j}$, c_i and b_i, in (4.1b) and (4.1c) determine the RK method. These parameters are collected in the *Butcher array* $\mathbf{A} = [c, A, b^T]$. The Butcher array is often displayed in the form:

[†] The term control samples will be clarified shortly.

$$
\mathbf{A} \quad = \quad \begin{array}{c|cccc}
c_1 & 0 & & & \\
c_2 & a_{2,1} & 0 & & \\
\vdots & & \ddots & \ddots & \\
c_s & a_{s,1} & & a_{s,s-1} & 0 \\
\hline
 & b_1 & \cdots & b_{s-1} & b_s
\end{array}
$$

The following assumption on the b parameters will hold throughout this chapter (conditions on the c parameters will be added later):

Assumption 4.1. For all $i \in \mathbf{s}$, $b_i > 0$ and $\sum_{i=1}^{s} b_i = 1$. □

Remark 4.2. The condition $\sum_{i=1}^{s} b_i = 1$ is satisfied by all convergent RK methods. Other conditions must be satisfied to achieve higher order convergence for multi-stage RK methods. □

Now, in our case, $\widetilde{h}(t,x) = h(x, u(t))$ and the elements $u(t)$ of the subspaces L_N will be allowed to be discontinuous from the left at certain pre-specified points. Hence, $\widetilde{h}(\cdot, x)$ is discontinuous and special care must be taken to ensure accurate integration. For this purpose, the values $u(\tau_{k,i})$ must sometimes be replaced by left limits as appropriate for the particular choice of the subspace L_N. We will refer to these values as *control samples* and denote them by $u[\tau_{k,i}]$ where, if necessary, $u[\tau_{k,i}] = \lim_{t \uparrow \tau_{k,i}} u(t)$. The specific definition of $u[\tau_{k,i}]$ depends on the definition of L_N, but clearly if $u(\cdot)$ is continuous at $\tau_{k,i}$ then $u[\tau_{k,i}] = u(\tau_{k,i})$.

The recursion (4.1c) evaluates $\widetilde{h}(\cdot,\cdot)$ s times for each time-step $k \in \mathcal{N}$. If we collect the corresponding s control samples into a matrix $\omega_k \doteq (u[\tau_{k,1}] \cdots u[\tau_{k,s}])$, we can replace equations (4.1b) and (4.1c) with

$$
\bar{x}_{k+1} = \bar{x}_k + \Delta \sum_{i=1}^{s} b_i K_{k,i} \ , \quad \bar{x}_0 = x(0) = \xi \ , \quad k \in \mathcal{N}, \tag{4.3a}
$$

where $K_{k,i} \doteq K_i(\bar{x}_k, \omega_k)$ which is defined by the recursion

$$
K_1(x,\omega) = h(x,\omega_1) \ , \quad K_i(x,\omega) = h(x + \Delta \sum_{j=1}^{i-1} a_{i,j} K_j(x,\omega)\ , \omega_i) \ , \ i = 2, \ldots, s \ , \tag{4.3b}
$$

where ω_i is the i-th column of ω. Equations (4.3a,b) can be written equivalently as

$$
\bar{x}_{k+1} = \bar{x}_k + \Delta \sum_{i=1}^{s} b_i h(\bar{Y}_{k,i}, \omega_i) \ , \quad \bar{x}_0 = x(0) = \xi \ , \quad k \in \mathcal{N}, \tag{4.3c}
$$

where, for each k,

$$
\bar{Y}_{k,1} = \bar{x}_k \quad , \quad \bar{Y}_{k,i} = \bar{x}_k + \Delta \sum_{j=1}^{i-1} a_{i,j} h(\bar{Y}_{k,j}, \omega_j) \ . \tag{4.3d}
$$

The quantities $\bar{Y}_{k,i}$ are intermediate estimates of $x(\tau_{k,i})$.

We will define the control subspace L_N, in such a way that there is a one-to-one correspondence between elements $u \in L_N$ and the samples of $u[t_k + c_i\Delta]$ used by the RK method with step-size $\Delta = 1/N$. The definition of L_N is somewhat complicated by the fact that some of the c_i elements of the Butcher array may have the same value. This causes the RK method to use samples at times $t_k + c_i\Delta$ more than once and hence leads to a reduction of the dimension in the associated subspace L_N. To keep track of the distinct values of the c_i elements of the Butcher array, we define the ordered set of indices

$$I \doteq \{ i_1, i_2, \ldots, i_r \} \doteq \{ i \in \mathbf{s} \,|\, c_j \neq c_i, \forall\, j \in \mathbf{s},\, j < i \}, \tag{4.4a}$$

and let

$$I_j \doteq \{ i \in \mathbf{s} \,|\, c_i = c_{i_j},\, i_j \in I \} \ , \quad j \in \mathbf{r} \, . \tag{4.4b}$$

Thus, the total number of distinct values taken by the elements c_i in the Butcher array is r. For example, if $c = \{ 0, 1/2, 1/2, 1 \}$ (as in the most commonly used fourth order RK method), then $r = 3$, $I = \{ i_1 = 1, i_2 = 2, i_3 = 4 \}$, $I_1 = \{ 1 \}$, $I_2 = \{ 2, 3 \}$, and $I_3 = \{ 4 \}$. If each c_i is distinct, then $r = s$, $i_j = j$, and I_j is the singleton $\{ j \}$. Otherwise, $r < s$ and $i_j \geq j$ for each $j \in \mathbf{r}$.

By construction of the set I, the r distinct sampling times in the interval $[t_k, t_{k+1}]$, $k \in \mathcal{N}$ are given by τ_{k,i_j}, $j \in \mathbf{r}$, $i_j \in I$. Corresponding to each sampling time there is a control sample $u[\tau_{k,i_j}] \in \mathbb{R}^m$. The collection of these control samples can be viewed as a vector $\bar{u} \in \times_N \times_r \mathbb{R}^m$, where the symbol $\times_N$ indicates the Cartesian product of N spaces. We will partition vectors $\bar{u} \in \times_N \times_r \mathbb{R}^m$ into N blocks, as follows:

$$\bar{u} = (\bar{u}_0, \bar{u}_1, \ldots, \bar{u}_{N-1}) \,, \tag{4.5a}$$

where each block $\bar{u}_k \in \times_r \mathbb{R}^m$, $k \in \mathcal{N}$, is of the form

$$\bar{u}_k = (\bar{u}_{k,1}, \ldots, \bar{u}_{k,r}) \,, \tag{4.5b}$$

with $\bar{u}_{k,j} \in \mathbb{R}^m$, $j \in \mathbf{r}$, corresponding to the samples $u[\tau_{k,i_j}]$, $i_j \in I$, used by the RK integration during the k-th time interval. Our algebraic expressions are simplified if we treat $\bar{u}$ as the $m \times Nr$ matrix $[\bar{u}_{0,1} \cdots \bar{u}_{0,r} \cdots \bar{u}_{N-1,1} \cdots \bar{u}_{N-1,r}]$, *i.e.*, we will identify $\times_N \times_r \mathbb{R}^m$ with the space $\mathbb{R}^{m \times Nr}$ of $m \times Nr$ matrices. Similarly, in algebraic expressions, will treat $\bar{u}_k$ as the $m \times r$ matrix $[\bar{u}_{k,1} \cdots \bar{u}_{k,r}]$. The standard inner product on $\times_N \times_r \mathbb{R}^m$ is the l_2 Euclidean inner product given by

$$\langle \bar{u}, \bar{v} \rangle_{l_2} = \sum_{N=0}^{N-1} \sum_{j=1}^{r} \langle \bar{u}_{k,j}, \bar{v}_{k,j} \rangle \,. \tag{4.5c}$$

Let G be the $r \times s$ matrix defined by

$$G = \begin{bmatrix} \mathbb{1}_1^T & & & \\ & \mathbb{1}_2^T & & \\ & & \ddots & \\ & & & \mathbb{1}_r^T \end{bmatrix} \tag{4.5d}$$

where, for each $j \in \mathbf{r}$, $\mathbb{1}_j^T = (1, 1, \ldots, 1)$ is a row vector of dimension $|I_j|$ ($|I_j|$ is the number of elements in I_j). Then we can associate the components $\bar{u}_k$, $k \in \mathcal{N}$, of a vector $\bar{u} \in \underset{N}{\times}\underset{r}{\times} \mathbb{R}^m$, with the matrices ω_k used by the RK method (4.3a,b) by setting $\omega_k = \bar{u}_k G = [\bar{u}_{k,1} \cdots \bar{u}_{k,r}]G$.

We now present two control representations that define subspaces $L_N^i \subset L_{\infty,2}^m[0,1]$, $i = 1, 2$, $N \in \mathbb{N}$, of dimension Nrm, such that $\cup_{N=1}^{\infty} L_N^1$ and $\cup_{N=1}^{\infty} L_N^2$ are dense in $L_{\infty,2}^m[0,1]$. Both representations reduce to simple square pulses for Euler's method ($r = 1$). The basis functions $\{ e_l \Phi_{N,k,j}^i \}_{j=1,k=1,l=1}^{N,r,m}$, $i = 1, 2$, with e_l the l-th unit vector in $\mathbb{R}$ and $\Phi_{N,l,k}^i : [0,1] \to \mathbb{R}^m$, that we use to construct the spaces L_N^i are not orthonormal. Hence, for numerical calculations, we associate with these spaces Nrm-dimensional spaces of real coefficients of the form

$$\bar{L}_N^i \doteq \left(\underset{N}{\times}\underset{r}{\times} \mathbb{R}^m \, , \langle \cdot, \cdot \rangle_{\bar{L}_N^i} \, , \, \| \cdot \|_{\bar{L}_N^i} \right) , \; i = 1, 2 \, , \; N \in \mathbb{N} \, , \tag{4.5e}$$

where the inner products and norms are chosen so that for any $u, v \in L_N^i$, with $u(t) = \sum_{j=1,k=1}^{N,r} \bar{u}_{k,j} \, \Phi_{N,j,k}^i(t)$ and $v(t) = \sum_{j=1,k=1}^{N,r} \bar{v}_{k,j} \, \Phi_{N,j,k}^i(t)$, $t \in [0,1]$,

$$\langle u, v \rangle_2 = \langle \bar{u}, \bar{v} \rangle_{\bar{L}_N^i} \quad , \quad \|u\|_2 = \|\bar{u}\|_{\bar{L}_N^i} , \tag{4.5f}$$

where $\bar{u} \in \underset{N}{\times}\underset{r}{\times} \mathbb{R}^m$ is defined in (4.5b,c). The spaces $\bar{L}_N^i$ will be needed to define gradients for the cost and constraint functions of the approximating problems as well as in setting up numerical implementations of optimal control algorithms. Figure 4.1, which follows the definitions of L_N^i below, illustrates the relationship between the various control spaces.

The reason that we choose an L_2 norm preserving, nonstandard inner product on $\bar{L}_N^i$ is that if we use the standard l_2 inner product and norm on $\bar{L}_N^i$ (as is commonly done), we might, unwittingly, cause serious deterioration in the performance of numerical algorithms which solve the approximating problems in the coefficient spaces $\bar{L}_N^i$. The extent of this ill-conditioning effect is illustrated in Section 6. Of course, if our basis for L_N had been orthonormal, then standard l_2 inner product would be the appropriate choice. The purpose of defining different control representations is first, that the solutions of the approximating problems have different properties for the different representations (this is discussed at the end of this section) and, second, some results for the second representation are used to provide results for the first representation (Conjecture 5.11 and formula (7.19b)).

Representation R1

(Piecewise r-th order polynomials)

Assumption 4.3. For all $i \in \mathbf{s}$, $c_i \in [0,1]$. $\square$

For each $k \in \mathcal{N}$, define the sub-intervals $T_k^1 \doteq [t_k, t_{k+1})$ and define pulse functions

$$\Pi_{N,k}^1(t) \doteq \begin{cases} 1 & \text{if } t \in T_k^1 \\ 0 & \text{elsewhere} \end{cases} . \tag{4.6a}$$

Then, define the finite dimensional control subspace L_N^1 as follows:

$$L_N^1 \doteq \{\, u \in L_2^m[0,1] \,|\, u(t) = \sum_{k=0}^{N-1} \sum_{j=1}^{r} \bar{u}_{k,j}\, \Phi_{N,k,j}^1(t) \ ,\ \bar{u}_{k,j} \in \mathbb{R}^m, \forall\, t \in [0,1] \,\} , \tag{4.6b}$$

where

$$\Phi_{N,k,j}^1(t) \doteq \phi_{N,k,j} \Pi_{N,k}^1(t) \ ,\ \ k \in \mathcal{N}, \tag{4.6c}$$

with $\phi_{N,k,j}(t)$ the j-th Lagrange polynomial for the points $\{\, \tau_{k,i_j} \,\}_{j=1}^{r}$, $i_j \in I$, defined by

$$\phi_{N,k,j}(t) \doteq \prod_{\substack{l\,=\,1 \\ l \neq j}}^{r} \frac{(t - \tau_{k,i_l})}{(\tau_{k,i_j} - \tau_{k,i_l})} \ ,\ \ k \in \mathcal{N},\ \ j \in \mathbf{r} \ , \tag{4.6d}$$

with the property that $\phi_{N,k,j}(\tau_{k,i_l}) = 1$ if $l = j$ and $\phi_{N,k,j}(\tau_{k,i_l}) = 0$ if $l \neq j$. By construction of the set I, $i_l, i_j \in I$ implies that $\tau_{k,i_j} \neq \tau_{k,i_l}$ if $l \neq j$. Hence, the functions $\phi_{N,k,j}(\cdot)$ are well-defined, and the functions $\Phi_{N,k,j}^1(\cdot)$ are linearly independent. The function $\phi_{N,k,j}(t)$ is thus the unique r-th order polynomial that interpolates $\{\, (\tau_{k,i_j}, \bar{u}_{k,j}) \,\}_{j=1}^{r}$ on the interval $[t_k, t_{k+1}]$. The control samples for L_N^1 are given by

$$u[\tau_{k,i}] \doteq \begin{cases} u(\tau_{k,i}) & \text{if } \tau_{k,i} \in T_k^1 \\ \lim_{t \uparrow \tau_{k,i}} u(t) & \text{if } \tau_{k,i} = t_{k+1} \end{cases} \ ,\ k \in \mathcal{N},\ i \in I \ . \tag{4.6e}$$

Proposition 4.4. Let L_N^1 be defined as in (4.6b) and define the map

$$V_{\mathbf{A},N}^1 : L_N^1 \to \mathop{\times}_{N} \mathop{\times}_{r} \mathbb{R}^m$$

$$u \mapsto \{\{\, u[\tau_{k,i_j}] \,\}_{j=1}^{r} \,\}_{k=0}^{N-1} \ ,\ \ i_j \in I \ , \tag{4.6f}$$

with $u[\cdot]$ given by (4.6e). Suppose Assumption 4.3 holds. Then $V_{\mathbf{A},N}^1$ is invertible.

Proof. Let $u(t) = \sum_{j=0}^{N-1} \sum_{k=1}^{r} \bar{u}_{k,j} \Phi_{N,k,j}^1(t)$ be an arbitrary element of L_N^1. Assumption 4.3 implies that $\tau_{k,i_j} \in T_k^1$. Next, it follows from (4.6e), that $u[\tau_{k,i_j}] = \sum_{j=1}^{r} \bar{u}_{k,j} \phi_{N,k,j}(\tau_{k,i_j}) = \bar{u}_{k,j}$, because of the interpolation property of Lagrange polynomials. Hence $V_{\mathbf{A},N}^1$ is invertible. $\square$

The polynomial pulse functions $\{ \Phi^1_{N,k,j}(t) \}_{k=0,j=0}^{N-1,r-1}$ are linearly independent, but they are neither orthogonal nor normal with respect to the L_2 inner product and norm. To complete the definition of the spaces $\bar{L}^1_N$ in (4.5e), we will now define the required inner product, which, in turn, defines the norm. First, let $u \in L^1_N$ and note that we can write each r-th order polynomial piece $\sum_{j=1}^r \bar{u}_{k,j}\phi_{N,k,j}(t)$ in (4.6b) as a power series $\alpha_k P(t - t_k)$, where α_k is an $m \times r$ matrix of coefficients and the function $P : \mathbb{R} \to \mathbb{R}^r$ is defined by

$$P(t) \doteq [\, 1 \;\; t/\Delta \cdots (t/\Delta)^{r-1} \,]^T \; . \tag{4.7}$$

If $\bar{u} = V^1_{\mathbf{A},N}(u)$, then from Proposition 4.4, $\bar{u}_{k,j} = \alpha_k P(c_{i_j}\Delta)$, $j \in \mathbf{r}$, $i_j \in I$. Hence, $\bar{u}_k = [\, \bar{u}_{k,1} \cdots \bar{u}_{k,r} \,] = \alpha_k T^{-1}$ where

$$T^{-1} \doteq \Big[P(c_{i_1}\Delta) \;\; P(c_{i_2}\Delta) \cdots P(c_{i_r}\Delta) \Big] = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ c_{i_1} & c_{i_2} & & c_{i_r} \\ \vdots & & \ddots & \\ c_{i_1}^{r-1} & c_{i_2}^{r-1} & & c_{i_r}^{r-1} \end{bmatrix}_{r \times r} . \tag{4.8}$$

The matrix T^{-1} is a Vandermonde matrix and the r values c_{i_j}, $i_j \in I$, are distinct. Therefore, T^{-1} is non-singular and $\alpha_k = \bar{u}_k T$. Hence, for each $k \in \mathcal{N}$, $u(t) = \bar{u}_k T P(t - t_k)$ for $t \in [t_k, t_{k+1})$.

We now define the inner product between two vectors $\bar{u}, \bar{v} \in \bar{L}^1_N$, with $u = (V^1_{\mathbf{A},N})^{-1}(\bar{u})$ and $v = (V^1_{\mathbf{A},N})^{-1}(\bar{v})$, by

$$\begin{aligned} \langle \bar{u}, \bar{v} \rangle_{\bar{L}^1_N} = \langle u, v \rangle_2 &= \sum_{k=0}^{N-1} \int_0^\Delta \langle u(t_k + t), v(t_k + t) \rangle \, dt \\ &= \sum_{k=0}^{N-1} \int_0^\Delta \langle \bar{u}_k \, T \, P(t), \bar{v}_k \, T \, P(t) \rangle \, dt \\ &= \Delta \sum_{k=0}^{N-1} \operatorname{trace}\!\Big(\bar{u}_k \, T \, \frac{1}{\Delta} \int_0^\Delta P(t) \, P(t)^T dt \, T^T \, \bar{v}_k^T \Big) \; , \\ &= \Delta \sum_{k=0}^{N-1} \operatorname{trace}\!\big(\bar{u}_k \, M_1 \, \bar{v}_k^T \big) \; , \end{aligned} \tag{4.9a}$$

where T was defined by (4.8), $P(\cdot)$ was defined in (4.7), and

$$\boxed{ M_1 \doteq T \left[\frac{1}{\Delta} \int_0^\Delta P(t) \, P(t)^T dt \right] T^T = T \, \mathrm{Hilb}(r) \, T^T } \; , \tag{4.9b}$$

is an $r \times r$ symmetric, positive definite matrix with

$$
\mathrm{Hilb}(r) = \begin{bmatrix} 1 & 1/2 & 1/3 & & 1/r \\ 1/2 & 1/3 & 1/4 & \cdots & 1/(r+1) \\ 1/3 & 1/4 & 1/5 & & \\ & \vdots & & \ddots & \\ 1/r & 1/(r+1) & & & 1/(2r-1) \end{bmatrix}_{r\times r} \tag{4.9c}
$$

the Hilbert matrix whose i, j-th entry is $1/(i+j-1)$. Note that both $\mathrm{Hilb}(r)$ and T are ill-conditioned matrices. However, the product in (4.9b) is well-conditioned (the product corresponds to switching from the power-series polynomial representation back to the Lagrange expansion). The matrix M_1 is positive definite because $\mathrm{Hilb}(r)$ is positive definite and T is non-singular. Given $\bar{u} \in \bar{L}_N^1$, its norm is $\|\bar{u}\|^2_{\bar{L}_N^1} = \langle \bar{u}, \bar{u} \rangle_{\bar{L}_N^1}$. Finally, if we define the N-block diagonal matrix

$$
\mathbf{M}_N \doteq \mathrm{diag}[\,\Delta_0 M_1\,, \Delta_1 M_1\,, \ldots, \Delta_{N-1} M_1\,]\,, \tag{4.9d}
$$

with $\Delta_k \doteq t_{k+1} - t_k = 1\,/\,N$ (in this chapter), then we can express the inner product given by (4.9a) more succinctly as

$$
\langle \bar{u}, \bar{v} \rangle_{\bar{L}_N^1} = \langle \bar{u}\mathbf{M}_N, \bar{v} \rangle_{l_2} = \mathrm{trace}(\bar{u}\mathbf{M}_N \bar{v}^T) \tag{4.9e}
$$

We have introduced the notation of Δ_k here in anticipation of using non-uniform meshes in later chapters.

Remark 4.5. A special class of functions within representation **R1** is the subspace of r-th order, m dimensional splines [63]. The dimension of the spline subspace is only a fraction of the dimension of L_N^1. Our results for **R1** can be extended to splines; this extension is presented Section 7. □

Representation R2

(Piecewise constant functions)

For $j \in \mathbf{r}$, I_j defined in (4.4b), let

$$
\widetilde{b}_j \doteq \sum_{i \,\in\, I_j} b_i, \tag{4.10a}
$$

$$
d_j \doteq \Delta \sum_{i=1}^{j} \widetilde{b}_i\,, \quad d_0 \doteq 0\,. \tag{4.10b}
$$

If all the c_i elements of the Butcher array have distinct values then $d_j = \Delta \sum_{i=1}^{j} b_i$. At this point, we can replace Assumption 4.1 with the following weaker assumption:

Assumption 4.1' For all $j \in \mathbf{r}$, $\widetilde{b}_j > 0$ and $d_r = \Delta$. □

Note that Assumption 4.1' implies that for all $j \in \mathbf{r}$, $d_j > d_{j-1}$, and that $t_k + d_j \in [t_k, t_{k+1}]$, $k \in \mathcal{N}$.

Next, we introduce one additional assumption which is stronger than Assumption 4.3 used for representation **R1**.

Assumption 4.6. For $j \in \mathbf{r}$ and $i_j \in I$, $d_{j-1} \le c_{i_j}\Delta \le d_j$, so that $\tau_{k,i_j} \in [t_k + d_{j-1}, t_k + d_j]$. □

Now, for each $k \in \mathcal{N}$ and $j \in \mathbf{r}$, define the sub-intervals $T^2_{k,j} \doteq [t_k + d_{j-1}, t_k + d_j)$ and let the basis functions $\Phi^2_{N,k,j} : \mathbb{R} \to \mathbb{R}$ be defined by

$$\Phi^2_{N,k,j}(t) \doteq \begin{cases} 1 & \text{if } t \in T^2_{k,j} \\ 0 & \text{elsewhere} \end{cases} . \tag{4.11a}$$

Then, define the finite dimensional control subspace L^2_N as follows:

$$L^2_N \doteq \{\, u \in L^m_2[0,1] \mid u(t) = \sum_{k=0}^{N-1} \sum_{j=1}^{r} \bar{u}_{k,j}\, \Phi^2_{N,k,j}(t)\,,\ \bar{u}_{k,j} \in \mathbb{R}^m,\ \forall\, t \in [0,1] \,\}\ . \tag{4.11b}$$

The control samples for L^2_N are given by

$$u[\tau_{k,i_j}] \doteq \begin{cases} u(\tau_{k,i_j}) & \text{if } \tau_{k,i_j} \in T^2_{k,j} \\ \lim_{t \uparrow \tau_{k,i_j}} u(t) & \text{if } \tau_{k,i_j} = t_k + d_j \end{cases} ,\ k \in \mathcal{N},\ i_j \in I\ ,\ j \in \mathbf{r}\ . \tag{4.11c}$$

Proposition 4.7. Let L^2_N be defined as in (4.11b) and define the map

$$V^2_{\mathbf{A},N} : L^2_N \to \underset{N}{\times} \underset{r}{\times} \mathbb{R}^m$$

$$u \mapsto \{\{\, u[\tau_{k,i_j}] \,\}_{j=1}^{r} \,\}_{k=0}^{N-1}\ ,\ i_j \in I\ , \tag{4.11d}$$

with $u[\cdot]$given by (4.11c). Suppose Assumptions 4.1' and 4.6 hold. Then $V^1_{\mathbf{A},N}$ is invertible.

Proof. Assumption 4.1' ensures that the support for each $\Phi^2_{N,k,j}(\cdot)$ is of nonzero length. This ensures a one-to-one correspondence between the elements of L^2_N and the vector coefficients $\bar{u}_{k,j}$, in (4.11b). Next, Assumption 4.6 together with definition (4.11c) of $u[\cdot]$ implies that for any $u \in L^2_N$, with $u(t) = \sum_{k=0}^{N-1} \sum_{j=1}^{r} \bar{u}_{k,j}\, \Phi^2_{N,k,j}(t)$, $u[\tau_{k,j}] = \bar{u}_{k,j}$ for all $k \in \mathcal{N}$ and $j \in \mathbf{r}$. Hence, $V^2_{\mathbf{A},N}$ is invertible. □

To complete the definition, in (4.5e), of the spaces $\bar{L}^2_N$ we will now define the required inner product and norm. We define the inner product between two vectors $\bar{u}, \bar{v} \in \bar{L}^2_N$, with $u = (V^2_{\mathbf{A},N})^{-1}(\bar{u})$ and $v = (V^2_{\mathbf{A},N})^{-1}(\bar{v})$, by

$$\langle \bar{u}, \bar{v} \rangle_{\bar{L}^2_N} = \langle u, v \rangle_2 = \sum_{k=0}^{N-1} \sum_{j=1}^{r} \int_{d_{j-1}}^{d_j} \langle u(t_k + t), v(t_k + t) \rangle\, dt$$

$$= \Delta \sum_{k=0}^{N-1} \sum_{j=1}^{r} \tilde{b}_j \langle \bar{u}_{k,j}, \bar{v}_{k,j} \rangle \, dt$$

$$= \Delta \sum_{k=0}^{N-1} \operatorname{trace}\big(\bar{u}_k \, M_2 \, \bar{v}_k\big) \, , \tag{4.12a}$$

where,

$$\boxed{M_2 = \begin{bmatrix} \tilde{b}_1 & & 0 \\ & \ddots & \\ 0 & & \tilde{b}_r \end{bmatrix}} \, . \tag{4.12b}$$

Since all $\tilde{b}_j > 0$, M_2 is diagonal, positive definite. Given $\bar{u} \in \bar{L}_N^2$, its norm is $\|\bar{u}\|_{\bar{L}_N^2}^2 = \langle \bar{u}, \bar{u} \rangle_{\bar{L}_N^2}$. Finally, if we define the N-block diagonal matrix

$$\mathbf{M}_N \doteq \operatorname{diag}[\, \Delta_0 M_2 \, , \Delta_1 M_2 \, , \ldots , \Delta_{N-1} M_2 \,] \, , \tag{4.12c}$$

with $\Delta_k \doteq t_{k+1} - t_k = 1/N$ (in this chapter), then we can express the inner product given by (4.12a) more succinctly as

$$\langle \bar{u}, \bar{v} \rangle_{\bar{L}_N^2} = \langle \bar{u}\mathbf{M}_N, \bar{v} \rangle_{l_2} = \operatorname{trace}(\bar{u}\mathbf{M}_N \bar{v}^T) \, . \tag{4.12d}$$

Remark 4.8. In place of (4.10b), we could have used the alternate definition $d_j \doteq \Delta \sum_{i=1}^{j} b_i$ and set $\bar{u}_{k,j} = u[\tau_{k,j}]$ for all $j \in \mathbf{s}$, $k \in \mathcal{N}$. In this way, samples corresponding to repeated values of c_j in the Butcher array would be treated as independent values and the space L_N would have to be correspondingly enlarged. However, Proposition 6.2 in Section 6 indicates that (4.10b) is the preferable definition. □

The relationship between the spaces $L_{\infty,2}^m[0,1]$, L_N and $\bar{L}_N$ and the relationship between a function $u \in L_N^1$ and its control samples $\bar{u} = V_{\mathbf{A},N}(u)$ are illustrated in Figure 4.1.

2.4.2 Definition of Approximating Problems

For $N \in \mathbb{N}$, let

$$H_N \doteq \mathbb{R}^n \times L_N \, , \tag{4.13a}$$

where $L_N = L_N^1$ for representation **R1** or $L_N = L_N^2$ for representation **R2**. Since $H_N \subset H_{\infty,2}$, it inherits the inner product from $H_{\infty,2}$ which, for $\eta', \eta'' \in H_N$, with $\eta' = (\xi', u')$ and $\eta'' = (\xi'', u'')$, is given by

$$\langle \eta', \eta'' \rangle_H \doteq \langle \xi', \xi'' \rangle + \langle u', u'' \rangle_2 \, . \tag{4.13b}$$

Also, for any $\eta \in H_N$, $\|\eta\|_H^2 = \langle \eta, \eta \rangle_H$. Similarly, for $N \in \mathbb{N}$, we define the coefficient spaces

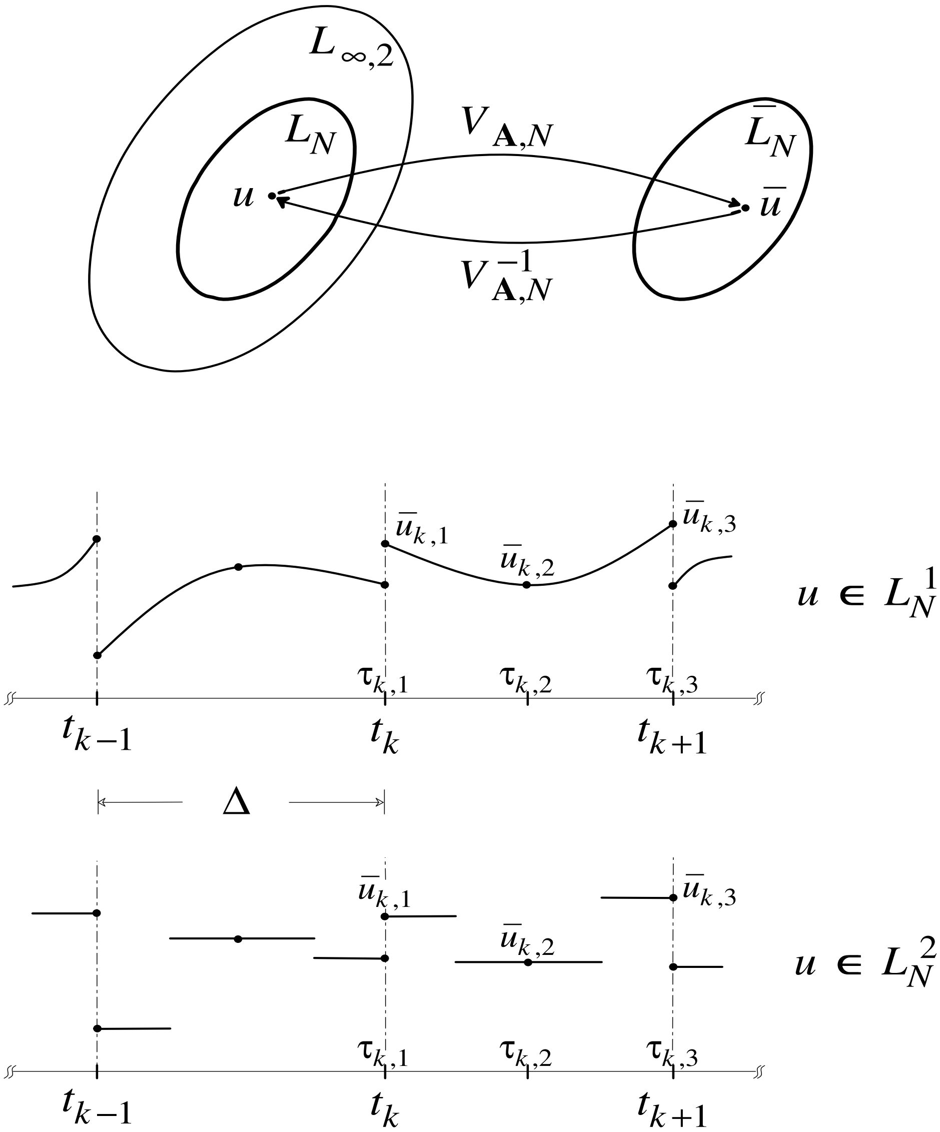


Fig. 4.1: The diagrams above depict the relationship between the various control spaces. On top, the map $V_{\mathbf{A},N}$, which is a bijection, identifies control functions in the finite dimensional spaces L_N with their control samples in the coefficient spaces $\bar{L}_N$. The spaces L_N are subsets of the infinite dimensional spaces $L^m_{\infty,2}[0,1]$. The two bottom plots show a portion of a single control, $u \in L_N$ for an RK method with $c = (0, \frac{1}{2}, \frac{1}{2})$ and $b = (\frac{1}{6}, \frac{2}{3}, \frac{1}{6})$. Since $r = 3$ there are three control samples per interval. The middle plot shows $u \in L^1_N$; u is composed of third order polynomial pieces. The bottom plot shows $u \in L^2_N$; u is piecewise constant. For the k-th interval the samples are taken at times $\tau_{k,j} = t_k + c_j\Delta$, $j = 1, 2, 3$, where $\Delta = 1/N$ is the step-size. Notice that the samples at $\tau_{k,1}$ and $\tau_{k,3}$ occur at points of discontinuities in $u(\cdot)$. The definition of the control samples, $\bar{u}_{k,j} = u[\tau_{k,j}]$, ensures that the samples on the k-th interval are taken from the k-th polynomial piece for $u \in L^1_N$, and for $u \in L^2_N$, the k, j-th sample is taken from the k, j-th piece. Note that this picture would look exactly the same for a four stage RK methods with $c = (0, \frac{1}{2}, \frac{1}{2}\ 1)$ and $b = (\frac{1}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{6})$ since, in this case, there is a repeated sampling time ($c_2 = c_3$ and $\tilde{b}_2 = \frac{2}{3}$).

$\bar{H}_N$ by

$$\bar{H}_N \doteq \mathbb{R}^n \times \bar{L}_N \ , \tag{4.14a}$$

where $\bar{L}_N = \bar{L}_N^1$ or $\bar{L}_N = \bar{L}_N^2$. The inner product on $\bar{H}_N$ is defined by

$$\langle \bar{\eta}', \bar{\eta}'' \rangle_{\bar{H}_N} \doteq \langle \xi', \xi'' \rangle + \langle \bar{u}', \bar{u}'' \rangle_{\bar{L}_N} \ , \tag{4.14b}$$

and the norm correspondingly. Let $W_{\mathbf{A},N} : H_N \to \bar{H}_N$ be defined by $W_{\mathbf{A},N}(\eta) = (\xi, V^1_{\mathbf{A},N}(u))$ for representation **R1** and $W_{\mathbf{A},N}(\eta) = (\xi, V^2_{\mathbf{A},N}(u))$ for representation **R2**, where $\eta = (\xi, u)$. Then we see that $W_{\mathbf{A},N}$ is a nonsingular map and, with our definition of the norms on $\bar{H}_N$, provides an isometric isomorphism between H_N and $\bar{H}_N$. Thus, we can use the spaces H_N and $\bar{H}_N$ interchangeably.

Next, we define control constraint sets for the approximating problems, as follows. Let U be the convex, compact set used to define $\mathbf{U}$ in (3.3a). Then, with $\kappa_U < \infty$, we define

$$\bar{\mathbf{U}}^1_N \doteq \{ \ \bar{u} \in \bar{L}^1_N \mid \bar{u}_{k,j} \in U, \ j \in \mathbf{r} \ , \ \|\bar{u}_k T_j\|_\infty \le \frac{\Delta}{(j-1)(r-1)} \kappa_U, \ j = 2, \ldots, r \ \ \forall \, k \in \mathcal{N} \} \tag{4.15a}$$

$$\bar{\mathbf{U}}^2_N \doteq \{ \ \bar{u} \in \bar{L}^2_N \mid \bar{u}_{k,j} \in U \ \ \forall \, j \in \mathbf{r}, k \in \mathcal{N} \} \ , \tag{4.15b}$$

where T_j is the j-th column of the matrix T, defined by its inverse in (4.8), and $\Delta = 1/N$, as before. Finally, we define the constraint sets for the approximating problems by

$$\mathbf{H}_N \doteq \mathbb{R}^n \times V^{-1}_{\mathbf{A},N}(\bar{\mathbf{U}}_N) \ \subset \ H_N \ , \tag{4.15c}$$

and their reflections in coefficient space:

$$\bar{\mathbf{H}}_N \doteq \mathbb{R}^n \times \bar{\mathbf{U}}_N \ \subset \bar{H}_N \ , \tag{4.15d}$$

with $\bar{\mathbf{U}}_N = \bar{\mathbf{U}}^1_N$ and $V_{\mathbf{A},N} = V^1_{\mathbf{A},N}$ for representation **R1** and $\bar{\mathbf{U}}_N = \bar{\mathbf{U}}^2_N$ and $V_{\mathbf{A},N} = V^2_{\mathbf{A},N}$ for representation **R2**. We assume that $\rho_{\max}$ was chosen large enough in (3.3c) to ensure that $\mathbf{H}_N \subset \mathbf{B}$.

Remark 4.9. The constraints on $\|\bar{u}_k T_j\|_\infty$ appearing in the definition of $\mathbf{U}^1_N$ were introduced to ensure that each polynomial piece, $\sum_{j=1}^r \bar{u}_{k,j} \Phi^1_{N,k,j}(\cdot)$, of $u = V^{-1}_{\mathbf{A},N}(\bar{u})$ is Lipschitz continuous on $[t_k, t_{k+1}]$ with Lipschitz constant κ_U, independent of N. That is, $\|u_N(\tau_1) - u_N(\tau_2)\| \le \kappa_U$ for all $\tau_1, \tau_2 \in [t_k, t_{k+1}]$, $k \in \mathcal{N}$. This piecewise Lipschitz constant is needed to establish that the accuracy of the RK integration increases at least linearly with decreasing step-size (Lemma A.1 and Lemma 4.10*(i)*, but see Remark A.2). The need for this piecewise Lipschitz continuity is demonstrated in Remark 4.13. In the next section we will show that the control samples of solutions to the approximating problems we define do not depend on the control representation (Proposition 5.5). Because of this, we will conjecture (Conjecture 5.11) that the piecewise Lipschitz continuity constraints in the definition of $\bar{\mathbf{U}}^1_N$ can be dispensed with if the assumptions required for the approximating problems defined with control representation **R2** (which are strong assumptions than those required for representation **R1**) are met. □

Next, with $\eta = (\xi, u) \in H_N$ and $\bar{\eta} = (\xi, \bar{u}) = W_{\mathbf{A},N}(\eta)$, we will denote the solutions of (4.3a,b), with $\omega_k = \bar{u}_k G$, $k \in \mathcal{N}$, by $\{ \bar{x}_k^{\bar{\eta}} \}_{k=0}^{N}$ or, equivalently, $\{ \bar{x}_k^{\eta} \}_{k=0}^{N}$. The variable $\bar{x}_k^{\bar{\eta}}$ is thus the computed estimate of $x^{\eta_N}(t_k)$. Finally, for $\nu \in \mathbf{q}$, let $f_N^\nu : H_N \to \mathbb{R}$ and $\bar{f}_N^\nu : \bar{H}_N \to \mathbb{R}$ be defined by

$$f_N^\nu(\eta) \doteq \zeta^\nu(\xi, \bar{x}_N^\eta) \equiv \bar{f}_N^\nu(\bar{\eta}) \doteq \zeta^\nu(\xi, \bar{x}_N^{\bar{\eta}}) \,, \quad \nu \in \mathbf{q} \,, \tag{4.16}$$

where $\zeta^\nu(\cdot,\cdot)$ was used to define $f^\nu(\cdot)$ in (3.4c). We can now state the approximating problems as:

$$\mathbf{CP}_N \qquad \min_{\eta \in \mathbf{H}_N} \{ \psi_{o,N}(\eta) \mid \psi_{c,N}(\eta) \le 0 \} \,, \tag{4.17a}$$

where $\psi_{o,N}(\eta) \doteq \max_{\nu \in \mathbf{q}_o} f_N^\nu(\eta)$ and $\psi_{c,N}(\eta) \doteq \max_{\nu \in \mathbf{q}_c + q_o} f_N^\nu(\eta)$, or equivalently, in the form in which they must be solved numerically:

$$\overline{\mathbf{CP}}_N \qquad \min_{\bar{\eta} \in \bar{\mathbf{H}}_N} \{ \bar{\psi}_{o,N}(\bar{\eta}) \mid \bar{\psi}_{c,N}(\bar{\eta}) \le 0 \} \,, \tag{4.17b}$$

where $\bar{\psi}_{o,N}(\bar{\eta}) \doteq \max_{\nu \in \mathbf{q}_o} \bar{f}_N^\nu(\bar{\eta})$ and $\bar{\psi}_{c,N}(\bar{\eta}) \doteq \max_{\nu \in \mathbf{q}_c + q_o} \bar{f}_N^\nu(\bar{\eta})$. By defining the feasible set $\mathbf{F}_N \doteq \{ \eta \in \mathbf{H}_N \mid \psi_{c,N}(\eta) \le 0 \}$, we can write $\mathbf{CP}_N$ in the equivalent form of problem $\mathbf{P}_N$ in (2.1b).

Note that for any $u \in \mathbf{U} \cap L_N^i$, $i = 1, 2$, where $\mathbf{U}$ was defined in (3.3a), $\bar{u} = V_{\mathbf{A},N}^i(u)$ satisfies $\bar{u}_{k,j} \in U$, for $k \in \mathcal{N}$, $j \in \mathbf{r}$, because $u(t) \in U$ for all $t \in [0, 1]$. Hence, for representation **R2**, (4.15b,c) imply that $\mathbf{H} \cap H_N \subset \mathbf{H}_N$. Conversely, $\bar{u} \in \bar{\mathbf{U}}_N^2 \Longleftrightarrow (V_{\mathbf{A},N}^2)^{-1}(\bar{u}) \in \mathbf{U}$, and therefore $\mathbf{H}_N \subset \mathbf{H} \cap H_N$. Consequently, for representation **R2**, $\mathbf{H}_N = \mathbf{H} \cap H_N$. Unfortunately, for representation **R1** $\mathbf{H}_N \ne \mathbf{H} \cap H_N$. First, $\mathbf{H} \cap H_N \not\subset \mathbf{H}_N$ because elements $u \in \mathbf{U} \cap L_N^1$ do not necessarily satisfy the Lipschitz continuity constraint imposed by (4.15a). Second, if $r \ge 2$ (except for the case $r = 2$ and the Butcher array elements $c = (0, 1)$), $\mathbf{H}_N \not\subset \mathbf{H} \cap H_N$ because, given $\bar{u} \in \bar{L}_N^1$, generally $\| V_{\mathbf{A},N}^1(\bar{u}) \|_\infty > \| \bar{u} \|_\infty$, [63, p. 24]. Hence, if $\{ \eta_N = (\xi_N, u_N) \}_{N \in \mathbf{N}}$, $\mathbf{N} \subset \mathbb{N}$, is a sequence of approximate solutions to the problems $\mathbf{CP}_N$ using representation **R1**, it is possible for the u_N to violate the control constraints in **CP**. However, as we will see, the limit points of such a sequence do satisfy the control constraints in **CP**. This problem of constraint violations for representation **R1** could have been avoided by choosing $\mathbf{H}_N \doteq \mathbf{H} \cap H_N$ (as in [43]) and letting $\bar{\mathbf{H}}_N \doteq W_{A,N}(\mathbf{H}_N)$, but the set $\bar{\mathbf{H}}_N$ would then be difficult to characterize and we would have to impose a Lipschitz continuity constraint directly on the set $\mathbf{H}$, which would be unacceptable.

Nesting. The theory of consistent approximations is stated in terms of nested subspaces H_N. This allows the approximate solution of an approximating problem $\mathbf{CP}_{N_1}$ to be used as a "warm-start" for an algorithm solving an approximating problem $\mathbf{CP}_{N_2}$ with a higher discretization level ($N_2 > N_1$) (see [55,57]).

For representation **R1**, for any $N \in \mathbb{N}$, $N \geq 1$, $L_N^1 \subset L_{2N}^1$, and therefore doubling the discretization level nests the subspaces. If $u \in L_N^1$, then $\bar{v} = V_{\mathbf{A},2N}^1(u)$ can be determined from $\bar{u} = V_{\mathbf{A},N}^1(u)$ using (4.7) and (4.8), as follows. For $k \in \mathcal{N}$ and $j \in \mathbf{r}$, $\bar{v}_{2k}^j = \bar{u}_k \, T \, P(c_j / 2N)$ and $\bar{v}_{2k+1}^j = \bar{u}_k \, T \, P((c_j + 1) / 2N)$. For representation **R2**, $L_N^2 \subset L_{dN}^2$ where d is the smallest common denominator of the parameters b_j, $j \in \mathbf{s}$, in the Butcher array, which is finite assuming, as is typically the case, that the b_j are rational. Thus, the discretization level must be increased by factors of d to achieve nesting. If $u \in L_N^2$ and $\bar{u} = V_{\mathbf{A},N}^2(u)$, then $\bar{v} = V_{\mathbf{A},dN}^2(u)$ is given, for $k \in \mathcal{N}$, $i, j \in \mathbf{r}$, and $l = 1, \ldots, d$, by $\bar{v}_{dk+l}^i = \bar{u}_{k,j}$ for $d_{j-1} \leq l / d < d_j$, where d_j is defined in (4.10b).

2.4.3 Epiconvergence

We are now ready to establish epiconvergence of the approximating problems. First we present convergence properties for the solutions computed by Runge-Kutta integration on H_N. The proof of the following lemma, given in the Appendix A, differs from standard Runge-Kutta results because of the presence of (possibly discontinuous) controls in the differential equations.

Lemma 4.10. For representation **R1**, suppose that Assumptions 3.1*(a)*, 4.1' and 4.3 hold. For representation **R2**, suppose that Assumptions 3.1*(a)*, 4.1', and 4.6 hold.

(i) Convergence. For any bounded subset $S \subset \mathbf{B}$, there exist $\kappa < \infty$ and $N^* < \infty$, such that for any $\eta \in S \cap \mathbf{H}_N$ and $N \geq N^*$,

$$\| x^\eta(t_k) - \bar{x}_k^\eta \| \leq \frac{\kappa}{N} \quad , \quad k \in \{ 0, 1, \ldots, N \} . \tag{4.18a}$$

(ii) Order of Convergence. Additionally, suppose the Runge-Kutta method is order ρ, (see [61,62]) and $h(\cdot, \cdot)$ is $\rho - 1$ times Lipschitz continuously differentiable. Let

$$\mathbf{H}_N^{(\rho)} \doteq \{ \eta = (\xi, u) \in \mathbf{H}_N \mid \| \frac{d^{\rho-1}}{dt^{\rho-1}} \big(u(t_1) - u(t_2) \big) \| \leq \kappa' \;\; \forall \, t_1, t_2 \in [t_k, t_{k+1}) \, , \;\; k \in \mathcal{N} \} \tag{4.18b}$$

where $\kappa' < \infty$ is independent of N. Then for representation **R1**, there exist $\kappa < \infty$ and $N^* < \infty$ such that, if either $\eta \in S \cap \mathbf{H}_N^{(\rho)}$, or if $\eta \in S \cap \mathbf{H}_N$ and $h(x, u) = \tilde{h}(x) + Bu$, where B is an $n \times m$ constant matrix, then for any $N \geq N^*$,

$$\| x^\eta(t_k) - \bar{x}_k^\eta \| \leq \frac{\kappa}{N^\rho} \quad , \quad k \in \{ 0, 1, \ldots, N \} . \tag{4.18c}$$

Bound (4.18c) also holds for representation **R2** for any $\eta \in S \cap \mathbf{H}_N$ if $h(x, u) = \tilde{h}(x) + Bu$. □

In proving consistency, we will need to add a version of Slater's constraint qualification on the problem **CP**.

Assumption 4.11. For every $\eta \in \mathbf{H}$ such that $\psi_c(\eta) \le 0$, there exists a sequence $\{ \eta_i \}_{i=1}^{\infty}$ such that $\eta_i \in \mathbf{H}$, $\psi_c(\eta_i) < 0$, and $\eta_i \to \eta$ as $i \to \infty$. □

Theorem 4.12 (Epiconvergence). For representation **R1**, suppose that Assumptions 3.1, 4.1', 4.3 and 4.11 hold and let $d = 2$. For representation **R2**, suppose that Assumptions 3.1, 4.1', 4.6 and 4.11 hold and let d be the least common denominator for the elements b_j, $j \in \mathbf{s}$, of the Butcher array. Let $\mathbf{N} = \{ d^l \}_{l=1}^{\infty}$. Then, the problems $\{ \mathbf{CP}_N \}_{N \in \mathbf{N}}$ converge epigraphically to the problem **CP** as $N \to \infty$.

Proof. Let $S \subset \mathbf{B}$ be bounded. Then, by Assumption 3.1*(b)* and Lemma 4.10*(i)*, there exist $\kappa', \kappa < \infty$ such that for any $\nu \in \mathbf{q}$ and for any $\eta_N \in S \cap \mathbf{H}_N$,

$$|f^{\nu}(\eta_N) - f_N^{\nu}(\eta_N)| = |\zeta^{\nu}(\xi_N, x^{\eta_N}(1)) - \zeta^{\nu}(\xi_N, \bar{x}_N^{\eta_N})| \le \kappa' \|x^{\eta_N}(1) - \bar{x}_N^{\eta_N}\| \le \frac{\kappa}{N} . \quad (4.19a)$$

Now, let $\nu' \in \mathbf{q}_o$ be such that $\psi_o(\eta_N) = f^{\nu'}(\eta_N)$. Then,

$$\psi_o(\eta_N) - \psi_{o,N}(\eta_N) = f^{\nu'}(\eta_N) - \psi_{o,N}(\eta_N) \le f^{\nu'}(\eta_N) - f_N^{\nu'}(\eta_N) \le \frac{\kappa}{N} . \quad (4.19b)$$

By reversing the roles of $\psi_o(\eta_N)$ and $\psi_{o,N}(\eta_N)$ we can conclude that

$$|\psi_o(\eta_N) - \psi_{o,N}(\eta_N)| \le \frac{\kappa}{N} . \quad (4.20a)$$

Similarly,

$$|\psi_c(\eta_N) - \psi_{c,N}(\eta_N)| \le \frac{\kappa}{N} . \quad (4.20b)$$

Now, given $\eta \in \mathbf{H}$ such that $\psi_c(\eta) \le 0$, there exists, by Assumption 4.11, a sequence $S = \{ \eta_i \}_{i \in \mathbf{N}}$, with $\eta_i \in \mathbf{H}$, such that $\eta_i \to \eta$ as $i \to \infty$ (hence S is a bounded set), and $\psi_c(\eta_i) < 0$ for all i. Now, clearly for each i, there exists $N_i \in \mathbf{N}$ and $\eta'_{N_i} \in \mathbf{H}_{N_i}$ such that *(a)* $\kappa / N_i \le -1/2\psi_c(\eta_i)$, *(b)* $\|\eta_{N_i}' - \eta_i\| \le 1 / N_i$, since, for both control representations, the union of the subspaces H_N is dense in H_2 which contains $H_{\infty,2}$ and $\mathbf{H} \cap H_N \subset \mathbf{H}_N$, *(c)* $\psi_c(\eta_{N_i}') \le 1/2\psi_c(\eta_i)$ due to Theorem 3.2*(iii)*, and *(d)* $N_i < N_{i+1}$. It follows from (4.20b) that $\psi_{c,N_{i+k}}(\eta_{N_i}') \le \psi_c(\eta_{N_i}') + \kappa / N_i \le 1/2\psi_c(\eta_i) + \kappa / N_i \le 0$ for any $i, k \in \mathbb{N}$. Now consider the sequence $S'' = \{ \eta_M'' \}_{M \in \mathbf{N}}$ defined as follows: if $M = N_i$ for some $i \in \mathbf{N}$, then $\eta_M'' = \eta_{N_i}'$ for $M = N_i, N_i + d, N_i + 2d, \ldots, N_{i+1} - d$. Then we see that $\psi_{c,M}(\eta_M'') \le 0$ for all $M \in \mathbf{N}$, $\eta_M'' \to \eta$ as $M \to \infty$ (hence S'' is bounded), and by (4.20a) and Theorem 3.2*(iii)* that $\lim_{M \in \mathbf{N}} \psi_{o,M}(\eta_M'') = \psi_o(\eta)$. Thus, part *(a)* of Definition 2.1 is satisfied.

Now let $S = \{ \eta_N \}_{N \in K}$, $K \subset \mathbf{N}$, be a sequence with $\eta_N = (\xi_N, u_N) \in \mathbf{H}_N$ and $\psi_{c,N}(\eta_N) \le 0$ for all $N \in K$, and suppose that $\eta_N \to^K \eta = (\xi, u)$. First, we want to show that $\eta \in \mathbf{H}$. For any $v \in \mathbb{R}^m$, let $d(v, U) \doteq \min_{v' \in U} \| v - v' \|$. Since $\bar{u} = V^i_{\mathbf{A},N}(u_N) \in \bar{\mathbf{U}}^i_N$, $i = 1, 2$, for each N, $\bar{u}_{k,j} \in U$ for all $k \in \mathcal{N}$, $j \in \mathbf{r}$. For representation **R1**, $\overline{\lim}_{t \in [0,1], N \in K} d(u_N(t), U) = 0$ since elements, $u_N \in \mathbf{U}^1_N$ are piecewise Lipschitz continuous polynomials, with Lipschitz constant independent of N, defined over progressively smaller intervals†. For representation **R2**, $d(u_N(t), U) = 0$ for all $N \in \mathbf{N}$ and $t \in [0, 1]$ since $u_N \in \mathbf{U}^2_N$ is piecewise constant. This implies that $u \in \mathbf{U}$; hence $\eta \in \mathbf{H}$. Furthermore, $\psi_c(\eta) \le 0$ by (4.20b) and the continuity of $\psi_c(\cdot)$. Finally, by (4.20a) and Theorem 3.2*(iii)*, $\lim_{N \in K} \psi_{o,N}(\eta_N) = \psi_o(\eta)$. Thus, part *(b)* of Definition 2.1 holds. □

Remark 4.13. In [42], Hager empirically observes that RK methods with $b_j = 0$ for some j, such as the modified Euler method, cannot be used to discretize optimal control problems. This requirement, formalized in Assumption 4.1, is used in our proof of epiconvergence. However, for Representation **R1**, epiconvergence of $\mathbf{P}_N$ to $\mathbf{P}$ can be established even if, for some j, $\tilde{b}_j \le 0$. This is because of the Lipschitz continuity constraint imposed on the set $\mathbf{U}^1_N$ in (4.15a).

Nonetheless, our experimental evidence suggests that using an RK method with $\tilde{b}_j \le 0$ is unwise. For example, the three stage, third order RK method with Butcher array

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{4} & \frac{1}{4} & & \\ 1 & -\frac{7}{5} & \frac{12}{5} & \\ \hline & -\frac{1}{6} & \frac{8}{9} & \frac{5}{18} \end{array}$$

was used to discretize the problem described in Section 6 at discretization level $N = 10$. The solutions u_N^* for different values of Lipschitz constant κ_U are plotted in Figure 4.2a. For comparison, the solutions of the approximating problems produced with the third order RK method with Butcher array

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

are presented in Figure 4.2b. For both, with κ_U small, the quadratic polynomial pieces in each time interval are forced to be fairly flat. But, as κ_U is increased, the solutions for the "bad"

† It can also be shown by contradiction that $d(u_N(\cdot), U) \to 0$ a.e. on $[0, 1]$ without requiring, in (4.15a), elements of $\mathbf{U}^1_N$ to have a uniform piecewise Lipschitz constant.

method become increasingly worse and the control solutions remain pushed against the Lipschitz continuity constraints. On the other hand, the solutions for the "good" method become better as κ_U in increased. In fact, when κ_U is bigger than the Lipschitz constant of the true solution u^*, the Lipschitz continuity constraints are inactive for the "good" method (see Remark 4.9). This is seen in Figure 4.2b since the solutions for $\kappa_U = 1$ and $\kappa_U = 10$ are identical. As κ_U is increased from 0.1 to 10, the error $\max_{k,j} |{u_N}^*[\tau_{k,j}] - u^*(\tau_{k,j})|$ goes from 0.0332 to 7.9992e-4 for the "good" method and goes from 0.0332 to 1.9119 for the "bad" method.

The conditions imposed by Assumptions 4.3 and 4.6 on the c parameters of the Butcher array are needed because of the discontinuities in the controls $u \in L_N^i$, $i = 1, 2$. □

2.4.4 Factors in Selecting the Control Representation

The choice of selecting $L_N = L_N^1$ versus $L_N = L_N^2$ depends on the relative importance of approximation error versus constraint satisfaction. It follows from the proof of epiconvergence, that irrespective of which representation is used, if $\{ \eta_N \}_{N \in \mathbf{N}}$ is a sequence such that $\eta_N \in \mathbf{H}_N$, and $\eta_N \to \eta$, then $\eta \in \mathbf{H}$. Thus η satisfies the control constraints. However, as mentioned earlier, if representation **R1** is used, then η_N may not satisfy the control constraints for any finite N (except for the case $r = 2$ and $c = (0, 1)$). Since a numerical solution must be obtained after a finite number of iterations, representation **R2** should be used if absolute satisfaction of control constraints is required.

If some violation of control constraints is permissible, then representation **R1** may be preferable to representation **R2** (although, see comment about transformation of simple control bounds in Section 6) because a tighter bound for the error of the approximate solution can be established for **R1** than for **R2**. To see this, let ${\eta_N}^* = ({\xi^*}_N, {u_N}^*)$, $N \in \mathbb{N}$, be a local minimizer of the finite-dimensional problem $\mathbf{CP}_N$. This solution is computed by setting ${\eta_N}^* = W_{\mathbf{A},N}^{-1}({\bar{\eta}_N}^*)$, where ${\bar{\eta}_N}^*$ is the result of a numerical algorithm implemented on a computer using the formulae to be presented in the following sections. The error, $\|u^* - {u_N}^*\|_2$, of the approximate control solutions ${u_N}^*$ can be determined as follows. Assume that ${u_N}^* \to u^*$ as $N \to \infty$ and that u^* is a local minimizer of **CP** (if the ${u_N}^*$ solutions are uniformly strict minimizers then u^* must be a local minimizer by Theorem 2.2). Let $\vec{u}^* \in \bigtimes_N \bigtimes_r \mathbb{R}^m$ be such that ${\vec{u}^*}_{k,j} = u^*(\tau_{k,j})$, for $k \in \mathcal{N}$, $j \in \mathbf{r}$ (assuming $u^*(\tau_{k,j})$ exists). Then, with ${\bar{u}_N}^* = V_{\mathbf{A},N}({u_N}^*)$,

$$\|u^* - {u_N}^*\|_2 \leq \|u^* - V_{\mathbf{A},N}^{-1}(\vec{u}^*)\|_2 + \|V_{\mathbf{A},N}^{-1}(\vec{u}^*) - {u_N}^*\|_2 = \|u^* - V_{\mathbf{A},N}^{-1}(\vec{u}^*)\|_2 + \|\vec{u}^* - {\bar{u}_N}^*\|_{\bar{L}_N} \quad (4.21)$$

By Proposition 5.5 in the next section, the quantity $\|\vec{u}^* - {\bar{u}_N}^*\|_{\bar{L}_N}$ is not affected by the choice of control representations. For smooth, unconstrained problems discretized by symmetric RK methods, a bound for $\|\vec{u}^* - {\bar{u}_N}^*\|_\infty$ can be found in [57, Thm. 3.1] (see Proposition 6.2 in for an

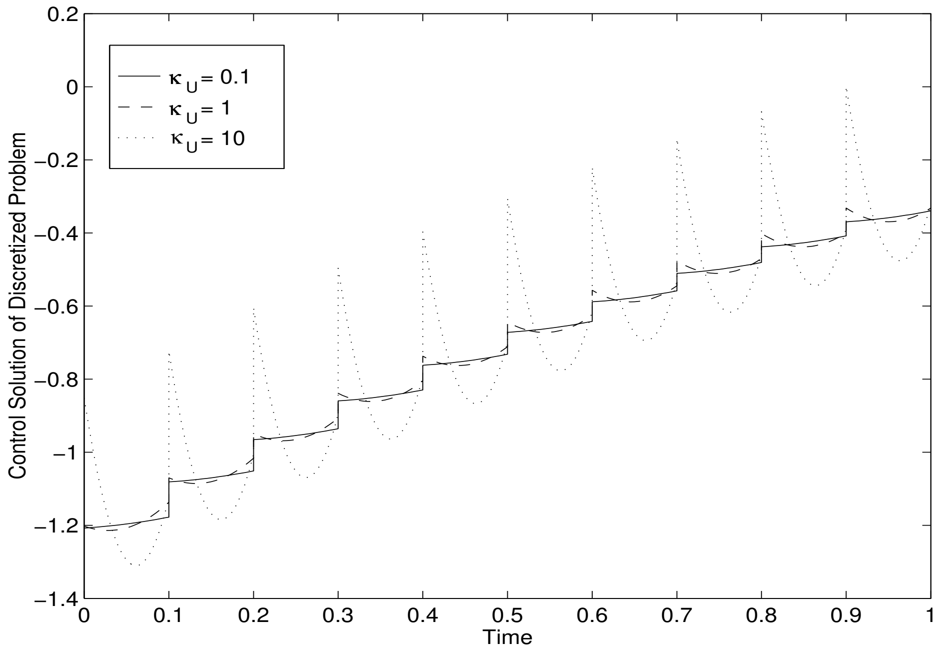


Fig. 4.2a: Effect of the Lipschitz constant κ_U on the solution of problem (6.3) discretized with an RK method that satisfies the Assumptions of Theorem 4.12 but has $b_1 < 0$. The solution gets worse as κ_U is increased.

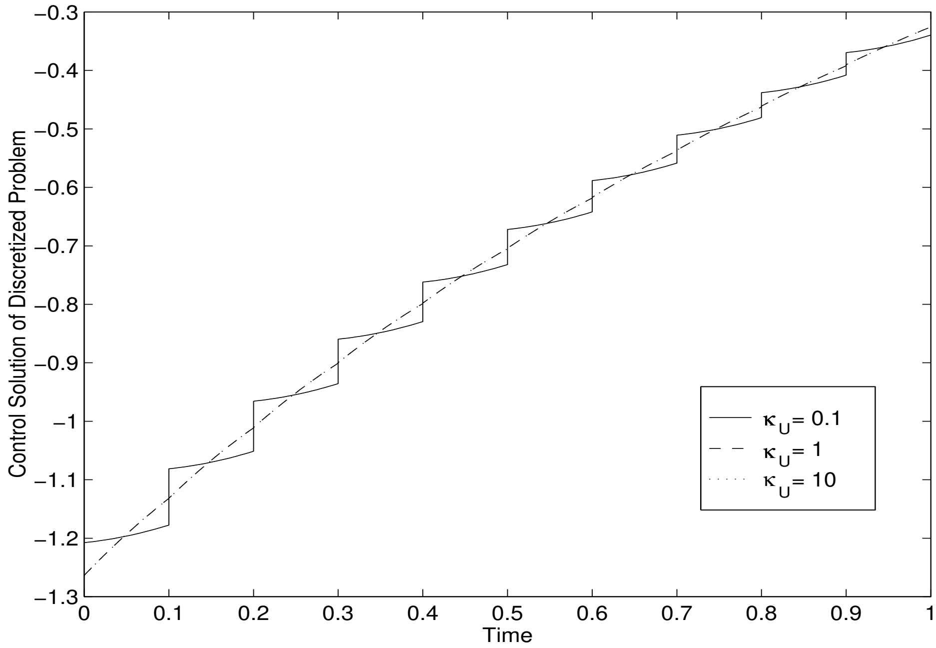


Fig. 4.2b: Effect of the Lipschitz constant κ_U on the solution of problem (6.3) discretized with an RK method that has all $b_j > 0$. The solution gets better as κ_U is increased until the point where the Lipschitz continuity constraints on u_N^* become inactive, as is the case for $\kappa_U = 1$ and $\kappa_U = 10$.

improved bound for RK4). The quantity $\|u^* - V_{\mathbf{A},N}^{-1}(\vec{u}^*)\|_2$ is the error between u^* and the element of L_N^1 or L_N^2 that interpolates $u^*(t)$ at $t = \tau_{k,j}$, $k \in \mathcal{N}$ and $j \in \mathbf{r}$. The piecewise polynomials of representation **R1** are generally better interpolators for $u^*(\cdot)$, except near non-smooth points, than the functions of **R2**. For $u^*(\cdot)$ sufficiently smooth, $\|u^* - V_{\mathbf{A},N}^{-1}(\vec{u}^*)\|_\infty$ is of order r for representation **R1** (see [63]), but only first order for representation **R2**.

2.5 OPTIMALITY FUNCTIONS FOR THE APPROXIMATING PROBLEMS

In order to develop optimality functions for our approximating problems, we must determine the gradients of the cost and constraint functions for the approximating problems.

2.5.1 Computing Gradients

At each time step, the RK integration formula is a function of the current state estimate $\bar{x}_k$ and the r control samples $\bar{u}_k = (\bar{u}_{k,1}, \ldots, \bar{u}_{k,r})$. So, let $F : \mathbb{R}^n \times (\times_s \mathbb{R}^m) \to \mathbb{R}^n$ be defined by

$$F(x, w) = x + \Delta \sum_{i=1}^{s} b_i K_i(x, wG) , \tag{5.1}$$

where $w = (w_1, \ldots, w_r) \in \times_r \mathbb{R}^m$ is being treated as the $m \times r$ matrix $[w_1 \cdots w_r]$, G was defined in (4.5d), and $K_i(x, \omega)$ was defined in (4.3b) (with $\omega = wG \in \mathbb{R}^{m \times s}$). Then, referring to (4.3a,b), we see that for any $\bar{\eta} = (\xi, \bar{u}) \in \bar{H}_N$, with $\bar{H}_N$ defined in (4.14a),

$$\bar{x}_{k+1}^{\bar{\eta}} = F(\bar{x}_k, \bar{u}_k) , \quad \bar{x}_0 = \xi , \quad k \in \mathcal{N}. \tag{5.2}$$

The derivative of $F(\cdot,\cdot)$ with respect to the j-th component of w is, with I_j defined in (4.4b),

$$\begin{aligned} F_{w_j}(x, w) &= \Delta \frac{\partial}{\partial w_j} \sum_{i=1}^{s} b_i K_i(x, wG) \\ &= \Delta \sum_{l \in I_j} \frac{\partial}{\partial \omega_l} \sum_{i=1}^{s} b_i K_i(x, \omega) \\ &= \Delta \sum_{l \in I_j} \left[b_l h_u(Y_l(x, \omega), w_j) + \Delta \sum_{i=1}^{s} b_i h_x(Y_i(x, \omega), \omega_i) \sum_{p=1}^{i-1} \frac{\partial}{\partial \omega_l} K_p(x, \omega) \right], \end{aligned} \tag{5.3a}$$

where $\omega = wG$ and $Y_i(x, \omega) \doteq x + \Delta \sum_{j=1}^{i-1} a_{i,j} K_j(x, \omega)$.

The next theorem provides an expression for the gradients of the functions $f_N^\nu(\cdot)$, $\nu \in \mathbf{q}$, given by (4.16). For $\eta = (\xi, u) \in \mathbf{H}$, we will use the notation $d_\xi \bar{f}_N(\eta) \doteq \frac{d}{d\xi} \bar{f}_N(\eta)$ and

$$d_{\bar{u}}\bar{f}_N(\eta) \doteq \left(\frac{d}{d\bar{u}_{0,1}}\,\bar{f}_N(\bar{u}) \cdots \frac{d}{d\bar{u}_{0,r}}\,\bar{f}_N(\bar{u}) \;\cdots\; \frac{d}{d\bar{u}_{N-1,1}}\,\bar{f}_N(\bar{u}) \cdots \frac{d}{d\bar{u}_{N-1,r}}\,\bar{f}_N(\bar{u}) \right), \tag{5.3b}$$

to indicate the derivative with respect to ξ and the discrete-time derivative with respect to the control samples $\bar{u} = V_{\mathbf{A},N}(u)$, of $\bar{f}_N(W_{\mathbf{A},N}(\eta))$. Note that $d_{\bar{u}}\bar{f}_N(\eta) \in \mathbb{R}^{m \times Nr}$ is a "short-fat" matrix.

Theorem 5.1 (Gradients of Approximating Functions). Let $N \in \mathbb{N}$, $\eta \in H_N$ and $\bar{\eta} = W_{\mathbf{A},N}(\eta)$. Also, let $\mathbf{M}_N \in \mathbb{R}^{Nr \times Nr}$ be the N-block diagonal matrix defined by

$$\mathbf{M}_N \doteq \operatorname{diag}[\,\Delta_0 M\,, \Delta_1 M\,, \ldots, \Delta_{N-1} M\,]\,, \tag{5.4}$$

where $M = M_1$ for representation **R1** and $M = M_2$ for representation **R2**, and again, we introduce the notation $\Delta_k \doteq t_{k+1} - t_k$ in anticipation of using non-uniform meshes in later chapters. Then, for each $\nu \in \mathbf{q}$, the gradient of $f_N^\nu(\cdot)$, $\nabla f_N^\nu : H_N \to H_N$, is given by

$$\nabla f_N^\nu(\eta) = \big(\, \nabla_\xi f_N^\nu(\eta), \nabla_u f_N^\nu(\eta) \,\big) = \big(\, d_\xi \bar{f}_N^\nu(\eta)\,, V_{\mathbf{A},N}^{-1}\big(\, d_{\bar{u}}\bar{f}_N^\nu(\eta)\,\mathbf{M}_N^{-1}\big)\big) \tag{5.5a}$$

where $V_{\mathbf{A},N} = V_{\mathbf{A},N}^1$ for representation **R1**, $V_{\mathbf{A},N} = V_{\mathbf{A},N}^2$ for representation **R2** and $d\bar{f}_N^\nu(\eta) \doteq (\, d_\xi \bar{f}_N^\nu(\eta), d_{\bar{u}}\bar{f}_N^\nu(\eta)\,) \in \bar{H}_N$ is defined by

$$d_\xi \bar{f}_N^\nu(\eta) = \nabla_\xi \zeta^\nu(\xi, \bar{x}_N^{\bar{\eta}}) + \bar{p}_0^{\nu,\bar{\eta}}\,, \tag{5.5b}$$

$$d_{\bar{u}}\bar{f}_N^\nu(\eta)_{k,j} = F_{w_j}(\bar{x}_k^{\bar{\eta}}, \bar{u}_k)^T\, \bar{p}_{k+1}^{\nu,\bar{\eta}}\,, \quad k \in \mathcal{N}, \quad j \in \mathbf{r}\,, \tag{5.5c}$$

with $\bar{p}_k^{\nu,\bar{\eta}}$ determined by the adjoint equation

$$\bar{p}_k^\nu = F_x(\bar{x}_k^{\bar{\eta}}, \bar{u}_k)^T\, \bar{p}_{k+1}^\nu\,; \quad \bar{p}_N^\nu = \zeta_x^\nu(\xi, \bar{x}_N^{\bar{\eta}})^T\,, \quad k \in \mathcal{N}, \tag{5.5d}$$

and where $F_x(\cdot,\cdot)$ and $F_{w_j}(\cdot,\cdot)$ denote the partial derivatives of $F(x, w)$ with respect to x and the j-th component of w.

Proof. First, we note that $V_{\mathbf{A},N}^1$ is invertible by Proposition 4.4 and $V_{\mathbf{A},N}^2$ is invertible by Proposition 4.7. Next, referring to [2, p. 68], we see that $d_\xi \bar{f}_N^\nu(\bar{\eta})$ is the gradient of $\bar{f}_N^\nu(\eta)$ with respect to ξ. Similarly, $d_{\bar{u}}\bar{f}_N^\nu(\eta)$ is the gradient of $\bar{f}_N^\nu(\bar{\eta})$ with respect to $\bar{u} \in \times_N \times_r \mathbb{R}^m$ using the standard l_2 (Euclidean) inner product. Hence, the Gâteaux differential of f_N^ν is given by

$$Df_N^\nu(\eta\,;\delta\eta) = D\bar{f}_N^\nu(\bar{\eta}\,;\delta\bar{\eta}) = \langle d_\xi \bar{f}_N^\nu(\eta), \delta\xi \rangle + \langle\, d_{\bar{u}}\bar{f}_N^\nu(\eta), \delta\bar{u} \rangle_{l_2}$$

$$= \langle\, d_\xi \bar{f}_N^\nu(\eta), \delta\xi \,\rangle + \langle\, d_{\bar{u}}\bar{f}_N^\nu(\eta)\,\mathbf{M}_N^{-1}, \delta\bar{u} \rangle_{\bar{L}_N}$$

$$= \langle\, d_\xi \bar{f}_N^\nu(\eta), \delta\xi \,\rangle + \langle V_{\mathbf{A},N}^{-1}(\, d_{\bar{u}} \bar{f}_N^\nu(\eta)\, \mathbf{M}_N^{-1}), \delta u \rangle_2 \ , \tag{5.6}$$

where $\delta\eta = (\delta\xi, \delta u) \in H_N$ and $\delta\bar{\eta} = (\delta\xi, \delta\bar{u}) = W_{\mathbf{A},N}(\delta\eta)$. Since by definition of $\nabla f_N^\nu(\eta)$, $Df_N^\nu(\eta\,;\delta\eta) = \langle \nabla f_N^\nu(\eta), \delta\eta \rangle_H$ for all $\delta\eta \in H_N$, the desired result follows from (5.6). □

Note that for $\eta \in H_N$, $\bar{\eta} = W_{\mathbf{A},N}(\eta)$, $d_{\bar{u}} \bar{f}_N^\nu(\eta)_k \in \bigtimes_r \mathbb{R}^m$ and

$$\big(\nabla_u f_N^\nu(\eta)[\tau_{k,i_1}] \cdots \nabla_u f_N^\nu(\eta)[\tau_{k,i_r}]\big) = \frac{1}{\Delta} \big(\, d_{\bar{u}} \bar{f}_N^\nu(\eta)_{k,1} \cdots d_{\bar{u}} \bar{f}_N^\nu(\eta)_{k,r} \,\big)\, M^{-1} \tag{5.7}$$

where $i_j \in I$, $j \in \mathbf{r}$ and $\nabla_u f_N^\nu(\eta)[\tau_{k,i_j}]$ is computed according according to (4.6e) or (4.11c).

Remark 5.2. At this point, we can draw one very important conclusion. For every $\nu \in \mathbf{q}$, the steepest descent direction, in $\bar{H}_N$, for the function $\bar{f}_N^\nu(\cdot)$, at $\bar{\eta}$, is given by $-\big(d_\xi \bar{f}_N^\nu(\eta), d_{\bar{u}} \bar{f}_N^\nu(\eta)\, \mathbf{M}_N^{-1}\big)$, and not by $-\big(d_\xi \bar{f}_N^\nu(\eta), d_{\bar{u}} \bar{f}_N^\nu(\eta)\big)$ which is the steepest descent direction that one would obtain using the standard l_2 inner product on $\bigtimes_N \bigtimes_r \mathbb{R}^m$. The naive approach of solving the discrete-time optimal control problem $\mathbf{CP}_N$ using the latter steepest descent directions amounts to a change of metric that can result in severe ill-conditioning, as we illustrate in Section 6. □

We can now define optimality functions for the approximating problems, using the form of the optimality function presented in (3.9b), for the original problem. For $\mathbf{CP}_N$, we define $\theta_N : H_N \to \mathbb{R}$, with $\sigma > 0$ and the set $\mathbf{H}_N$ defined in (4.15c), by

$$\theta_N(\eta) \doteq \min_{\eta' \in \mathbf{H}_N} \max \left\{ \max_{\nu \in \mathbf{q}_o} \tilde{f}^{\,\nu}_N(\eta, \eta') - \psi_{o,N}(\eta) - \sigma\psi_{c,N}(\eta)_+ ,\ \max_{\nu \in \mathbf{q}_c + q_o} \tilde{f}^{\,\nu}_N(\eta, \eta') - \psi_{c,N}(\eta)_+ \right\} \tag{5.8a}$$

where $\psi_{c,N}(\eta)_+ \doteq \max\{\, 0, \psi_{c,N}(\eta) \,\}$, and for $\nu \in \mathbf{q}$,

$$\tilde{f}^{\,\nu}_N(\eta, \eta') \doteq f_N^\nu(\eta) + \langle \nabla f_N^\nu(\eta), \eta' - \eta \rangle_H + \tfrac{1}{2} \| \eta' - \eta \|_H^2 \ . \tag{5.8b}$$

If needed for a particular numerical algorithm (*e.g.* [64]), $\theta_N(\eta) = \bar{\theta}_N(\bar{\eta})$, where $\bar{\eta} = W_{\mathbf{A},N}(\eta)$ and

$$\bar{\theta}_N(\bar{\eta}) \doteq \min_{\bar{\eta}' \in \bar{\mathbf{H}}_N} \tfrac{1}{2} \| \bar{\eta}' - \bar{\eta} \|_{\bar{H}_N}^2 + \Theta_N(\bar{\eta}, \bar{\eta}') \ , \tag{5.9a}$$

with

$$\Theta_N(\bar{\eta}, \bar{\eta}') = \max \Big\{ \max_{\nu \in \mathbf{q}_o} \bar{f}_N^\nu(\bar{\eta}) + \langle \big(d_\xi \bar{f}_N^\nu(\eta), d_{\bar{u}} \bar{f}_N^\nu(\eta)\, \mathbf{M}_N^{-1}\big), \bar{\eta}' - \bar{\eta} \rangle_{\bar{H}_N} - \bar{\psi}_{o,N}(\bar{\eta}) - \sigma\bar{\psi}_{c,N}(\bar{\eta})_+ \ ,$$

$$\max_{\nu \in \mathbf{q}_c + q_o} \bar{f}_N^\nu(\bar{\eta}) + \langle \big(d_\xi \bar{f}_N^\nu(\eta), d_u \bar{f}_N^\nu(\eta)\, \mathbf{M}_N^{-1}\big), \bar{\eta}' - \bar{\eta} \rangle_{\bar{H}_N} - \bar{\psi}_{c,N}(\bar{\eta})_+ \Big\} \ , \tag{5.9b}$$

and the set $\bar{\mathbf{H}}_N$ is defined in (4.15d).

It should be obvious that these optimality functions are well defined because of the form of the quadratic term and the fact that the minimum is taken over a set of finite dimension. The following theorem confirms that (5.8a) satisfies the definition for an optimality function. The proof is essentially the same as the proof in [58, Thms. 3.6,3.7].

Theorem 5.3. *(i)* $\theta_N(\cdot)$ is continuous; *(ii)* for every $\eta \in H_N$, $\theta_N(\eta) \le 0$; *(iii)* if $\hat{\eta} \in \mathbf{H}_N$ is a local minimizer for $\mathbf{CP}_N$ then $\theta_N(\hat{\eta}) = 0$. □

Remark 5.4. It can also be shown that $\theta_N(\hat{\eta}) = 0$ for $\hat{\eta} \in \mathbf{H}_N$ if and only if $d_2 \Psi_N(\hat{\eta}, \hat{\eta}\,; \eta - \hat{\eta}) \ge 0$ for all $\eta \in \mathbf{H}_N$ where

$$\Psi_N(\eta, \eta') \doteq \max \{ \psi_{o,N}(\eta) - \psi_{o,N}(\eta') - \sigma \psi_{c,N}(\eta')_+ \,, \psi_{c,N}(\eta) - \psi_{c,N}(\eta')_+ \} \ . \tag{5.9c}$$

□

Proposition 5.5. The stationary points for problem $\overline{\mathbf{CP}}_N$, that is, the points $\bar{\eta} \in \bar{\mathbf{H}}_N$ such that $\bar{\theta}_N(\bar{\eta}) = 0$, do not depend on the control representation.

Proof. First, $\bar{\eta} \in \bar{\mathbf{H}}_N$ is such that $\bar{\theta}_N(\bar{\eta}) = 0$ if and only if $\Theta_N(\bar{\eta}, \bar{\eta}') = 0$ for all $\bar{\eta}' \in \bar{\mathbf{H}}_N$. The "if" direction is obvious. For the "only if" direction, $\bar{\theta}_N(\bar{\eta}) = \min_{\hat{\eta}' \in \bar{\mathbf{H}}_N} \{ 1/2\|\bar{\eta}' - \bar{\eta}\|^2_{\bar{H}_N} + \Theta_N(\bar{\eta}, \bar{\eta}') \} = 0$. This implies that $\Theta_N(\bar{\eta}, \bar{\eta}') = 0$ because $\Theta_N(\bar{\eta}, \bar{\eta}')$ is linear in $\bar{\eta}'$ whereas $1/2\|\bar{\eta}' - \bar{\eta}\|^2_{\bar{H}_N}$ is quadratic in $\bar{\eta}'$. Second, let $\delta\bar{\eta} = (\delta\xi, \delta\bar{u}) = \bar{\eta}' - \bar{\eta}$. Then, for each $\nu \in \mathbf{q}$,

$$\langle \big(d_\xi \bar{f}^\nu_N(\eta), d_{\bar{u}} \bar{f}^\nu_N(\eta) \mathbf{M}_N^{-1} \big), \bar{\eta}' - \bar{\eta} \rangle_{\bar{H}_N} = \langle d_\xi \bar{f}^\nu_N(\eta), \delta\bar{\xi} \rangle + \langle d_{\bar{u}} \bar{f}^\nu_N(\eta), \delta\bar{u} \rangle_{l_2} \,, \tag{5.9d}$$

since $\mathbf{M}_N$ is non-singular. Hence, $\Theta_N(\bar{\eta}, \bar{\eta}')$ does not depend on the control representation. Thus, the points $\bar{\eta}$ such that $\bar{\theta}_N(\bar{\eta}) = 0$ do not depend on the control representation. □

This proposition says that the numerical solution of the discretized problem is the same for either control representation. The search directions and the control functions $(V^1_{\mathbf{A},N})^{-1}(u^*)$ and $(V^2_{\mathbf{A},N})^{-1}(u^*)$ will, of course, be different.

2.5.2 Consistency of the Approximations

To complete our demonstration of consistency of approximations we will show that the optimality functions of the approximating problems satisfy condition (2.3). In fact, we will show that the optimality of the approximating problems hypoconverge to the optimality function of the original problem (*i.e.*, $-\theta_N \xrightarrow{\text{Epi}} -\theta$). First we will present a simple algebraic condition which implies convergence of the gradients. We will use the column vector $\tilde{b} \in \mathbb{R}^r$ given by

$$\tilde{b} = (\tilde{b}_1 \cdots \tilde{b}_r)^T \tag{5.10a}$$

with components $\tilde{b}_j$ defined in (4.10a), and also the values d_j defined in (4.10b).

Theorem 5.6. For representation **R1**, suppose that Assumptions 3.1, 4.1' and 4.3 hold. For representation **R2**, suppose that Assumptions 3.1, 4.1', and 4.6 hold. For $N \in \mathbb{N}$, let H_N be defined as in (4.13a), with $L_N = L_N^1$ or $L_N = L_N^2$, and let $f_N^\nu : H_N \to \mathbb{R}$, $\nu \in \mathbf{q}$, be defined by (4.16). Let $M = M_1$ if $L_N = L_N^1$ and let $M = M_2$ if $L_N = L_N^2$. Let S be a bounded subset of $\mathbf{B}$. If

$$M^{-1}\tilde{b} = \mathbb{1} \;, \tag{5.10b}$$

where $\mathbb{1}$ is a column vector of r ones, then there exists a $\kappa < \infty$ and an $N^* < \infty$ such that for all $\eta = (\xi, u) \in S \cap \mathbf{H}_N$ and $N \geq N^*$,

$$\|\nabla f^\nu(\eta) - \nabla f_N^\nu(\eta)\|_H \leq \frac{\kappa}{N} \;. \tag{5.10c}$$

Proof. To simplify notation, we replace $\bar{x}_k^{\bar{\eta}}$ by $\bar{x}_k$, and $\bar{p}_k^{\nu,\bar{\eta}}$ by $\bar{p}_k^\nu$. Let $S \subset \mathbf{B}$ be bounded and let $\eta = (\xi, u) \in S \cap \mathbf{H}_N$. Let $\bar{u} = V_{\mathbf{A},N}(u)$ and $\bar{\eta} = (\xi, \bar{u})$ where $V_{\mathbf{A},N} = V_{\mathbf{A},N}^1$ for representation **R1** and $V_{\mathbf{A},N} = V_{\mathbf{A},N}^2$ for representation **R2**. For each $j \in \mathbf{r}$ and $k \in \mathcal{N}$, $F_{w_j}(\bar{x}_k, \bar{u}_k)$ is given by (5.3a). So, with $Y_{k,i} \doteq \bar{x}_k + \Delta \sum_{j=1}^{i-1} a_{i,j} K_j(\bar{x}_k, \omega_k)$ and $\omega_k = \bar{u}_k G$, there exists $\kappa_1 < \infty$ such that

$$\|F_{w_j}(\bar{x}_k, \bar{u}_k) - \Delta \tilde{b}_j h_u(\bar{x}_k, \bar{u}_{k,j})\|$$

$$\leq \|F_{w_j}(\bar{x}_k, \bar{u}_k) - \Delta \sum_{l \in I_j} b_l h_u(Y_{k,l}, \bar{u}_{k,j})\| + \|\Delta \sum_{l \in I_j} b_l h_u(Y_{k,l}, \bar{u}_{k,j}) - \Delta \tilde{b}_j h_u(\bar{x}_k, \bar{u}_{k,j})\|$$

$$\leq \Delta^2 \| \sum_{l \in I_j} \sum_{i=1}^{s} b_i h_x(Y_{k,i}, \omega_{k,i}) \sum_{p=1}^{i-1} \frac{\partial}{\partial \omega^l} K_p(\bar{x}_k, \omega_k)\| + \Delta \sum_{l \in I_j} b_l \|h_u(Y_{k,l}, \bar{u}_{k,j}) - h_u(\bar{x}_k, \bar{u}_{k,j})\|$$

$$\leq \kappa_1 \Delta^2 \;, \tag{5.11a}$$

where we have used the Lipschitz continuity of $h_u(\cdot,\cdot)$ and the fact that S bounded implies that $\bar{x}_k$ and $\bar{u}_{k,j}$ are bounded, which implies that for all $j \in \mathbf{r}$, $\|h_u(\bar{x}_k, \bar{u}_{k,j})\|$ and $\|h_x(\bar{x}_k, \bar{u}_{k,j})\|$ are bounded. Therefore, it follows from (5.5c) that

$$d_{\bar{u}} \bar{f}_N^\nu(\eta)_k = \left[F_{w^1}(\bar{x}_k, \bar{u}_k)^T \bar{p}_{k+1}^\nu \cdots F_{w^r}(\bar{x}_k, \bar{u}_k)^T \bar{p}_{k+1}^\nu\right]$$

$$= \Delta\left[\tilde{b}_1 h_u^T(\bar{x}_k, \bar{u}_{k,1}) \bar{p}_{k+1}^\nu \cdots \tilde{b}_r h_u^T(\bar{x}_k, \bar{u}_{k,r}) \bar{p}_{k+1}^\nu\right] + O(\Delta^2) \;, \tag{5.11b}$$

where $\lim_{\Delta \to 0} |O(\Delta)/\Delta| < \infty$. From equation (5.5a), $V_{\mathbf{A},N}(\nabla_u f_N^\nu(\eta)) = d_{\bar{u}} \bar{f}_N^\nu(\eta) \mathbf{M}_N^{-1}$. Therefore, from (5.11b) we obtain, for each $k \in \mathcal{N}$,

$$V_{\mathbf{A},N}(\nabla_u f_N^\nu(\eta))_k = \frac{\Delta}{\Delta}\left(\tilde{b}_1 h_u(\bar{x}_k, \bar{u}_{k,1})^T \bar{p}_{k+1}^\nu \cdots \tilde{b}_r h_u(\bar{x}_k, \bar{u}_{k,r})^T \bar{p}_{k+1}^\nu\right) M^{-1} + \frac{O(\Delta^2)}{\Delta} \;. \tag{5.11c}$$

At this point we must deal with our two control representations separately. For representation **R1**, $u(\cdot) \in \mathbf{U}_N^1$ is a Lipschitz continuous polynomial on each interval $[t_k, t_{k+1})$, with Lipschitz constant κ_U given in (4.15a). Thus, for any $i_j, i_l \in I$, with $j, l \in \mathbf{r}$ and I defined in (4.4a),

$$\|\bar{u}_{k,j} - \bar{u}_{k,l}\| = \|u[\tau_{k,i_j}] - u[\tau_{k,i_l}]\| \leq \kappa_U \|\Delta(c_{i_j} - c_{i_l})\| \leq \kappa_U \Delta \, , \tag{5.12}$$

where Assumption 4.3 was used to justify the last inequality. Now, let

$$D \doteq [\widetilde{b}_1 h_u^T(\bar{x}_k, \bar{u}_{k,1})\bar{p}_{k+1}^\nu \cdots \widetilde{b}_r h_u^T(\bar{x}_k, \bar{u}_{k,r})\bar{p}_{k+1}^\nu] M^{-1} \, , \tag{5.13a}$$

and let D_j, $j \in \mathbf{r}$, denote the j-th column of D, so that, from (5.11c),

$$\nabla_u f_N^\nu(\eta)[\tau_{k,i_j}] = V_{\mathbf{A},N}(\nabla_u f_N^\nu(\eta))_{k,j} = D_j + O(\Delta) \, . \tag{5.13b}$$

It follows from Assumptions 3.1 fI(a) and 4.1', equation (5.12) and the fact that $\bar{p}_{k+1}^\nu$ is bounded for any $\eta \in S$, that there exists $\kappa_2, \kappa_3 < \infty$, such that for any $j \in \mathbf{r}$ and $i_j \in I$, and with $M_{i,j}^{-1}$ denoting the i, j-th entry of M^{-1},

$$\|D_j - h_u(\bar{x}_k, \bar{u}_{k,j})^T p_{k+1}^\nu \sum_{i=1}^r \widetilde{b}_i M_{i,j}^{-1}\| \leq \| \sum_{i=1}^r \widetilde{b}_i [\, h_u(\bar{x}_k, \bar{u}_{k,i}) - h_u(\bar{x}_k, \bar{u}_{k,j}) \,]^T p_{k+1}^\nu M_{i,j}^{-1}\|$$

$$\leq \sum_{i=1}^r \kappa_2 \|\bar{u}_{k,i} - \bar{u}_{k,j}\| \, \|p_{k+1}^\nu M_{i,j}^{-1}\| \leq \kappa_3 \Delta \, . \tag{5.13c}$$

Also, if $M^{-1}\widetilde{b} = \mathbb{1}$ then $\sum_{i=1}^r M_{i,j}^{-1} \widetilde{b}_i = 1$ since M is symmetric. Hence for any $j \in \mathbf{r}$,

$$\|D_j - h_u(x_k, u_{k,j})^T \bar{p}_{k+1}^\nu\| \leq \kappa_3 \Delta \, . \tag{5.13d}$$

Therefore, from (5.13b),

$$\nabla_u f_N^\nu(\eta)[\tau_{k,i_j}] = h_u(x_k, \bar{u}_{k,j})^T \bar{p}_{k+1}^\nu + O(\Delta) \, . \tag{5.13e}$$

For representation **R2**, $\bar{u}(\cdot)$ is not Lipschitz continuous on $[t_k, t_{k+1})$, so (5.12) does not hold. However, since $M = M_2$ is diagonal, equation (5.13e) is seen to be true directly from equation (5.11c) if $M^{-1}\widetilde{b} = \mathbb{1}$.

Next, since S is bounded, *(i)* by Lemmas 4.10*(i)* and A.4 there exists $\kappa_4 < \infty$ such that $\|\bar{x}_k - x^\eta(t_k)\| \leq \kappa_4 \Delta$ and $\| \bar{p}_{k+1}^\nu - p^{\eta,\nu}(t_{k+1})\| \leq \kappa_4 \Delta$ and *(ii)* $\bar{p}_{k+1}^\nu$ and $h_u(\bar{x}^k, u[\tau_{k,i_j}])$ are bounded. Thus, making use of Theorem 3.2*(v)* and equation (5.13e), the fact that both $x^\eta(\cdot)$ and $p^{\nu,\eta}(\cdot)$ are Lipschitz continuous, and $u[\tau_{k,i_j}] = \bar{u}_{k,j}$, we conclude that there exists $\kappa_5 < \infty$ such that

$$\|\nabla_u f^\nu(\eta)[\tau_{k,i_j}] - \nabla_u f_N^\nu(\eta)[\tau_{k,i_j}]\|$$

$$= \|h_u(x^\eta(\tau_{k,i_j}), u[\tau_{k,i_j}])^T p^{\nu,\eta}(\tau_{k,i_j}) - h_u(\bar{x}_k, u[\tau_{k,i_j}])^T \bar{p}_{k+1}^\nu\| + O(\Delta) \leq \kappa_5 \Delta \, . \tag{5.14}$$

Next, for $j \in \mathbf{r}$, $i_j \in I$, $k \in \mathcal{N}$, and $t \in [0, 1]$ we have that

$$\|\nabla_u f^\nu(\eta)(t) - \nabla_u f_N^\nu(\eta)(t)\| \le \|\nabla_u f^\nu(\eta)(t) - \nabla_u f^\nu(\eta)[\tau_{k,i_j}]\| + \|\nabla_u f^\nu(\eta)[\tau_{k,i_j}] - \nabla_u f_N^\nu(\eta)[\tau_{k,i_j}]\|$$

$$+ \|\nabla_u f_N^\nu(\eta)[\tau_{k,i_j}] - \nabla_u f_N^\nu(\eta)(t)\| . \tag{5.15a}$$

The second term in (5.15a) is order $O(\Delta)$ by (5.14). We will show that the first and third terms in (5.15a) are also order $O(\Delta)$. First consider representation **R1**. It follows by inspection of (3.6b) in Theorem 3.2*(v)* that $\nabla_u f^\nu(\eta)(\cdot)$ is Lipschitz continuous on $t \in [t_k, t_{k+1})$, $k \in \mathcal{N}$, because $u \in L_N^1$ is Lipschitz continuous on these intervals. Since $\nabla_u f_N^\nu(\eta)(\cdot) \in L_N$, it is also Lipschitz continuous on these intervals. Finally, by Assumption 4.3 $\tau_{k,i_j} \in [t_k, t_{k+1}]$ for all $k \in \mathcal{N}$. Thus, the first and third terms are of order $O(\Delta)$ for all $t \in [0,1]$. For representation **R2**, $\nabla_u f_N^\nu(\eta)(\cdot) \in H_N$ is constant on $t \in [t_k + d_{j-1}, t_k + d_j)$, $j \in \mathbf{r}$ and $k \in \mathcal{N}$. Since $u \in L_N^2$ is constant on these intervals, it again follows by inspection of (3.6b) in Theorem 3.2*(v)* that $\nabla_u f^\nu(\eta)(\cdot)$ is Lipschitz continuous on these intervals. Finally, by Assumption 4.6, $\tau_{k,i_j} \in [t_k + d_{j-1}, t_k + d_j]$, for all $k \in \mathcal{N}$ and $j \in \mathbf{r}$. Since $d_0 = 0$ and $d_r = \Delta$, the first and third terms are of order $O(\Delta)$ for all $t \in [0,1]$. We conclude that there exist $\kappa_6 < \infty$ such that

$$\|\nabla_u f^\nu(\eta)(t) - \nabla_u f_N^\nu(\eta)(t)\| \le \kappa_6 \Delta \,, \quad t \in [0,1] \tag{5.15b}$$

which implies that

$$\|\nabla_u f^\nu(\eta) - \nabla_u f_N^\nu(\eta)\|_2 \le \kappa_6 \Delta \,. \tag{5.15c}$$

Next we consider the gradient with respect to initial conditions ξ. From Theorem 3.2*(v)* and (5.5b), $\|\nabla_\xi f^\nu(\eta) - d_\xi \bar{f}_N^\nu(\eta)\| \le \|\nabla_\xi \zeta^\nu(\xi, x^\eta(1)) - \nabla_\xi \zeta^\nu(\xi, \bar{x}_N)\| + \|p^{\nu,\eta}(0) - \bar{p}_0^\nu\|$. Thus, since S is bounded, it follows from Assumption 3.1*(b)* and Lemmas 4.10 and A.4 that there exists $\kappa_7 < \infty$ such that

$$\|\nabla_\xi f^\nu(\eta) - d_\xi \bar{f}_N^\nu(\eta)\| \le \kappa_7 \big(\|x^\eta(1) - \bar{x}_N\| + \|p^{\nu,\eta}(0) - \bar{p}_0^\nu\|\big) \le \kappa_7 \Delta \,. \tag{5.16}$$

Combining (5.15c) and (5.16), we see that there exists $\kappa < \infty$ such that for any $\eta_N \in S \cap H_N$,

$$\|\nabla f^\nu(\eta_N) - \nabla f_N^\nu(\eta_N)\|_H \le \frac{\kappa}{N} \,. \tag{5.17}$$

□

The following proposition states conditions for (5.10b) to hold.

Proposition 5.7.

(a) Suppose $M = M_1$. Then (5.10b) holds if and only if the coefficients of the Butcher array satisfy

$$\sum_{j=1}^{s} b_j c_j^{l-1} = \frac{1}{l}\ , \quad l = 1, \ldots, r\ . \tag{5.18}$$

(b) Suppose $M = M_2$. Then (5.10b) holds if and only if for all $j \in \mathbf{r}$, $\tilde{b}_j > 0$.

Proof. *(a)* For $M = M_1$, it follows from (4.9b) that $M^{-1}\tilde{b} = \mathbb{1}$ if and only if

$$T^{-T}\mathrm{Hilb}(s)^{-1}T^{-1}\tilde{b} = \mathbb{1}\ . \tag{5.19a}$$

Now, it is easy to see that

$$T^{-1}\tilde{b} = \begin{pmatrix} \sum_{j=1}^{r} \tilde{b}_j \\ \sum_{j=1}^{r} \tilde{b}_j c_{i_j} \\ \vdots \\ \sum_{j=1}^{r} \tilde{b}_j c_{i_j}^{r-1} \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{s} b_j \\ \sum_{j=1}^{s} b_j\, c_j \\ \vdots \\ \sum_{j=1}^{s} b_j\, c_j^{r-1} \end{pmatrix} = \begin{pmatrix} 1 \\ 1/2 \\ \vdots \\ 1/r \end{pmatrix}, \tag{5.19b}$$

where the last equality holds if and only if (5.18) holds. Note that $T^{-1}\tilde{b}$ is then the first column of $\mathrm{Hilb}(r)$. Consequently,

$$\mathrm{Hilb}(r)^{-1}T^{-1}\tilde{b} = \mathrm{Hilb}(r)^{-1}\begin{pmatrix} 1 \\ 1/2 \\ \vdots \\ 1/r \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{5.19d}$$

which leads us to conclude that

$$M^{-1}\tilde{b} = T^{-T}\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & c_{i_1} & \cdots & c_{i_1}^{r-1} \\ 1 & c_{i_2} & & c_{i_2}^{r-1} \\ & & \ddots & \\ 1 & c_{i_r} & \cdots & c_{i_r}^{r-1} \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}. \tag{5.19e}$$

(b) For $M = M_2$, given by (4.12b), M^{-1} is non-singular if and only if $\tilde{b}_j \neq 0$. Also, (5.10b) holds if and only if $M\mathbb{1} = \tilde{b}$. Clearly then, if $\tilde{b}_j \neq 0$, (5.10b) holds because

$$M\mathbb{1} = \begin{pmatrix} \tilde{b}_1 & & \\ & \ddots & \\ & & \tilde{b}_r \end{pmatrix}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \tilde{b}\ . \tag{5.20}$$

□

Remark 5.8. The conditions (5.18) on the coefficients of the Butcher array for representation **R1** are necessary conditions for the RK methods to be r-th order accurate [61,62]. The condition with $l = 1$ in (5.18) is the same as the second part of Assumption 4.1'. □

Theorem 5.9. For representation **R1**, suppose that Assumptions 3.1, 4.1' and 4.3 and equation (5.18) hold and let $d = 2$. For representation **R2**, suppose that Assumptions 3.1, 4.1', and 4.6 hold and let d be the least common denominator for the elements b_j, $j \in \mathbf{s}$, of the Butcher array. Let $\mathbf{N} \doteq \{ d^l \}_{l=1}^{\infty}$ and suppose that $\{ \eta_N \}_{N \in K}$, $K \subset \mathbf{N}$, is such that $\eta_N \in \mathbf{H}_N$ for all $N \in K$ and $\eta_N \to \eta$ as $N \to \infty$. Then $\theta_N(\eta_N) \to^K \theta(\eta)$ as $N \to \infty$.

Proof. Let $\tilde{\Psi} : \mathbf{H} \times \mathbf{H} \to \mathbb{R}$ be defined by

$$\tilde{\Psi}(\eta, \eta') \doteq \max \left\{ \max_{\nu \in \mathbf{q}_o} \tilde{f}^{\nu}(\eta, \eta') - \psi_o(\eta) - \sigma \psi_c(\eta)_+ \, , \, \max_{\nu \in \mathbf{q}_c + q_o} \tilde{f}^{\nu}(\eta, \eta') - \psi_c(\eta)_+ \right\}, \tag{5.21a}$$

and $\tilde{\Psi}_N : \mathbf{H}_N \times \mathbf{H}_N \to \mathbb{R}$ be defined by

$$\tilde{\Psi}_N(\eta, \eta') \doteq \max \left\{ \max_{\nu \in \mathbf{q}_o} \tilde{f}^{\nu}_N(\eta, \eta') - \psi_{o,N}(\eta) - \sigma \psi_{c,N}(\eta)_+ \, , \, \max_{\nu \in \mathbf{q}_c + q_o} \tilde{f}^{\nu}_N(\eta, \eta') - \psi_{c,N}(\eta) \right\}, \tag{5.21b}$$

so that, $\theta(\eta) = \min_{\eta' \in \mathbf{H}} \tilde{\Psi}(\eta, \eta')$, and $\theta_N(\eta) = \min_{\eta' \in \mathbf{H}_N} \tilde{\Psi}_N(\eta, \eta')$. Now, suppose that $\{ \eta_N \}_{N \in K}$ is a sequence such that, for all N, $\eta_N \in \mathbf{H}_N$ and $\eta_N \to^K \eta$. From the proof of Theorem 4.12, $\eta \in \mathbf{H}$. Let $\hat{\eta} \in \mathbf{H}$ be such that $\theta(\eta) = \tilde{\Psi}(\eta, \hat{\eta})$, and let $\{ \eta'_N \}_{N \in K}$ be any sequence such that, for all N, $\eta'_N \in \mathbf{H}_N$ and $\eta'_N \to^K \hat{\eta}$. Then,

$$\theta_N(\eta_N) \le \tilde{\Psi}_N(\eta_N, \eta'_N) \le \tilde{\Psi}(\eta_N, \eta'_N) +$$

$$\max \left\{ \max_{\nu \in \mathbf{q}_o} \{ \tilde{f}^{\nu}_N(\eta_N, \eta'_N) - \tilde{f}^{\nu}(\eta_N, \eta'_N) \} - [\psi_{o,N}(\eta_N) - \psi_o(\eta_N)] - [\sigma \psi_{c,N}(\eta_N)_+ - \sigma \psi_c(\eta_N)_+] \, , \right.$$

$$\left. \max_{\nu \in \mathbf{q}_c + q_o} \{ \tilde{f}^{\nu}_N(\eta_N, \eta'_N) - \tilde{f}^{\nu}(\eta_N, \eta'_N) \} - [\psi_{c,N}(\eta_N) - \psi_c(\eta_N)] \right\} \tag{5.22}$$

It follows from Theorem 4.12, Theorem 5.6, Proposition 5.7 and the fact that $\{ \eta_N \}_{N \in K}$ is a bounded set, that each part of the max term on the right hand side of (5.22) converges to zero as $N \to \infty$. The quantity $\tilde{\Psi}(\eta_N, \eta'_N)$ converges to $\theta(\eta)$ since $\eta_N \to^K \eta$, $\eta'_N \to^K \hat{\eta}$ and $\tilde{\Psi}(\cdot, \cdot)$ is continuous. Thus, taking limits of both sides of equation (5.22), we obtain that $\overline{\lim}\, \theta_N(\eta_N) \le \theta(\eta)$ (this proves that (2.3) holds for the optimality functions of the approximating problems). Now, for all $N \in K$, let $\hat{\eta}_N \in \mathbf{H}_N$ be such that $\theta_N(\eta_N) = \tilde{\Psi}_N(\eta_N, \hat{\eta}_N)$. Then, $\theta(\eta_N) \le \tilde{\Psi}(\eta_N, \hat{\eta}_N)$ and proceeding in a similar fashion as (5.22) and taking limits, we see that $\theta(\eta) \le \underline{\lim}\, \theta_N(\eta_N)$. Hence, together with the previous result, we can conclude that

$\theta_N(\eta_N) \to^K \theta(\eta)$ as $N \to \infty$. $\square$

Since the union of the spaces H_N is dense in $H_{\infty,2}$, and Theorem 5.9 holds, it follows that the hypographs of the optimality functions $\theta_N(\cdot)$ converge to the hypograph of the optimality function $\theta(\cdot)$, in the Kuratowski sense, *i.e.*, $-\theta_N(\cdot) \stackrel{\text{Epi}}{\to} -\theta(\cdot)$.

The following corollary is a direct result of Theorem 4.12 (epiconvergence) and Theorem 5.9:

Corollary 5.10. (Consistency) For representation **R1**, suppose that Assumptions 3.1, 4.1', 4.3 and 4.11 and equation (5.18) hold. For representation **R2**, suppose that Assumptions 3.1, 4.1', 4.6 and 4.11 hold. Let $\mathbf{N} = \{ d^l \}_{l=1}^{\infty}$ where $d = 2$ for representation **R1** and d is the least common denominator of the $\widetilde{b}_j$, $j \in \mathbf{s}$, for representation **R2**. Then the approximating pairs $(\mathbf{CP}_N, \theta_N)$, $N \in \mathbf{N}$, are consistent approximations to the pair $(\mathbf{CP}, \theta)$. $\square$

We conclude this section with a conjecture concerning the constraints on $\|\bar{u}_k T_j\|_\infty$ used to define $\bar{\mathbf{U}}_N^1$ in (4.15a). Recall from Remark 4.9 that these constraints impose a Lipschitz continuity constraint on the individual polynomial pieces of $u \in \mathbf{U}_N^1 = V_{\mathbf{A},N}^{-1}(\bar{\mathbf{U}}_N^1)$ that is needed to ensure accurate RK integration for controls defined by representation **R1**. Clearly, the addition of these constraints, which do not appear in the original problem **CP**, is a nuisance. Conjecture 5.11 proposes conditions under which these constraints are not needed to define consistent approximating problems $(\mathbf{CP}_N, \theta_N)$ using control representation **R1**. Assumption 4.6 (needed for control representation **R2**) is required in place of Assumption 4.3.

Conjecture 5.11. Suppose that the approximating problems $\mathbf{CP}_N$ are defined according (4.17a) with $\mathbf{H}_N \doteq \mathbb{R} \times V_{\mathbf{A},N}^{-1}(\bar{\mathbf{U}}_N^1)$ where

$$\bar{\mathbf{U}}_N^1 \doteq \{ \bar{u} \in \bar{L}_N^1 \mid \bar{u}_k^j \in U \ \ \forall\, j \in \mathbf{r}, k \in \mathcal{N} \} \ . \tag{5.23}$$

Furthermore, assume that Assumption 3.1, 4.1', 4.6 and 4.11 and (5.18) hold. Let $\mathbf{N} = \{ 2^l \}_{l=1}^{\infty}$. Then the approximating pairs $(\mathbf{CP}_N, \theta_N)$, $N \in \mathbf{N}$, are consistent approximations to the pair $(\mathbf{CP}, \theta)$. $\square$

The basis for this conjecture is the fact that, according to Proposition 5.5, the control samples of the approximating problem solutions do not depend on the control representation. Since we have shown that the approximating problems, along with their optimality functions, $(\mathbf{CP}_N, \theta_N)$, defined with control representation **R2** are consistent approximations to $(\mathbf{CP}_N, \theta_N)$, we know that the convergence results of Theorem 2.2 hold. In particular, we know that strict local minimizers of $\mathbf{CP}_N$ converge to strict local minimizers of **CP**. But this must also be true under representation **R1** with $\mathbf{CP}_N$ defined according to Conjecture 5.11 since the control samples for the sequence of solutions generated by solving these approximating problems is the same as for the approximating problems defined using representation **R2**. Thus, if a sequence $\{ \bar{u}_N{}^* \}$ of control samples corresponding to the solutions for the problems $\{ \mathbf{CP}_N \}$ is such that $(V^2_{\mathbf{A},N})^{-1}(\bar{u}_N{}^*) \to^K u^*$, then it is also true that $(V^1_{\mathbf{A},N})^{-1}(\bar{u}_N{}^*) \to^K u^*$. For each N, let $u_N{}^* = (V^1_{\mathbf{A},N})^{-1}(\bar{u}_N{}^*)$. Then, because $\{ u_N{}^* \}_{k \in K}$ is a convergent sequence, there exists $\kappa_U < \infty$ such that $\| u_N(\tau_1) - u_N(\tau_2) \| \le \kappa_U$ for all $\tau_1, \tau_2 \in [t_k, t_{k+1}]$, $k \in \mathcal{N}$. Hence, Lemma 4.10*(i)* holds which implies that Theorem 4.12 and Corollary 5.10 hold. Note that, even if this conjecture is not true, the main convergence results provided by the theory of consistent approximations do hold for the reason just presented.

2.6 COORDINATE TRANSFORMATIONS AND NUMERICAL RESULTS

The problems $\overline{\mathbf{CP}}_N$ can be solved using existing optimization methods (*e.g.*, [64,65]). These methods, however, are defined on Euclidean space and existing code would have to be modified for use on the coefficient spaces $\bar{L}^i_N$, $i = 1,2$. To avoid this difficulty, we will now define a change of coordinates in coefficient space that implicitly defines an orthonormal basis for the subspace L^i_N, and hence turns the coefficient space into a Euclidean space.

Let $L_N = L^1_N$ or L^2_N and, correspondingly, $V_{\mathbf{A},N} = V^1_{\mathbf{A},N}$ or $V^2_{\mathbf{A},N}$. Recall from (5.5a) that, for $\eta = (\xi, u) \in H_N$ and $\nu \in \mathbf{q}$, $\nabla_u f^\nu_N(\eta) = V^{-1}_{\mathbf{A},N}\big(d_{\bar{u}} \bar{f}^\nu_N(\eta)\, \mathbf{M}^{-1}_N \big)$, where $\bar{\eta} = (\xi, \bar{u}) = W_{\mathbf{A},N}(\eta)$ and $d_{\bar{u}} \bar{f}^\nu_N(\eta)$, defined in (5.5c), is the gradient of $\bar{f}^\nu_N(\cdot)$ with respect to the standard l_2 inner product on $\times_N \times_r \mathbb{R}^m$. The gradient of $\bar{f}^\nu_N(\cdot)$ with respect to the inner product on $\bar{L}_N$ is given by $\nabla_u \bar{f}^\nu_N(\bar{\eta}) = V_{\mathbf{A},N}\big(\nabla_u f^\nu_N(\eta) \big) = d_{\bar{u}} \bar{f}^\nu_N(\eta)\, \mathbf{M}^{-1}_N$, and satisfies

$$\langle \nabla_u f^\nu_N(\eta), \delta u_N \rangle_2 = \langle \nabla_u \bar{f}^\nu_N(\bar{\eta}), \delta \bar{u} \rangle_{\bar{L}_N} = \langle d_{\bar{u}} \bar{f}^\nu_N(\eta), \delta \bar{u} \rangle_{l_2} , \tag{6.1}$$

for any $\delta u \in H_N$ and $\delta \bar{u} = V_{\mathbf{A},N}(\delta u)$. Introduce a new coefficient space, $\tilde{L}_N = \times_N \times_r \mathbb{R}^m$, endowed with the standard l_2 inner product and norm, and a transformation $Q : \bar{L}_N \to \tilde{L}_N$ defined by

$$\tilde{u} = Q(\bar{u}) = \bar{u}\, \mathbf{M}^{1/2}_N , \tag{6.2a}$$

where $\mathbf{M}_N$ is defined in (5.4). Let $\tilde{\eta} = (\xi, \tilde{u})$ and for each $\nu \in \mathbf{q}$, let $\tilde{f}^\nu_N : \mathbb{R}^n \times \tilde{L}_N \to \mathbb{R}$ be defined by

$$\boxed{\tilde{f}^\nu_N(\tilde{\eta}) \doteq \bar{f}^\nu_N\big((\xi, \tilde{u}\, \mathbf{M}^{-1/2}_N) \big)} . \tag{6.2b}$$

Finally, let $\bar{\eta} = (\xi, Q^{-1}(\tilde{u}))$. Then, by the chain rule,

$$\boxed{\nabla_{\tilde u} \tilde f^{\nu}_N(\tilde\eta) = Q^{-1}\big(\nabla_{\bar u}\bar f^{\nu}_N(\bar\eta)\big) = d_u \bar f^{\nu}_N(\eta)\,\mathbf{M}_N^{-1/2}} \,. \tag{6.2c}$$

Thus, $\langle \nabla_{\tilde u} \tilde f^{\nu}_N(\tilde\eta), \delta\tilde u\rangle_{l_2} = \langle \nabla_{\bar u}\bar f^{\nu}_N(\bar\eta), \delta\bar u\rangle_{\bar L_N} = \langle \nabla_u f^{\nu}_N(\eta), \delta u\rangle_2$, where $\delta\tilde u = Q(\delta\bar u)$.

Remark 6.1. Implicitly, the transformation Q creates an orthonormal basis for L_N because under this transformation the inner-product and norm on L_N are equal to the l_2 inner-product and norm on the coefficient space. With this transformation, the approximating problems $\overline{\mathbf{CP}}_N$ can be solved using standard nonlinear programming methods without introducing ill-conditioning. It is important to note, however, that control constraints are also transformed. Thus, the constraint $\bar u \in \bar{\mathbf{U}}_N$ becomes $\tilde u\,\mathbf{M}_N^{-1/2} \in \bar{\mathbf{U}}_N$. For representation **R1**, since $\mathbf{M}_N^{-1/2}$ is not diagonal (except if $r = 1$), this means that the transformed control constraints will, for each $k \in \mathcal{N}$, involve linear combinations of the control samples $\tilde u_{k,j}$, $j \in \mathbf{r}$.

We will now present a numerical example that shows, in particular, that this transformation can make a substantial difference in the performance of an algorithm.

Example. Consider the following linear-quadratic problem taken from [42]:

$$\min_{u\,\in\,\mathbf{U}} f(u)\,,\quad f(u) \doteq x_2^u(1)\,, \tag{6.3a}$$

where $x(t) = (x_1(t), x_2(t))^T$ and

$$\dot x = \begin{bmatrix} 0.5x_1 + u \\ 0.625x_1^2 + 0.5x_1u + 0.5u^2 \end{bmatrix},\quad x(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\quad t \in [0,1]\,. \tag{6.3b}$$

The solution to this problem is given by

$$u^*(t) = -(\tanh(1-t) + 0.5)\cosh(1-t)/\cosh(1)\,,\quad t \in [0,1]\,, \tag{6.4}$$

with optimal cost $f(u^*) = e^2\sinh(2)/(1+e^2)^2 \approx 0.380797$.

The approximating cost functions are $f_N(u) = (\,0\;1\,)\,\bar x^u_N$ where $\{\,\bar x^u_k\,\}_{k=0}^N$ is the RK solution for a given control $u \in L_N$. We discretized the dynamics using the following common RK methods of order 3 and 4 respectively:

$$\mathbf{A}_3 \;=\; \begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \qquad\qquad \mathbf{A}_4 \;=\; \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

The matrices $\mathbf{M}_N$ used to define the transformation Q in (6.2a) are given by (4.9d) and (4.12c) with

$$M_1 = \frac{1}{30}\begin{bmatrix} 4 & 2 & -1 \\ 2 & 16 & 2 \\ -1 & 2 & 4 \end{bmatrix}, \qquad M_2 = \frac{1}{6}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.5}$$

The matrices M_1 and M_2 for method $\mathbf{A}_3$ are the same as for method $\mathbf{A}_4$ since $c_2 = c_3 = 1/2$ implies $r = 3$ and $\tilde{b}_2 = 2/3$.

We solved the approximating problems using steepest descent with the step-size determined by an Armijo rule augmented with a quadratic fit based on the value of $f_N(\cdot)$ at the last two evaluations in the line search. The stopping criterion was $\|d_{\bar{u}} \bar{f}_N\|_2 \le (3.1e-4)/N$† and the initial guess was $u(t) = 0,\ t \in [0,1]$. Table 6.1 shows the number of iterations required to solve the approximating problems for different discretization levels N with and without the transformation (6.2a,b). We see that solving the discretized problems without the transformation requires about five times the number of iterations required for solving the problem with the transformation. The situation can be even worse for other RK methods. The choice of representation **R1** versus representation **R2** and the RK method had no effect on the number of iterations required.

	Number of Iterations	
N	$M = M_i\ ,\ i = 1,2$	$M = \frac{1}{N} I$
10	4	19
20	4	19
40	5	23
80	5	24

Table 6.1: Conditioning Effect of the Transformation Q on Approximating Problems (RK3).

We will now show why it is advantageous to treat the repeated control samples for method $\mathbf{A}_4$ as a single sample (*cf.* Remark 4.9). Let $\{ u_N^* \}_{N \in \mathbf{N}}$, $\mathbf{N} \subset \mathbb{N}$, be solutions of $\mathbf{CP}_N$ and

† Higher precision was difficult to achieve when the Q transformation was not used.

suppose ${u_N}^* \to u^*$ where u^* is a solution of **CP**. In [57, Thm. 3.1], Hager establishes, for symmetric RK methods [66,67], a tight upper bound on the error $E_N^u \doteq \|V_{\mathbf{A},N}(u^*) - V_{\mathbf{A},N}({\bar{u}_N}^*)\|_{l_\infty}$, of second order in $\Delta = 1/N$ for smooth, unconstrained problems. Note that $V_{\mathbf{A},N}(u^*)_{k,j} = u^*(\tau_{k,j})$, $k \in \mathcal{N}$ and $j \in \mathbf{r}$ because $u^*(\cdot)$ is continuous (in fact, smooth) for smooth problems [28,36]. Hager used the problem given in (6.3) to demonstrate the tightness of this bound. For the particular RK method given by the Butcher array $\mathbf{A}_2$, we can state the following improved result (which, according to Proposition 5.5, does not depend on the control representation):

Proposition 6.2. Let $\mathbf{CP} \doteq \min_{u \in \mathbf{U}} f(x^u(1))$, u unconstrained. Suppose the approximating problems $\mathbf{CP}_N$ are produced by discretizing **CP** with the fourth order RK method with Butcher array $\mathbf{A}_4$. Further, suppose that $\|x^{u^*}(1) - \bar{x}^{u^*{}_{N,N}}\| = O(\Delta^4)$, that is, at the RK integration is fourth order accurate at u^*. Let $\{ {u_N}^* \}_{N \in \mathbf{N}}$, $\mathbf{N} \subset \mathbb{N}$, be solutions of $\mathbf{CP}_N$ and suppose ${u_N}^* \to u^*$ where u^* is a solution of **CP**. Then $E_N^u \doteq \|V_{\mathbf{A},N}(u^*) - V_{\mathbf{A},N}({u_N}^*)\|_{l_\infty} = O(\Delta^3)$.

Sketch of Proof. In [42], it is shown, using a reasonable non-singularity assumption on the Hessians of $f_N(\cdot)$, that the accuracy of the solutions of the approximating problems is determined by N times the size of the discrete-time derivative (using the standard l_2 inner-product) of the approximating problem at $\vec{u}^* \doteq V_{\mathbf{A},N}(u^*)$, that is, $E_N^u \sim N\|d_{\bar{u}}\bar{f}_N(\vec{u}^*)\|$. This, in turn, is a function of the accuracy of the state and adjoint approximations. For the RK method under consideration, Hager shows that, for $k \in \mathcal{N}$, the variables $\vec{u}^*{}_{k,1} = u^*(t_k)$ and $\vec{u}^*{}_{k,3} = u^*(t_k + \Delta)$ are third order approximations to $u^*(t_k)$ and $u^*(t_k + \Delta)$, respectively. Thus, we need only show that $\vec{u}^*{}_{k,2} = u^*(t_K + \Delta/2)$ is a third order approximation to $u^*(t_k + \Delta/2)$.

Let $Y_{k,2} = \bar{x}_k + \frac{\Delta}{2} h(\bar{x}_k, \bar{u}_{k,1})$ and $Y_{k,3} = \bar{x}_k + \frac{\Delta}{2} h(Y_{k,2}, \bar{u}_{k,2})$ represent intermediate values used by the RK method at the k-th time-step. In Hager's notation, $Y_{k,2} = y(1,k)$ and $Y_{k,3} = y(2,k)$. Hager introduces a clever transformation, specific to symmetric RK methods, for the adjoint variables so that they can be viewed as being calculated with the same RK method used to compute the state variables, but run backwards in time. The intermediate adjoint variables of interest here are denoted by $q(2,k)$ and $q(1,k)$. With this transformation, the discrete-time derivative for the approximating problems have the same form as the continuous-time gradient for the original problem. Since $c_2 = c_3 = 1/2$, $d_{\bar{u}}\bar{f}_N(\vec{u}^*)_{k,2} = \frac{2}{3}\Delta[\frac{1}{2} h_u(Y_{k,2}, \vec{u}^*{}_{k,2})^T q(1,k) + \frac{1}{2} h_u(Y_{k,3}, \vec{u}^*{}_{k,2})^T q(2,k)]$. Further, since $2\frac{\Delta}{3} h_u(x^{u^*}(t_k + \frac{\Delta}{2}), u^*(t_k + \frac{\Delta}{2}))^T p^{u^*}(t_k + \frac{\Delta}{2}) = 0$, $\|d_{\bar{u}}\bar{f}_N(\vec{u}^*)_{k,2}\|$ is bounded by $\frac{2}{3}\Delta$ times the maximum of $\|(Y_{k,2} + Y_{k,3})/2 - x^{u^*}(t_k + \frac{\Delta}{2})\|$ and $\|(q(2,k) + q(1,k))/2 - p^{u^*}(t_k + \frac{\Delta}{2})\|$. Let

$$w(k) \doteq \frac{Y_{k,2} + Y_{k,3}}{2} = \bar{x}_k + \frac{\Delta}{4}\,[h(\bar{x}_k, \vec{u}^*{}_{k,1}) + h(\bar{x}_k + \frac{\Delta}{2} h(\bar{x}_k, \vec{u}^*{}_{k,1}), \vec{u}^*{}_{k,2})]$$

$$= \bar{x}_k + \frac{\Delta'}{2}\,[h(\bar{x}_k, \vec{u}^*_{k,1}) + h(\bar{x}_k + \Delta' h(\bar{x}_k, \vec{u}^*_{k,1}), \vec{u}^*_{k,2})]\,, \tag{6.6}$$

where $\Delta' = \Delta/2$. Thus, w(k) is produced by the improved Euler rule applied to $\bar{x}_k$. Since the local truncation error for the improved Euler rule is order $O(\Delta^3)$ and $\bar{x}_k$ is order $O(\Delta^4)$, $\| w(k) - x^{u^*}(t_k + \frac{\Delta}{2)}\|$ is order $O(\Delta^3)$. In the same way, it can be shown that $\| q(2,k) + q(1,k))/2 - p^{u^*}(t_k + \frac{\Delta}{2})\|$ is $O(\Delta^3)$. Thus, we can conclude that $\| d_{\bar{u}} \bar{f}_N(u^*)_{k,2}\| = O(\Delta^4)$ for all $k \in \mathcal{N}$. This implies that the solutions of the approximating problems satisfy $\| \vec{u}^*_{N\,k,j} - u^*(\tau_{k,i_j})\| = O(\Delta^3)$ for all $k \in \mathcal{N}$ and $j \in \mathbf{r}$. □

Table 6.2 summarizes our numerical results using the RK method with Butcher array $\mathbf{A}_4$. The first column gives the discretization level. Columns 2 and 3 show that doubling the discretization results in an eight-fold reduction in the control error. Thus, as predicted by Proposition 6.2, E^u_N is $O(\Delta^3)$. The next two columns, agreeing with Hager's observations that the optimal trajectories of the approximating problem converge to those of the original problem with the same order as the order of the symmetric RK method, show that $E^f_N \doteq |f(u^*) - f_N(\bar{u}_N{}^*)|$ is order $O(\Delta^4)$. The numbers in columns 2 and 4 were obtained by solving the discretized problems to full precision. Finally, we include in the last two columns the number of iterations required to solve the approximate problem with and without the transformation Q. The stopping criterion was the same as used for Table 6.1. As with the previous method, the effect of the Q transformation is quite significant. The solution of the untransformed problem requires about five times the number of iterations required to solve the transformed problem.

	Accuracy of Solutions				Number of Iterations	
N	E^u_N	E^u_N / E^u_{2N}	E^f_N	E^f_N / E^f_{2N}	$M = M_i\,,\ i = 1,2$	$M = \frac{1}{N} I$
10	1.48e-4	7.91	2.86e-7	16.22	4	21
20	1.87e-5	7.99	1.76e-8	16.13	5	21
40	2.34e-6	7.62	1.09e-9	16.07	5	23
80	3.07e-7		6.80e-11		5	23

Table 6.2: Order of Convergence; Conditioning Effect of the Transformation Q (RK4).

The last table shows the accuracy of the gradients for the approximating problems produced with the second RK method (Butcher array $\mathbf{A}_4$) evaluated at the control $u(t) = -1 + 2t$. The first column shows the discretization level N. The second and third columns confirm that the gradients, $\nabla \bar{f}_N(\bar{u}) = d_{\bar{u}} \bar{f}_N(u) \mathbf{M}_N^{-1}$, for the approximating problems converge to the gradients of the original problem. Note that, based on the proof of Theorem 5.6, it is enough to show that the gradients converge at the points τ_{k,i_j}, $k \in \mathcal{N}$, $j \in \mathbf{r}$, and $i_j \in I$. The fourth column of Table 6.3 shows that the gradients that result if one uses the standard l_2 inner product on $\bigtimes_N \bigtimes_r \mathbb{R}^m$ do not converge.

N	$M = M_1$ $\|V_{\mathbf{A},N}(\nabla f(u)) - \nabla \bar{f}_N(\bar{u})\|_{l_\infty}$	$M = M_2$ $\|V_{\mathbf{A},N}(\nabla f(u)) - \nabla \bar{f}_N(\bar{u})\|_{l_\infty}$	$M = \frac{1}{N} I$ $\|V_{\mathbf{A},N}(\nabla f(u)) - N d_{\bar{u}} \bar{f}_N(\bar{u})\|_{l_\infty}$
10	1.67e-3	6.46e-4	1.48
20	3.77e-4	8.31e-5	1.48
40	9.94e-5	1.05e-5	1.48
80	2.55e-5	1.33e-6	1.48

Table 6.3: Convergence of Gradients.

2.7 APPROXIMATING PROBLEMS BASED ON SPLINES

In this section, we use splines as the finite dimensional basis elements in the construction of approximating problems for optimal control problems with endpoint inequality constraints and box-type control constraints. One of the early references that used splines for the solution optimal control problems is [35]. We show that the resulting approximating problems, along with their optimality functions, are consistent approximations to the original problem with its optimality function assuming that Conjecture 5.11 is true. In the process, we will develop some results for splines that are interesting for their own sake. For clarity, the results below are stated only in terms of control variables u rather than the initial state/control pair $\eta = (\xi, u)$. The treatment of variable initial conditions is unaffected by the use of splines.

We will construct our finite dimensional control spaces using spline basis functions (B-splines). Thus, for $r \in \mathbb{N}$, $r \geq 1$, let

$$L_N^{(r)} \doteq \{\, u \in L_{\infty,2}^m[0,1] \;\mid\; u(t) = \sum_{k=1}^{N+r-1} \alpha_k \phi_k(t)\,,\;\; t \in [0,1] \,\}\;, \tag{7.1a}$$

where $\alpha_k \in \mathbb{R}^m$, $\phi_k : [0,1] \to \mathbb{R}$ are the basis function with $\phi_k(t) = B_{k,r,\mathbf{t}_N}(t)$, defined below, and r is the order (one more than the degree) of the polynomials that make up the spline pieces. For an excellent presentation of spline theory, we refer the interested reader to [63]. The subscript $\mathbf{t}_N$ in $B_{k,r,\mathbf{t}_N}(t)$ is the knot sequence upon which the B-splines are defined. We will not consider knot sequences with repeated interior knots although many of our results hold in that case also.

Rather, we will consider two knot sequences which are constructed by adding endpoint knots to the set of breakpoints $\{ t_k \}_{k=0}^{N}$ (note that, unlike in [63], our breakpoint sequence begins with the index $k = 0$ rather than $k = 1$.). The two knot sequences are

Uniform knot sequence. The knot sequence is

$$\mathbf{t}_N \doteq \{ k / N \}_{k=-r+1}^{N+r-1} . \tag{7.1b}$$

General knot sequence. The knot sequence is

$$\mathbf{t}_N \doteq \{ t_k \}_{k=-r+1}^{N+r-1} \tag{7.1c}$$

where $\{ t_k \}$ is a sequence of not necessarily uniformly spaced breakpoints which satisfy

$$t_{-r+1} = \cdots = t_0 < t_1 < \cdots < t_{N-1} < t_N = \cdots = t_{N+r-1} . \tag{7.1d}$$

The uniform knot sequence can only be used for uniformly spaced breakpoints. The purpose of its introduction is solely to make some results cleaner and easier to see. The spacing of the breakpoints, $\{ t_k \}_{k=0}^{N}$, for the general knot sequence may or may not be uniform. In our notation, the knot sequences begin with index $k = -r + 1$ (rather than $k = 1$ as in [63]).

With these knot sequences, the B-splines constitute a basis for the $N + r - 1$ dimensional space of $r-2$ times continuously differentiable splines of order r with breakpoints at times $\{ t_k \}_{k=0}^{N}$. Since splines are just piecewise polynomials between breakpoints with continuity and smoothness constraints at the breakpoints, $L_N^{(r)} \subset L_N^1$ where L_N^1 is defined in Section 4 for representation **R1** with r-th order polynomial pieces. The control samples, $u[\tau_{k,j}]$, $k = 0, \ldots, N-1$, $j \in \mathbf{r}$, used by the RK integration method given in (4.3a,b) are related to the spline coefficients by $u[\tau_{k,j}] = \sum_{k=1}^{N+r-1} \alpha_k \phi_k(\tau_{k,j})$.

We will use B-splines normalized so that $\sum_{k=1}^{N+r-1} B_{k,r,\mathbf{t}_N}(t) = 1$ for all $t \in [0,1]$. For a given knot sequence, these B-splines can be written (see [63]) in terms of the following recursion on the spline order r:

$$B_{k,r+1,\mathbf{t}_N}(t) = \frac{t - t_{k-r-1}}{t_{k-1} - t_{k-r-1}} B_{k-1,r,\mathbf{t}_N}(t) + \frac{t_k - t}{t_k - t_{k-r}} B_{k,r,\mathbf{t}_N}(t) \quad , \quad r \geq 1 , \tag{7.2a}$$

$$B_{k,1,\mathbf{t}_N}(t) = \begin{cases} 1 \, , & t_{k-1} \leq t < t_k \\ 0 \, , & \text{otherwise} \end{cases} . \tag{7.2b}$$

If $\mathbf{t}_N$ is the uniform knot sequence, the domain of the B-splines extends outside of the range $t \in [0,1]$. This is for the purpose of construction only; the functions $u(t)$, given by (7.1a), are defined only on $t \in [0,1]$. An important feature of B-splines is that the support of each basis function, $\phi_k(t)$, is limited to $[t_{k-r}, t_k]$. This is important for efficient computation of $u(t)$ from its spline coefficients and of gradients for the cost and constraint functions.

As an example of B-splines defined on a uniform knot sequence, we express the basis functions for cubic splines ($r = 4$) explicitly (compare with [35] where the B-spline normalization is different):

$$B_{k,4,\mathbf{t}_N}(t) = \frac{1}{6\Delta^3}\begin{cases} (t - t_{k-4})^3 \ , & t_{k-4} \le t \le t_{k-3} \\ \Delta^3 + 3\Delta^2(t - t_{k-3}) + 3\Delta(t - t_{k-3})^2 - 3(t - t_{k-3})^3 \ , & t_{k-3} \le t \le t_{k-2} \\ 4\Delta^3 - 6\Delta(t - t_{k-2})^2 + 3(t - t_{k-2})^3 \ , & t_{k-2} \le t \le t_{k-1} \\ \Delta^3 - 3\Delta^2(t - t_{k-1}) + 3\Delta(t - t_{k-1})^2 - (t - t_{k-1})^3 \ , & t_{k-1} \le t \le t_k \end{cases} , \tag{7.2c}$$

where $\Delta = 1/N$. As another example, we plot the B-spline functions for a quadratic spline defined on the general knot sequence $\{ 0, 0, 0, 0.1, 0.25, 0.3, 0.4, 0.4, 0.4 \}$ in Figure 7.1.

We now formulate the approximating problems using splines. The control constraint sets for the approximating problems are defined as,

$$\mathbf{U}_N^{(r)} \doteq \{ u \in L_N^{(r)} \mid \alpha_k \in U, \ k = 1, \ldots, N + r - 1 \} \tag{7.3}$$

where, for this section, we assume that U, used to define $\mathbf{U}$ in (3.3a), is of the form

---

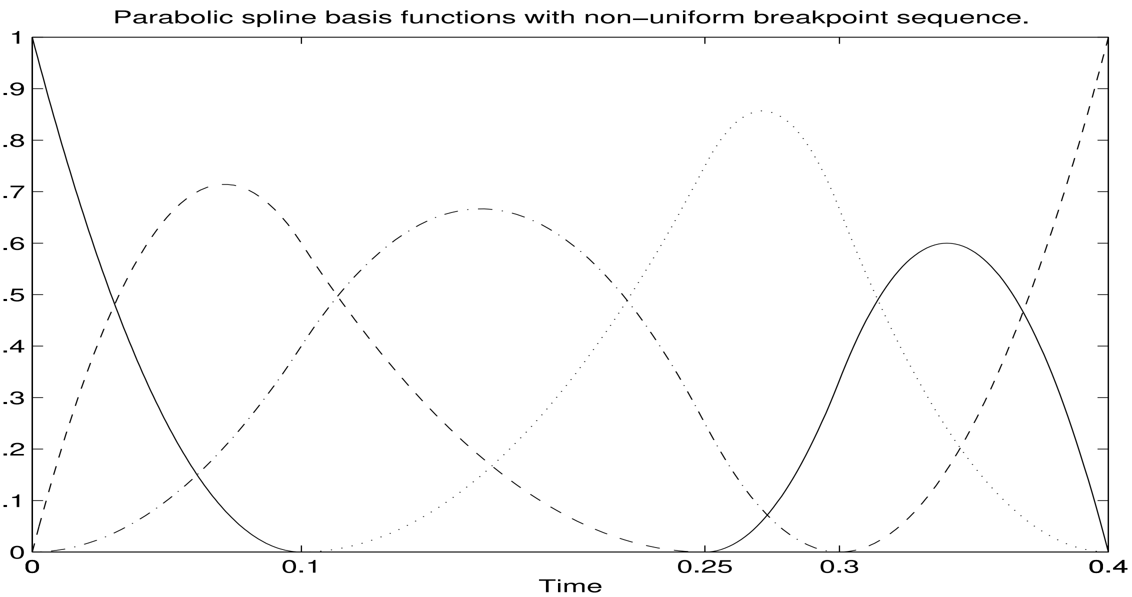


Fig. 7.1: A plot of the six B-spline functions used to construct quadratic splines defined on this general knot sequence.

$$U \doteq \{ v = [v^1, \ldots, v^m]^T \in \mathbb{R}^m \mid -\infty < a^i \le v^i \le b^i < \infty \,,\ i = 1, \ldots, m \} \ . \tag{7.3b}$$

Thus, the spline coefficients for each component of the control have constant bounds.

The approximating problems are thus:

$$\mathbf{CP}_N \qquad \min_{u \in \mathbf{U}_N^{(r)}} \{ \psi_{o,N}(u) \mid \psi_{c,N}(u) \le 0 \} \ , \tag{7.3c}$$

with $\psi_{o,N}(u)$ and $\psi_{c,N}(u)$ as defined in (4.17). We will keep the definition of the optimality functions the same as given in Section 5. Note that the decision parameters for these problems transcribed into coefficient space are the coefficients α_k, $k = 1, \ldots, N + r - 1$, rather than the Nr control samples $u[\tau_{k,j}]$, $k = 0, \ldots, N-1$, $j = 1, \ldots, r$ for the approximating problems defined in Section 4. Thus, the number of decision parameters needed for splines is substantially less than the number needed for the same order general piecewise polynomials. This is the motivation for using splines.

The next three results state properties of the spline subspaces that are needed to prove epi-convergence of the approximating problems to the original problem. Proposition 7.1 and Corollary 7.2 apply only to uniform knot sequences. Corollary 7.2 is a non-recursive version of the subdivision result given in [68, Thm 3.1]; the method of proof is completely different. A similar result for general knots sequences, expressed in terms of a recursion formula, can be found in [69,70].

Proposition 7.1. (Nesting of Basis Functions) Given an integer $\rho \ge 1$, let $\mathbf{t}_N = \{ k/N \}_{k=-\rho+1}^{k=N+\rho-1}$ and $\mathbf{t}_N{}' = \{ k/2N \}_{k=-\rho+1}^{k=2N+\rho-1}$. Then,

$$B_{k,\rho,\mathbf{t}_N}(t) = \frac{1}{2^{\rho-1}} \sum_{i=1}^{\rho+1} \sigma_{\rho,i} B_{2k-\rho+i-1,\rho,\mathbf{t}_N{}'}(t) \ ,\ \ k = 1, \ldots, N + \rho - 1 \ , \tag{7.4}$$

where $\sigma_{\rho,i}$ is the i-th coefficient of the polynomial $(t+1)^\rho$.

Proof. We prove (7.4) by induction on ρ. It is clear from (7.2b) that (7.4) holds for $\rho = 1$. Now we will show that if (7.4) holds for $\rho = r$, then it holds for $\rho = r + 1$. From (7.2a),

$$B_{k,r+1,\mathbf{t}_N}(t) = \frac{t - t_{k-r-1}}{r\Delta} B_{k-1,r,\mathbf{t}_N}(t) + \frac{t_k - t}{r\Delta} B_{k,r,\mathbf{t}_N}(t) \ . \tag{7.5a}$$

Substituting (7.4) into this expression while letting $\Delta' = \Delta/2$ and $t_k = t'_{2k}$ gives us

$$B_{k,r+1,\mathbf{t}_N}(t) = \frac{t - t'_{2(k-r-1)}}{2r\Delta'} \frac{1}{2^{r-1}} \sum_{i=1}^{r+1} \sigma_{r,i} B_{2(k-1)-r+i-1,r,\mathbf{t}_N{}'}(t) + \frac{t'_{2k} - t}{2r\Delta'} \frac{1}{2^{r-1}} \sum_{i=1}^{r+1} \sigma_{r,i} B_{2k-r+i-1,r,\mathbf{t}_N{}'}(t)$$

$$= \frac{1}{2^r}\left\{\sigma_{r,1}\,\frac{t - t'_{2(k-r-1)}}{r\Delta'}\,B_{2(k-1)-r,r,\mathbf{t}_N'}(t) + \sigma_{r,2}\,\frac{t - t'_{2(k-r-1)}}{r\Delta'}\,B_{2k-1-r,r,\mathbf{t}_N'}(t)\right.$$

$$+ \sum_{j=2}^{r}\left(\sigma_{r,j+1}\,\frac{t - t'_{2(k-r-1)}}{r\Delta'} + \sigma_{r,j-1}\,\frac{t'_{2k} - t}{r\Delta'}\right)B_{2(k-1)-r+j,r,\mathbf{t}_N'}(t)$$

$$\left. + \sigma_{r,r}\,\frac{t'_{2k} - t}{r\Delta'}\,B_{2k-1,r,\mathbf{t}_N'}(t) + \sigma_{r,r+1}\,\frac{t'_{2k} - t}{r\Delta'}\,B_{2k,r,\mathbf{t}_N'}(t)\right\}$$

$$= \frac{1}{2^r}\left\{\sigma_{r,1}\,\frac{t - t'_{2(k-r-1)}}{r\Delta'}\,B_{2(k-1)-r} + \sigma_{r,1}\,\frac{t'_{2k-r-1} - t}{r\Delta'}\,B_{2k-r-1} + (\sigma_{r,1} + \sigma_{r,2})\,\frac{t - t'_{2(k-r)-1}}{r\Delta'}\,B_{2k-r-1}\right.$$

$$+ \sum_{j=2}^{r}\left((\sigma_{r,j-1} + \sigma_{r,j})\,\frac{t'_{2(k-1)-r+j} - t}{r\Delta'} + (\sigma_{r,j} + \sigma_{r,j+1})\,\frac{t - t'_{2(k-r-1)+j}}{r\Delta'}\right)B_{2(k-1)-r+j}$$

$$\left. + (\sigma_{r,r} + \sigma_{r,r+1})\,\frac{t'_{2k-1} - t}{r\Delta'}\,B_{2k-1} + \sigma_{r,r+1}\,\frac{t - t'_{2k-r-1}}{r\Delta'}\,B_{2k-1} + \sigma_{r,r+1}\,\frac{t'_{2k} - t}{r\Delta'}\,B_{2k}\right\}$$

where we have abbreviated $B_{k,r,\mathbf{t}_N'}(t)$ with B_k and we have used the following facts:

(i) since $r\sigma_{r,1} - \sigma_{r,2} = 0$,

$$\sigma_{r,2}(t - t'_{2(k-r-1)}) = (r\sigma_{r,1} - \sigma_{r,2})\Delta' + \sigma_{r,2}(t - t'_{2(k-r-1)})$$

$$= \sigma_{r,1}(t'_{2k-r-1} - t + t - t'_{2(k-r)-1}) + \sigma_{r,2}(t - t'_{2(k-r-1)} - \Delta')$$

$$= \sigma_{r,1}(t'_{2k-r-1} - t) + (\sigma_{r,1} + \sigma_{r,2})(t - t'_{2(k-r)-1})\,, \tag{7.5b}$$

(ii) since $\sigma_{r,r} - r\sigma_{r,r+1} = 0$,

$$\sigma_{r,r}(t'_{2k} - t) = \sigma_{r,r}(t'_{2k} - t) - (\sigma_{r,r} - r\sigma_{r,r+1})\Delta'$$

$$= \sigma_{r,r}(t'_{2k} - t - \Delta') + \sigma_{r,r+1}(t'_{2k-1} - t + t - t'_{2k-r-1})$$

$$= (\sigma_{r,r} + \sigma_{r,r+1})(t'_{2k-1} - t) + \sigma_{r,r+1}(t - t'_{2k-r-1})\,, \tag{7.5c}$$

(iii) and since $\sigma_{r,j-1}(r+2-j) - r\sigma_{r,j} + j\sigma_{r,j+1} = 0$, $j = 2, \ldots, r$, we see that

$$\sum_{j=2}^{r} \sigma_{r,j-1}\,\frac{t'_{2k} - t}{r\Delta'} + \sigma_{r,j+1}\,\frac{t - t'_{2(k-r-1)}}{r\Delta'}$$

$$= \sum_{j=2}^{r} \sigma_{r,j-1}\,\frac{t'_{2k} - t}{r\Delta'} + \sigma_{r,j+1}\,\frac{t - t'_{2(k-r-1)}}{r\Delta'} - \frac{1}{r}\,(\sigma_{r,j-1}(r+2-j) - r\sigma_{r,j} + j\sigma_{r,j+1})$$

$$= \sum_{j=2}^{r} \sigma_{r,j-1} \frac{t'_{2k} - t}{r\Delta'} + \sigma_{r,j+1} \frac{t - t'_{2(k-r-1)}}{r\Delta'} - \sigma_{r,j-1} \frac{r+2-j}{r} - \sigma_{r,j+1} \frac{j}{r}$$

$$+ \sigma_{r,j} \frac{t'_{2(k-1)-r+j} - t + t - t'_{2(k-r-1)+j}}{r\Delta'}$$

$$= \sum_{j=2}^{r} \sigma_{r,j-1} \frac{t'_{2k} - t}{r\Delta'} - \sigma_{r,j-1} \frac{(r+2-j)\Delta'}{r\Delta'} - \sigma_{r,j} \frac{t'_{2(k-1)-r+j} - t}{r\Delta'}$$

$$+ \sigma_{r,j} \frac{t - t'_{2(k-r-1)+j}}{r\Delta'} + \sigma_{r,j+1} \frac{t - t'_{2(k-r-1)}}{r\Delta'} - \sigma_{r,j+1} \frac{j\Delta'}{r\Delta'}$$

$$= \sum_{j=2}^{r} (\sigma_{r,j-1} + \sigma_{r,j}) \frac{t'_{2(k-1)-r+j} - t}{r\Delta'} + (\sigma_{r,j} + \sigma_{r,j+1}) \frac{t - t'_{2(k-r-1)+j}}{r\Delta'} , \tag{7.5d}$$

which all derive from the fact that $\sigma_{r,1} = 1$ and $\sigma_{r,j} = \frac{d^{j-1}}{dt^{j-1}} \frac{(t+1)^r}{(j-1)!} \Big|_{t=0} = \frac{r(r-1)\cdots(r-j+2)}{(j-1)!}$, $j = 2, \ldots, r+1$. Now, rearranging terms slightly, we get

$$B_{k,r+1,\mathbf{t}_N}(t) = \frac{1}{2^r} \Bigg\{ \sigma_{r,1} \frac{t - t'_{2(k-r-1)}}{r\Delta'} B_{2(k-1)-r} + \sigma_{r,1} \frac{t'_{2k-r-1} - t}{r\Delta'} B_{2k-r-1}$$

$$+ \sum_{j=1}^{r} (\sigma_{r,j} + \sigma_{r,j+1}) \left(\frac{t - t'_{2(k-r-1)+j}}{r\Delta'} B_{2(k-1)-r+j} + \frac{t'_{2k-r+j-1} - t}{r\Delta'} B_{2k-r+j-1} \right)$$

$$+ \sigma_{r,r+1} \frac{t - t'_{2k-r-1}}{r\Delta'} B_{2k-1} + \sigma_{r,r+1} \frac{t'_{2k} - t}{r\Delta'} B_{2k} \Bigg\} . \tag{7.5e}$$

Referring to (7.2a) and noting that $\sigma_{r,1} = \sigma_{r+1,1} = 1$, $\sigma_{r,r+1} = \sigma_{r+1,r+2} = 1$, and $\sigma_{r,j} + \sigma_{r,j+1} = \sigma_{r+1,j+1}$, $j = 1, \ldots, r+1$, we see that

$$B_{k,r+1,\mathbf{t}_N}(t) = \frac{1}{2^r} \sum_{i=1}^{r+2} \sigma_{r+1,i} B_{2k-r+i-2,r+1,\mathbf{t}_N{}'}(t) , \tag{7.5f}$$

which verifies that (7.4) holds for $\rho = r+1$. □

Corollary 7.2. Let $r \geq 1$ and $\sigma_{r,i}$ be the i-th coefficient of the polynomial $(t+1)^r$. Then, for splines defined on uniform knot sequences, given $u \in L_N^{(r)}$ with coefficients α_k, $k = 1, \ldots, N+r-1$, u is also a member of $L_{2N}^{(r)}$ with coefficients β_k, $k = 1, \ldots, 2N+r-1$ given, for r odd, by

$$\beta_k = \frac{1}{2^{r-1}} \begin{cases} \sum_{i=1}^{\frac{r+1}{2}} \alpha_{(k+r)/2-i+1}\, \sigma_{r,2i-1} & , \quad k \text{ odd} \\ \sum_{i=1}^{\frac{r+1}{2}} \alpha_{(k+r+1)/2-i}\, \sigma_{r,2i} & , \quad k \text{ even} \end{cases} , \tag{7.6a}$$

and, for r even, by

$$\beta_k = \frac{1}{2^{r-1}} \begin{cases} \sum_{i=1}^{\lceil \frac{r+1}{2} \rceil} \alpha_{(k+r)/2-i+1}\, \sigma_{r,2i-1} & , \quad k \text{ even} \\ \sum_{i=1}^{\lfloor \frac{r+1}{2} \rfloor} \alpha_{(k+r-1)/2-i+1}\, \sigma_{r,2i} & , \quad k \text{ odd} \end{cases} , \tag{7.6b}$$

where $\lceil p \rceil$ is the smallest integer n such that $n \geq p$ and $\lfloor p \rfloor$ is the largest integer n such that $n \leq p$.

Proof. In the following, set $B_{k,r,\mathbf{t}_N{}'}(t) \equiv 0$ if $k < 1$ or $k > 2N + r - 1$. From equation (7.1a) and Proposition 7.1,

$$u(t) = \sum_{k=1}^{N+r-1} \alpha_k B_{k,r,\mathbf{t}_N}(t) = \sum_{k=1}^{N+r-1} \alpha_k \frac{1}{2^{r-1}} \sum_{i=1}^{r+1} \sigma_{r,i} B_{2k-r+i-1,r,\mathbf{t}_N{}'}(t)$$

$$= \sum_{\substack{k'=1 \\ k' \text{ odd}}}^{2(N+r-1)} \alpha_{\frac{k'+1}{2}} \frac{1}{2^{r-1}} \sum_{i=1}^{r+1} \sigma_{r,i} B_{k'-r+i,r,\mathbf{t}_{N'}}(t)$$

$$= \sum_{\substack{k'=1 \\ k' \text{ odd}}}^{2(N+r-1)-1} \alpha_{\frac{k'+1}{2}} \frac{1}{2^{r-1}} \left(\sum_{j=1}^{\lceil \frac{r+1}{2} \rceil} \sigma_{r,2j-1} B_{k'-r+2j-1,r,\mathbf{t}_{N'}}(t) + \sum_{j=1}^{\lfloor \frac{r+1}{2} \rfloor} \sigma_{r,2j} B_{k'-r+2j,r,\mathbf{t}_N{}'}(t) \right)$$

$$= \sum_{k'=1}^{2(N+r-1)} \frac{1}{2^{r-1}} \begin{cases} \sum_{j=1}^{\lceil \frac{r+1}{2} \rceil} \alpha_{\frac{k'+1}{2}}\, \sigma_{r,2j-1} B_{k'-r+2j-1,r,\mathbf{t}_{N'}}(t) & k' \text{ odd} \\ \sum_{j=1}^{\lfloor \frac{r+1}{2} \rfloor} \alpha_{\frac{k'}{2}}\, \sigma_{r,2j} B_{k'-r+2j-1,r,\mathbf{t}_N{}'}(t) & k' \text{ even} \end{cases}$$

$$
= \frac{1}{2^{r-1}} \begin{cases} \sum_{j=1}^{\lceil \frac{r+1}{2} \rceil} \sum_{k=2j-r}^{2N+r-3+2j} \alpha_{\frac{k+r}{2}+1-j}\, \sigma_{r,2j-1} B_{k',r,\mathbf{t}_{N'}}(t) & k+r \text{ even} \\ \sum_{j=1}^{\lfloor \frac{r+1}{2} \rfloor} \sum_{k=2j-r}^{2N+r-3+2j} \alpha_{\frac{k+r+1}{2}-j}\, \sigma_{r,2j} B_{k',r,\mathbf{t}_N}{}'(t) & k+r \text{ odd} \end{cases} . \tag{7.7a}
$$

Thus, if r is odd, we can write, abbreviating $B_{k,r,\mathbf{t}_N}{}'(t)$ with B_k,

$$
u(t) = \frac{1}{2^{r-1}} \left\{ \sum_{k=2-r}^{2N+r-1} \begin{pmatrix} \alpha_{\frac{k+r}{2}}\, \sigma_{r,1} \\ \alpha_{\frac{k+r-1}{2}}\, \sigma_{r,2} \end{pmatrix} B_k + \sum_{k=4-r}^{2N+r+1} \begin{pmatrix} \alpha_{\frac{k+r}{2}-1}\, \sigma_{r,3} \\ \alpha_{\frac{k+r-1}{2}-1}\, \sigma_{r,4} \end{pmatrix} B_k + \cdots + \sum_{k=1}^{2(N+r-1)} \begin{pmatrix} \alpha_{\frac{k+1}{2}}\, \sigma_{r,r} \\ \alpha_{\frac{k}{2}}\, \sigma_{r,r+1} \end{pmatrix} B_k \right\},
$$

where the top row is for k odd and the bottom row is for k even. If r is even, we can write

$$
u(t) = \frac{1}{2^{r-1}} \left\{ \sum_{k=2-r}^{2N+r-1} \begin{pmatrix} \alpha_{\frac{k+r}{2}}\, \sigma_{r,1} \\ \alpha_{\frac{k+r-1}{2}}\, \sigma_{r,2} \end{pmatrix} B_k + \sum_{k=4-r}^{2N+r+1} \begin{pmatrix} \alpha_{\frac{k+r}{2}-1}\, \sigma_{r,3} \\ \alpha_{\frac{k+r-1}{2}-1}\, \sigma_{r,4} \end{pmatrix} B_k + \cdots + \sum_{k=1}^{2(N+r)-1} \begin{pmatrix} \alpha_{\frac{k}{2}}\, \sigma_{r,r+1} \\ 0 \end{pmatrix} B_k \right\},
$$

where, the top row is for k even and the bottom row is for k odd. Now, by collecting the terms for $k \in \{ 1, \ldots, 2N+r-1 \}$ and forming the expression

$$
u(t) = \sum_{k=1}^{2N+r-1} \beta_k B_{k,r,\mathbf{t}_N}{}'(t) , \tag{7.7b}
$$

we see that the coefficients β_k are as given by (7.6a,b). □

Lemma 7.3. Let $\mathbf{N} = \{ 2^n \}_{n=1}^{\infty}$. Then, $L_{N_1}^{(r)} \subset L_{N_2}^{(r)}$ for any $N_1, N_2 \in \mathbf{N}$ such that $N_1 < N_2$. Furthermore,

(a) Given $u \in \mathbf{U}$ and $N = 2^n < \infty$, there exists $j_n \in \mathbf{N}$, $j_n < \infty$ and $u_{j_n} \in \mathbf{U}_{j_n}^{(r)}$ such that $\| u - u_{j_n} \| < 1/N$.

(b) Suppose there is a sequence $\{ u_N \}_{N \in \mathbf{N}}$ such that $u_N \in \mathbf{U}_N^{(r)}$ and $u_N \to u$. Then $u \in \mathbf{U}$.

Proof. The nesting of subspaces defined on uniform knot sequences follows directly from Corollary 7.2. Nesting for subspaces defined on general knot sequences follows from the knot subdivision results in 70 . and [69]

(a) This result is obvious for the case $r = 1$ (since $u_N \in L_N^{(1)}$ is piecewise constant). So assume that $r \geq 2$. Since $u \in L_{\infty,2}^m[0,1] \subset L_2^m[0,1]$ we have, for any $\varepsilon > 0$, that there exists $u'_\varepsilon \in C^m[0,1]$ (space of continuous functions, $u(\cdot)$, with $u(t) \in \mathbb{R}^m$, $t \in [0,1]$) such that $\| u - u'_\varepsilon \|_2 \leq \varepsilon$, [71, Theorem 3.14 (p. 69)]. Choose $\varepsilon = 2/(5+m)N$. We will construct from u'_ε

another function, u_ε such that $\|u - u_\varepsilon\|_2 \le \varepsilon$ and $a + \varepsilon \le u_\varepsilon(t) \le b - \varepsilon$ for all $t \in [0,1]$. This allows us to approximate u_ε with an r-th order spline in such away that by allowing enough knots for this spline (and using the fact that the spline subspaces nest) its coefficients α_k satisfy $\alpha_k \in U$.

With $(\cdot)^i$ denoting the i-th row of a vector, define the continuous function $u_\varepsilon : [0,1] \to \mathbb{R}^m$ as follows:

$$u^i_\varepsilon(t) = \begin{cases} b^i - \varepsilon & \text{if } u'^i_\varepsilon(t) > b^i - \varepsilon \,, \\ u'^i_\varepsilon(t) & \text{if } a^i + \varepsilon \le u'^i_\varepsilon(t) \le b^i - \varepsilon \,, \\ a^i + \varepsilon & \text{if } u'^i_\varepsilon(t) < a^i + \varepsilon \,, \end{cases} \quad i \in \mathbf{m} \,,\ t \in [0,1] \,, \tag{7.8a}$$

Note that, for all $t \in [0,1]$, $a^i \le u^i(t) \le b^i$ since $u(t) \in U$. Thus, if $u'^i_\varepsilon < a_i + \varepsilon$, then either *(i)* $a^i \le u^i(t) \le a^i + \varepsilon = u^i_\varepsilon(t)$, in which case, $0 \le u^i_\varepsilon(t) - u^i(t) \le u^i_\varepsilon(t) - a^i = \varepsilon$, and therefore

$$(u^i_\varepsilon(t) - u^i(t))^2 = \varepsilon^2 \le u'^i_\varepsilon - u^i(t))^2 + \varepsilon^2 \,, \tag{7.8b}$$

or, *(ii)* $u^i(t) > a^i + \varepsilon$, in which case $0 < u^i(t) - u^i_\varepsilon(t) < u^i(t) - u'^i_\varepsilon(t)$, and therefore

$$(u^i_\varepsilon - u^i(t))^2 < (u'^i_\varepsilon(t) - u^i(t))^2 \le (u'^i_\varepsilon(t) - u^i(t))^2 + \varepsilon^2 \,. \tag{7.8c}$$

A similar argument holds for the case $u'^i_\varepsilon > b_i - \varepsilon$. For the case $b^i \le u'^i_\varepsilon \le a^i$, $(u^i_\varepsilon(t) - u^i(t))^2 = ((u'_\varepsilon)^i(t) - u^i(t))^2$. Thus, in all cases, $(u^i(t) - u^i_\varepsilon(t))^2 \le (u^i(t) - u'^i_\varepsilon(t))^2 + \varepsilon^2$. Therefore, we have

$$\|u - u_\varepsilon\|_2^2 = \int_0^1 \sum_{i=1}^m (u^i(t) - u^i_\varepsilon(t))^2 dt \le \int_0^1 \sum_{i=1}^m \big((u^i(t) - u'^i_\varepsilon(t))^2 + \varepsilon^2\big)\, dt = \|u - u'_\varepsilon\|_2^2 + m\varepsilon^2 \,. \tag{7.8d}$$

Thus, $\|u - u_\varepsilon\|_2 \le (1+m)\varepsilon$. Since $u(\cdot)$ is a continuous function, for each $i \in \mathbf{m}$, the modulus of continuity for u^i_ε, $\omega(u^i_\varepsilon, \sigma) \doteq \max\{\, |u^i_\varepsilon(t_1) - u^i_\varepsilon(t_2)| \;\big|\; |t_1 - t_2| \le \sigma \,\}$, goes to zero as $\sigma \to 0$. Thus, by [63, Theorem XII.1 (p. 170)], there exists an integer $N_1 = 2^{n_1} < \infty$ and a spline $u_{N_1} \in L^{(r)}_{N_1}$ such that

$$\|u_\varepsilon - u_{N_1}\|_2 \le \|u_\varepsilon - u_{N_1}\|_\infty \le \frac{\varepsilon}{2} \,. \tag{7.8e}$$

Let $D_{r,\infty}$ be as given on page 155 of [63] (for all $r \ge 2$, $1 \le D_{r,\infty} < \infty$). Since u_{N_1} is a spline with bounded coefficients, it is Lipschitz continuous. Hence, there exists $n_2 \in \mathbb{N}$, $n_1 < n_2 < \infty$, such that, with $N_2 = 2^{n_2}$,

$$\|u_{N_1}(t_1) - u_{N_1}(t_2)\| \le \frac{\varepsilon}{D_{r,\infty} - 1} \,, \quad \forall\, t_1, t_2 \in [0,1] \text{ such that } |t_1 - t_2| \le (r-1)/N_2 \,. \tag{7.8f}$$

Now, for $k = 1, \ldots, N_2 + r$, define the intervals $T_k \doteq [t_{k-r}, t_{k-1}]$, with $t_k = k/N_2$, and define the quantities $M^i_k = \max_{t \in T_k} u^i_{N_1}(t)$, and $m^i_k = \min_{t \in T_k} u^i_{N_1}(t)$. Since for $t_1, t_2 \in T_k$,

$|t_1 - t_2| \leq (r-1)/N_2$, we see that

$$M_k^i - m_k^i = \max_{t_1, t_2 \in T_k} \|u_{N_1}^i(t_1) - u_{N_1}^i(t_2)\| \leq \frac{\varepsilon}{D_{r,\infty} - 1}\ , \text{ for } t_1, t_2 \in T_k\ ,\ i \in \mathbf{m}\ . \tag{7.8g}$$

Thus,

$$D_{r,\infty} \leq \frac{\varepsilon}{M_k^i - m_k^i} + 1\ ,\ \forall\, i \in \mathbf{m}\ . \tag{7.8h}$$

Next, since $L_{N_1} \subset L_{N_2}$ by Corollary 7.2, $u_{N_1} \in L_{N_2}$. Hence, there exists $\{\, \alpha_k \,\}_{k=1}^{N_2+r} \subset \mathbb{R}^m$ such that $u_{N_1}(t) = \sum_{k=1}^{N_2+r} \alpha_k \phi_k(t)$. Thus, by [63, Corollary XI.2 (p. 156)],

$$|\alpha_k^i - \frac{M_k^i + m_k^i}{2}| \leq D_{r,\infty} \frac{M_k^i - m_k^i}{2} \leq \tfrac{1}{2}\,\varepsilon + \frac{M_k^i - m_k^i}{2}\ , \tag{7.8i}$$

where we used (7.8h) for the second inequality. Therefore, $-\frac{1}{2}\,\varepsilon + m_k^i \leq \alpha_k^i \leq \frac{1}{2}\,\varepsilon + M_k^i$. But, from (7.8e), we see that $M_k^i \doteq \max_{t \in T_k} u_{N_1}^i(t) \leq \max_{t \in T_k} u_\varepsilon^i(t) + \frac{1}{2}\,\varepsilon = b^i - \frac{1}{2}\,\varepsilon$ and $m_k^i \doteq \min_{t \in T_k} u_{N_1}^i(t) \geq \min_{t \in T_k} u_\varepsilon^i(t) - \frac{1}{2}\,\varepsilon = a^i + \frac{1}{2}\,\varepsilon$. Thus, $a^i \leq \alpha_k^i \leq b^i$ which implies that $\alpha_k \in U$. Finally, by (7.8d) and (7.8e),

$$\|u - u_{N_1}\|_2 \leq \|u - u'_\varepsilon\|_2 + \|u'_\varepsilon - u_\varepsilon\|_2 + \|u_\varepsilon - u_{N_1}\|_2 \leq \varepsilon + (1+m)\varepsilon + \frac{\varepsilon}{2} = \frac{1}{N}\ , \tag{7.8j}$$

since $\varepsilon = 2/(5+m)N$. Thus, the proposition holds with $j_n = 2^{n_2}$ and $u_{j_n} \in \mathbf{U}_{j_n}^{(r)}$.

(b) Referring to [63, Corollary XI.1 (p. 155)] and using the fact that the coefficients for each component, $i = 1, \ldots, m$, of the control satisfies $a^i \leq \alpha_k^i \leq b^i$ for each $k = 1, \ldots, N+r-1$, we see that $\mathbf{U}_N^{(r)} \subset \mathbf{U}$. The result follows immediately since $\mathbf{U}$ is closed. □

Remark 7.4. From the proof of Proposition 7.3*(b)*, we see that $\mathbf{U}_N^{(r)} \subset \mathbf{U}$. Thus, from the definition of $\mathbf{U}_N$ in Section 4 for representation **R1**, we have that $\mathbf{U}_N^{(r)} \subset \mathbf{U} \cap L_N^{(r)} \subset \mathbf{U} \cap L_N \subset \mathbf{U}_N$. Hence, while control constraint violations are possible for $u \in \mathbf{U}_N$, they are not possible for $u \in \mathbf{U}_N^{(r)}$. □

Theorem 7.5. (Epiconvergence) Suppose that Assumptions 3.1*(a)*, 4.1 and 4.9 hold and that Conjecture 5.11 is true. Let $\mathbf{N} = \{\, 2^n \,\}_{n=1}^{\infty}$. Then, the problems $\{\, \mathbf{CP}_N \,\}_{N \in \mathbf{N}}$ converge epigraphically to the problem **CP** as $N \to \infty$.

Proof. Given $u \in \mathbf{U}$, there exists, by Assumption 4.9, a sequence $\{\, u_N \,\}_{N=1}^{\infty}$ such that $u_N \in \mathbf{U}$, $u_N \to u$ and $\psi_c(u_N) < 0$. By Lemma 7.3*(a)*, for each $N = 2^n$, there exists $j_n \in \mathbf{N}$ and $u_{j_n}{}' \in \mathbf{U}_{j_n}^{(r)}$ such that $\|u_N - u_{j_n}{}'\| \leq 1/N$. It now follows from the proof in Theorem 4.10 that part *(a)* of Definition 2.1 is satisfied. That part *(b)* of Definition 2.1 is satisfied follows from Lemma 7.3*(b)* and the proof in Theorem 4.10. □

To show consistency of approximations, what remains is to compute the gradients of the cost and constraint functions with respect to elements of $L_N^{(r)}$ and show that the optimality functions for the approximating problems satisfy condition (2.3). To compute the gradients $\nabla_u f_N^{\nu}(u)$, we first define the spline coefficient space

$$\bar{L}_N^{(r)} \doteq \Big(\mathop{\times}_{N+r-1} \mathbb{R}^m \ , \ \langle \cdot , \cdot \rangle_{\bar{L}_N^{(r)}} \ , \ \| \cdot \|_{\bar{L}_N^{(r)}} \Big) \tag{7.9a}$$

and the map

$$S_{N,r} : L_N^{(r)} \to \bar{L}_N^{(r)} \ , \tag{7.9b}$$

which takes elements $u = \sum_{k=1}^{N+r-1} \alpha_k \phi_k(t)$ and maps them to $\bar{\alpha} = \{ \alpha_k \}_{k=1}^{N+r-1}$, with $\alpha_k \in \mathbb{R}^m$. When the quantity $\bar{\alpha} \in \bar{L}_N^{(r)}$ appears in a linear algebra statement, it is to be considered a "short-fat" matrix

$$\alpha = [\ \alpha_1 \ \cdots \ \alpha_{N+r-1}\ ] \in \mathbb{R}^{m \times N+r-1} \ . \tag{7.9c}$$

It is clear that $S_{N,r}$ is a linear bijection. Proceeding as in Section 4, we define the inner product on $\bar{L}_N^{(r)}$ in the following way. Given $\alpha, \beta \in \bar{L}_N^{(r)}$, let $u = S_{N,r}^{-1}(\alpha)$ and $v = S_{N,r}^{-1}(\beta)$. The inner product must satisfy

$$\begin{aligned} \langle \alpha, \beta \rangle_{\bar{L}_N^{(r)}} = \langle u, v \rangle_{L_2} &= \int_0^1 \langle \textstyle\sum_{k=1}^{N+r-1} \alpha_k \phi_k(t) , \sum_{l=1}^{N+r-1} \beta_l \phi_l(t) \rangle \, dt \\ &= \sum_{k=1}^{N+r-1} \sum_{l=1}^{N+r-1} \langle \alpha_k, \beta_l \rangle \int_0^1 \phi_k(t) \phi_l(t) \, dt = \langle \alpha \, \mathbf{M}_\alpha, \beta \rangle_{l_2} \ . \end{aligned} \tag{7.10a}$$

Thus, $\mathbf{M}_\alpha$ is the $(N+r-1) \times (N+r-1)$ matrix whose k, l-th entry is given by

$$[\mathbf{M}_\alpha]_{k,l} = \int_0^1 \phi_k(t) \phi_l(t) \, dt \ . \tag{7.10b}$$

An alternate means of determining $\mathbf{M}_\alpha$ is to make use of the fact that $L_N^{(r)} \subset L_N^1$ where L_N^1 was defined in (4.6) as the subspace of piecewise polynomial controls. This will allow us to derive a more useful formula for $\mathbf{M}_\alpha$ and enable us to use the results for L_N^1 in Section 5 to show consistency. Let $\mathbf{M}_N$ be as defined in (4.9b) with $M = M_1$, the quadrature matrix for representation **R1**. Recall from Section 4 that, given $u \in L_N^1$, $\bar{u} = V_{\mathbf{A},N}(u) \in \bar{L}_N^1$ are its control samples. Thus, from (7.1a), the composite map $V_{\mathbf{A},N} \circ S_{N,r}^{-1}$, which computes the control samples of $u \in L_N^{(r)}$ from its spline coefficients $\alpha = S_{N,r}(u)$, satisfies

$$\bar{u} = V_{\mathbf{A},N} \circ S_{N,r}^{-1}(\alpha) = \alpha \, \Phi_{\mathbf{A},N}^T \ , \tag{7.11a}$$

where $\Phi_{\mathbf{A},N}$ is an $Nr \times (N+r-1)$ matrix whose $(lr + j, k)$-th entry, $l = 0, \ldots, N-1$, $j = 1, \ldots, r$, and $k = 1, \ldots, N+r-1$ is $\phi_k(\tau_{l,i_j})$ with $i_j \in I$. The index set I is defined in (2.4.4). In other words, the elements of the k-th column of $\Phi_{\mathbf{A},N}$ are the control samples of the k-th B-spline function $\phi_k(\cdot)$. Thus,

$$\begin{aligned}
\langle u, v \rangle_{L_2} = \langle S_{N,r}(u)\,\mathbf{M}_\alpha, S_{N,r}(v) \rangle_{l_2} &= \langle V_{\mathbf{A},N}(u)\,\mathbf{M}_N, V_{\mathbf{A},N}(v) \rangle_{l_2} \\
&= \langle V_{\mathbf{A},N} \circ S_{N,r}^{-1} \circ S_{N,r}(u)\,\mathbf{M}_N, V_{\mathbf{A},N} \circ S_{N,r}^{-1} \circ S_{N,r}(v) \rangle_{l_2} \\
&= \langle S_{N,r}(u)\,\Phi_{\mathbf{A},N}^T\,\mathbf{M}_N, S_{N,r}(v)\,\Phi_{\mathbf{A},N}^T \rangle_{l_2} \,. \qquad (7.11b)
\end{aligned}$$

Therefore,

$$\boxed{\mathbf{M}_\alpha = \Phi_{\mathbf{A},N}^T \mathbf{M}_N \Phi_{\mathbf{A},N}} \,. \qquad (7.11c)$$

It is not obvious that (7.11c) is equivalent to (7.10b) and hence independent of the Butcher array $\mathbf{A}$. To see that this is so, note that the k, j-th element of $\mathbf{M}_\alpha$, as given in (7.11c), is

$$[\mathbf{M}_\alpha]_{k,l} = \Big[\; \phi_k(\tau_{0,i_1}) \cdots \phi_k(\tau_{0,i_r}) \cdots \phi_k(\tau_{N-1,i_1}) \cdots \phi_k(\tau_{N-1,i_r}) \;\Big] \mathbf{M}_N \begin{bmatrix} \phi_l(\tau_{0,i_1}) \\ \vdots \\ \phi_l(\tau_{N-1,i_r}) \end{bmatrix}. \qquad (7.11d)$$

This is just the inner-product of $V_{\mathbf{A},N}(\phi_k(t))$ and $V_{\mathbf{A},N}(\phi_l(t))$ in $\bar{L}_N^1$. Hence, because of the way M_1 was defined in Section 4, $[\mathbf{M}_\alpha]_{k,l} = \langle \phi_k(t), \phi_l(t) \rangle_{L_2}$.

We are now in a position to compute the gradients of the cost and constraint functions with respect to elements of the finite-dimensional Hilbert space $L_N^{(r)}$. For $u \in L_N^{(r)}$, we will use the shorthand notation

$$d_\alpha \bar{f}_N(u) \doteq \left(\frac{d}{d\alpha_1}\, \bar{f}_N(S_{N,r}(u)) \;\cdots\; \frac{d}{d\alpha_{N+\rho-1}}\, \bar{f}_N(S_{N,r}(u)) \right) \qquad (7.12a)$$

to denote the derivative of $\bar{f}_N(\alpha)$, $\alpha = S_{N,r}(u)$, with respect to the spline coefficients α. This quantity is the gradient with respect to the Euclidean norm but not the norm that we have defined on $\bar{L}_N^{(r)}$. Note that $d_\alpha \bar{f}_N(u) \in \mathbb{R}^{m \times (N+\rho-1)}$ is a "short-fat" matrix; if $m = 1$ (single input system), then $d_\alpha \bar{f}_N(u)$ is a row vector. Let $f_N : L_N^{(r)} \to \mathbb{R}$. Then, the differential of f_N is a bounded linear operator on $L_N^{(r)}$ which, by the Riesz representation theorem, can be represented as $\langle \nabla f_N(u), \delta u \rangle$ where $\nabla f_N(u) \in L_N^{(r)}$. Therefore, using (7.10a) and the definition of directional derivatives, we have, for $u \in L_N^{(r)}$, and $\delta u \in L_N^{(r)}$,

$$\langle \nabla_u f_N(u), \delta u \rangle = \langle S_{N,r}(\nabla_u f_N(u))\,\mathbf{M}_\alpha\,, \delta\alpha \rangle_{l_2} = \langle\, d_\alpha \bar{f}_N(u)\,, \delta\alpha \rangle_{l_2}\,, \qquad (7.12b)$$

where $\delta\alpha = S_{N,r}(\delta u)$, $\alpha = S_{N,r}(u)$ and, by using the chain rule along with (7.11a),

$$\boxed{d_\alpha \bar{f}_N(u) = d_{\bar{u}} \bar{f}_N(u)\, \Phi_{\mathbf{A},N}}\ , \tag{7.12c}$$

with $d_{\bar{u}} \bar{f}_N(u)$ defined by (5.5c). Thus, for $u \in L_N^{(r)}$,

$$\nabla_u f_N(u) = S_{N,r}^{-1}\big(\, d_{\bar{u}} \bar{f}_N(u) \Phi_{\mathbf{A},N}\, \mathbf{M}_\alpha^{-1} \big)\ . \tag{7.12d}$$

It is important to note that the expression in (7.12d) for the gradient on $L_N^{(r)}$ is not the same as the gradient $\nabla f_N(u) = V_{\mathbf{A},N}^{-1}(d_{\bar{u}} \bar{f}_N(u)\, \mathbf{M}_N^{-1})$ in L_N^1 restricted to $L_N^{(r)} \subset L_N^1$ because the definition of the gradient depends on the space of perturbations upon which the differential of $f_N(\cdot)$ is allowed to act. However, from (7.12d) and (7.11a), we can relate the samples of $\nabla f_N(u)$ on $L_N^{(r)}$ to the discrete-time derivative of $\bar{f}_N(\cdot)$ as follows,

$$V_{\mathbf{A},N}(\nabla f_N(u)) = V_{\mathbf{A},N} \circ S_{N,r}^{-1}(d_{\bar{u}} \bar{f}_N(u) \Phi_{\mathbf{A},N} \mathbf{M}_\alpha^{-1}) = d_{\bar{u}} \bar{f}(u) \Phi_{\mathbf{A},N} \mathbf{M}_\alpha^{-1} \Phi_{\mathbf{A},N}^T\ . \tag{7.13}$$

We note that $\Phi_{\mathbf{A},N} \mathbf{M}_\alpha^{-1} \Phi_{\mathbf{A},N}^T \neq \mathbf{M}_N^{-1}$.

The expression in (7.13) can be used in the proof of Theorem 5.6 to show that there exists $\kappa < \infty$ such that $\| \nabla_u f_N(u) - \nabla_u f(u) \| \leq \kappa / N$ for all $u \in L_N^{(r)} \subset H_N$. The derivation of (5.13d) starting from (5.11d) must be modified by taking into account the fact that entries of $\Phi_{\mathbf{A},N} \mathbf{M}_\alpha^{-1} \Phi_{\mathbf{A},N}^T$ go to zero away from the diagonal and $\Phi_{\mathbf{A},N} \mathbf{M}_\alpha^{-1} \Phi_{\mathbf{A},N}^T \mathbb{1} \to \mathbb{1}$ as $N \to \infty$, where $\mathbb{1}$ is a compatible column vector of ones. Therefore, the optimality functions hypoconverge by the result of Theorem 5.9 and thus satisfy condition (2.3). This, along with Theorem 7.5, shows that the approximating problems $\mathbf{CP}_N$, with feasible sets $\mathbf{H}_N^{(r)}$ and optimality functions θ_N given by (5.8a) using (7.12d) as the expression for the gradients, are consistent approximations to $(\mathbf{CP}, \theta)$. We state this result as a theorem:

Theorem 7.6. Suppose that Assumptions 3.1, 4.1 and 4.9 and equation (5.18) hold and that Conjecture 5.11 is true. Let $\mathbf{N} = \{\, 2^n \,\}_{n=1}^{\infty}$. Then, with $\mathbf{CP}_N$ as defined in (7.3c) and θ_N as defined in (5.8a), the family of approximating pairs $(\mathbf{CP}_N, \theta_N)$, $N \in \mathbf{N}$, constitute consistent approximations for the pair $(\mathbf{CP}, \theta)$. □

Example (Linear Splines --- uniform knot sequence)

In this case, $r = 2$ and the basis functions are given by

$$\phi_k(t) = \begin{cases} (t - t_{k-2})/\Delta & \text{if } t \in [t_{k-2}, t_{k-1}] \\ (t_k - t)/\Delta & \text{if } t \in [t_{k-1}, t_k] \end{cases}\ . \tag{7.14}$$

Let $u, v \in L_N^{(r)}$ and $\alpha = S_{N,r}(u)$ and $\beta = S_{N,r}(v)$. Since these hat functions have a support of only two time intervals (2Δ), $\mathbf{M}_\alpha$, given by equation (7.10b), is

$$\mathbf{M}_\alpha = \frac{\Delta}{(2r-1)!}\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & & \ddots & & & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix}. \tag{7.15}$$

Example (Cubic Splines --- uniform knot sequence)

In this case, r=4 and the basis functions are given by (7.2c). Assuming $k \le l$, $\int_0^1 \phi_k(t)\phi_l(t)\,dt = \int_a^b \phi_k(t)\phi_l(t)\,dt$ where $a = \max\{\,0, t_{l-4}\,\}$ and $b = \min\{\,t_k, 1\,\}$ since each B-spline has support of width 4Δ. In particular, $[\mathbf{M}_\alpha]_{k,l} = 0$ if $|k-l| > 3$. Thus,

$$\mathbf{M}_\alpha = \frac{\Delta}{(2r-1)!}\begin{bmatrix} 20 & 129 & 60 & 1 & & & & \\ 129 & 1208 & 1062 & 120 & 1 & & & \\ 60 & 1062 & 2396 & 1191 & 120 & 1 & & \\ 1 & 120 & 1191 & 2416 & 1191 & 120 & 1 & \\ & 1 & 120 & 1191 & 2416 & 1191 & 120 & 1 \\ & & \ddots & & \ddots & & & \ddots \end{bmatrix}. \tag{7.16}$$

Remark 7.7. Formula (7.10b) for determining $\mathbf{M}_\alpha$ can be applied in a straightforward manner using either a numerical integration routine or by evaluating the integrals (whose integrands are piecewise polynomials) analytically. Some care must be taken when applying formula (7.11c). Specifically, it must be remembered that the order r of the splines in $L_N^{(r)}$ will be smaller than the number of stages s in the RK method if some of the c_i values in the Butcher array are repeated. For example, the fourth-order method RK4 used in Section 6 with $c = (0, 1/2, 1/2, 1)$, has $r = 3$. In this case, formula (7.11c) applies to quadratic splines, not cubic splines. Recall, though, that (7.11c) is actually independent of the RK parameters. So, for a ρ-th order spline, it is easiest to just choose $c = (c_1, \ldots, c_\rho)$ where $c_j = 1/j$, $j = 1, \ldots, \rho$, and not consider which RK method is going to be used.

Both formulas also work when the knot sequence contains repeated interior knots. In this case, care must be taken at any point where the spline is discontinuous to ensure that the values of $\phi_k(\tau_{l,i_j})$ are evaluated at the correct side of the discontinuity. □

Remark 7.8. For $r \ge 2$, $\mathbf{M}_\alpha^{-1}$ is a dense matrix. However, $\mathbf{M}_\alpha$ is banded and diagonally dominant. Hence, for $u \in L_N^{(r)}$, we can find $\bar{d}(u) \doteq S_{N,r}(\nabla_u f_N(u)) = d_\alpha \bar{f}_N(u) M_\alpha^{-1}$ efficiently by solving

$$\bar{d}(u)\,\mathbf{M}_\alpha = d_\alpha \bar{f}_N(u)\,, \tag{7.17}$$

where $\alpha = S_{N,r}(u)$, using Gaussian elimination without pivoting [72, page 119]. □

2.7.1 Implementation of Spline Coordinate Transformation.

The idea of coordinate transformations introduced in Section 6 can also be used with spline representations for the finite dimensional control subspaces. In this case we use the transform matrix $\mathbf{M}_\alpha$ to set up the coordinate transformation:

$$\tilde{\alpha} = \alpha \mathbf{M}_\alpha^{1/2} \tag{7.18a}$$

$$\boxed{\tilde{f}_N(\tilde{\alpha}) \doteq \bar{f}_N(\tilde{\alpha}\,\mathbf{M}_\alpha^{-1/2})}\ . \tag{7.18b}$$

Correspondingly, with $u = S_{N,r}^{-1}(\tilde{\alpha}\,\mathbf{M}_\alpha^{-1/2})$, the gradient is given by the expression

$$\boxed{\nabla_{\tilde{\alpha}} f_N(u) = d_\alpha \bar{f}_N(\tilde{\alpha}\,\mathbf{M}_\alpha^{-1/2})\mathbf{M}_\alpha^{-1/2}}\ . \tag{7.18c}$$

Because $\mathbf{M}_\alpha^{1/2}$ and its inverse are dense matrices, a factorization approach is needed in the implementation of this transformation. Since, $\mathbf{M}_\alpha$ is symmetric it has a Schur decomposition $\mathbf{M}_\alpha = UDU^T$ with U a unitary matrix and D a diagonal matrix, see [72, page 410]. Thus, because $U^T = U^{-1}$, $\mathbf{M}_\alpha^{1/2} = UD^{1/2}U^T$ and $\mathbf{M}_\alpha^{-1/2} = UD^{-1/2}U^T$ (see [72, page 540]). Note that the factorization need be performed only once.

Besides the computational burden of computing $\mathbf{M}_\alpha^{1/2}$, the fact that $\mathbf{M}_\alpha^{1/2}$ is not a diagonal matrix causes another problem: simple bound constraints on $\bar{\alpha}$ are transformed into general linear constraints on $\tilde{\alpha}$ (*cf.* Remark 6.1). Both of these problems can be alleviated for second order splines using the following considerations. First, we recall from (7.11c) that $\mathbf{M}_\alpha = \Phi_{\mathbf{A},N}^T \mathbf{M}_N \Phi_{\mathbf{A},N}$. In this expression, $\mathbf{M}_N$ is the block diagonal matrix

$$\mathbf{M}_N \doteq \mathrm{diag}[\Delta_0 M_1, \Delta_1 M_1, \ldots, \Delta_{N-1} M_1] \tag{7.19a}$$

where M_1 is the quadrature matrix defined for control representation **R1** in (4.9b). Next, note from (7.11a) that given a vector $\alpha \in \bar{L}_N^{(r)}$, $\alpha\Phi_{\mathbf{A},N}^T = V_{\mathbf{A},N} \circ S_{N,r}^{-1}(\alpha) \in \bar{L}_N^1$. Thus, $\mathbf{M}_\alpha$ is just $\mathbf{M}_N$ transformed from the basis for L_N^1 to the basis for $L_N^{(r)}$. Now, according to Proposition 5.5, the control samples of solutions to the discretized problem $(\mathbf{P}_N, \theta_N)$, and hence, the spline that interpolates those samples, do not depend on which control representation, **R1** or **R2**, is being used. Hence, we can construct $\mathbf{M}_N$ from the quadrature matrix M_2 instead of M_1 without affecting the solutions of $(\mathbf{P}_N, \theta_N)$. For $r = 2$, we have

$$M_2 = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}.$$

Thus, letting

$$\mathbf{M}_N \doteq \text{diag}[\ \Delta_0 M_2\ ,\ \Delta_1 M_2\ ,\ldots,\ \Delta_{N-1} M_2\ ]\ , \tag{7.19b}$$

we get the transformation matrix

$$\mathbf{M}_\alpha = \begin{bmatrix} \Delta_0 & & & & & \\ & \frac{\Delta_0 + \Delta_1}{2} & & & & \\ & & \frac{\Delta_1 + \Delta_2}{2} & & & \\ & & & \ddots & & \\ & & & & \frac{\Delta_{N-2} + \Delta_{N-1}}{2} & \\ & & & & & \Delta_{N-1} \end{bmatrix} \tag{7.19c}$$

which is diagonal. Note that this idea will not produce a diagonal $\mathbf{M}_\alpha$ for higher order splines. However, in the next chapter, we argue against using higher order splines for problems with control constraints anyway.

In the following table, we list the number of iterations required to solve a few optimal control problems with and without using the coordinate transformation. For each problem, we used a uniformly spaced integration mesh with a discretization level of $N = 50$. The convergence tolerance was set at $\varepsilon = 10^{-6}$ when using the coordinate transformation and $\varepsilon = 10^{-6} / \sqrt{N}$ when not using the coordinate transformation (to make the convergence criterion equivalent). This is equivalent to setting $M_\alpha = \frac{1}{N} I$ where I is the identity matrix. For problems with endpoint and/or trajectory constraints, we set the constraint violation tolerance at 10^{-4}.

For the two unconstrained problems, we used both the projected descent algorithm (P-Descent) in conjunction with the L-BFGS (limited memory quasi-Newton method) and an SQP method (NPSOL). We solved the problems using both a second order (RK2) and a fourth order (RK4) integration methods. The order of the spline representation is indicated by r. For second order splines, $r = 2$, we used $\mathbf{M}_\alpha$ as given by (7.11c) and by (7.19c). The number of iterations required when using expression (7.19c) is the first number in the $r = 2$ column and the number of iterations required when using (7.11c) is the second number.

The optimal control problems listed in table 7.1 are described in Appendix B. The last three rows of this table pertain to problems with control bounds. We did not, therefore, solve these problems with third order splines.

		RK2		RK4			
		With	Without	With		Without	
Problem	Method	$r = 2$	$r = 2$	$r = 2$	$r = 3$	$r = 2$	$r = 3$
LQR	P-Descent	5 , 5	12[1]	10 , 5	5	19[1]	26[1]
LQR	SQP	5 , 15	15	10 , 5	5	26	39
Rayleigh	P-Descent	13 , 15	14	16 , 14	14	19	28[1]
Rayleigh	SQP	18 , 23	40	18 , 18	20	56	78
Constr. Rayleigh	SQP	24 , 35	20	22 , 22	24	26	34
Switch	SQP	1 , 18	8	12 , 1	6	18	25
Bang	SQP	12	9	12		9	
Goddard	SQP	26	37[2]	30		38[2]	
Goddard[3]	SQP	24	38[2]	22		29[2]	

Table 7.1: Number of iterations required to solve various optimal control problems with and without the use of the spline coordinate transformation (7.18). The order of the spline representation is indicated by r. Columns with two numbers show the number of iterations required when $\mathbf{M}_\alpha$ is determined with formula (7.19c) or (7.11c) respectively. Constr. Rayleigh is the Rayleigh problem with the endpoint constraint $x_1(2.5) = 0$.

In this table a superscript [1] indicates that the projected descent method terminated without achieving the requested tolerance because the step-size became too small. A superscript [2] indicates that the SQP reached the requested tolerance but that iterates, $\{ \alpha^i \}$, failed to converge in the sense that $\|\alpha^{i+1} - \alpha^i\|$ was not small enough. The last two rows are results for the Goddard maximum ascent rocket problem. This is a singular optimal control problem and was solved with a 10^{-4} penalty on the variation of the control derivative (see Chapter 4.5). The last row, Goddard[3], includes a trajectory constraint on the dynamic pressure. We can make the following observation from the data in this table.

- With the exception of of constrained Rayleigh using RK2 and Bang, the number of iterations when using the coordinate transformation is less than without the transformation; sometimes substantially less.
- In several cases, the optimization procedure was not able to terminate successfully when the coordinate transformation was not used.
- In comparing the results of the first row of this table to Table 6.1, we see that the effect of the coordinate transformation on the number of iterations is less pronounced for splines than it is for controls in L_N^1 or L_N^2. This is due to the spline smoothness.

- For problems without control constraints, formula (7.11c) for $\mathbf{M}_\alpha$, when using second order splines, gives better results for RK4 than formula (7.19c) gives. The reverse is true with RK2. We have no explanation for this behavior. This suggests the following rule for which formula for $\mathbf{M}_\alpha$ to use: if the problem has control bounds or if using RK2, use formula (7.19b), otherwise use (7.11c).

2.8 CONCLUDING REMARKS

We have shown that a large class of Runge-Kutta integration methods can be used to construct consistent approximations to continuous time optimal control problems. The construction is not unique: it is determined by the selection of families of finite dimensional subspaces of the control space. Because the elements of these subspaces are discontinuous functions, appropriate extensions of Runge-Kutta methods must be used. Not all convergent Runge-Kutta methods, however, produce consistent approximations. This was observed both numerically and by failure to prove consistency of approximation with these methods. We have considered two selections of control subspaces, one defined by piecewise polynomial functions and one by piecewise constant functions. Splines can also be used and are treated in Section 7. Each selection has some advantages and some disadvantages. A final selection has to be made on the basis of secondary considerations such as the importance of approximate solutions satisfying the original control constraints, the form that the control constraints take in the discrete-time optimal control problems, or the accuracy with which the differential equation is integrated.

As in our construction, the basis functions that are used implicitly to define the finite dimensional control subspaces may turn out to be non-orthonormal. In this case a non-Euclidean inner product and corresponding norm should be used in solving the resulting approximating discrete-time optimal control problems. Neglecting to do so amounts to a change of coordinates that can lead to serious ill-conditioning. This ill-conditioning is demonstrated in Section 6 and Section 7.

Finally, the use of the framework of consistent approximations opens up the possibility of developing optimal discretization strategies, such as those considered for semi-infinite programming in [57]. Such a strategy provides rules for selecting the number of approximating problems to be used as well as the discretization level, the order of the RK method, and the number of iterations of a particular optimization algorithm to be applied for each such approximating problem, so as to minimize the computing time needed to reach a specified degree of accuracy in solving an optimal control problem.

Chapter 3

PROJECTED DESCENT METHOD FOR PROBLEMS WITH SIMPLE BOUNDS

3.1 INTRODUCTION

In this chapter, we consider a class of finite dimensional optimization problems that arises from the discretization of optimal control problems with simple control bounds. Because the discussion in this Chapter is within the realm mathematical programming, we will maintain the convention of using the x, rather than the control variable u, to represent the decision variables of a mathematical program.

Consider the problem

P $$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to } \; x^i \geq 0 \,, \;\; i = 1, \ldots, n \,,$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and $x = (x^1, x^2, \ldots, x^n)$.

Algorithms for solving problem **P** based on the projection of a descent direction were first proposed by Goldstein [73] and Levitin and Polyak [74]. In [75], Bertsekas used the projection operator defined in [73,74] to construct a projected gradient descent algorithm with an Armijo step-size rule for solving **P**. Whenever a sequence constructed by this algorithm enters a sufficiently small neighborhood of a local minimizer $\hat{x}$ satisfying standard second order sufficiency conditions, it gets trapped and converges to this local minimizer. Furthermore, in this case, the active constraint set at $\hat{x}$ is identified in a finite number of iterations. This fact was used in [76] to construct a modified projected Newton method, again using the projection operator defined in [73,74], with a modified Armijo step-size rule. The algorithm in [76] employs the Newton search direction only in the estimated subspace of non-binding (inactive) variables, and uses the gradient direction in the estimated subspace of binding (active) variables. Under reasonable assumptions, Bertsekas showed that his projected modified Newton method for solving **P** is globally convergent with Q-quadratic rate. The algorithm in [76] is easily extended to problems with simple bounds of the form $b_l^i \leq x^i \leq b_u^i$, $i = 1, \ldots, n$ where $b_l^i \leq b_u^i$. Bertsekas also provides an extension for handling general linear constraints of the form $b_l \leq Ax \leq b_u$. The Bertsekas projected Newton method was further extended to handle general convex constraints in [77]. The efficiency of this family of algorithms derives from the fact that *(a)* the search direction computation is simple, *(b)* any number of constraints can be added to or removed from the active constraint set at each iteration, and *(c)* under the standard second order sufficiency condition the algorithms identify the correct active constraint set after a finite number of iterations. This last fact implies that the rate of convergence depends only on the rate of convergence of the algorithm in the subspace of decision variables that are unconstrained at the solution.

Another example of a projection based algorithm for solving optimization problems with simple bounds can be found in [78] where Quintana and Davison present a *conceptual* algorithm with exact line search based upon a modified version of the Fletcher-Reeves conjugate gradient method in function space combined with a projection operator. This algorithm was intended for the solution of optimal control problems with bounded controls. The soundness of the algorithm in [78] is not clear because the proof of convergence assumes that there exists, at each iteration k, a step-size $\alpha_k > 0$ that causes a decrease in function value. However, the authors do not show that such a positive step-size exists. Furthermore, Quintana and Davison require an *a posteriori* assumption (their eqn. (26)) that is not directly related to the problem or the method under consideration.

More recently, there have been some papers based on ideas related to Bertsekas' projected gradient scheme. A trust region algorithm for problems with simple bounds is analyzed in [79]. On each iteration of this algorithm, a projection operator is used to find the generalized *Cauchy point* of a quadratic model to the objective function. Then the quadratic model is further minimized on the intersection of the trust region with the feasible set while keeping the variables bounded at the Cauchy point fixed. This algorithm is extended to problems with general constraints in [80] using an augmented Lagrangian approach. A scheme similar to that proposed in [79] is given in [81]. However, in [81] the quadratic model is based on a positive definite, limited-memory BFGS estimate of the Hessian. Therefore, a trust region is not needed and the approximate minimizer of the model is used to construct a search direction for a projected line minimization of the objective function. The projected gradient idea is also used in [82] and [83] to rapidly identify the active constraint set for bound constrained quadratic programming. Finally, the authors of [84] have extended the projected Newton method of [76] to optimal control problems with control bounds.

Our results extend those provided in [76] by showing that the concept of Bertsekas' projection method can be used with any search direction and step-size rules that satisfy general conditions similar to those in [85]. For example, we show that our version of the projection method can be used with search directions that are determined by a conjugate gradient in the subspace of unconstrained decision variables. The extension to conjugate-gradient methods is particularly valuable for solving large-scale optimization problems with simple bound constraints because conjugate-gradient methods do not require much additional storage or computation beyond that required by the steepest descent method but, in practice, perform considerably better than steepest descent. We also use our results to construct an algorithm based on the limited-memory quasi-Newton method, L-BFGS (described in [86]), for the search direction computations. Unlike the L-BFGS method used in [81], our update is only used to estimate the Hessian in the unconstrained subspace and our search direction is obtain directly rather than as an approximate solution of a quadratic subproblem.

The remainder of this chapter consists of three sections. In Section 2, we define the projection operator and state an algorithm model that can use any search directions that satisfy certain conditions. The algorithm uses a modified Armijo rule for the step-size selection. We prove convergence of this algorithm model and the fact that it identifies the correct active constraint set in a finite number of iterations under second order sufficiency conditions. We also show how to incorporate other step-size rules into our algorithm in a way that preserves the convergence properties. Several of our proofs are similar to those in [76]. As an example of the construction of admissible search directions for our algorithm, we use the Polak-Ribère conjugate gradient formula which numerical experience has shown to be more effective than the Fletcher-Reeves formulation (an explanation for this empirical result is given in [87]). We provide one example of the fact that standard rate of convergence results for the conjugate gradient method still hold for its projected version. To conclude Section 2, we describe an extension of the algorithm model that handles simple bounds of the form $b_l^i \leq x^i \leq b_u^i$, $i = 1, \ldots, n$. In Section 3, we present numerical results obtained in solving two optimal control problems with simple control bounds. We use three implementations of our algorithm based on steepest descent, conjugate gradient, and the limited-memory quasi-Newton (L-BFGS) method for the search direction computations. These numerical results indicate that the projected conjugate gradient method and the projected L-BFGS method perform significantly better than the projected steepest descent method. Finally, in Section 4, we state our concluding remarks.

3.2 ALGORITHM MODEL FOR MINIMIZATION SUBJECT TO SIMPLE BOUNDS

The algorithm to be presented is described with the help of the following notation: for any $z \in \mathbb{R}^n$, the projection operator $[\cdot]_+$ is given by

$$[z]_+ \doteq \begin{bmatrix} \max\{\, 0, z^1 \,\} \\ \vdots \\ \max\{\, 0, z^n \,\} \end{bmatrix}, \tag{2.1a}$$

and, for any search direction $d \in \mathbb{R}^n$, $x \in \mathbb{R}^n$ and step-size $\lambda \in \mathbb{R}_+$,

$$x(\lambda, d) \doteq [x + \lambda d]_+ \,. \tag{2.1b}$$

For any index set $I \subset \{\, 1, \ldots, n \,\}$ and $x, y \in \mathbb{R}^n$, we define $\langle x, y \rangle_I \doteq \sum_{i \in I} x^i y^i$, and $\|x\|_I^2 \doteq \langle x, x \rangle_I$. Without subscripts, $\langle \cdot, \cdot \rangle$ and $\| \cdot \|$ denote the Euclidean inner product and norm, respectively, on $\mathbb{R}^n$. Let

$$B(\bar{x}, \rho) \doteq \{\, x \in \mathbb{R}^n \mid \|x - \bar{x}\| \le \rho \,\} \tag{2.1c}$$

denote the closed ball of radius ρ around $\bar{x}$. Finally, let

$$\mathcal{F} \doteq \{\, x \in \mathbb{R}^n \mid x^i \ge 0 \,,\ i = 1, \ldots, n \,\} \tag{2.1d}$$

denote the feasible set for problem **P**.

Definition 2.1. A point $\hat{x} \in \mathcal{F}$ is said to be a *stationary point* for the problem **P** if directional derivative of $f(x)$ at $\hat{x}$ is non-decreasing in all feasible directions:

$$df(\hat{x}\,; x - \hat{x}) \ge 0 \,,\ \forall\, x \in \mathcal{F}, \tag{2.2a}$$

or equivalently, for $i = 1, \ldots, n$,

$$\frac{\partial f(\hat{x})}{\partial x^i} \ge 0 \,, \quad \text{and} \quad \frac{\partial f(\hat{x})}{\partial x^i} = 0 \ \text{ if } \hat{x}^i > 0 \,. \tag{2.2b}$$

□

Active and almost active bounds. The projected descent algorithm model (Algorithm Model PD) which we will present requires, for each iterate x_k, the definition of sets $I_k = I(x_k) \subset \{\, 1, 2, \ldots, n \,\}$ and $A_k = A(x_k) \subset \{\, 1, 2, \ldots, n \,\}$. The set A_k contains the indices of the "active" or "almost active" bounds at iteration k and the set I_k is the complement of A_k in $\{\, 1, \ldots, n \,\}$. With $g(x) = \nabla f(x)$, we define†

$$w(x) \doteq \|x - [x - g(x)]_+\| \,, \tag{2.3a}$$

† More generally, $w(x)$ can be defined as $w(x) \doteq \|x - [x - Dg(x)]_+\|$ where D is a positive definite diagonal matrix.

and

$$\varepsilon(x) \doteq \min \{ \varepsilon, w(x) \} \ , \tag{2.3b}$$

where $\varepsilon > 0$ is a parameter in Algorithm Model PD. We can see that $\varepsilon(x) = 0$ if and only if $x \in \mathcal{F}$ is a stationary point because the requirement that $x \in \mathcal{F}$ and that (2.2b) hold is equivalent to the requirement that

$$\max \{ -g^i(x), -x^i \} = 0 \ , \ \ i = 1, \ldots, n \tag{2.3c}$$

which, upon addition of x^i, $i = 1, \ldots, n$, to both sides, yields $[x - g(x)]_+ = x$, *i.e.* that $w(x) = 0$. Next, for $x \in \mathcal{F}$, we define

$$A(x) \doteq \{ \ i \in 1, \ldots, n \ \big| \ 0 \le x^i \le \varepsilon(x) \ , \ g^i(x) > 0 \ \} \ , \tag{2.4a}$$

and

$$I(x) \doteq \{ \ i \in 1, \ldots, n \ \big| \ i \notin A(x) \ \} \ = \ \{ \ i \in 1, \ldots, n \ \big| \ x^i > \varepsilon(x) \text{ or } \ g^i(x) \le 0 \ \} \ . \tag{2.4b}$$

To understand the logic behind the definition of the active constraint index set $A(x)$, first consider the situation corresponding to $\varepsilon = 0$. In this case, if $i \in A(x)$, $x^i = 0$ and $g^i(x) > 0$. Thus, x^i is at its bound and, moreover, any movement in $\mathcal{F}$ away from that bound will cause an increase in the objective function. Hence our algorithm will be constructed to leave such an x^i unchanged. When $\varepsilon > 0$, as in Algorithm Model PD below, the set $A(x)$ also includes indices of variables that are almost at their bounds and, because $g^i(x) > 0$, are likely to hit their bounds during the line search. Thus, given $x \in \mathcal{F}$, the set $A(x)$ tends to identify the active constraints at a ‘‘nearby’’ point on the boundary of $\mathcal{F}$.

Note that in Algorithm Model PD, below, the search directions are specified only to the extent that they satisfy three conditions (stated in (2.5a,b,c)). It is clear that the direction of steepest descent, and more generally, any direction of the form $d_k = -D_k g_k$ where D_k is a symmetric, positive definite matrix that is diagonal with respect to indices $i \in A_k$ and has eigenvalues bounded from above and away from zero, satisfies these conditions. In the sequel we will show how d_k satisfying these conditions can be constructed using standard algorithms.

The most important property of Algorithm Model PD is that it identifies the correct active constraint set in a finite number of iterations. Once the correct active constraint set is identified, the active variables x_i remain at the value of zero while on the orthogonal, ‘‘unconstrained’’ subspace Algorithm Model PD behaves as an unconstrained optimization algorithm. Because of this, the rate of convergence of Algorithm Model PD is that associated with whatever method is used to determine the components of the search direction d_k in the ‘‘unconstrained’’ subspace.

Algorithm Model PD:

Data: $\alpha, \beta \in (0,1)$, $M \in \mathbb{N}$, $\sigma_1 \in (0,1)$, $\sigma_2 \in (1,\infty)$, $\varepsilon \in (0,\infty)$, $x_0 \in \mathcal{F}$.

Step 0: Set $k = 0$.

Step 1: Compute $g_k = \nabla f(x_k)$ and set $A_k = A(x_k)$, $I_k = I(x_k)$. If $\|g_k\|_{I_k} = 0$ and $x_k^i = 0$ for all $i \in A_k$, stop.

Step 2: Select scalars m_k^i, $i \in A_k$, and a search direction d_k satisfying the following conditions:

$$d_k^i = -m_k^i g_k^i \quad , \quad \sigma_1 \le m_k^i \le \sigma_2 \quad , \quad \forall\, i \in A_k \;, \tag{2.5a}$$

$$\langle d_k, g_k \rangle_{I_k} \le -\sigma_1 \|g_k\|_{I_k}^2 \;, \tag{2.5b}$$

$$\|d_k\|_{I_k} \le \sigma_2 \|g_k\|_{I_k} \;. \tag{2.5c}$$

Step 3: Compute the step-size $\lambda_k = \beta^m$ where m is the smallest integer greater than $-M$ such that λ_k satisfies the Armijo-like rule:

$$f(x_k(\lambda_k, d_k)) - f(x_k) \;\le\; \alpha \left\{ \lambda_k \langle g_k, d_k \rangle_{I_k} \;-\; \langle g_k, x_k - x_k(\lambda_k, d_k) \rangle_{A_k} \right\} \;. \tag{2.6a}$$

Set

$$x_{k+1} = x_k(\lambda_k, d_k) = [x_k + \lambda_k d_k]_+ \;. \tag{2.6b}$$

Step 4: Replace k by $k+1$ and go to Step 1. □

Note that in (2.5a) one can choose $m_k^i = 1$ for all $i \in A_k$. Then the search direction in the subspace of active constraints is the steepest descent direction. The criterion $\|g_k\|_{I_k} = 0$ is not a practical test; in a numerical implementation it would instead be required that $\|g_k\|_{I_k}$ be smaller than a given tolerance. The ε in the algorithm description is needed in (2.3b) to determine the active constraint index set.

Remark 2.2. It is easy to see that the the right-hand side of (2.6a) is non-positive. The first term of the bracketed expression is non-positive because $\langle g_k, d_k \rangle_{I_k} \le -\sigma_1 \|g_k\|_{I_k}^2$ by (2.5b). The second term is non-negative:

$$\langle g_k, x_k - x_k(\lambda_k, d_k) \rangle_{A_k} \;\ge 0, \tag{2.7}$$

because for all $i \in A_k$, $g_k^i > 0$ and $d_k^i = -m_k^i g_k^i \le -\sigma_1 g_k^i < 0$ and hence $x_k^i - x_k^i(\lambda, d_k) \ge 0$ for all $\lambda \ge 0$.

□

Remark 2.3. The requirements in (2.5a,b,c) are similar to those used in the Polak-Sargent-Sebastian Theorem of convergence for abstract, iterative minimization processes [85]; they ensure that $\|d_k\|$ is bounded below by $\sigma_1\|g_k\|$ and bounded above by $\sigma_2\|g_k\|$ and that d_k does not become orthogonal to g_k. To wit, let θ_k be the angle between the vectors d_k and $-g_k$. From (2.5a,b) we have that

$$\langle d_k, g_k \rangle \leq -\sigma_1 \|g_k\|_{A_k}^2 + \langle d_k, g_k \rangle_{I_k} \leq -\sigma_1 \|g_k\|^2 \ , \tag{2.8a}$$

and from (2.5a,c)

$$\|d_k\|^2 \leq \sigma_2^2 \|g_k\|_{A_k}^2 + \|d_k\|_{I_k}^2 \leq \sigma_2^2 \|g_k\|^2 \ . \tag{2.8b}$$

Using these expression in (2.8a) we see that

$$\cos\theta_k = \frac{-\langle d_k, g_k \rangle}{\|d_k\| \|g_k\|} \geq \frac{\sigma_1}{\sigma_2} > 0 \ . \tag{2.8c}$$

□

Remark 2.4. In [76], the search directions are given by $d_k = -D^k g_k$ where the D^k are symmetric, positive definite matrices with elements, D_{ij}^k, that are diagonal with respect to the indices $i \in A_k$, *i.e.*,

$$D_{ji}^k = D_{ij}^k = 0 \ \ , \forall\, i \in A_k \ , \ j = 1, 2, \ldots, n \ , \ j \neq i \ , \tag{2.8d}$$

and are required to satisfy

$$\gamma_1 w(x_k)^{q_1} \|z\|^2 \leq z^T D^k z \leq \gamma_2 w(x_k)^{q_2} \|z\|^2 \ , \ \ \forall\, z \in \mathbb{R}^n \ , \tag{2.8e}$$

where γ_1 and γ_2 are positive scalars, q_1 and q_2 are non-negative integers and $w(\cdot)$ is defined in (2.3a). It is easy to see that in the case $q_1 = q_2 = 0$, with $\gamma_1 \in (0, 1)$ and $\gamma_2 \in (1, \infty)$, $d_k = -D_k g_k$ will satisfy the conditions required by (2.5a,b,c). If we replace the constants σ_1 and σ_2 by $\sigma_1 w(x_k)^{q_1}$ and $\sigma_2 w(x_k)^{q_2}$, respectively, in (2.5a,b,c), the search directions $d_k = -D^k g_k$ satisfy these tests for all non-negative, integer q_1 and q_2. □

Before proving convergence, we will show that the step-size rule is well defined and that the stopping criterion in Step 1 of Algorithm Model PD is satisfied by a point x_k if and only if x_k is a stationary point.

Proposition 2.5. Let x_k, d_k be any iterate and corresponding search direction constructed by Algorithm Model PD, *i.e.*, d_k satisfies the conditions in (2.5a,b,c). Then

(a) x_k is a stationary point for problem **P** if and only if $x_k(\lambda, d_k) = x_k$ for all $\lambda \geq 0$;

(b) x_k is a stationary point for problem **P** if and only if $\|g_k\|_{I_k} = 0$ and $x_k^i = 0$ for all $i \in A_k$;

(c) if x_k is not a stationary point for problem **P** then there exists $\bar{\lambda} > 0$ such that

$$f(x_k(\lambda, d_k)) - f(x_k) \leq \alpha \left\{ \lambda \langle g_k, d_k \rangle_{I_k} - \langle g_k, x_k - x_k(\lambda, d_k) \rangle_{A_k} \right\}, \ \forall \lambda \in [0, \bar{\lambda}) , \quad (2.9)$$

i.e., the step-size rule (2.6a) is well defined at x_k and will be satisfied with $\lambda_k \geq \min \{ \beta^{-M}, \beta\bar{\lambda} \}$.

Proof.

(a) Suppose that x_k is a stationary point. Then (2.2b) implies that $g_k^i = 0$ for all $i \in I_k$. Hence, $d_k^i = 0$ for all $i \in I_k$, since $\|d_k\|_{I_k} \leq \sigma_2 \|g_k\|_{I_k}$. Hence $x_k^i(\lambda, d_k) = [x_k^i + \lambda d_k^i]_+ = x_k^i$ for all $\lambda \geq 0$, $i \in I_k$. Now, if $i \in A_k$, then $g_k^i > 0$ and, since x_k is stationary, it follows from (2.2b) that $x_k^i = 0$. Hence for all $i \in A_k$, $x_k^i(\lambda, d_k) = [-\lambda m_k^i g_k^i]_+ = 0 = x_k^i$ for all $\lambda \geq 0$. Thus, $x_k(\lambda, d_k) = [x_k]_+ = x_k$ for all $\lambda \geq 0$.

Next, suppose that $x_k(\lambda, d_k) = x_k$ for all $\lambda \geq 0$. Then $d_k^i = 0$ if $x_k^i > 0$ and $d_k^i \leq 0$ if $x_k^i = 0$. Let the index sets $I_1(x_k)$, $I_2(x_k)$ be defined by

$$I_1(x_k) \doteq \{ i \in I_k \mid x_k^i > 0 \} \ , \quad I_2(x_k) \doteq \{ i \in I_k \mid x_k^i = 0 \} \ , \quad (2.10a)$$

so that $I_k = I_1(x_k) \cup I_2(x_k)$. It follows from the above that if $i \in I_1(x_k)$, then $d_k^i = 0$, and if $i \in I_2(x_k)$, then $d_k^i \leq 0$, and also $g_k^i \leq 0$ (by definition of I_k since $x_k^i = 0$). Thus,

$$\langle g_k, d_k \rangle_{I_k} = \langle g_k, d_k \rangle_{I_1(x_k)} + \langle g_k, d_k \rangle_{I_2(x_k)} = \langle g_k, d_k \rangle_{I_2(x_k)} \geq 0 \ . \quad (2.10b)$$

But, from (2.5b), $\langle g_k, d_k \rangle_{I_k} \leq -\sigma_1 \|g_k\|_{I_k^2}$. Therefore, $\|g_k\|_{I_k} = 0$ and hence $g_k^i = 0$ for all $i \in I_k$. For $i \in A_k$, $d_k^i = -m_k^i g_k^i < 0$. Since $x_k^i(\lambda, d_k) = x_k^i$, for all $\lambda \geq 0$, this implies that $x_k^i = 0$. Thus we have that for all $i \in I_k$, $g_k^i = 0$ and for all $i \in A_k$, $g_k^i > 0$ and $x_k^i = 0$. Consequently, x_k is a stationary point.

(b) Suppose that x_k is a stationary point. If $i \in A_k$ then, because $g_k^i > 0$ for all $i \in A_k$, it follows from (2.2b), that $x_k^i = 0$. If $i \in I_k$, then either *(i)* $x_k^i > 0$ in which case it follows directly from (2.2b) that $g_k^i = 0$, or *(ii)* $x_k^i = 0$ in which case, by definition (2.4b) of $I_k \doteq I(x_k)$, $g_k^i \leq 0$; then, because $g_k^i \geq 0$ by (2.2b), we must have $g_k^i = 0$. To complete the proof, suppose that $\|g_k\|_{I_k} = 0$ and $x_k^i = 0$ for all $i \in A_k$. Then, since $x_k^i \geq 0$ for all $i \in I_k$ and, since $g_k^i > 0$ for all $i \in A_k$, it follows that (2.2b) holds for all i.

(c) Suppose that x_k is not a stationary point. Define the following index sets:

$$I_3(x_k) \doteq \{\, i \in I_k \,\big|\, x_k^i > 0 \,,\text{ or } \ (x_k^i = 0 \ \text{ and } \ d_k^i > 0)\,\} \ , \tag{2.11a}$$

$$I_4(x_k) \doteq \{\, i \in I_k \,\big|\, x_k^i = 0 \,,\, d_k^i \le 0 \,\} \,, \tag{2.11b}$$

$$A_1(x_k) \doteq \{\, i \in A_k \,\big|\, x_k^i > 0 \,\} \ , \ A_2(x_k) \doteq \{\, i \in A_k \,\big|\, x_k^i = 0 \,\} \ . \tag{2.11c}$$

First note that $I_3(x_k) \cup A_1(x_k)$ is not empty. To see this, suppose that $i \in I_4(x_k) \cup A_2(x_k)$ for all $i = 1, \ldots, n$. Then $x_k^i = 0$ and $d_k^i \le 0$ (since $d_k^i = -m_k^i g_k^i < 0$ for $i \in A_2(x_k)$) which implies that $x_k^i(\lambda, d_k) = [x_k^i + \lambda d_k^i]_+ = 0 = x_k^i$ for all i. Consequently, by part *(a)* of this proposition, x_k must be a stationary point; this is a contradiction. Now, let

$$\lambda_1 = \sup \{\, \lambda \,\big|\, x_k^i + \lambda d_k^i \ge 0 \,,\ i \in I_3(x_k)\,\} \ , \tag{2.12a}$$

$$\lambda_2 = \sup \{\, \lambda \,\big|\, x_k^i + \lambda d_k^i \ge 0 \,,\ i \in A_1(x_k)\,\} \ . \tag{2.12b}$$

Clearly $\lambda_1 > 0$ (possibly infinite) and $\lambda_2 > 0$. If $I_3(x_k)$ is empty let $\lambda_1 = \infty$ or, if $A_1(x_k)$ is empty, let $\lambda_2 = \infty$. Now, if $i \in I_4(x_k)$, $x_k^i(\lambda, d_k) = [\lambda d_k^i]_+ = 0 = x_k^i$ for all $\lambda \ge 0$. Similarly, if $i \in A_2(x_k)$, $x_k^i(\lambda, d_k) = [-\lambda m_k^i g_k^i]_+ = 0 = x_k^i$ for all $\lambda \ge 0$. On the other hand, if $i \in I_3(x_k)$ and $\lambda \in [0, \lambda_1)$, then $x_k^i(\lambda, d_k) = x_k^i + \lambda d_k^i$ and if $i \in A_1(x_k)$ and $\lambda \in [0, \lambda_2]$, then $x_k^i(\lambda, d_k) = x_k^i - \lambda m_k^i g_k^i$. Therefore, with

$$\bar{d}_k^i = \begin{cases} d_k^i & \text{if } i \in I_3(x_k) \cup A_1(x_k) \\ 0 & \text{otherwise} \end{cases} \quad , \quad i = 1, \ldots, n \ , \tag{2.13a}$$

it follows that

$$x_k(\lambda, d_k) = [x_k + \lambda d_k]_+ = x_k + \lambda \bar{d}_k \ , \ \forall\, \lambda \in [0, \min\{\, \lambda_1, \lambda_2 \,\}) \ . \tag{2.13b}$$

Next, from (2.13a), we obtain

$$\langle \bar{d}_k, g_k \rangle = \langle d_k, g_k \rangle_{I_3(x_k)} + \langle d_k, g_k \rangle_{A_1(x_k)} \ . \tag{2.14a}$$

Now, from (2.5b), $\langle d_k, g_k \rangle_{I_k} = \langle d_k, g_k \rangle_{I_3(x_k)} + \langle d_k, g_k \rangle_{I_4(x_k)} \le -\sigma_1 \| g_k \|_{I_k^2}$. But, for $i \in I_4(x_k)$, $d_k^i \le 0$ and $g_k^i \le 0$, so $\langle d_k, g_k \rangle_{I_4(x_k)} \ge 0$. Thus, $\langle d_k, g_k \rangle_{I_3(x_k)} \le -\sigma_1 \| g_k \|_{I_k^2}$. This, together with (2.5a) and (2.14a), implies that

$$\langle \bar{d}_k, g_k \rangle \le -\sigma_1 \| g_k \|_{I_k}^2 - \sigma_1 \| g_k \|_{A_1(x_k)}^2 \ . \tag{2.14b}$$

Since x_k is not a stationary point, there exists at least one $i \in I_k \cup A_1(x_k)$ such that $g_k^i \ne 0$. Hence $\langle \bar{d}_k, g_k \rangle < 0$, *i.e.* $\bar{d}_k$ is a feasible descent direction. Next, it follows from (2.13b) that, for all $\lambda \in [0, \min\{\, \lambda_1, \lambda_2 \,\})$, the Armijo-like step-size rule in (2.6a) is equivalent to the following requirement on λ,

$$f(x_k + \lambda \bar{d}_k) - f(x_k) \le \alpha \lambda \langle g_k, \bar{d}_k \rangle_{A_k \cup I_3(x_k)} + \alpha \lambda \langle g_k, d_k \rangle_{I_4(x_k)} . \tag{2.14c}$$

But for all $i \in I_4(x_k)$, $g_k^i \le 0$ and $d_k^i \le 0$. Therefore the last term in the (2.14c) is non-negative. Hence (2.14c) is satisfied if the following, harder, condition is satisfied,

$$f(x_k + \lambda \bar{d}_k) - f_k \le \alpha \lambda \langle g_k, \bar{d}_k \rangle . \tag{2.14d}$$

But this is the usual Armijo rule applied to an unconstrained problem which can always be satisfied with a positive step-size when $\langle g_k, \bar{d}_k \rangle < 0$. Hence, there exists $0 < \bar{\lambda} \le \min \{ \lambda_1, \lambda_2 \}$ such that (2.9) holds. □

We will now show that Algorithm Model PD produces a sequence of iterates whose accumulation points are stationary points. The following assumption will be used:

Assumption 2.6. The gradient $\nabla f(\cdot)$ is Lipschitz continuous on bounded subsets of $\mathcal{F}$; *i.e.*, given any bounded set $S \subset \mathcal{F}$, there exists a scalar $L < \infty$ such that

$$\| \nabla f(x) - \nabla f(y) \| \le L \| x - y \| , \ \forall \, x, y \in S . \tag{2.15}$$

□

Lemma 2.7. Suppose that the sequences $\{ x_k \}_{k=0}^{\infty}$ with $x_k \in \mathcal{F}$, $\{ d_k \}_{k=0}^{\infty}$ with $d_k \in \mathbb{R}^n$, and $\{ \lambda_k \}_{k=0}^{\infty}$ with $\lambda_k \in \mathbb{R}$ bounded and non-negative, are such that (2.5a,b,c) and (2.6a) are satisfied by the triplet $\{ x_k, d_k, \lambda_k \}$ for all k. Then, for any $\bar{x} \in \mathcal{F}$ that is not a stationary point, there exists a $\bar{\rho} > 0$ and a $\bar{\delta} > 0$ (depending on $\bar{x}$) such that

$$f(x_k(\lambda_k, d_k)) - f(x_k) \le - \bar{\delta} \tag{2.16}$$

for all k such that $x_k \in B(\bar{x}, \bar{\rho})$.

Proof. We will first show that there exists $\bar{\lambda} > 0$, depending on $\bar{x}$, such that $\lambda_k \ge \bar{\lambda}$ for all k such that x_k is sufficiently close to $\bar{x}$. We will then use this $\bar{\lambda}$ to derive $\bar{\rho} > 0$ and $\bar{\delta} > 0$ such that (2.16) holds. Let $S \subset \mathcal{F}$ be a bounded neighborhood of $\bar{x}$. For any $x_k \in S$, $\nabla f(x_k)$ is bounded because it is a continuous function. By (2.5a,c) d_k is also bounded. This implies that $x_k(\lambda_k, d_k)$, $\lambda_k \in [0, \beta^{-M}]$, is bounded. Thus, by Assumption 2.6 we have that there exists an $L < \infty$ such that for $s \in [0, 1]$,

$$\| g_k - \nabla f(x_k - s[x_k - x_k(\lambda, d_k)]) \|_2 \le sL \| x_k - x_k(\lambda, d_k) \| , \ \forall \, x_k \in S . \tag{2.17}$$

Expanding the left-hand side of (2.6a) we have, for $x_k \in S$ and $\lambda \in [0, \beta^{-M}]$,

$$f(x_k(\lambda, d_k)) - f(x_k) = \langle g_k, x_k(\lambda, d_k) - x_k \rangle$$

$$+ \int_0^1 \langle \nabla f(x_k - s[x_k - x_k(\lambda, d_k)]) - g_k, x_k(\lambda, d_k) - x_k \rangle \, ds$$

$$\leq \langle g_k, x_k(\lambda, d_k) - x_k \rangle + \| x_k(\lambda, d_k) - x_k \| \int_0^1 sL \| x_k(\lambda, d_k) - x_k \| \, ds$$

$$= \langle g_k, x_k(\lambda, d_k) - x_k \rangle + \frac{L}{2} \| x_k(\lambda, d_k) - x_k \|^2 \, . \tag{2.18}$$

Now, for $i \in A_k$, $x_k^i(\lambda, d_k) = [x_k^i - \lambda m_k^i g_k^i]_+ \geq x_k^i - \lambda m_k^i g_k^i$ so that $x_k^i - x_k^i(\lambda, d_k) \leq \lambda m_k^i g_k^i$. Thus,

$$\lambda \sum_{i \in A_k} m_k^i g_k^i [x_k^i - x_k^i(\lambda, d_k)] \geq \| x_k - x_k(\lambda, d_k) \|_{A_k}^2 \, . \tag{2.19}$$

Now consider the sets $I_{5,k} \doteq \{ i \in I_k \mid g_k^i > 0 \}$ and $I_{6,k} \doteq \{ i \in I_k \mid g_k^i \leq 0 \}$. If $i \in I_{5,k}$ then $x_k^i > \varepsilon_k$ (for otherwise $i \in A_k$). Since $\hat{x}$ is not a stationary point, $\| \bar{x} - [\bar{x} - \nabla f(\bar{x})]_+ \| > 0$ (see discussion of equation (2.3c)). Thus, since $w(\cdot)$ is continuous and $\varepsilon(x) = \min \{ \varepsilon, w(x) \}$, there exists $\rho_1 > 0$ and $\bar{\varepsilon} > 0$ such that *(i)* $\varepsilon(x_k) \geq \bar{\varepsilon}$ for all $x_k \in B(\bar{x}, \rho_1)$ and *(ii)* $B(\bar{x}, \rho_1) \subset S$. Let $\Delta < \infty$ be such that $\| g_k \| \leq \Delta / \sigma_1$ for all $x_k \in B(\bar{x}, \rho_1)$. Then, for all k such that $x_k \in B(\bar{x}, \rho_1)$, we have from (2.5b) that $|d_k^i| \leq \Delta$ for all $i \in I_k$. Hence, for $\lambda \in [0, \bar{\varepsilon} / \Delta]$ and $i \in I_{5,k}$, $x_k^i(\lambda, d_k) = x_k^i + \lambda d_k^i$ and

$$\sum_{i \in I_{5,k}} g_k^i [x_k^i - x_k^i(\lambda, d_k)] = -\lambda \langle g_k, d_k \rangle_{I_{5,k}} \, , \ \forall \, \lambda \in [0, \bar{\varepsilon} / \Delta] \, . \tag{2.20a}$$

Next, for all $\lambda \geq 0$, $x_k^i - x_k^i(\lambda, d_k) \leq -\lambda d_k^i$, and since $g_k^i \leq 0$ for $i \in I_{6,k}$, we have that

$$\sum_{i \in I_{6,k}} g_k^i [x_k^i - x_k^i(\lambda, d_k)] \geq -\lambda \langle g_k, d_k \rangle_{I_{6,k}} \, , \ \forall \, \lambda \geq 0 \, . \tag{2.20b}$$

Combining these last two expressions gives us

$$\langle g_k, x_k - x_k(\lambda, d_k) \rangle_{I_k} \geq -\lambda \langle g_k, d_k \rangle_{i \in I_k} \, , \ \forall \, \lambda \in [0, \bar{\varepsilon} / \Delta] \tag{2.20c}$$

for all k such that $x_k \in B(\bar{x}, \rho_1)$. Finally, from (2.5b,c) we have that $-\langle d_k, g_k \rangle_{I_k} \geq \sigma_1 \| g_k \|_{I_k}^2 \geq \frac{\sigma_1}{\sigma_2^2} \| d_k \|_{I_k}^2$, and since, for any $i \in \{ 1, \ldots, n \}$ and all $\lambda \geq 0$, $|x_k^i - x_k^i(\lambda)| \leq \lambda |d_k^i|$, we have that

$$\| x_k - x_k(\lambda, d_k) \|_{I_k}^2 \leq \lambda^2 \| d_k \|_{I_k}^2 \leq -\lambda^2 \frac{\sigma_2^2}{\sigma_1} \langle d_k, g_k \rangle_{I_k} \, . \tag{2.21}$$

Thus, from (2.20c), we see that, for all $x_k \in B(\bar{x}, \rho_1)$,

$$\langle g_k, x_k(\lambda, d_k) - x_k \rangle = -\langle g_k, x_k - x_k(\lambda, d_k)\rangle_{A_k} - \langle g_k, x_k - x_k(\lambda, d_k)\rangle_{I_k}$$

$$\leq -\langle g_k, x_k - x_k(\lambda, d_k)\rangle_{A_k} + \lambda \langle g_k, d_k \rangle_{I_k} \,, \tag{2.22a}$$

and, from (2.19) and (2.21),

$$\frac{L}{2} \| x_k(\lambda, d_k) - x_k \| = \frac{L}{2} \| x_k(\lambda, d_k) - x_k \|_{A_k}^2 + \frac{L}{2} \| x_k(\lambda) - x_k \|_{I_k}^2$$

$$\leq \frac{\lambda \sigma_2 L}{2} \langle g_k, x_k - x_k(\lambda, d_k)\rangle_{A_k} - \lambda^2 \frac{\sigma_2^2 L}{2\sigma_1} \langle g_k, d_k \rangle_{I_k} \,, \ \forall\, \lambda \in [0, \bar{\varepsilon} \,/\, \Delta] \,. \tag{2.22b}$$

Substituting the expressions (2.22a,b) into (2.18), we obtain that for all $\lambda \in [0, \bar{\varepsilon} \,/\, \Delta]$ and $x_k \in B(\bar{x}, \rho_1)$,

$$f(x_k(\lambda, d_k)) - f(x_k) \leq \lambda \big(1 - \lambda \frac{\sigma_2^2 L}{2\sigma_1}\big) \langle g_k, d_k \rangle_{I_k} + \big(\frac{\lambda \sigma_2 L}{2} - 1\big) \langle g_k, x_k - x_k(\lambda, d_k)\rangle_{A_k} \,. \tag{2.23}$$

Comparing this with (2.6a), and noting from (2.5b) and (2.7) that $\langle g_k, d_k \rangle_{I_k} \leq 0$ and $\langle g_k, x_k - x_k(\lambda, d_k)\rangle_{A_k} \geq 0$, we see that the Armijo-like rule is satisfied for any $\lambda \geq 0$ such that $\lambda \leq \bar{\varepsilon} / \Delta$, $\lambda - \lambda^2 L \sigma_2^2 / 2\sigma_1 \geq \alpha \lambda$ and $\lambda \sigma_2 L / 2 - 1 \leq -\alpha$. Since $\beta \in (0,1)$ and the step-size rule requires the smallest m such that $\lambda = \beta^m$ satisfies (2.6a), we see that for all $x_k \in B(\bar{x}, \rho_1)$,

$$\lambda_k \geq \bar{\lambda} \doteq \min \left\{ \frac{\bar{\varepsilon}}{\Delta}, \frac{2\sigma_1(1-\alpha)}{\sigma_2^2 L}, \beta^{-M} \right\} > 0 \,. \tag{2.24}$$

Now we will use (2.24) to show that (2.16) holds. For any $i \in I_k$,

$$\langle g_k, d_k \rangle_{I_k} \leq -\sigma_1 \| g_k \|_{I_k}^2 \leq -\sigma_1 (g_k^i)^2 \,. \tag{2.25a}$$

Also, for all $i \in A_k$, $g_k^i > 0$ and $x_k^i - x_k^i(\lambda_k, d_k^i) \geq 0$. Thus, for any $i \in A_k$, $x_k \in B(\bar{x}, \rho_1)$,

$$\langle g_k, x_k - x_k(\lambda_k, d_k)\rangle_{A_k} \geq g_k^i (x_k^i - x_k^i(\lambda_k, d_k))$$

$$\geq g_k^i (x_k^i - x_k^i(\bar{\lambda}, d_k))$$

$$\geq (g_k^i) \min \{ x_k^i, \bar{\lambda} / m_k^i g_k^i \} = \min \{ g_k^i x_k^i, \bar{\lambda} / m_k^i \} \,, \tag{2.25b}$$

where $\bar{\lambda}$ is given by (2.24). Let $\bar{g} \doteq \nabla f(\bar{x})$. Since $\bar{x}$ is not stationary, there must exist an $i_0 \in \{1, \ldots, n\}$ such that either *(i)* $\bar{g}^{i_0} < 0$ or *(ii)* $\bar{x}^{i_0} > 0$ and $\bar{g}^{i_0} \neq 0$. By continuity of $\nabla f(\cdot)$, there exist $\rho_2 \in (0, \rho_1]$ such that

$$\tfrac{1}{2} |\bar{g}^{i_0}| \leq |g_k^{i_0}| \leq 2 |\bar{g}^{i_0}| \,, \ \forall\, x_k \in B(\bar{x}, \rho_2) \,. \tag{2.25c}$$

Thus, if $\bar{x}^{i_0} = 0$, then $\bar{g}^{i_0} < 0$ and, for all $x_k \in B(\bar{x}, \rho_2)$, $g_k^{i_0} < 0$. Hence, $i_0 \in I_k$ and, by (2.25a,c),

$$\langle g_k, d_k \rangle_{I_k} \le -\sigma_1 (\bar g^{i_0})^2 / 4 \doteq -\delta_{1.} \tag{2.26a}$$

If $\bar x^{i_0} > 0$, then we must have $\bar g^{i_0} > 0$. Thus, from (2.25c), if $x_k \in B(\bar x, \rho_2)$ then $i_0 \in A_k$. So, if $\bar x^{i_0} > 0$, then for all $k \in K$ such that $\|x_k - \bar x\| \le \min\{ \rho_2, \bar x^{i_0}/2 \}$, we have from (2.25b,c),

$$-\langle g_k, x_k - x_k(\lambda_k, d_k) \rangle_{A_k} \le -\min\{ \bar g^{i_0} \bar x^{i_0}/4, \bar\lambda / m_k^{i_0} \} \doteq -\delta_2 \,. \tag{2.26b}$$

Now, from (2.6a) and (2.16), we have

$$f(x_{k+1}) - f(x_k) \le f(x_k(\lambda_k, d_k)) - f(x_k) \le \alpha \{ \lambda_k \langle g_k, d_k \rangle_{I_k} - \langle g_k, x_k - x_k(\lambda_k, d_k) \rangle_{A_k} \} \,. \tag{2.27}$$

Since it is always the case that $\langle g_k, d_k \rangle_{I_k} \le -\sigma_1 \|g_k\|^2_{I_k} \le 0$ and $-\langle g_k, x_k - x_k(\lambda_k, d_k) \rangle_{A_k} \le 0$, we have from (2.26a,b) and (2.27), that

$$f(x_{k+1}) - f(x_k) \le -\bar\delta < 0 \,, \ \forall x_k \in B(\bar x, \bar\rho) \,, \ k \in K \,, \tag{2.28}$$

where $\bar\delta = \alpha\delta_1$ and $\bar\rho = \rho_2 > 0$ if $\bar x^{i_0} = 0$ or $\bar\delta = \alpha\delta_2$ and $\bar\rho = \min\{ \rho_2, \bar x^{i_0}/2 \} > 0$ if $\bar x^{i_0} > 0$. □

Theorem 2.8. Suppose that Assumption 2.6 holds. Let $\{ x_k \}_{k=0}^{\infty}$ with $x_k \in \mathcal{F}$, $\{ d_k \}_{k=0}^{\infty}$ with $d_k \in \mathbb{R}^n$, and $\{ \lambda_k \}_{k=0}^{\infty}$ with $\lambda_k \in \mathbb{R}$ bounded and non-negative, be sequences such that, for all k, (2.5a,b,c) and (2.6a) are satisfied by the triplet $\{ x_k, d_k, \lambda_k \}$ and

$$f(x_{k+1}) \le f(x_k(\lambda_k, d_k)) \,. \tag{2.29}$$

Then any accumulation point, $\hat x$, of $\{ x_k \}_{k=0}^{\infty}$ is a stationary point of problem **P**.

Proof. We will prove this result be contradiction. Suppose that $\hat x$ is an accumulation point of $\{ x_k \}_{k=0}^{\infty}$. Then there exists an infinite set $K \subset \mathbb{N}$ such that $\lim_{k \in K} x_k = \hat x$. By continuity of $f(\cdot)$, this implies that $\lim_{k \in K} f(x_k) = f(\hat x)$. Additionally, for each $k \in \mathbb{N}$, $f(x_{k+1}) \le f(x_k(\lambda_k, d_k) \le f(x_k)$ since the right hand side of (2.6a) is non-positive (*cf.* Remark 2.2). Hence, $f(x_k) \to f(\hat x)$ and, therefore,

$$f(x_k) - f(x_{k+1}) \to 0 \ \text{ as } k \to \infty \,. \tag{2.30}$$

Now, to establish the contradiction, suppose that $\hat x$ is not a stationary point. Then, by (2.29) and Lemma 2.7, there exists $\hat\rho > 0$ and $\hat\delta > 0$ such that $f(x_k) - f(x_{k+1}) \ge f(x_k) - f(x_k(\lambda_k, d_k)) \ge \hat\delta$ for all $x_k \in B(\hat x, \hat\rho)$. But this contradicts (2.30). Therefore $\hat x$ must be a stationary point. □

Corollary 2.9. Suppose that Assumption 2.6 holds. If $\{ x_k \}_{k=0}^{k_f}$ is a finite sequence generated by Algorithm Model PD, then x_{k_f} is a stationary point of problem **P**. If $\{ x_k \}_{k=0}^{\infty}$ is an infinite sequence generated by Algorithm Model PD, then every accumulation point of $\{ x_k \}_{k=0}^{\infty}$ is a stationary point of problem **P**.

Proof. First suppose $\{ x_k \}_{k=0}^{k_f}$ is a finite sequence. Then by Proposition 2.5*(b)*, x_{k_f} is a stationary point. If $\{ x_k \}_{k=0}^{\infty}$ is an infinite sequence then, by Theorem 2.8 (we have, trivially, $f(x_{k+1}) = f(x_k(\lambda_k, d_k))$, every accumulation point is a stationary point. □

Next, we proceed towards a proof that under suitable conditions, after a finite number of iterations, Algorithm Model PD reverts to an unconstrained optimization algorithm on the subspace defined by the non-binding variables at a strict local minimizer limit point. Let $\mathcal{B}(x)$ denote the set of all binding constraints at x, *i.e.*

$$\mathcal{B}(x) \doteq \{ i \mid x^i = 0 \} \tag{2.31}$$

(this $\mathcal{B}(x)$ should not be confused with $B(x, \rho)$, the closed ball of radius ρ around x). We will use the following alternative statement of the standard second order sufficiency condition with strict complementary slackness for a stationary point $\hat{x}$ to be a strict local minimizer for a problem **P** with $f(\cdot)$ twice continuously differentiable[†]:

$$z^T \nabla^2 f(\hat{x}) z > 0 \;, \; \forall z \in \{ z \in \mathbb{R}^n \mid z^i = 0 \;, \; \forall i \in \mathcal{B}(\hat{x}) \} \;, \tag{2.32a}$$

and

$$\frac{\partial f(\hat{x})}{\partial x^i} > 0 \;, \; \forall i \in \mathcal{B}(\hat{x}) \;. \tag{2.32b}$$

Theorem 2.10. Suppose $f(\cdot)$ is twice continuously differentiable. Consider a sequence $\{ x_k \}_{k=0}^{\infty}$ produced by Algorithm Model PD. If $\{ x_k \}_{k=0}^{\infty}$ has an accumulation point $\hat{x}$ such that (2.32a,b) hold, then $x_k \to \hat{x}$ and there exists an $N < \infty$ such that

$$A_k = \mathcal{B}(x_k) = \mathcal{B}(\hat{x}) \;, \; \forall k \geq N + 1 \;. \tag{2.33}$$

Proof. Since $f(\cdot)$ is twice continuously differentiable, Assumption 2.6 holds. Thus, by Corollary 2.9, $\hat{x}$ is a stationary point. It therefore follows from (2.2b) and the definition of $w(x)$ in (2.3a) that $w(\hat{x}) = 0$. Hence, because $w(\cdot)$ is continuous, there exists a $\rho_1 > 0$ such that $\varepsilon(x) \doteq \min \{ \varepsilon, w(x) \} = w(x)$ for all $x \in B(\hat{x}, \rho_1)$. Now, since *(i)* x^i, $i \in \mathcal{B}(\hat{x})$, is arbitrarily close to zero if x sufficiently close to $\hat{x}$, and *(ii)* $g(\cdot) \doteq \nabla f(\cdot)$ is continuous, and thus by (2.32b), bounded away from zero for x sufficiently close to $\hat{x}$, we have that for x sufficiently close to $\hat{x}$, $g^i(x) \geq x^i$ for all $i \in \mathcal{B}(\hat{x})$. Thus, there exists $\rho_2 \in (0, \rho_1]$ such that for all $x \in B(\hat{x}, \rho_2)$

$$[x^i - g^i(x)]_+ = 0 \;, \; \forall i \in B(\hat{x}) \tag{2.34a}$$

[†] See, for example, [88, pp. 316-317]: *viz.*, equation (2.2b) holds, the Hessian of the Lagrangian $L(x) \doteq f(x) - \mu^T x$ is positive definite on the subspace $\{ z \in \mathbb{R}^n \mid z^i = 0 \;, \; i \in \mathcal{B}(\hat{x}) \}$ and the multipliers $\mu^i = \partial f(\hat{x}) / \partial x^i$ are positive for all $i \in \mathcal{B}(\hat{x})$ and zero for all $i \notin \mathcal{B}(\hat{x})$.

$$g^i(x) > 0\ ,\ \forall\, i \in B(\hat{x})\ . \tag{2.34b}$$

From (2.34a) it follows that for $x \in B(\hat{x}, \rho_2)$,

$$\varepsilon^2(x) = w^2(x) = \|x - [x - g(x)]_+\|^2 = \sum_{i \,\in\, \mathcal{B}(\hat{x})} (x^i)^2 + \sum_{i \,\notin\, \mathcal{B}(\hat{x})} (x^i - [x^i - g^i]_+)^2 \geq \sum_{i \,\in\, \mathcal{B}(\hat{x})} (x^i)^2 \tag{2.35a}$$

and hence,

$$x^i \leq \varepsilon(x)\ ,\ \forall\, i \in B(\hat{x}) \text{ and } x \in B(\hat{x}, \rho_2)\ . \tag{2.35b}$$

Also, since $(\hat{x})^i > 0$ for all $i \notin B(\hat{x})$ and $\varepsilon(\hat{x}) = w(\hat{x}) = 0$, there exist scalars $\tilde{\varepsilon} > 0$ and $\rho_3 \in (0, \rho_2]$ such that

$$x^i \geq \tilde{\varepsilon} > \varepsilon(x)\ ,\ \forall\, i \notin \mathcal{B}(\hat{x}) \text{ and } x \in B(\hat{x}, \rho_3)\ . \tag{2.35c}$$

Thus, we see from (2.34b), (2.35b), (2.35c) and the definition of $A(x)$ given in (2.4a), that

$$A(x) = B(\hat{x})\ ,\ \forall\, x \in B(\hat{x}, \rho_3)\ . \tag{2.36}$$

To establish the next point, we need to assert that there exists $\bar{\lambda} > 0$ such that $\lambda_k \geq \bar{\lambda}$ for all k such that $x_k \in B(\hat{x}, \rho_3)$. We cannot directly use (2.24) in the proof of Lemma 2.7 because that quantity was derived by assuming, in the derivation of equation (2.20a), that $\hat{x}$ was *not* a stationary point. However, from (2.35c) and (2.36), we know that if $x_k \in B(\hat{x}, \rho_3)$ then $x_k^i > \tilde{\varepsilon}$ for $i \in I_k$. We can therefore use $\tilde{\varepsilon}$ for equation (2.20a) in lieu of the $\bar{\varepsilon}$. Thus, $\lambda_k \geq \bar{\lambda}$ for all k such that $x_k \in B(\hat{x}, \rho_3)$, where $\bar{\lambda}$ is defined in (2.24) with $\bar{\varepsilon}$ replaced by $\tilde{\varepsilon}$. Now for any k, i such that $x_k \in B(\hat{x}, \rho_3)$ and $i \in A_k$ we have $d_k^i = -m_k^i g_k^i < 0$, $x_k^i \leq \varepsilon(x_k)$ and by (2.36), $i \in \mathcal{B}(\hat{x})$. Thus, from (2.32b), the continuity of $\nabla f(\cdot)$ and the fact that $\lambda_k \geq \bar{\lambda}$ for all k, there exists a $\rho_4 \in (0, \rho_3]$ such that for any $i \in A_k$ and $x_k \in B(\hat{x}, \rho_4)$, $0 \leq x_{k+1}^i = [x_k^i - \lambda_k m_k^i g_k^i]_+ \leq [x_k^i - \bar{\lambda} m_k^i g_k^i]_+ = 0$. This implies that $i \in \mathcal{B}(x_{k+1})$. Hence,

$$A_k \subset \mathcal{B}(x_{k+1})\ ,\ \forall\, x_k \in B(\hat{x}, \rho_4)\ . \tag{2.37}$$

On the other hand, for any k, i such that $x_k \in B(\hat{x}, \rho_4)$ and $i \notin A_k$, we have from (2.36) that $i \notin \mathcal{B}(\hat{x})$ and hence, by (2.35c), $x_k^i > \tilde{\varepsilon}$. Since $\hat{x}$ is a stationary point, we see from (2.2b) that $\partial f(\hat{x}) / \partial x^i = 0$ for all $i \notin \mathcal{B}(\hat{x})$. Also, λ_k is bounded, $\varepsilon(\cdot)$ is continuous, and by (2.5c), $\|d_k\|_{I_k} < \sigma_2 \|g_k\|_{I_k}$. Therefore, since $\|x_{k+1} - x_k\| \leq \|x_{k+1} - x_k\|_{A_k} + \|x_{k+1} - x_k\|_{I_k} \leq \varepsilon(x_k) + \sigma_2 \|g_k\|_{I_k}$ is arbitrarily small for x_k sufficiently close to $\hat{x}$, there exists $\rho_5 \in (0, \rho_4]$ such that if $x_k \in B(\hat{x}, \rho_5)$, then *(i)* $x_{k+1} \in B(\hat{x}, \rho_4)$ and *(ii)* $x_{k+1}^i > \varepsilon(x_{k+1})$ for $i \notin A_k$. It follows from *(i)* and (2.36) that $A_k = A_{k+1} = \mathcal{B}(\hat{x})$. From *(ii)* it follows that $i \notin A_k$ implies that $i \notin \mathcal{B}(x_{k+1})$ and hence, $\mathcal{B}(x_{k+1}) \subset A_k$. These facts together with (2.37) allow us to conclude that

$$\mathcal{B}(x_{k+1}) = A_{k+1} = A_k = \mathcal{B}(\hat{x})\ ,\ \forall\, x_k \in B(\hat{x}, \rho_5)\ . \tag{2.38}$$

Hence, the $(k+1)$-th iteration is equivalent to an unconstrained minimization on the subspace $\{\ x \in \mathbb{R}^n \mid x^i = 0 \text{ for } i \in \mathcal{B}(\hat{x})\ \}$. Therefore, since (2.5c) and (2.32a) hold and since $\hat{x}$ is a stationary point, we can invoke Proposition 1.12 in [89] which states that there exists an open set $N(\hat{x})$ containing $\hat{x}$ such that $N(\hat{x}) \subset B(\hat{x}, \rho_5)$ and with the property that for any k such that $x_{k+1} \in N(\hat{x})$ and $\mathcal{B}(x_{k+1}) = \mathcal{B}(\hat{x})$, $x_{k+2} \in N(\hat{x})$ and, by (2.38), $\mathcal{B}(x_{k+2}) = \mathcal{B}(\hat{x})$. By continuing in this manner, we can conclude that if there is an N such that $x_{N+1} \in N(\hat{x})$ and $\mathcal{B}(x_{N+1}) = \mathcal{B}(\hat{x})$, then $x_k \in N(\hat{x})$ and $\mathcal{B}(x_k) = \mathcal{B}(\hat{x})$ for all $k \geq N+1$. We show that such an N exists as follows. Using the same arguments that led up to (2.38), there exists $\rho_6 \in (0, \rho_5]$ such that if $x_k \in B(\hat{x}, \rho_6)$ then $x_{k+1} \in N(\hat{x}) \subset B(\hat{x}, \rho_5)$ and, by (2.38), $\mathcal{B}(x_{k+1}) = \mathcal{B}(\hat{x})$. Since $\hat{x}$ is an accumulation point, there exists $N < \infty$ such that $x_N \in B(\hat{x}, \rho_6)$. Thus, $x_{N+1} \in N(\hat{x})$ and $\mathcal{B}(x_{N+1}) = \mathcal{B}(\hat{x})$ and therefore $x_k \in N(\hat{x})$ and $\mathcal{B}(x_k) = \mathcal{B}(\hat{x})$ for all $k \geq N+1$. This allows us to conclude, also using Proposition 1.12 in [89], that $x_k \to \hat{x}$. □

It follows from Theorem 2.10 that, under the conditions stated, Algorithm Model PD will identify the constrained components of the solution $\hat{x}$ after a finite number of iterations N. Hence, for all $k \geq N+1$, $x_k^i = 0$ if $i \in \mathcal{B}(\hat{x})$ and $x_k^i > 0$ if $i \notin \mathcal{B}(\hat{x})$. Consequently, for all $k \geq N+1$, Algorithm Model PD reduces to an unconstrained optimization algorithm on the subspace $\{\ x \in \mathbb{R}^n \,|\, x^i = 0\ , \forall\, i \in \mathcal{B}(\hat{x})\ \}$ and its rate of convergence is governed entirely by the rules used in the construction of the components d_k^i, $i \in I_k$, of the search direction.

The use of other step-size rules.

Typically, the unconstrained portion, d_k^i, $i \in I_k$, of the search direction required by Algorithm Model PD is constructed from a standard method such as the conjugate gradient method or a variable metric method (we demonstrate this construction in Section 3). Depending on the method used to construct the unconstrained direction, it may be useful to require the step-size to satisfy a stronger condition than the Armijo rule. For instance, the conjugacy of search directions produced by a conjugate gradient algorithm depends strongly on the accuracy of the line search and, therefore, it is usually more efficient to use a step-size that provides a more accurate line minimization than the Armijo step-size.

In order to incorporate a more accurate line search method into Algorithm Model PD, we propose the following modification which preserves the results of Theorem 2.10.

Modified Step-size Procedure: Let $\sigma_3 \in (0, 1]$ and $\sigma_4 \in (0, \infty)$ be given. At iteration k of Algorithm Model PD, let λ_k be the Armijo step-size satisfying (2.6a) and let $\lambda'_k \in [0, \sigma_4]$ be another step-size. If $\lambda'_k \geq \sigma_3 \lambda_k$, then set $\tilde{x} = x_k(\lambda'_k, d_k)$. Otherwise, set

$$\tilde{x}^i = \begin{cases} x_k^i(\lambda'_k, d_k) & \text{if } i \in I_k \\ x_k^i(\sigma_3 \lambda_k, d_k) & \text{if } i \in A_k \end{cases} . \tag{2.39}$$

If $f(\tilde{x}) \leq f(x_k(\lambda_k, d_k))$ then set $x_{k+1} = \tilde{x}$. Otherwise, set $x_{k+1} = x_k(\lambda_k, d_k)$. □

Proposition 2.11. Suppose $f(\cdot)$ is twice continuously differentiable. Consider a sequence $\{ x_k \}_{k=0}^{\infty}$ produced by Algorithm Model PD using the Modified Step-size Procedure. If $\{ x_k \}_{k=0}^{\infty}$ has an accumulation point $\hat{x}$ that satisfies the second order sufficiency conditions (2.32a,b), then $x_k \rightarrow \hat{x}$ and there exists an $N < \infty$ such that

$$A_k = \mathcal{B}(x_k) = \mathcal{B}(\hat{x}) \,, \ \forall\, k \geq N + 1 \,. \tag{2.40}$$

Proof. Let $\{ x_k \}_{k=0}^{\infty}$ be a sequence of iterates produced by Algorithm Model PD using the Modified Step-size Procedure. First, by construction $f(x_{k+1}) \leq f(x_k(\lambda_k, d_k))$ for all k. Therefore, by Theorem 2.8, any accumulation point, $\hat{x}$, of $\{ x_k \}_{k=0}^{\infty}$ must be a stationary point of problem **P**. This being the case, Proposition 2.11 follows from the proof of Theorem 2.10 with the following modification. In deriving (2.37), we used the fact that $\lambda_k \geq \bar{\lambda} > 0$ for all k such that $x_k \in B(\hat{x}, \rho_3)$. However, the new step-size, λ'_k, may not be bounded away from zero. Nonetheless, we can show that (2.37) still holds. First, if $f(\tilde{x}) > f(x_k(\lambda_k, d_k))$ then $x_{k+1} = x_k(\lambda_k, d_k)$. Clearly then, (2.37) holds since x_{k+1} is determined from the unmodified step-size rule in Algorithm Model PD as in Theorem 2.10. If, on the other hand, $f(\tilde{x}) \leq f(x_k(\lambda_k, d_k))$, then by construction of $\tilde{x}$ we have, for all $i \in A_k$, that $x_{k+1}^i = \tilde{x}^i \leq [x_k^i - \sigma_3 \lambda_k g_k^i]_+ \leq [x_k^i - \sigma_3 \bar{\lambda} g_k^i]_+$. Now, by (2.36), the continuity of $\nabla f(\cdot)$ and the fact that $\partial f(\hat{x}) / \partial x^i > 0$ for all $i \in \mathcal{B}(\hat{x})$, there exists $\rho_4 \in (0, \rho_3]$ such that $[x_k^i - \sigma_3 \bar{\lambda} g_k^i]_+ = 0$ for all $i \in A_k$ and $x_k \in B(\hat{x}, \rho_4)$. Thus, if $x_k \in B(\hat{x}, \rho_4)$ and $i \in A_k$, then $x_{k+1}^i = 0$ and, hence, $i \in \mathcal{B}(x_{k+1})$. So, again, (2.37) holds. The rest of the proof of Theorem 2.10 holds without further modification. □

The important aspect of the Modified Step-size Procedure is that once the active constraint set is identified and those variables in the active set are at their bounds, $x_k^i(\lambda_k, d_k) = x_k^i(\lambda'_k, d_k) = 0$, for all $i \in A_k$. Therefore, $\tilde{x} = x_k(\lambda_k', d_k)$ and the algorithm behaves as an unconstrained algorithm using the step-size λ'_k (so long as $f(x_k(\lambda'_k, d_k) \leq f(x_k(\lambda_k, d_k)))$. This fact can used to obtain the properties associated with almost any step-size procedure. For instance, consider the *strong Wolfe criterion*:

$$f(x_k + \alpha_k d_k) - f(x_k) \le \alpha_1 \lambda_k \langle g_k, d_k \rangle \tag{2.41a}$$

$$| \langle \nabla f(x_k + \lambda_k d_k), d_k \rangle | \le -\alpha_2 \langle g_k, d_k \rangle \,, \tag{2.41b}$$

with $0 < \alpha_1 < \alpha_2 < \frac{1}{2}$. This step-size rule can be implemented in Algorithm Model PD using the Modified Step-size Procedure by requiring λ'_k to satisfy

$$f(x_k(\lambda'_k, d_k)) - f(x_k) \ \le\ \alpha_1 \left\{ \lambda'_k \langle g_k, d_k \rangle_{I_k} \ -\ \langle g_k, x_k - x_k(\lambda'_k, d_k) \rangle_{A_k} \right\} . \tag{2.41c}$$

and

$$| \langle \nabla f(x_k(\lambda'_k, d_k)), d_k \rangle_{I_k} | \le -\alpha_2 \langle g_k, d_k \rangle_{I_k} \,. \tag{2.41d}$$

Then, by Proposition 2.11, Algorithm Model PD reverts to an unconstrained minimization over the subspace $\{ i \mid x_k^i \notin \mathcal{B}(\hat{x}) \}$ in a finite number of iterations and, therefore, conditions (2.41c,d) become equivalent to conditions (2.41a,b).

The usefulness of Proposition 2.11 is demonstrated in our next result which deals with the rate of convergence of an implementation of Algorithm Model PD that uses the Polak-Ribière conjugate gradient rule to construct the components d_k^i, $i \in I_k$, of the search direction d_k. Corollary 2.12 states that Algorithm Model PD with the Modified Step-size procedure using exact line searches, with search directions d_k given by (2.45a,b,c) and restarts imposed every $m+1$ iterations, has iterates that converge $(m+1)$-step linearly with a root rate constant that depends on only the smallest $n-r-m$ eigenvalues of the Hessian at the solution restricted to the unconstrained subspace. Here, $r = |\mathcal{B}(\hat{x})|$ is the number of constraints binding at the solution. Since it follows from the interlacing eigenvalue property of symmetric matrices [72, Cor. 8.1.4] that the condition of the restricted Hessian is no worse than that of the Hessian itself, the presence of bounds on the decision variables can only serve to reduce the convergence rate constant. For problems that include penalty functions, if m is taken to be the number of penalized constraints then Corollary 2.12 shows that the $(m+1)$-step convergence root rate constant is independent of the size of the penalty constant (see [90]).

Corollary 2.12. Suppose that

(a) in problem **P**, $f(\cdot)$ is three times continuously differentiable with positive definite Hessian $H(x)$ on $\mathcal{F}$ and that $\hat{x}$, the unique global minimizer of **P**, satisfies the sufficient conditions (2.32a,b)†, and that $\mathcal{B}(\hat{x}) \doteq \{ n-r+1, \ldots, n \}$, with $1 \le r \le n$ (achieved by renumbering the

† The convexity of the constraint set and the strict convexity of the objective function guarantees that **P** has a unique global minimizer.

variables, if necessary).

(b) $\{ x_k \}$ is a sequence produced by Algorithm Model PD using the Modified Step-size Procedure with search directions d_k determined as follows (with $d_{-1} = 0 \in \mathbb{R}^n$):

$$\tilde{d}^{\,i}_k = -g^i_k + \mu_k d^i_{k-1} \ , \ i \in I_k \ , \ \text{where} \quad \mu_k = \mu^{PR}_k \doteq \frac{\langle g_k, g_k - g_{k-1} \rangle_{I_k}}{\|g_{k-1}\|^2_{I_k}} \ , \tag{2.42a}$$

$$d^i_k = \begin{cases} \tilde{d}^{\,i}_k & \text{if} \ \ \langle \tilde{d}_k, g_k \rangle_{I_k} \leq -\sigma_1 \|g_k\|^2_{I_k} \ \text{ and } \ \|\tilde{d}_k\|_{I_k} \leq \sigma_2 \|g_k\|_{I_k} \\ -g^i_k & \text{otherwise} \end{cases} \ , \ \forall\, i \in I_k \ , \tag{2.42b}$$

$$d^i_k = -m^i_k g^i_k \ , \ \sigma_1 \leq m^i_k \leq \sigma_2 \ , \ \forall\, i \in A_k \ , \tag{2.42c}$$

and with the step-size λ'_k determined by an exact line search, and with restarts ($d^i_k = -g^i_k$ for $i \in I_k$) imposed every $m + 1 \leq n - r$ iterations.

Let $H_{1,1}(\hat{x})$ denote the upper-left $(n - r) \times (n - r)$ diagonal block of $H(\hat{x})$, and let a denote its minimum eigenvalue and b its $(m + 1)$-th largest ($(n - m - r)$-th smallest) eigenvalue. If, in Algorithm Model PD, $\sigma_2 \geq 1 + b / a$, then for any $\delta > 0$ there exists $N < \infty$ such that for all $k \geq N / (m + 1)$,

$$\|x_{(k+n)(m+1)} - \hat{x}\| \leq c_k \left[\frac{b - a}{b + a} + \delta \right]^n \ , \ n = 0, 1, 2, \ldots \tag{2.43}$$

where c_k is a bounded constant.

Proof. By Proposition 2.11, there exists $N_1 < \infty$ such that for all $k \geq N_1 + 1$, $\mathcal{B}(x_k) = \mathcal{B}(\hat{x})$, *i.e.*, for all $k \geq N_1 + 1$, $x^i_k = 0$ for $i \in \mathcal{B}(\hat{x})$ and for all $i \notin \mathcal{B}(\hat{x})$, d^i_k is determined by equations (2.42a,b). Furthermore, it can be shown (see [2], equations 2.64, 2.65 and 2.66) that with the choice for σ_2 given in the Corollary statement the tests in (2.42b) will not fail for $k \geq N_1 + 1$. Thus, the search direction $\tilde{d}_k = (d^1_k \cdots d^r_k)$, $k \geq N_1 + 1$, is determined by the unconstrained, partial conjugate gradient method, with restarts every $m + 1$ iterations, applied to the unconstrained subspace $\{ x \in \mathbb{R}^n \mid x^i = 0 \ , \ i \in \mathcal{B}(\hat{x}) \}$. It follows from Corollary 5.1 in [90] that there exists a finite $N \geq N_1 + 1$ such that (2.43) holds. □

Remark 2.13. If exact line searches were not used in the implementation of the conjugate gradient method given in Corollary 2.12, the tests in (2.42b) would provide an automatic restarting mechanism. It is possible to avoid restarts altogether (after a finite number of iterations) even without using exact line minimization if the search direction d^i_k, $i \in I_k$, in (2.42a,b) are constructed using the Fletcher-Reeves conjugate gradient method ($\mu_k = \mu^{FR}_k \doteq \|g_k\|^2_{I_k} / \|g_{k-1}\|^2_{I_k}$), or using the Polak-Ribière modified so that $\mu_k = \mu^{PR}_k$ if $|\mu^{PR}_k| \leq |\mu^{FR}_k|$ and $\mu_k = \mu^{FR}_k$ if $|\mu^{PR}_k| > |\mu^{FR}_k|$.

To achieve this the Modified Step-size Procedure must be used with λ'_k satisfying the strong Wolfe conditions given by (2.41c,d). Proposition 2.11 shows that Algorithm Model PD using the Modified Step-size Procedure reverts to an unconstrained minimization after a finite number of iterations and, for the remaining iterations, $x^i_{k+1} = x^i_k + \lambda'_k d^i_k$, $i \notin \mathcal{B}(\hat{x})$. For these iterations, the results in [91] show that the tests in (2.5b,c) will always be satisfied if $\sigma_1 = \frac{1-2\alpha_2}{1-\alpha_2}$ and $\sigma_2 = \frac{1}{1-\alpha_2}$. $\square$

Extension to upper and lower bounds. Algorithm Model PD can easily be extended to deal with upper and lower bounds of the form $b^i_l \leq x^i \leq b^i_u$, $i = 1, \ldots, n$. Merely replace the projection operator $[\cdot]_+$ with the projection operator $[\cdot]_\#$, defined for $z \in \mathbb{R}^n$ and $i = 1, \ldots, n$, by

$$[z]^i_\# \doteq \begin{cases} b^i_l & \text{if } z^i \leq b^i_l \,, \\ z^i & \text{if } b^i_l < z^i < b^i_u \,, \\ b^i_u & \text{if } z^i \geq b^i_u \,, \end{cases} \tag{2.44}$$

define the feasible set as $\mathcal{F} \doteq \{ \, x \in \mathbb{R}^n \mid b^i_l \leq x_i \leq b^i_u \,,\ i = 1, \ldots, n \, \}$ and set $A_k = A(x_k)$ where, for $x \in \mathcal{F}$,

$$A(x_k) \doteq \{ \, i \;\big|\; b^i_l \leq x^i_k \leq b^i_l + \varepsilon(x_k) \text{ and } g^i_k > 0 \,, \text{ or } b^i_u - \varepsilon(x_k) \leq x^i_k \leq b^i_u \text{ and } g^i_k < 0 \, \} \tag{2.45}$$

The set I_k is defined, as before, as the complement of A_k in $\{ \, 1, 2, \ldots, n \, \}$.

3.3 COMPUTATIONAL RESULTS

One source of large-scale optimization problems is discretizations of optimal control problems. An optimal control problem can be discretized by replacing the differential equations describing the system dynamics with a system of difference equations that describes some integration algorithm applied to the differential equations, and by replacing the infinite dimensional function space of controls with a finite dimensional subspace of parameterized controls. The result is a standard nonlinear programming problem whose decision variables are the control parameters. The number of decision variables in the nonlinear program is equal to the dimension of the approximating control subspace. For optimal control problems with control bounds, the nonlinear program is in the form of problem **P** and is suitable for solution by Algorithm Model PD.

We used Algorithm Model PD to solve a discretization of the following optimal control problem:

OCP $\underset{u \in L_2[0,2.5]}{\text{minimize}} \left[\; Cx_1^2(2.5) + \int_0^{2.5} x_1^2(t) + u^2(t)\; dt \; \right]$

subject to

$$\dot{x}_1(t) = x_2(t) \quad ; \quad x_1(0) = -5 \; ,$$

$$\dot{x}_2(t) = -x_1(t) + [1.4 - 0.14x_2^2(t)]x_2(t) + 4u(t) \quad ; \quad x_2(0) = -5 \; .$$

$$u(t) \geq -4|t - 1.5| \; , \; \forall t \in [0, 2.5] \; ,$$

with $C \geq 0$ a parameter.

The discretization was carried out using a second order Runge-Kutta method (the explicit trapezoidal rule) with $u(\cdot)$ restricted to the subspace of continuous, piecewise linear functions. Specifically, the decision variables for the discretized problem are $u = (u^0, u^1, \ldots, u^{n-1}) \in \mathbb{R}^n$ where $u^i = u(t_i)$ are the values of $u(\cdot)$ at the breakpoints $t_i = i(2.5/1000)$, $i = 0, \ldots, 1000$. Thus $n = 1001$. The use of Runge-Kutta integration for discretization of optimal control problems is described in detail in Chapter 2. We utilized the natural coordinate transformation, given by (2.7.19c), associated with this discretization in order to prevent unnecessary ill-conditioning. For this case, the transformation is given by $\tilde{u} = \mathbf{M}_N^{\frac{1}{2}} u$ where, due to continuity imposed at the breakpoints t_k, $\mathbf{M}_N$ is an $n \times n$ diagonal matrix with diagonal$(M) = [\frac{1}{2} \; 1 \; 1 \cdots 1 \; 1 \; \frac{1}{2}] / N$[†].

In addition to the coordinate transformation required by the theory of consistent approximations, we also pre-scale the problem by multiplying $f(\cdot)$ by the factor γ, where γ is defined as follows:

$$S = (1 + \|u_0\|_\infty) / (100 \|g_0\|_\infty) \tag{3.1a}$$

$$\delta u = [u_0 - S g_0]_\# \; , \tag{3.1b}$$

$$\gamma = \frac{1}{2} \left| \frac{\langle \delta u, \delta u \rangle_{I_0}}{f(u_0 + \delta u) - f(u_0) - \langle g_0, \delta u \rangle_{I_0}} \right| \; , \tag{3.1c}$$

with $[\cdot]_\#$ as given by (2.44). This pre-scaling makes it likely that a step-size of one is accepted in the first iteration of the algorithm (γ is the distance along the projected steepest descent direction, δu, to the minimum of a quadratic fit to $f(\cdot)$) and it acts as a normalization on the problem so that the tests in (2.5a,b,c) and the numerical termination criterion are less scale sensitive.

[†] Other coordinate transformations are also sometimes useful. For example, a coordinate transformation for use with the conjugate gradient method applied to optimal control problems with a special structure is discussed, and shown to be extremely effective, in [92]. Another possibility is to use the inverse of the diagonal of the Hessian. This matrix can be efficiently computed using a recursive algorithm similar to the one described in [93]. However, our experience indicates that this is not effective for optimal control problems discretized as described above.

For purposes of comparison, we solved the discretized optimal control problem with a projected steepest descent algorithm, a projected conjugate gradient method and a projected version of the limited memory quasi-Newton algorithm (L-BFGS) presented in [86,94].

The projected steepest descent algorithm uses the search directions $d_k = -g_k$.

For the projected conjugate gradient method we used the search directions given in equations (2.42a,b,c) with $m_k^i = 1$ for all $i \in A_k$, $\sigma_1 = 0.2$ and $\sigma_2 = 10$. After computing a step-size λ_k that satisfies the Armijo-like rule in (2.6a), we construct a quadratic approximation $q(\lambda)$ to $f(u_k(\lambda, d_k))$ such that $q(0) = f(u_k)$, $q'(0) = g_k$ and $q(\lambda_k) = f(u_k(\lambda_k, d_k))$ and set λ'_k equal to the minimizer of this quadratic. We used this λ'_k in the Modified Step-size Procedure with $\sigma_3 = 1$ and $\sigma_4 = \beta^{-M}$ (the same upper bound as for λ_k). This procedure requires one extra function evaluation per iteration.

The L-BFGS algorithm computes an approximation G_k to the inverse of the Hessian based on a limited number of applications of the BFGS quasi-Newton update formula. At each iteration, the algorithm uses vectors $s_k \doteq u_k - u_{k-1}$ and $y_k \doteq g_k - g_{k-1}$ stored over a fixed number of previous iterations. The procedure is as follows: Let $\hat{m} = \min\{\, k, m-1 \,\}$. Then, at iteration k, G_k is computed from $\hat{m}+1$ BFGS updates applied to a starting estimate G_k^0 of the inverse Hessian (which can be different at each iteration) according to

$$\begin{aligned} G_{k+1} = {} & (V_k^T \cdots V_{k-\hat{m}}^T)\, G_k^0\, (V_{k-\hat{m}} \cdots V_k) \\ & + \rho_{k-\hat{m}} (V_k^T \cdots V_{k-\hat{m}+1}^T)\, s_{k-\hat{m}}\, s_{k-\hat{m}}^T (V_{k-\hat{m}+1} \cdots V_k) \\ & + \rho_{k-\hat{m}+1} (V_k^T \cdots V_{k-\hat{m}+2}^T)\, s_{k-\hat{m}+1}\, s_{k-\hat{m}+1}^T (V_{k-\hat{m}+2} \cdots V_k) \\ & \;\vdots \\ & + \rho_k s_k s_k^T \,, \end{aligned} \tag{3.2}$$

where $\rho_k = 1/(y_k^T s_k)$, $V_k = I - \rho_k y_k s_k^T$. Here we use I (without any subscript) to denote the $n \times n$ identity matrix. As in [94], we let $G_k^0 = I$ in the first iteration and during restarts, and on other iterations we let $G_k^0 = \gamma_k I$ where $\gamma_k = \langle y_k, s_k \rangle_{I_k} / \|y_k\|_{I_k}^2$ is a self-scaling term demonstrated in [95] to markedly improve the performance of quasi-Newton algorithms. An efficient, recursive procedure for computing $d_k = -G_k g_k$ without explicitly forming any matrices is given in [86].

The L-BFGS algorithm has proven to be quite effective [96]. We used this algorithm, with $m = 12$, to compute the search directions d_k^i, $i \in I_k$, in the unconstrained subspace. This is accomplished by saving the full vectors s_k and y_k but restricting the inner product calculations in the recursive algorithm of [86] to the current estimate, I_k, of the unconstrained subspace. In (2.5a), we chose $m_k^i = \gamma_k$ for all $i \in A_k$. We also added the following tests:

(i) If $\langle y_k, s_k \rangle_{I_k} < 0.001 \|g_k\|^2_{I_k}$, set $\gamma_k = 1$ and do not use current information y_k and s_k for Hessian updates.

(ii) Restart if $\langle d_k, g_k \rangle_{I_k} > -0.2\gamma_k \|g_k\|^2_{I_k}$,

(iii) Restart if $\|d_k\|^2_{I_k} > 1000\gamma_k^2 \|g_k\|^2_{I_k}$,

The first test ensures that the Hessian estimate is positive definite. With the tests *(ii)* and *(iii)*, the search direction d_k satisfies the conditions in (2.5a,b,c) for some $\sigma_1 \in (0, 1)$ and $\sigma_2 \in (1, \infty)$ if we restrict the magnitude of γ_k so that

$$0.2\gamma_k \geq \sigma_1 \tag{3.3a}$$

and

$$\sqrt{1000}\gamma_k \leq \sigma_2 \ . \tag{3.3b}$$

Therefore, with this restriction, algorithm PD with L-BFGS search directions is convergent. In our implementation, we have $\sigma_1 = 0.2 \times 10^{-3}$ and $\sigma_2 = \sqrt{1000} \times 10^3$.

The remaining data required by Algorithm Model PD were chosen as follows: $\alpha = 1/2$, $\beta = 3/5$, $M = 20$, $u_0^i = 0$ for $i = 1, \ldots, n$, and $\varepsilon = 0.2$ except, for the projected L-BFGS method, $\alpha = 1/3$ was used to ensure that a step-size of 1 could be accepted close to a solution. The termination test in Step 1 of Algorithm Model PD was considered satisfied when

(i) $\|g_k\|_{I_k} / |I_k| < \varepsilon_{\mathrm{mach}}^{2/3}(1 + |f(u_k)|)$,

(ii) $f(u_k) - f(u_{k-1}) < 10\varepsilon_{\mathrm{mach}}(1 + |f(u_k)|)$,

(iii) $\|u_k - u_{k-1}\|_\infty < \varepsilon_{\mathrm{mach}}^{1/2}(1 + \|u_k\|_\infty)$,

(iv) $x_k^i = 0$ for all $i \in A_k$,

where the machine precision is $\varepsilon_{\mathrm{mach}} \approx 2.2204e-16$. Note that this is a very demanding termination criterion.

With $C = 0$, there were 171 binding constraints at the solution and these were identified after 7 iterations for the projected conjugate and projected L-BFGS methods and after 12 iterations for the projected steepest descent method. With $C = 100$, there were 436 binding constraints at the solution and these were identified after 19, 28 and 153 iterations, respectively, for the three methods.

The number of iterations, function evaluations, gradient evaluations and cpu time† required to reach termination for problem **OCP** with $C = 0$ are given in Table 3.1. The same information

† Experiments were run on a 60MHz Sun SparcStation 20 with 96MB internal memory and 1MB external cache. The algorithms were implemented in Matlab's M-script language with the exception of the L-BFGS search direction routine which was written in C.

is provided in Table 3.2 for problem **OCP** with $C = 100$. It is clear that the projected conjugate gradient and the projected L-BFGS methods perform significantly better than the project steepest descent method.

Method	Function calls	Gradient calls	Iterations	CPU Time
Conjugate Gradient	70	20	19	4.2 sec.
L-BFGS	43	14	13	2.7 sec.
Steepest Descent	97	29	28	5.3 sec.

Table 3.1: Work done to solve problem **OCP** with $C = 0$.

Method	Function calls	Gradient calls	Iterations	CPU Time
Conjugate Gradient	249	38	37	10.9 sec.
L-BFGS	163	38	37	8.3 sec.
Steepest Descent	1788	355	354	81.0 sec.

Table 3.2: Work done to solve problem **OCP** with $C = 100$.

The optimal solution, $\hat{u}$, of the discretized problem with $C = 100$ is shown in Figure 3.1.

3.4 CONCLUDING REMARKS

We have presented an implementable projected descent algorithm model, Algorithm Model PD, and proved its convergence for any search directions satisfying the conditions in equations (2.5a,b,c). This algorithm model solves a common class of problems involving simple bounds on the decision variables. It is particularly useful if the number of decision variable is large such as can occur in the discretization of optimal control problems with control bounds. Furthermore, many problems with simple bounds on the decision variables as well as some additional general constraints can be converted into the form of problem **P** using quadratic penalty functions or augmented Lagrangians. The Algorithm Model PD, when used in conjunction with a conjugate gradient method or the limited-memory BFGS method for determining the unconstrained portion of the search directions, has the advantage of requiring very little storage and work per iteration. Yet, the rate of convergence behavior can be expected to be the same as that of the unconstrained conjugate gradient or limited-memory BFGS methods after a finite number of iterations.

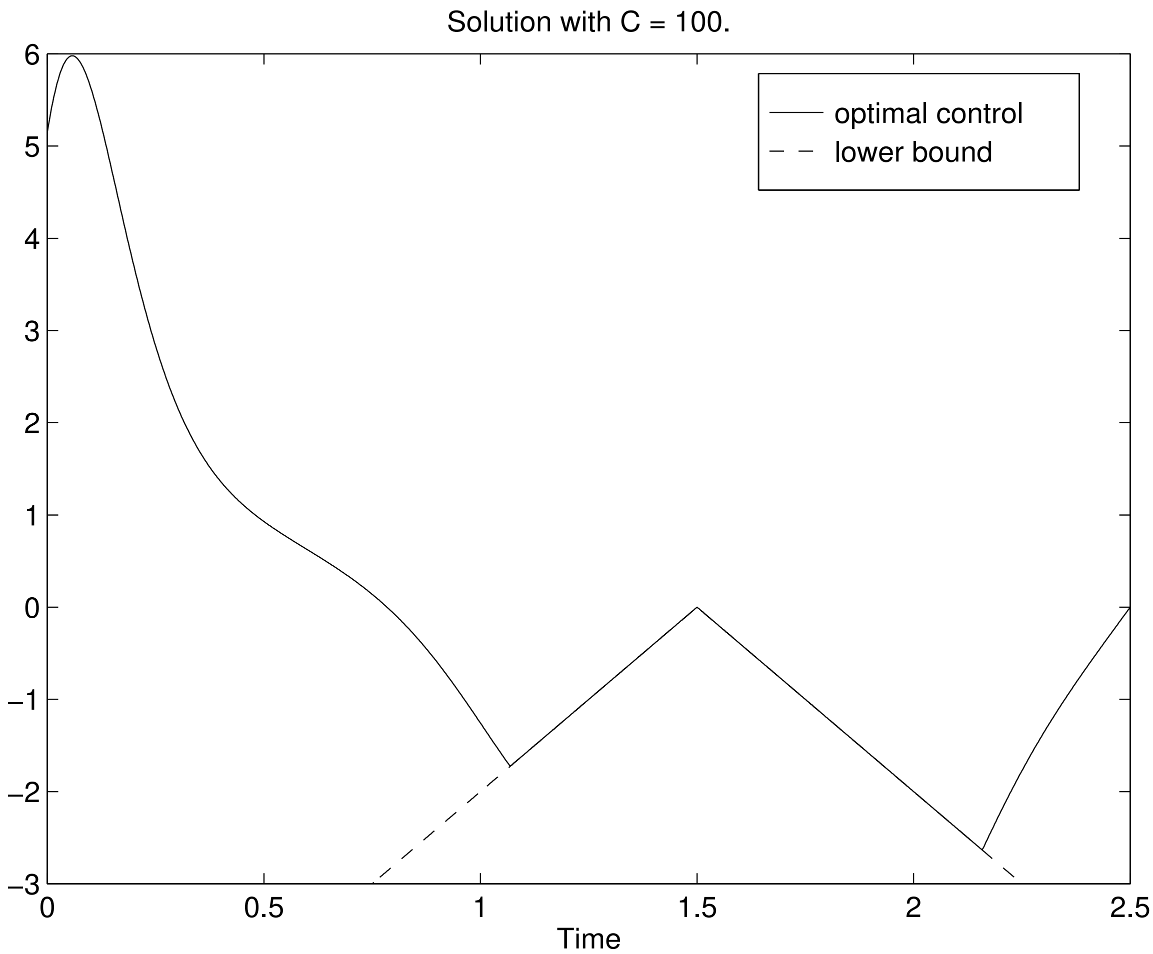


Fig. 3.1: Plot showing the optimal control for problem **OCP** with $C = 0$ ($n = 1000$).

Chapter 4

NUMERICAL ISSUES

4.1 INTRODUCTION

In Chapter 2, we provided a framework for the construction of approximating problems $\{ (\mathbf{P}_N, \theta_N) \}$ that are consistent approximations to an original problem $(\mathbf{P}, \theta)$. In Chapter 3, we presented an algorithm for solving a particular class of problems $\mathbf{P}_N$. In this chapter, we focus our attention on two practical issues that arise in the numerical implementation of an algorithm for solving a sequence of problems $\{ \mathbf{P}_N \}$ whose solutions $\{ \eta_N \}$ converge to a solution η^* of $\mathbf{P}$. These issues are, roughly speaking, *(i)* given an approximate solution ${\eta_{N_i}}^*$ to problem $\mathbf{P}_{N_i}$ at discretization level N_i and a new discretization level N_{i+1}, select a new integration mesh for problem $\mathbf{P}_{N_{i+1}}$ such that $\| {\eta_{N_{i+1}}}^* - \eta^* \|_{H_2}$ is as small as possible, and *(ii)* provide estimates of $\| {\eta_{N_i}}^* - \eta^* \|_{H_2}$.

These issues are important for practical computation because they allow for the discretization to be refined in a way that leads to more accurate solutions with less computation. Furthermore, because we cannot compute the entire sequence $\{ {\eta_N}^* \}$, it is desirable to know the error, $\| {\eta_{N_f}}^* - \eta^* \|_{H_2}$, of the finite-dimensional solution at some final discretization level N_f. A description of the optimal control problems used for the numerical examples in this chapter can be found in Appendix B. Throughout this chapter we presume that Assumption 2.3.1 holds.

Notation. In this chapter, we will be comparing solutions of approximating problems that are defined on different integration meshes. Thus we need to extend the notation of Chapter 2 to explicitly indicate the discretization level of the mesh. We will do this by adding the subscript N when necessary. We start by defining the *discretization mesh* for problem $\mathbf{P}_N$,

$$\mathbf{t}_N \doteq \{ t_{N,k} \}_{k=0}^{N} , \tag{1.1a}$$

where $t_{N,k}$ is the k-th mesh point at discretization level N. Since we will allow $\mathbf{t}_N$ to be a non-uniform mesh, we will denote the finite dimensional control subspaces by $L_{\mathbf{t}_N}$ instead of just L_N. Similar, the mapping from between controls $u \in L_{\mathbf{t}_N}$ and their samples $\bar{u} \in \bar{L}_{\mathbf{t}_N}$ will be denoted

$V_{\mathbf{A},\mathbf{t}_N} : L_{\mathbf{t}_N} \to \bar{L}_{\mathbf{t}_N}$. Recall from (2.4.5a,b) that elements $\bar{u} \in \bar{L}_{\mathbf{t}_N}$ are partitioned as follows:

$$\bar{u} = (\bar{u}_0, \bar{u}_1, \ldots, \bar{u}_{N-1}) \ , \quad \bar{u}_k = (\bar{u}_{k,1}, \ldots, \bar{u}_{k,r}) \ , \tag{1.1b}$$

with $\bar{u}_{k,j} \in \mathbb{R}^m$, $j = 1, \ldots, r$. Also recall that the subspaces $L_{\mathbf{t}_N}$ are defined such that the discretization mesh coincides with the breakpoints of the finite dimensional controls. Thus, we will also refer to $\mathbf{t}_N$ as the control *breakpoints*.

For spline controls we use $\mathbf{t}_N$ to refer either to the breakpoints or to the knot sequence constructed from these breakpoints, depending on the context. Thus, for example, we will re-write $L_N^{(\rho)}$ as $L_{\mathbf{t}_N}^{(\rho)}$ where, in this context, $\mathbf{t}_N$ is the general knot sequence for the ρ-th order spline subspace with breakpoints $\{ t_{N,k} \}_{k=0}^{N}$. The map from splines to their coefficients is $S_{\mathbf{t}_N,\rho} : L_{\mathbf{t}_N}^{(\rho)} \to \bar{L}_{\mathbf{t}_N}^{(\rho)}$. So, given $u \in L_{\mathbf{t}_N}^{(\rho)}$, $\alpha = S_{\mathbf{t}_N,\rho}(u)$ is the vector

$$\alpha = (\ \alpha_1\ ,\ \alpha_2\ ,\ldots,\ \alpha_{N+\rho-1}\ )\ , \tag{1.1c}$$

with $\alpha_k \in \mathbb{R}^m$, of coefficients for the spline u defined on the knot sequence $\mathbf{t}_N$. When used in linear algebra operations, elements of $\bar{L}_{\mathbf{t}_N}$ and $\bar{L}_{\mathbf{t}_N}^{(\rho)}$ will be treated, respectively, as $m \times Nr$ and $m \times (N + \rho - 1)$ matrices in keeping with our previous convention.

Non-uniform meshes. The discussion of approximating problems in Chapter 2 was based on a uniform discretization mesh $\{ t_{N,k} \}_{k \in \mathcal{N}}$ with $t_{N,k} = k/N$. In fact, the convergence results still hold for any sequence of meshes with the property that $\max_k t_{N,k+1} - t_{N,k} \to 0$ as $N \to \infty$. In practice, we also require $\min_k t_{N,k+1} - t_{N,k} > 0$ for all N. Both of these mesh characteristics are ensured if the following property holds for each N:

Definition 1.1 (Quasi-uniformity). Let $\mathbf{t}_N = \{ t_{N,k} \}_{k=0}^{N}$ be a mesh for problem $\mathbf{P}_N$ and let

$$\Delta_{N,k} \doteq t_{N,k+1} - t_{N,k} \ . \tag{1.2a}$$

Then $\mathbf{t}_N$ is said to be a *quasi-uniform* mesh if

$$\frac{\max_k \Delta_{N,k}}{\min_k \Delta_{N,k}} \leq \bar{\delta} \ , \tag{1.2b}$$

for some fixed constant $\bar{\delta} < \infty$ independent of N. We will refer to $\bar{\delta}$ as the *quasi-uniformity ratio*. □

This definition ensures that, for any sequence of quasi-uniform meshes,

$$\Delta_N \doteq \max_k \Delta_{N,k} \to 0 \text{ as } N \to \infty \ . \tag{1.2c}$$

4.2 INTEGRATION ORDER AND SPLINE ORDER SELECTION

The results of Chapter 2 show that Runge-Kutta integration methods satisfying the assumptions for Corollary 2.5.10 produce consistent approximations. Additionally, Lemma 2.4.10 states order of integration error results and Proposition 2.6.2 provides one result on the order of error of approximating problem solutions for unconstrained problems. We will now discuss the relationship of the integration and spline orders to the error, $\|\eta^* - {\eta_N}^*\|$, of the solutions of the approximating problems. For the remainder of this discussion, we will refer to this quantity simply as the *solution error*. A solution is said to have high accuracy if the solution error is small relative to the discretization level.

The desire to obtain high solution accuracy is, of course, the motivation for studying higher-order Runge-Kutta methods in the construction of the approximating problems $\mathbf{P}_N$. Unfortunately, very little is known, in general, about the relationship between integration error and the error of the solutions of $\mathbf{P}_N$. The primary obstacle to such an understanding is the fact that little is known about the smoothness of optimal controls (*i.e.*, the regularity of solutions) for constrained optimal control problems. In this section, we will collect some positive and some negative results concerning the smoothness of optimal controls and the solution errors of approximating problems. These results provide a basis for selecting the order of the Runge-Kutta method and the order of the control representation in the construction of $\mathbf{P}_N$.

In what follows, we will consider only the four, full-order RK methods whose Butcher arrays are displayed below. We stop at fourth order simply because there is no fifth order RK method with less than six stages.

$$\mathbf{A}_1 = \begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \qquad \mathbf{A}_2 = \begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

$$\mathbf{A}_3 = \begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \qquad \mathbf{A}_4 = \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

The first of these RK methods, $\mathbf{A}_1$, is obviously Euler's method and the second, $\mathbf{A}_2$, is an explicit trapezoidal rule (also known as the Euler-Cauchy or improved Euler method). The third order method, $\mathbf{A}_3$, is Kutta's formula and the fourth order method, $\mathbf{A}_4$, is the classical Runge-Kutta method. These four methods are selected because, from what is known about RK discretization of optimal control problems (both theoretically and experimentally), these provide the best results for the given number of stages. Needless to say, all of these methods satisfy the Assumptions 4.1, 4.3 and 4.6 required for consistent approximations. These methods also have a symmetry property† that is used in [42,Thm. 3.1] to derive some of the error bounds we will now discuss. For methods $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$, there are no repeated values in the c vector of the Butcher array. Thus, for these methods, $r = s$ where r is the number of control samples per integration interval and s is the number of stages in the RK methods. For method A_4 however, $c_2 = c_3 = \frac{1}{2}$, and therefore $r = 3$ and $s = 4$.

4.2.1 Solution error for unconstrained problems

The best error results are known for unconstrained problems with fixed initial conditions of the form

$$\mathbf{P} \qquad \min_{u \in L_{\infty,2}[0,1]} \{ f(u) \mid \dot{x} = h(x,u) \,,\; t \in [0,1] \} \,,$$

where $f(u) = \tilde{f}(x^u(1))$. We will assume that $\tilde{f}(\cdot)$ and $h(\cdot,\cdot)$ are sufficiently smooth so that all required derivatives exist. The approximate problems are defined, for a given discretization level N, by replacing $f(u)$ with $f_N(u_N) \doteq \tilde{f}(x^{u_N}_{N,N})$ where $u_N \in L_N$ and $x^{u_N}_{N,N}$ is the computed value of $x^{u_N}(t_N)$ obtained by Runge-Kutta integration on the mesh $\mathbf{t}_N$. The known error results depend on the assumption that the differential equations can be integrated with full-order accuracy at a solution u^*. That is,

$$\| x^{u^*}(t_k) - \bar{x}^{u^*}_{N,k} \| = O(\Delta_N^s) \,,\; \forall\, k \in \{ 0, \ldots, N \} \,, \tag{2.1}$$

where s is the order of the RK method. Essentially, this is the same as a regularity condition on u^*, namely $u^* \in C^{(s)}[0,1]$. The following table, based on [42] and Proposition 2.6.2, provides the order of $\max_{k,j} \| \bar{u}_N{}^*{}_{k,j} - u^*(\tau_{k,j}) \|$, where $\bar{u}_N{}^* = V_{\mathbf{A},\mathbf{t}_N}(u_N{}^*)$, relative to $\Delta_N \doteq \max_k \Delta_{N,k}$. The last column is only experimentally supported conjecture.

† Each method is equivalent to the transpose of itself run backwards in time [67].

RK Method	$\max_{k,j} \|\bar{u}_N{}^*{}_{k,j} - u^*(\tau_{k,j})\|$	$\max_{k} \|\bar{x}^*{}_k - x^*(t_k)\|$
$\mathbf{A}_1$	$O(\Delta_N^1)$	$O(\Delta_N^1)$
$\mathbf{A}_2$	$O(\Delta_N^2)$	$O(\Delta_N^2)$
$\mathbf{A}_3$	$O(\Delta_N^2)$	$O(\Delta_N^3)$
$\mathbf{A}_4$	$O(\Delta_N^3)$	$O(\Delta_N^4)$

Table 1.1: Order of solution errors for the approximating problems.

Note that for method $\mathbf{A}_3$, the order of the control solution error, $\max_{k,j} \|u_N{}^*{}_{k,j} - u^*(\tau_{k,j})\|$, is only $O(\Delta_N^2)$ even though $r = 3$. Because of this, it makes sense when using method $\mathbf{A}_3$ to depart from our convention of using r-th order polynomials for methods that use r control sample per integration interval and instead use second order polynomials. Generally, it may be advantageous, especially for constrained problems, to use splines of order lower than r. Thus, we will henceforth make it a convention that r represents the number of distinct controls samples required by the RK method during each integration time step while $\rho \le r$ represents the order of the control representation. General formulas for the gradient computations for controls $u \in L_N^{(\rho)}$ with ρ not necessarily equal to r are given in the RIOTS user's manual.

If representation **R1** (piecewise polynomials of order ρ, where ρ is the exponent of Δ_N given in the middle column of Table 1.1) is used for the controls u_N, then $\|u_N{}^* - u^*\|_{L_\infty} = O(\Delta_N^\rho)$ since u^* is assumed to be smooth. We will show this for the harder case of spline approximations below.

The bounds in Table 1.1 are derived assuming that their are no constraints between the control samples $\bar{u}_{k,j}$. Such is not the case for splines because the continuity and smoothness conditions on splines implicitly enforce constraints between the control samples. For example, if $u_N \in L_N^{(2)}$ (linear splines), then for each $k \in \{0, .., N-2\}$, $\bar{u}_{k,2} = \bar{u}_{k+1,1}$. That is, continuity is imposed across breakpoints. Proposition 2.2 below indicates that, nonetheless, the same error results do hold for finite-dimensional controls represented by splines. Before stating this proposition, we note the following result for spline approximations.

Theorem 2.1 [63,Thm. XII.1 (p. 170)]. Given a spline order ρ, there exists a constant $c_\rho < \infty$ such that for all knot sequences $\mathbf{t}_N$ with $t_{-r+1} = \cdots = t_0 = a < t_1 \cdots < t_{N-1} < b = t_N = \cdots = t_{N+\rho-1}$ and for all $u \in C^{(\rho)}[a,b]$,

$$\min_{v \in L_{\mathbf{t}_N}^{(\rho)}} \|u - v\|_\infty \le c_\rho \Delta_N^\rho \|u^{(\rho)}\|_\infty \,, \tag{2.2}$$

where $\Delta_N \doteq \max_k \Delta_{N,k}$ and $u^{(\rho)}$ is the ρ-th derivative of $u(t)$ with respect to t. □

We will now provided a bound for the solution error $\|u_N^* - u^*\|_{L_\infty}$ for approximating problems defined with spline controls. For the statement and proof of Proposition 2.2, we will use the following notation for derivatives of the approximating functions with respect to spline coefficients and with respect to control samples. First, let $u \in L_{\mathbf{t}_N}^{(\rho)}$ be a ρ-th order spline with coefficients $\alpha = S_{\mathbf{t}_N,\rho}(u)$. We define $\bar{f}_N(\alpha) \doteq f_N(S_{\mathbf{t}_N,\rho}^{-1}(\alpha))$. Then, we use the notation $d_\alpha \bar{f}_N : \mathbb{R}^{m\times(N+\rho-1)} \to \mathbb{R}^{m\times(N+\rho-1)}$ with

$$d_\alpha \bar{f}_N(\alpha) \doteq \left(\frac{d}{d\alpha_1}\,\bar{f}_N(\alpha) \cdots \frac{d}{d\alpha_{N+\rho-1}}\,\bar{f}_N(\alpha) \right) \tag{2.3a}$$

to denote the derivative of $\bar{f}_N(\alpha)$ with respect to the components of α. Since $L_{\mathbf{t}_N}^{(\rho)} \subset L_{\mathbf{t}_N}$ (the space of piecewise polynomials), we can also define $\bar{f}_N(\bar{u}) = f_N(V_{\mathbf{A},t_N}^{-1}(\bar{u}))$ where $\bar{u} = V_{\mathbf{A},t_N}(u)$. Whether the argument of $\bar{f}_N(\cdot)$ is spline coefficients or control samples is always clear from context. We will use the notation $d_{\bar{u}}\bar{f}_N : \mathbb{R}^{m\times Nr} \to \mathbb{R}^{m\times Nr}$ with

$$d_{\bar{u}}\bar{f}_N(\bar{u}) \doteq \left(\frac{d}{d\bar{u}_{0,1}}\,\bar{f}_N(\bar{u}) \cdots \frac{d}{d\bar{u}_{0,r}}\,\bar{f}_N(\bar{u}) \quad \cdots \quad \frac{d}{d\bar{u}_{N-1,1}}\,\bar{f}_N(\bar{u}) \cdots \frac{d}{d\bar{u}_{N-1,r}}\,\bar{f}_N(\bar{u}) \right), \tag{2.3b}$$

to denote the derivative of $\bar{f}_N(\bar{u})$ with respect to the control samples. As usual, r is the number of control samples per interval. Note that (2.3a,b) define gradients with respect to the Euclidean norm on the coefficient spaces; these derivatives are not the gradients with respect to the norms we have defined on $\bar{L}_{\mathbf{t}_N}^{(\rho)}$ and $\bar{L}_{\mathbf{t}_N}$.

Theorem 2.2 (Error of spline solutions). Let $u^* \in C^{(\rho)}[a,b]$ be a local solution of $\mathbf{P}$ where ρ is the exponent of $\Delta_N \doteq \max_k \Delta_{N,k}$ in Table 1.1 corresponding to the RK method under consideration and assume that (2.1) holds. Suppose that $\{\, u_N^* \,\}$ is a sequence such that $u_N^* \in L_{\mathbf{t}_N}^{(\rho)}$ with spline coefficients $\alpha_N^* = S_{\mathbf{t}_N,\rho}(u_N^*)$, u_N^* is a local solution of $\mathbf{P}_N$ and $u_N^* \to u^*$. Also, assume that for each N, $d_\alpha^2\bar{f}_N(\cdot)$ exists and is continuous, $d_\alpha^2\bar{f}_N(\alpha_N^*) > 0$ and $\Delta_N\|d_\alpha^2\bar{f}_N(\cdot)^{-1}\|$ is uniformly bounded with respect to N. Then $\|u^* - u_N^*\|_\infty \sim O(\Delta_N^\rho)$.

Proof. For each N, let $\hat{u}_N = \arg\min_{u \in L_{\mathbf{t}_N}^{(r)}} \|u^* - u\|_\infty$ and let $\hat{\alpha}_N = S_{\mathbf{t}_N,\rho}(\hat{u}_N)$. We will first obtain a bound for $\|\alpha^*_N - \hat{\alpha}_N\|$ in terms of $\|d_{\bar{u}}\bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(\hat{u}_N))\|$ which we will then show is of order $O(\Delta_N^\rho)$. Finally, we will use this bound to show that $\|u^* - u_N^*\|_\infty$ is of order $O(\Delta_N^\rho)$.

Expanding the derivative of $\bar{f}_N(\cdot)$ to first order, we have

$$d_\alpha\bar{f}_N(\hat{\alpha}_N) = d_\alpha\bar{f}_N(\alpha_N^*) + \int_0^1 (\hat{\alpha}_N - \alpha_N^*) d_\alpha^2\bar{f}_N(\alpha_N^* + s(\hat{\alpha}_N - \alpha_N^*))\, ds$$

$$= (\hat{\alpha}_N - \alpha_N^*) \int_0^1 d_\alpha^2 \bar{f}_N(\alpha_N^* + s(\hat{\alpha}_N - \alpha_N^*))ds \; . \tag{2.4a}$$

since $d_\alpha \bar{f}_N(\alpha_N^*) = 0$. Next, $u_N^* \to u^*$ and, from Theorem 2.1, $\hat{u}_N \to u^*$ as $N \to \infty$, we have that $\hat{u}_N \to u^*_N$ as $N \to \infty$. Now, since $\tilde{u}_N \doteq \hat{u}_N - u^*_N$ is itself a spline and $\tilde{u}_N \to 0$, it follows from Corollary XI.2 [63,p. 156] that

$$\hat{\alpha}_N \to \alpha^*_N \;\text{ as }\; N \to \infty \; . \tag{2.4b}$$

Using the hypotheses that $d_\alpha^2 \bar{f}_N(\cdot)$ is continuous and $d_\alpha^2 \bar{f}_N(\alpha_N^*) > 0$, we see from (2.4a) and (2.4b) that there exists an $N^* < \infty$ such that

$$\hat{\alpha}_N - \alpha_N^* = \left[\int_0^1 d_\alpha^2 \bar{f}_N(\alpha_N^* + s(\hat{\alpha}_N - \alpha_N^*))ds \right]^{-1} d_\alpha \bar{f}_N(\hat{\alpha}_N) \; , \; \forall N \geq N^* \; . \tag{2.5a}$$

Thus, since $\Delta_N \| d_\alpha^2 \bar{f}_N(\cdot)^{-1} \|$ is uniformly bounded with respect to N and $\hat{\alpha}_N \to \alpha^*$, there exists a constant $K < \infty$ such that

$$\Delta_N \| \alpha_N^* - \hat{\alpha}_N \|_\infty \leq K \| d_\alpha \bar{f}_N(\hat{\alpha}_N) \|_\infty = K \| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(\hat{u}_N)) \Phi_{\mathbf{A},\mathbf{t}_N} \|_\infty$$

$$\leq K \| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(\hat{u}_N)) \|_\infty \| \Phi_{\mathbf{A},\mathbf{t}_N} \|_\infty \; , \tag{2.5b}$$

where we have used (2.7.12c) for the equality. The $Nr \times (N + \rho - 1)$ matrix $\Phi_{\mathbf{A},\mathbf{t}_N}$ was defined by (2.7.11a): the k-th column of $\Phi_{\mathbf{A},\mathbf{t}_N}$ contains the value of the k-th B-spline evaluated at the Nr control sample times. Now, there exists a $\kappa_1 < \infty$ such that

ne 2

$$\| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(\hat{u}_N)) \|_\infty \leq \| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(\hat{u}_N)) - d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(u^*)) \|_\infty + \| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(u^*)) \|_\infty$$

$$\leq \kappa_1 \Delta_N \| V_{\mathbf{A},\mathbf{t}_N}(\hat{u}_N - u^*) \|_\infty + \| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(u^*)) \|_\infty$$

$$\leq \kappa_1 \Delta_N \| \hat{u}_N - u^* \|_\infty + \| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(u^*)) \|_\infty$$

$$\leq \kappa_1 c_\rho \Delta_N^{\rho+1} + \| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(u^*)) \|_\infty \; . \tag{2.6}$$

The second of these inequalities was obtained from the Lipschitz continuity of $d_{\bar{u}} \bar{f}_N(\cdot)$. The constant c_ρ in the fourth inequality comes from Theorem 2.1. The quantity $\| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(u^*)) \|_\infty$ is used in the proofs of Theorem 3.1 in [42] and Proposition 2.6.2 which provide the bounds in the middle column of Table 1.1. These proofs show that

$$\| d_{\bar{u}} \bar{f}_N(V_{\mathbf{A},\mathbf{t}_N}(u^*)) \|_\infty = \kappa_2 \Delta_N^{\rho+1} \tag{2.7}$$

for some constant $\kappa_2 < \infty$ independent of N. Combining (2.5b), (2.6) and (2.7) with the fact that

$\|\Phi_{\mathbf{A},\mathbf{t}_N}\|_\infty = 1$ (due to the normalization of the B-splines), we see that

$$\|\alpha_N{}^* - \hat{\alpha}_N\|_\infty \sim O(\Delta_N^\rho) \ , \tag{2.8a}$$

Noting that $\alpha_N{}^* - \hat{\alpha}_N$ are the spline coefficients for $u_N{}^* - \hat{u}_N$, we can use Corollary XI.3 in [63,p. 156] to show that (2.8a) implies that

$$\|u_N{}^* - \hat{u}_N\|_\infty \sim O(\Delta_N^\rho) \ . \tag{2.8b}$$

Finally, using (2.8b) and Theorem 2.1, we obtain the desired result:

$$\|u^* - u_N{}^*\|_\infty \leq \|u^* - \hat{u}_N\|_\infty + \|\hat{u}_N - u_N{}^*\|_\infty \sim O(\Delta_N^\rho) \ . \tag{2.9}$$

□

Remark 2.3. The assumption in Theorem 2.2 that $\Delta_N \|d_\alpha^2 \bar{f}_N(\alpha)^{-1}\|$ is uniformly bounded in Δ_N and α is essentially the same as the assumption used by Hager for Theorem 3.1 in [42]. In that paper, Hager argues that the assumption is reasonable. □

There are optimal control problems whose approximating problems based on RK integration have solutions of higher accuracy than those listed in Table 1.1. However, this does not occur generically even for linear/quadratic problems. Therefore, based on the result in Table 1.1, the order ρ of the control representation should be chosen equal to the order of error given in the second column. Choosing ρ too small reduces the benefit of the higher-order integration accuracy (it does not eliminate this benefit, see below). Choosing ρ too large results in extra computational work and can actually reduce the accuracy of solutions due to an over-parameterization effect of the controls. In particular, for problems without control bounds and/or trajectory constraints, a piecewise linear control representation should be used with method $\mathbf{A}_3$ and either a piecewise linear or a piecewise quadratic representation should be used with method $\mathbf{A}_4$.

4.2.2 Constrained Problems

It is difficult to extend error results of the type given in Theorem 2.2 to constrained problems for many reasons. To explore some of the issues involved, it is helpful to consider two separate sources of error: the error due to numerical integration and the error due to the limited approximating capabilities of finite dimensional controls. For the sake of brevity we will only consider spline controls and we will not deal with free initial conditions. Define

$$u^* = \arg\min_{u \in \mathbf{U}} \{ \ f(u) \mid u \in \Omega \ \} \ , \tag{2.10a}$$

$$u_N{}^* = \arg\min_{u \in \mathbf{U}_N^{(\rho)}} \{ \ f_N(u) \mid u \in \Omega \ \} \ , \tag{2.10b}$$

$$\hat{u}_N = \arg\min_{u \in \mathbf{U}_N^{(\rho)}} \{ f(u) \mid u \in \Omega \} \ , \tag{2.10c}$$

where the set $\Omega \subset L^m_{\infty,2}[0,1]$ represents state constraints. Any control constraints in the optimal control problems are specified in the sets $\mathbf{U}$ and $\mathbf{U}_N^{(\rho)}$. As usual, u^* represents the solution of $\mathbf{P}$ and $u_N{}^*$ represents the solution of $\mathbf{P}_N$. Additionally, we have defined $\hat{u}_N{}^*$ as the solution of $\mathbf{P}$ but with the control space restricted to $\mathbf{U}_N^{(\rho)}$. By the triangle inequality, the solution error for the approximating problem $\mathbf{P}_N$ satisfies

$$\| u_N{}^* - u^* \|_2 \le \| u_N{}^* - \hat{u}_N \|_2 + \| \hat{u}_N - u^* \|_2 = E_{\text{int}} + E_{\text{rep}} \ , \tag{2.11a}$$

where

$$E_{\text{int}} \doteq \| u_N{}^* - \hat{u}_N \|_2 \tag{2.11b}$$

$$E_{\text{rep}} \doteq \| \hat{u}_N - u^* \|_2 \tag{2.11c}$$

are the errors due to, respectively, the numerical integration and the finite dimensional representation of controls for the approximating problems. Ideally, the integration order and the spline order should be chosen to make E_{int} and E_{rep} roughly equal. Otherwise some computational effort is being wasted. Below, we will present some evidence that second order splines are the best choice when using RK methods $\mathbf{A}_2$ and $\mathbf{A}_3$. The choice of the appropriate spline order for use with RK method $\mathbf{A}_4$ and the choice of which RK method to use is more involved and is also discussed.

Considerations for spline order selection. Since the error bounds for constrained problems are usually worse than the error bounds for unconstrained problem, one can conclude from the results in Table 1.1 and Theorem 2.2 that, at most, only first or second order splines should be used in conjunction with RK methods $\mathbf{A}_2$ and $\mathbf{A}_3$ (for Euler's method the only choice is first order splines), and that only first, second or third order splines should be used with RK method $\mathbf{A}_4$. The selection of an appropriate order for the spline control subspace depends on how that order affects the representation error E_{rep}. The size of E_{rep} for a given mesh $\mathbf{t}_N$ depends primarily on the smoothness of the optimal control $u^*(\cdot)$. For many constrained optimal control problems, the optimal control $u^*(\cdot)$ is not even continuous. The strongest regularity result available for constrained optimal control problems, [97], provides conditions for the optimal control of strictly convex problems to be Lipschitz continuous. Thus, E_{rep} may be only of order $O(\Delta_N)$.

For our method of discretization, there is clearly no reduction in the solution error to be gained by using a higher order control representation when the optimal control is discontinuous[†].

On the other hand, even though it generally impossible (without knowing the location of discontinuities in the optimal control) to improve upon $E_{rep} \sim O(\Delta)$, there is little to be lost by using second order, instead of first order, splines in conjunction with RK methods $\mathbf{A}_2$, $\mathbf{A}_3$ and $\mathbf{A}_4$ because there is very little extra work involved in computing a solution to $\mathbf{P}_N$. At the same time it seems that, at "non-asymptotic" values of the discretization level N, second order splines produce somewhat better results than piecewise constant controls even for problems with discontinuous optimal controls. Moreover, for convex problems it has been shown in [28,31,98] that the Ritz-Trefftz method gives an approximation error of order $O(\Delta_N^{3/2})$ assuming sufficient smoothness of the problem data. This is due to the fact that $\frac{d}{dt} u^*(t) \in L_2[0,1]$. Hence, in this case, there is a clear benefit to using second order splines over piecewise constant controls.

Based on these considerations, the best choice for the control representation when using method $\mathbf{A}_2$, $\mathbf{A}_3$ or $\mathbf{A}_4$ for problems whose solutions are likely to be non-smooth (such as problems with control constraints) is second order splines. If there are no control constraints and there is reason to believe that the optimal control is smooth then third order splines should be used with RK method $\mathbf{A}_4$ since third order convergence can be achieved. There is a caveat to this statement, however. If the spline coordinate transformation given by (2.7.18) is to be used during the solution of $\mathbf{P}_N$ and the work used by optimization algorithm that solves $\mathbf{P}_N$ requires less than N^3 operations (such as the projected descent method of Chapter 3 based on conjugate-gradient or the limited-memory BFGS search directions), then at a some discretization level N, the work required to compute and apply the coordinate transformation will exceed the amount of work required by the optimization algorithm. In this case, quadratic splines should not be used because it will be less expensive to solve the problem at a higher discretization level using second order splines.

Finally, it is very common for $E_{\text{int}} \gg E_{\text{rep}}$ at low to moderate discretization levels. If this is the case, it makes sense to use method A_4 with linear splines because, as seen from (2.11a), the reduction in E_{rep} afforded by the use of quadratic splines will be rendered meaningless by the size of E_{int}.

To demonstrate the effect of the spline order on E_{rep}, we have solved an unconstrained problem and a constrained, free final time problem using first, second, third and fourth order splines on a uniform mesh with $N = 20$ intervals. The two problems are the unconstrained Rayleigh problem and the Bang problem which are described in Appendix 2. The solution errors are given

† Although, as discussed in Chapter 6, it may be possible with a two-phase optimization approach to achieve better error results. During the second phase some grid points are allowed to move towards discontinuities in $u^*(\cdot)$. The efficacy of this approach relies on the fact that, under suitable conditions, $u^*(\cdot)$ is analytic on every interval in which the binding constraints do not change, see [28,31].

in Table 2.1 and the solutions are plotted in Figure 2.2a and Figure 2.2b. We computed the solution error as

$$\| u_N{}^* - u^* \|_2 = \left[\int_0^T (u_N{}^*(t) - u^*(t))^2 dt \right]^{1/2}, \tag{2.12}$$

where T is the (optimal) final time. For problem Rayleigh, $T = 2.5$; for problem Bang, $T^* = 30$. All of these solutions were computed using a variable step-size integration method so that the integration error E_{int} would be negligible compared to the error from the spline approximations. The results show that, for the unconstrained problem, the error decreases substantially as the spline order is increased. On the other hand, using higher-order splines for the constrained problem does not cause a significant reduction in the error. It should be noted, however, that the error is slightly less for second order splines than for first order splines.

Spline order	$\| u_N{}^* - u^* \|_2$ (Rayleigh, $T = 2.5$)	$\| u_N{}^* - u^* \|_2$ (Bang, $T^* = 30$)
1	0.3119	1.7146
2	0.1071	1.2357
3	0.0166	1.4585
4	0.0092	1.4166

Table 2.1: Solution error for an unconstrained and a constrained problem as a function of spline order.

These results suggests that, for problems with control constraints, either a first or a second order control representation should be used.

Piecewise polynomial controls versus splines. Recall the $L^{(\rho)}_{\mathbf{t}_N} \subset L^1_{\mathbf{t}_N}$. That is, the spline control spaces are subspaces of the piecewise polynomial control subspaces. Because the controls in $L^1_{\mathbf{t}_N}$ are allowed to be discontinuous at mesh points t_k, it would seem beneficial to use $L_{\mathbf{t}_N}$ rather than splines if we could place some of the mesh points at the locations of any discontinuities in the optimal control. There are two reasons this does not help. First, we don't not know the locations of such discontinuities *a priori*. Second, even if we were able to place the control breakpoints at the locations of discontinuities in the optimal control it is important to realize that locations of discontinuities in the optimal control for **p** will not be the same as the location of discontinuities for the solutions of $\mathbf{P}_N$ because of the error E_{int} due to the fixed step-size integration (but see previous footnote). It should also be noted that we could allow discontinuities in the splines by defining the spline subspaces with repeated interior knots (see [63]).

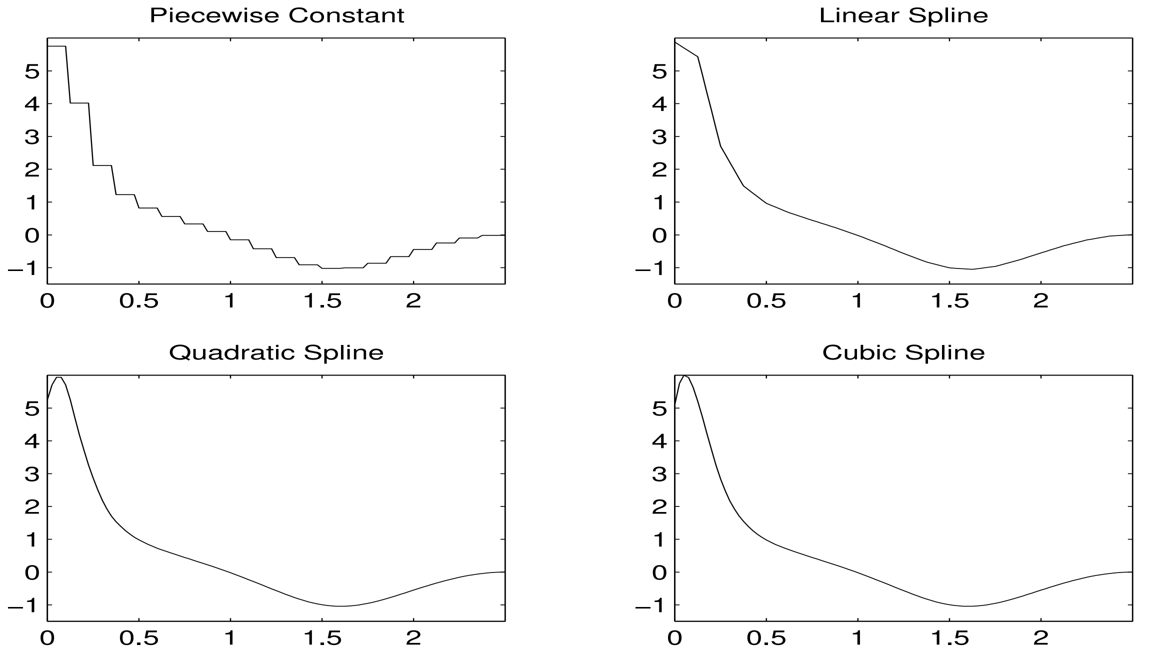


Fig. 2.2a: Effect of spline order on the solution $\hat{u}_N$ for the unconstrained problem Rayleigh. The plots show $\hat{u}_N(t)$ versus t.

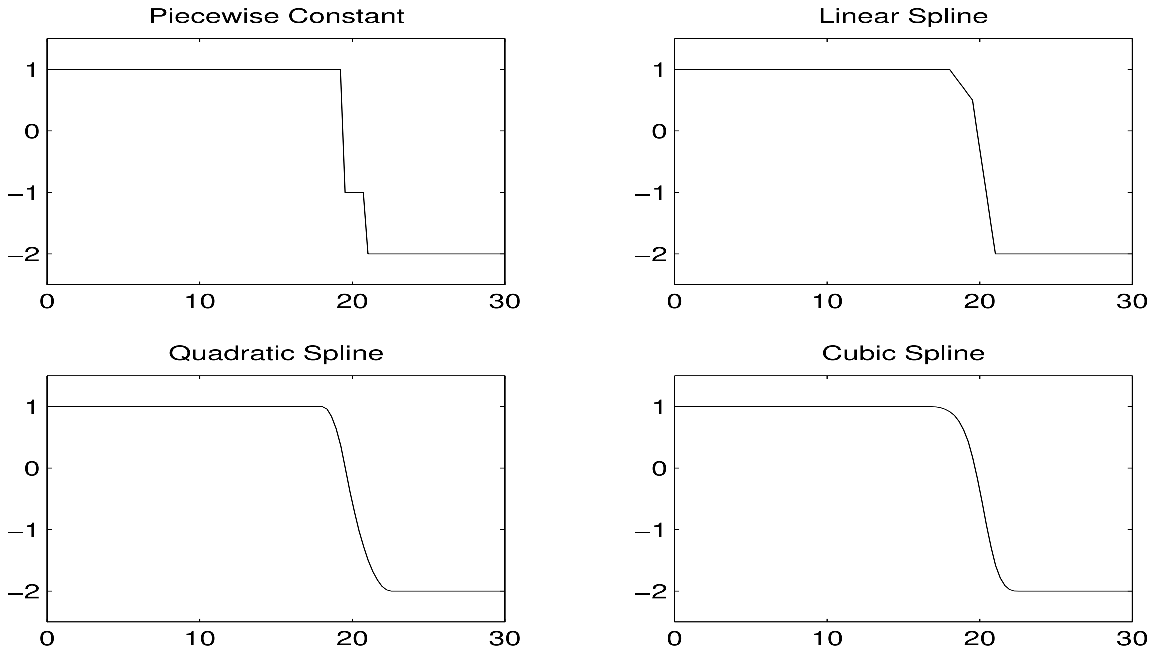


Fig. 2.2b: Effect of spline order on the solution $\hat{u}_N$ for the constrained problem Bang. The plots show $\hat{u}_N(t)$ versus t.

Considerations for integration order selection. The Runge-Kutta order should, ideally, be chosen to minimize the amount of work required to obtain a solution of specified accuracy. This choice depends on a combination of factors: the integration error which depends on the nonlinearity and stability of the differential equations $\dot{x} = h(x, u)$, the smoothness of solutions to the original optimal control problem, and the amount of work used by the optimization algorithm used to solve the approximating problems to perform each iteration. Generally speaking, however, none of these quantities is known in advance. Thus, we can only offer guidelines for the RK order selection. Further research is needed to provide a more systematic approach.

The amount of work, W_N, to solve a the discretized problem $\mathbf{P}_N$ is roughly

$$W_N \approx n_{\text{iter}}(W_{\text{int}} + W_{\text{opt}}) \,, \tag{2.13}$$

where n_{iter} is the number of iterations needed to achieve a certain optimality tolerance, W_{int} is the work required to integrate the system dynamics for a given η_N and to compute the gradients for each function, and W_{opt} is the amount of work (linear algebra) done by the optimization algorithm during each iteration (primarily in computing the search direction). Typically, W_{int} is linear in N (unless there are trajectory constraints) and linear in the integration order. The relationship between W_{opt} and N depends on the integration algorithm. Equation (2.13) is only approximate because the system dynamics usually have to be integrated more than once during line searches. But it does show that increasing the discretization level and/or the integration order (in order to decrease E_{int}) will increase W_N. At the same time, equation (2.11a) shows that just decreasing E_{int} does not necessarily imply that the overall solution error will decrease.

Without a more quantitative analysis, we consider RK method $\mathbf{A}_4$ to be the best choice, except in the two situations described below, because at low or moderate discretization levels, it is typically the case that $E_{\text{int}} \gg E_{\text{rep}}$. This is partly due to the fact that the errors in the control are, to some extent, "integrated out" by the system dynamics. Therefore, even if $E_{\text{rep}} \sim O(\Delta_N)$, the solution error will be smaller for low to moderate discretization levels if E_{int} is made smaller by using a higher order RK method. Also, for many unstable nonlinear systems, a certain amount of integration accuracy is required simply to be able to obtain a numerical solution to the differential equations. In this situation, the minimum required discretization level may be quite large for a low-order RK method. Thus, it is usually best to use method $\mathbf{A}_4$.

However, there are at least two situations in which it is may be better to use a lower order RK method. The first case is problems with reasonably behaved linear dynamics and control bounds. According to the preceding discussion, only first order second order splines should be used. Thus, $E_{\text{rep}} \sim O(\Delta_N^2)$ at best. On the other hand, because of the simple system dynamics, E_{int} is likely to be much smaller than E_{rep} if RK method $\mathbf{A}_4$ is used. Thus, more work will be done in the integrations while the overall error of the approximating solutions will be no greater than those produced using method $\mathbf{A}_2$ †.

The second case is problems that include trajectory constraints. When solving the discretized problems, the gradients for each point t_k in the trajectory constraint must be computed. Because trajectory constraints are evaluated at $N + 1$ such points, the amount of work to compute these gradients is proportional to sN^2 operations‡, where s is the number of stages in the RK method. This can amount to a great deal of work, especially if the gradients are computed using adjoint equations. The amount of work can, therefore, be substantially reduced by using a lower order RK method.

When solving problems with control and/or trajectory constraints at high discretization levels, it is probably disadvantageous to use anything but methods $\mathbf{A}_1$ or $\mathbf{A}_2$ since the overall solution error is, at best, $O(\Delta_N^2)$.

4.3 INTEGRATION ERROR AND MESH REDISTRIBUTION

From (2.11a), we see that $\|u^* - u_N^*\|_2 = E_{\text{int}} + E_{\text{rep}}$. It turns out that E_{int} and E_{rep} are, in fact, closely related. To see this, let $u : [0, 1] \to \mathbb{R}^m$ be continuous on each interval $[t_k, t_{k+1})$ and let $u_N \in L^1_{\mathbf{t}_N}$ be a piecewise polynomial function of order $\rho \geq 2$. Then, by Jackson's Theorem [63, p. 33], there exists a constant $K < \infty$ such that, on each interval $I_k \doteq [t_k, t_{k+1})$,

$$\|u - u_N\|_{I_k,\infty} \leq K \, \frac{\Delta_{N,k}}{\rho - 1} \, \omega_k(u_N \, ; \frac{\Delta_{N,k}}{2(\rho - 1)}) \ , \ \ k = 0, \ldots, N - 1 \ , \tag{3.1a}$$

where $\|u\|_{I_k,\infty} \doteq \max_{t \in I_k} \|u(t)\|$ and $\omega_k(u_N \, ; \cdot)$ is the modulus of continuity of $u_N(t)$ on the interval $t \in [t_k, t_{k+1}]$ defined by

$$\omega_k(u \, ; \delta) \doteq \max \{ \, \|u(t_1) - u(t_2)\| \mid t_1, t_2 \in [t_k, t_{k+1}] \ , \ \ |t_2 - t_1| \leq \delta \ \} \ . \tag{3.1b}$$

If $u(t)$ is continuously differentiable on each interval $(t_k, t_{k+1}$, then $\omega_k(u \, ; \delta) \leq \|\dot{u}\|_{I_k,\infty}\delta$. In particular, the approximation error $\|u - u_N\|_{I_k,\infty}$ is largest on intervals where $\|\dot{u}\|_{I_k,\infty}$ is largest. Jackson's theorem also applies to approximation of smooth functions in which case it is the norm of the higher derivatives of u that determine the approximation error.

---

† A secondary problem in this regard is that the integration accuracy of method $\mathbf{A}_4$ can make it difficult to refine the mesh based on the procedures discussed in the next section.

‡ In fact, the amount of work is proportional $N(N + 1)/2$ operations since the gradient of the trajectory constraint at time t_k is zero for $t > t_k$. Even if an ε-active set method (such as in [99]) is used to bypass unneeded gradient computations, the work is still proportional to N^2 operations.

At the same time, we have that the local integration error produced by one step of an s-th order RK method (see proof of Lemma 4.10 in Appendix A) is

$$\bar{x}_{N,k+1} = x_{N,k}(t_{k+1}) + e_{N,k}\Delta_{N,k}^{s+1} + O(\Delta_{N,k}^{s+2}) \,, \tag{3.2}$$

where $x_{N,k}(\cdot)$ is the solution of $\dot{x} = h(x,u)$ with $x(t_k) = \bar{x}_{N,k}$. The quantity $e_{N,k}\Delta_{N,k}^{s+1}$ is called the *principal local truncation error* (PLTE) and $\bar{x}_{N,k+1} - x_{N,k}(t_{k+1})$ is the *local truncation error* (LTE). We will refer to $e_{N,k}$ as the coefficient of the PLTE. It is possible to produce an expression for $e_{N,k}$ by comparing the Taylor expansion of $\bar{x}_{N,k+1}$ with the Taylor expansion for $x_{N,k}(t_{k+1})$, both around $\bar{x}_{N,k}$. However, this expression, which depends on the RK method, is quite complicated. As an example, $e_{N,k}$ for method $\mathbf{A}_2$ applied to a scalar system with u differentiable between mesh points is

$$e_{N,k} = \frac{1}{2}\left[\frac{1}{3}(h_{xx}h^2 + h_{ux}h\dot{u} + h_{ux}h\dot{u} + h_{uu}\dot{u}^2 + h_u\ddot{u}) - \frac{1}{6}(h_x h + h_u\dot{u})\right]. \tag{3.3}$$

The important aspect of the local truncation error is that $e_{N,k}$ depends on time derivatives of u. Therefore, regions where u_N provides a poor approximation to u according to (3.1a) are likely to coincide with regions where the LTE for the integration is large†. Thus, it is desirable to place many mesh breakpoints in time intervals where there is a relatively high level of local integration error, and fewer mesh points elsewhere. This leads to higher integration accuracy for a given discretization level and also a better sequence of breakpoints for the approximation of u^* by u_N.

4.3.1 Computing the local integration error

For a given N, we can compute $e_{N,k}$, $k = 0, \ldots, N-1$ approximately by comparing $\bar{x}_{N,k+1}$ to the value which would have been obtained by taking two RK half-steps from $\bar{x}_{N,k}$. As in equation (3.2), let $x_{N,k}(t_{k+1})$ be the solution of $\dot{x} = h(x,u)$ with $x(t_k) = \bar{x}_{N,k}$ and define the following quantities:

$$\tilde{x}_{N,k,1} = \frac{\Delta_{N,k}}{2} F(\bar{x}_{N,k}, \tilde{v}_{k1}) \tag{3.4a}$$

$$\tilde{x}_{N,k,2} = \frac{\Delta_{N,k}}{2} F(\tilde{x}_{N,k,1}, \tilde{v}_{k2}) \,, \tag{3.4b}$$

where

† This heuristic can fail if the local integration errors are very small. This can happen, for instance, when integrating a linear system with piecewise linear control using a fourth order integration method because the derivatives of the spline do not show up, or are multiplied by very small quantities, in the Taylor expansion for the LTE.

$$\tilde{v}_{k1} = (u[t_{N,k} + c_1 \tfrac{\Delta_{N,k}}{2}] \cdots u[t_{N,k} + c_s \tfrac{\Delta_{N,k}}{2}]) \tag{3.4c}$$

$$\tilde{v}_{k2} = (u[t_{N,k} + (1+c_1) \tfrac{\Delta_{N,k}}{2}] \cdots u[t_{N,k} + (1+c_s) \tfrac{\Delta_{N,k}}{2}]) \,. \tag{3.4d}$$

and $F(x, \tilde{v})$ is the composite function defined in (2.5.1) that implements one Runge-Kutta integration step for the given set of control samples $\tilde{v}$. The quantity $\tilde{x}_{N,k,2}$ thus defined represents the result of taking two RK half-steps from $\bar{x}_{N,k}$. These quantities are displayed in Figure 3.1. By making the approximation that the coefficient of the PLTE of the first half-step is the same as that for the second half-step (which is reasonable since the control is smooth over the whole interval), we obtain, by adding the PLTE for each half-step,

$$\tilde{x}_{N,k,2} = x_{N,k}(t_{N,k+1}) + 2e_{N,k}\left(\frac{\Delta_{N,k}}{2}\right)^{s+1} + O(\Delta_{N,k}^{s+2}) \,. \tag{3.5}$$

Combining this expression with (3.2) and dropping the higher order terms, we obtain the following estimate of the norm of the PLTE,

$$\boxed{|e_{N,k}|\Delta_{N,k}^{s+1} \approx \frac{|\bar{x}_{N,k+1} - \tilde{x}_{N,k,2}|}{(1 - 1/2^s)} \;,\;\; k = 0, \ldots, N-1} \,. \tag{3.6a}$$

For systems of differential equations, $e_{N,k}$ is a vector and $|\cdot|$ can be taken as any norm on $\mathbb{R}^n$. Instead of the absolute size of the PLTE, the relative size of the PLTE can be estimated using

$$\boxed{|e_{N,k}|\Delta_{N,k}^{s+1} \approx \frac{|\bar{x}_{N,k+1} - \tilde{x}_{N,k,2}| / (1 + |\bar{x}_{N,k}|)}{(1 - 1/2^s)} \;,\;\; k = 0, \ldots, N-1} \,. \tag{3.6b}$$

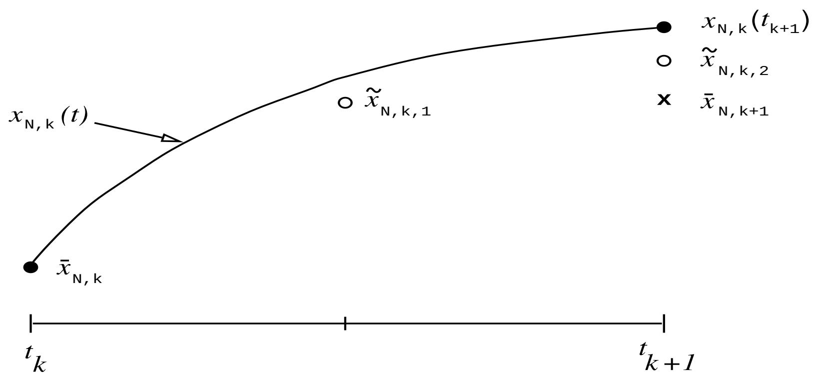


Fig. 3.1: The quantities is used in computing $e_{N,k}$.

Remark 3.1. It is important to state that this type of computation would not normally be done when solving differential equations because it involves essentially re-integrating the differential equations on a doubled mesh (although it is possible to reduce the amount of work by re-using quantities computed in the full-step length calculation for the half-step calculations). However, this amount of work is small relative to the number of simulation required to solve $\mathbf{P}_{N_i}$. Moreover, the work required to compute an integration mesh that achieves a certain accuracy with as small of a discretization level as possible is greatly offset by the savings in solving $\mathbf{P}_{N_{i+1}}$ that result from having fewer decision variables in the discretized problem. □

Remark 3.2. There are several other interesting methods for approximately computing the local truncation errors for Runge-Kutta integration. The method in [100], presented for RK4, is similar to ours except that it uses the doubled step computation for the integration result and use the single step computation for the purpose of constructing an embedded third order RK method to compute the error estimation. In this way, the error estimate is produced from a (3,4) pair with local extrapolation and requires no additional function evaluations. Another very efficient approach based on a similar idea is given in [101]. Also, it is possible to obtain error estimates by constructing interpolating polynomials as is done with linear multi-step methods [102]. A comparison of the efficiency and accuracy of error estimation methods is given in [103]. □

Global Integration Error. It is sometimes useful to have an estimate for the total integration error. Based on the convergence proofs for RK integration, it follows that

$$\|\bar{x}_{N,N} - x(t_N)\| \leq \sum_{k=0}^{N-1} |e_{N,k}|\Delta_{N,k}^{s+1} + O(\Delta_N^{s+1}) = O(\Delta_N^{s}), \tag{3.6c}$$

where $\Delta_N \doteq \max_k \Delta_{N,k}$. Hence, the quantity

$$\boxed{E_{\mathbf{t}_N} \doteq \sum_{k=0}^{N-1} |e_{N,k}|\Delta_{N,k}^{s+1}} \tag{3.6d}$$

provides an estimate of the total integration error.

4.3.2 Strategies for mesh refinement

The preceding discussion suggests that it may be advantageous to consider non-uniform meshes for the approximating problems. The non-uniform mesh would be chosen to distribute the estimates of the local integration error in such away that $\max_k |e_{N,k}|\Delta_{N,k}^{s+1}$ is minimized. There are two approaches for choosing an integration mesh. The first, *static mesh refinement*, uses the integration error evaluated at the solution, or approximate solution, η_{N_i} of $\mathbf{P}_{N_i}$ to produce $\mathbf{t}_{N_{i+1}}$ as a refinement of $\mathbf{t}_{N_i}$. In the second approach, *dynamic mesh refinement*, the approximating problem

$\mathbf{P}_N$ is modified to include the breakpoints of the mesh $\mathbf{t}_N$ as decision variables. In this way, the optimization problem that solves $\mathbf{P}_N$ also adjusts the integration mesh.

We choose the first approach because it easily fits into our theory of consistent approximations and we believe it to be the more efficient approach. Before discussing static refinement in more detail, we will describe dynamic refinement and explain why we feel that it is a less efficient approach for ultimately obtaining solutions to $\mathbf{P}$.

One method for introducing mesh breakpoints as decision variables in solving optimal control problems was given in [104]. In that paper the mesh points were allowed to move in order to help minimize the objective function in the optimization of $\mathbf{P}_N$. However, it was discovered that this led to the placement of breakpoints in a manner that reduced the objective function by allowing the violation of trajectory constraints to increase between mesh points. This problem was alleviated by constructing approximating polynomials to the state trajectories and ensuring constraint satisfaction for the polynomial over its whole interval of definition. In [41], Stryk notes that there are serious convergence and conditioning problems associated with this method. Other strategies based on equidistribution of the discretization error which avoid this problem altogether are developed in [34,41]. These methods lead to serious complication of the nonlinear program which must be solved to obtain a solution to $\mathbf{P}_N$. Specifically, a large system of nonlinear constraints that approximately equidistributes the integration error is added to the original nonlinear program. This results in a significant increase in the computation time.

The advantage of dynamic mesh refinement is that, for a given discretization level N, the mesh points can be placed quite accurately with respect to minimizing the integration error for that discretization level. However, this does not mean that the solution η_N of $\mathbf{P}_N$ is the best approximate solution of $\mathbf{P}$ that can be obtained for a discretization level N. Furthermore, adding so much extra computational burden in order to dynamically refine the mesh only makes sense if only one approximating problem is to be solved. Since we plan to solve a sequence of approximating problems $\mathbf{P}_{N_i}$, we can obtain meshes $\mathbf{t}_{N_i}$ that are almost the same as a mesh produced by dynamic refinement simply by refining the meshes between the solution of problems $\mathbf{P}_{N_i}$. The reason for this is that once η_{N_i} is close to η^*, the distribution of the local integration errors will not change much. Thus, the relative sizes of $t_{N_i,k+1} - t_{N_i,k}$ will change very little once $\|\eta_{N_{i+1}} - \eta_{N_i}\|$ becomes small.

It should be noted that neither dynamic mesh refinement nor static mesh refinement will be able to locate discontinuities in the optimal control or exit and entry points for trajectory constraints exactly. Nor will they be able to ensure that control and trajectory constraints are satisfied exactly for all $t \in [0, 1]$. A strategy for producing solutions of very high accuracy is discussed in the chapter on future research.

Static Mesh Refinement (Strategy 1, movable knots). We propose two strategies for static mesh refinement. Both attempt to choose the mesh points in order to approximately equidistribute the principal local truncation error $e_{N,k}\Delta_{N,k}^{s+1}$, $k = 0,..,N-1$. The first strategy is based on the algorithm `NEWKNT` for the repositioning of spline knot locations [63, pp. 182-184]. Given a mesh $\mathbf{t}_{N_i}$ and a solution η_{N_i} to $\mathbf{P}_{N_i}$ defined on that mesh, we seek to choose a new mesh $\mathbf{t}_{N_{i+1}} = \{ t_{N_{i+1},k} \}_{k=0}^{N_{i+1}}$ with the property that

$$|e_{N_{i+1},k}|\Delta_{N_{i+1},k}^{s+1} = |e_{N_{i+1},k+1}|\Delta_{N_{i+1},k+1}^{s+1} \ , \ \forall\, k \in \{ 0, \ldots, N_{i+1}-2 \} \ , \tag{3.7a}$$

where $\Delta_{N_{i+1},k} \doteq t_{N_{i+1},k+1} - t_{N_{i+1},k}$. Clearly, this is equivalent to choosing $\{ t_{N_{i+1},k} \}$ such that

$$|e_{N_{i+1},k}|^{1/s+1}\Delta_{N_{i+1},k} = |e_{N_{i+1},k+1}|^{1/s+1}\Delta_{N_{i+1},k+1} \ . \tag{3.7b}$$

We proceed by defining the piecewise constant function $E(t)$ which interpolates the values $(t_{N_i,k}, |e_{N_i,k}|\Delta_{N_i,k}^{s+1})$. Thus, for $t \in [t_k, t_{k+1})$, $E(t) = |e_{N_i,k}|\Delta_{N_i,k}^{s+1}$. Then, satisfying (3.7b) is equivalent to choosing $\{ t_{N_{i+1},k} \}$ such that for each $k \in \{ 0, \ldots, N_{i+1}-1 \}$,

$$\int_{t_{N_{i+1},k}}^{t_{N_{i+1},k+1}} E(t)^{1/s+1} dt = \frac{1}{N_{i+1}} \int_0^1 E(t)^{1/s+1} dt = \frac{1}{N_{i+1}} \sum_{j=0}^{N_i-1} |e_{N_i,j}|^{1/s+1}\Delta_{N_i,j} \ . \tag{3.7c}$$

This problem is easily solved, once N_{i+1} is specified, by constructing the continuous, monotone increasing, piecewise linear function

$$G(\tau) \doteq \int_0^\tau E(t)^{1/s+1} d\tau \tag{3.8a}$$

and setting

$$t_{N_{i+1},k} = G^{-1}\!\left(\frac{k}{N_{i+1}}\, G(1) \right), \ \ k = 0, \ldots, N_{i+1} \ . \tag{3.8b}$$

To choose N_{i+1} we can use the following heuristic. Since the total integration error is approximately given by

$$E_{\mathbf{t}_{N_i}} \doteq \sum_{k=0}^{N_i-1} |e_{N_i,k}|\Delta_{N_i,k}^{s+1} \approx \frac{K}{N_i^s} \ , \tag{3.9}$$

we could reduce by a factor of FAC the total integration error without redistributing by choosing a discretization level

$$N' = \lceil N_i (FAC)^{1/s} \rceil , \tag{3.10a}$$

where $\lceil r \rceil$ is the smallest integer larger than r. Since this value of N' does not taken into account the benefit of redistributing the mesh, we instead use

$$N_{i+1} = \max\{\, N_i, \lceil N_i \left(\frac{FAC}{\max_k \{ |e_{N_i,k}|/\bar{e} \}} \right)^{1/s} \rceil \} \ , \tag{3.10b}$$

where $\bar{e} = \frac{1}{N_i} \sum_{k=0}^{N-1} |e_{N_i,k}|$.

Finally, in order to ensure that the mesh refinement strategy will produce quasi-uniform meshes, we set

$$e_{N,k} = \max\{\, |e_{N,k}|, \max_k |e_{N,k}|/\bar{\delta} \,\} \ , \tag{3.11}$$

where $\bar{\delta}$ is the constant in Definition 2.1 of quasi-uniformity, before computing the new mesh. Also, an estimate, δ_Δ, of the effect of redistribution is computed as:

$$\delta_\Delta = \frac{\max_k |e_k|^{1/s+1} \Delta_{N,k}}{\min_k |e_k|^{1/s+1} \Delta_{N,k}} \ . \tag{3.12}$$

The larger δ_Δ is, the larger the benefit received from redistribution will be.

Static Mesh Refinement (Strategy 2, fixed knots). Our second strategy is based on the heuristic in [105] which allows mesh points to be added or deleted, but not moved. Thus, if no deletion occurs, the control subspaces nest, *i.e.* $L_{\mathbf{t}_{N_{i+1}}} \subset L_{\mathbf{t}_{N_i}}$. The estimates in (3.6a) or (3.6b) for the local truncation error are used to divide each interval $[t_k, t_{k+1}]$ into n_k subintervals. The only subtlety is that, whenever an interval is removed, the local truncation error associated with that interval is added to the local truncation error of the previous interval. The refinement is performed iteratively as follows:

Step 1: Compute the average local truncation error

$$\bar{e} = \frac{1}{N_i} \sum_{k=0}^{N_i-1} |e_{N_i,k}|^{1/s+1} \Delta_{N_i,k} \ , \tag{3.13a}$$

and compute the relative local truncation errors,

$$\bar{e}_k = \frac{|e_{N_i,k}|^{1/s+1} \Delta_{N_i,k}}{\bar{e}} \left(\frac{FAC}{\max_k \{ |e_{N_i,k}|/\bar{e} \}} \right)^{1/s} \ , \tag{3.13b}$$

where the second term, as in Strategy 1, equation (3.10b), specifies that the integration error should decrease by a factor of FAC.

Step 2: Choose $\sigma \in (0, \frac{1}{2})$. Determine which mesh points, t_k, are to be removed and add their relative local truncation errors, $\bar{e}_k$, to the relative local truncation error, $\bar{e}_{k-1}$, of the previous interval. A mesh point is to be removed (in Step 4) if $\bar{e}_k < \sigma$. The following loop implements this procedure:

for $k = N_i - 1$ **by -1 to 2,**
  if $\bar{e}_k < 0.25$
    $\bar{e}_{k-1} = \bar{e}_{k-1} + \bar{e}_k$
    $\bar{e}_k = 0$
  endif
  if $0.25 \le \bar{e}_k < 1$
    $\bar{e}_k = 1$
  endif
endfor

Step 3: For each $k = 0, \ldots, N-1$, let $n_k = [\bar{e}_k]$, where $[\bar{e}_k]$ is the integer nearest to $\bar{e}_k$. If $n_0 = 0$ set $n_0 = 1$ (so that the leftmost breakpoint will not be removed).

Step 4: Let $I = \{\ k \mid n_k \ge 1\ \}$ and create the new mesh

$$\mathbf{t}_{N_{i+1}} = \{\ \{\ t_{N_i,k} + \frac{i\Delta_{N_i,k}}{n_k}\ \}_{i=0}^{n_k-1}\ \}_{k \in I}\ . \tag{3.14}$$

□

Before performing the redistribution, we set, as in Strategy 1,

$$e_{N,k} = \max \{\ |e_{N,k}|, \max_k |e_{N,k}|/\bar{\delta}\ \}\ , \tag{3.15}$$

where $\bar{\delta}$ is the constant in Definition 2.1 of quasi-uniformity, in order to ensure that the mesh refinement strategies which use these local truncation errors will produce quasi-uniform meshes. As in Strategy 1, an estimate, δ_Δ, of the effect of redistribution can be computed as:

$$\delta_\Delta = \frac{\max_k |e_k|^{1/s+1}\Delta_{N,k}}{\min_k |e_k|^{1/s+1}\Delta_{N,k}}\ . \tag{3.16}$$

In the program that implements this redistribution strategy (see **distribute** in Chapter 5.7), a mechanism has been added before Step 2 that causes mesh points to be added at or near active trajectory constraints. Specifically, for each k such that $t_{N,k}$ is at or near an active trajectory constraint, $\bar{e}_k$ is set to

$$\bar{e}_k \leftarrow \bar{e}_k + 1 \tag{3.17}$$

Redistribution examples. The following plots demonstrate the effect and usefulness of mesh redistribution. We have set $\bar{\delta} = 50$ for both redistribution strategies and have set $\sigma = 1/4$ for Strategy 2. The first three plots were based on integrating the differential equations for the Rayleigh problem with the solution $u_N^* \in L_N^{(2)}$ of the approximating problem discretized using RK method $\mathbf{A}_2$ with $N = 50$. The first figure, Figure 3.2a, shows the local truncation errors, $\bar{x}_{N,k+1} - x_{N,k}(t_{k+1})$, $k = 0, \ldots, N-1$, produced by RK method $\mathbf{A}_2$ before and after mesh redistribution. We are actually plotting the time function

$$(\bar{x}_{N,k+1} - x_{N,k}(t_{k+1}))\Pi_k(t)\ ,\ \ k = 0, \ldots, N-1\ ,\ \ t \in [0, 2.5]\ ,$$

where

$$\Pi_k(t) = \begin{cases} 1 & , t \in [t_k, t_{k+1}) \\ 0 & , \text{otherwise.} \end{cases}$$

Notice that local truncation errors for the mesh produced by Strategy 1 are almost equidistributed. Strategy 2 does not quite achieve equidistribution and the number of mesh intervals increased from $N = 50$ to $N = 64$. Strategy 2 does, however, achieve nesting. Figure 3.2b is a close-up look at these local truncation after mesh redistribution. Finally, Figure 3.2c show the effect on the solution u_N^* before and after redistribution.

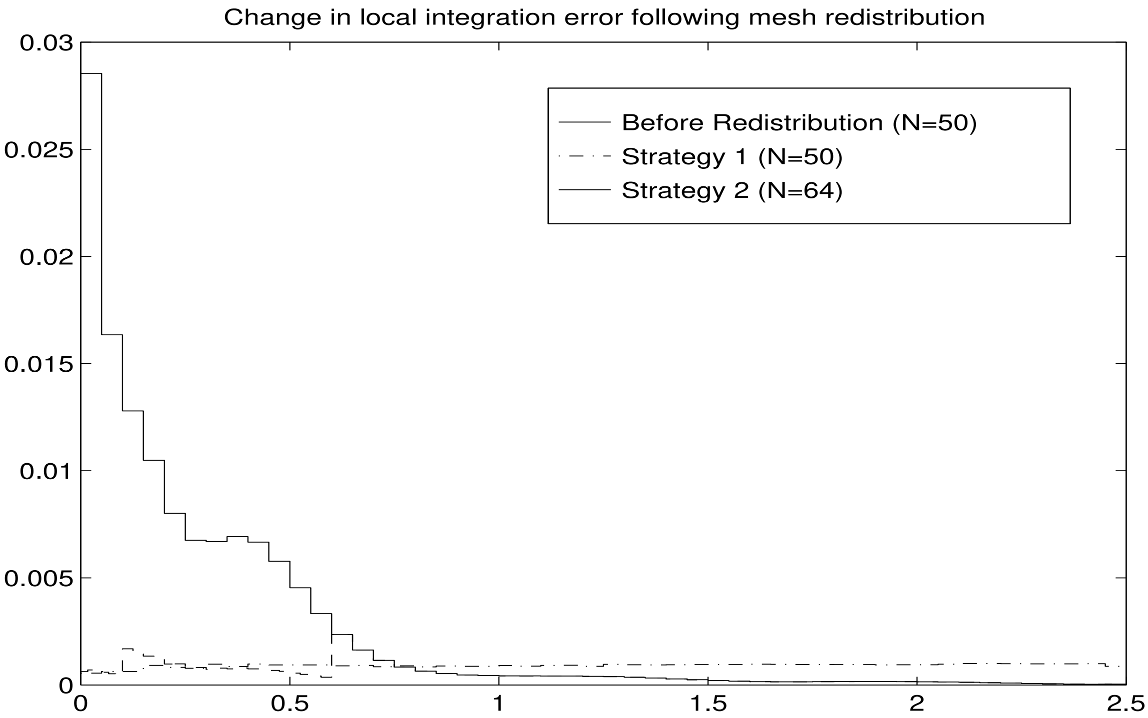


Fig. 3.2a: This plot shows the values of the local truncation versus time before and after mesh redistribution.

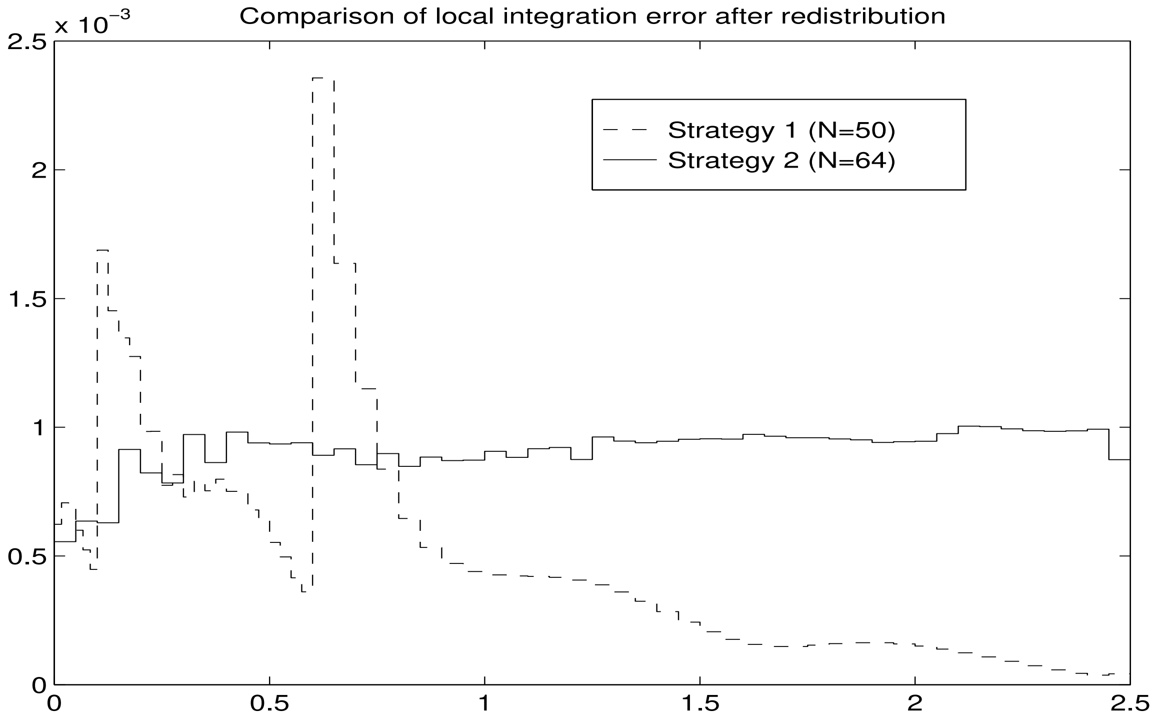


Fig. 3.2b: Closeup up view of the local truncation errors following the mesh redistribution by Strategies 1 and 2.

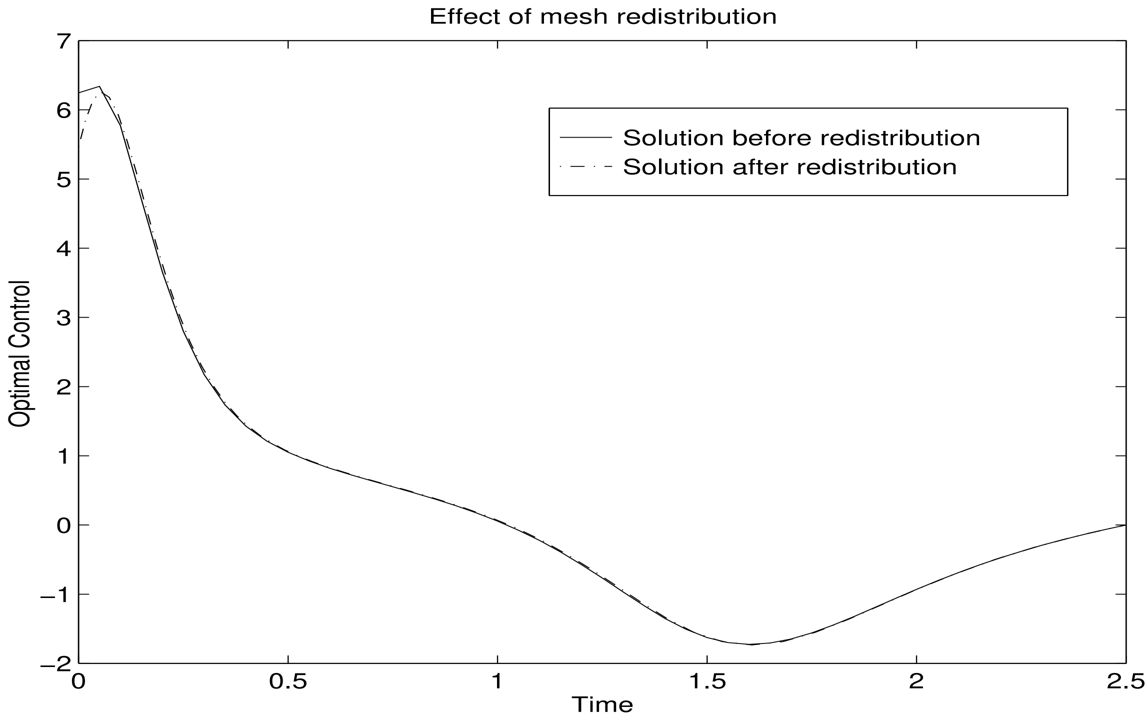


Fig. 3.2c: This plot compares of solution of the Rayleigh problem before and after mesh redistribution. Same number of points ($N = 50$) in both meshes.

The next table provides some quantitative results on the effect of the mesh redistribution. The error $\|u_N{}^* - u^*\|_2$ was calculated according to (2.12) with $T = 2.5$.

	total integration error	$\|u_N{}^* - u^*\|_2$
Before redistribution	0.1806	1.06e-1
After Strategy 1	0.0652	1.56e-2
After Strategy 2	0.0490	8.12e-3

Table 3.3: Integration error and solution error before and after mesh redistribution.

Note that the Strategy 1 results in almost a seven-fold decrease in the solution error without increasing the size of the mesh. For Strategy 2, the error is reduced by a factor of 13 with only a small increase in the number of mesh points.

As another example, we show the solution $u_N{}^* \in L^{(2)}_{\mathbf{t}_N}$ of problem Bang before and after mesh redistribution using Strategy 1. Again, we use RK method $\mathbf{A}_2$. In this case there are $N = 20$ intervals. The circles in Figure 3.4 indicate where the mesh points occur.

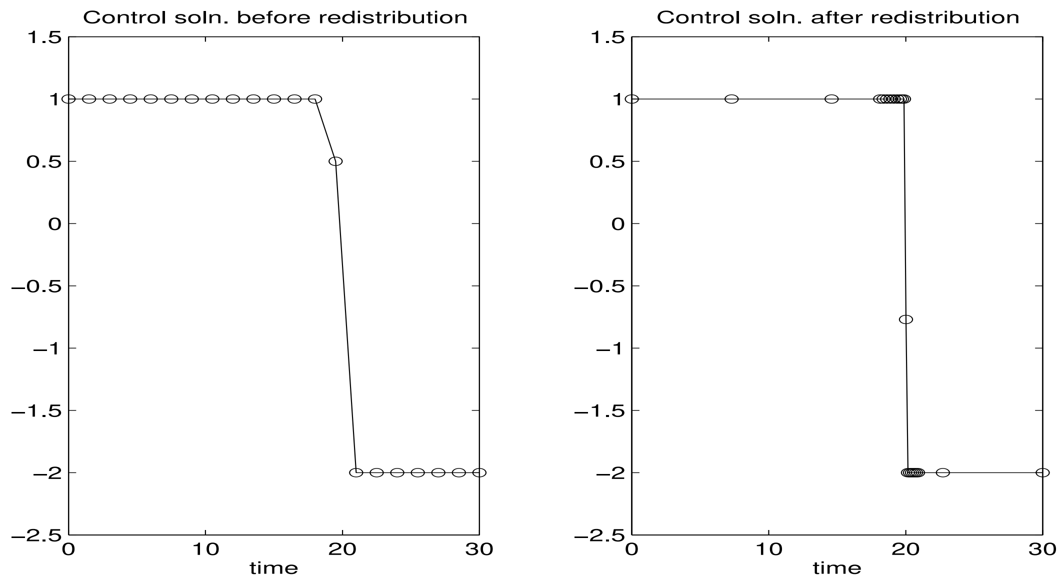


Fig. 3.4: Comparison of solution for problem Bang before and after mesh redistribution. Same number of points ($N = 20$) in both meshes. The optimal control is a bang-bang solution.

The next table provides quantitative results on the effect of redistribution. Again, $\|u_N{}^* - u^*\|_2$ is computed according to (2.12), this time with $T = T^* = 30$.

	total integration error	$\|u_N^* - u^*\|_2$
Before redistribution	8.44e-1	1.251
After Strategy 1	7.098e-3	0.465

Table 3.5: Integration error and solution error before and after mesh redistribution.

4.4 ESTIMATION OF SOLUTION ERROR

In this section we consider the sequence $\{ \eta_N \}$ of approximate solutions computed for the approximating problems $\{ \mathbf{P}_N \}$. For each N, we would like to be able to determine the solution error $\|\eta_N - \eta^*\|_{H_2}$. We will provide a formula for estimating this error that is based on heuristics. For simplicity we will consider problems of the form

$$\mathbf{P} \qquad \min_{\eta \in \mathbf{H}} \{ f(\eta) \mid g(\eta) = 0 \} \ , \tag{4.1a}$$

where $g : \mathbf{H} \to \mathbb{R}^q$ with $\mathbf{H} \doteq \mathbb{R}^n \times \mathbf{U}$ and $\mathbf{U}$, the feasible control set, is given by

$$\mathbf{U} \doteq \{ u \in L^m_{\infty,2}[0,1] \mid u(t) \in U \text{ for } t \in [0,1] \text{ a.e.} \} \tag{4.1b}$$

$$U \doteq \{ v \in \mathbb{R}^m \mid \|v\| \le \rho_{\max} \} \tag{4.1c}$$

with $\rho_{\max}$ sufficiently large so that, for all iterates, $u(t)$ is in the interior of U almost everywhere in $[0,1]$. Effectively then, there are no control constraints. The results are easily extended to problems with inequality endpoint and trajectory constraints. The extension to problems with control bounds is presented following the current discussion.

A useful quantity to consider for constrained problems is the augmented Lagrangian

$$L_c(\eta, \lambda) \doteq L(\eta, \lambda) + \frac{c}{2} g(\eta)^T g(\eta) \ , \tag{4.2a}$$

where

$$L(\eta, \lambda) \doteq f(\eta) + \lambda^T g(\eta) \tag{4.2b}$$

is the Lagrangian with λ a vector of multipliers.

The error estimate we derive below depends on a positive-definiteness property of the Hessian of the augmented Lagrangian. This property is associated with solution points that satisfy second-order sufficiency conditions for local optimality. Second-order sufficiency conditions are easily stated for finite dimensional optimization problems but are significantly more difficult to derive for optimal control problems. For instance, the proofs for the sufficiency conditions given in [3,106] are incorrect. In the statement of the following theorem, the required Fréchet differentiability of $f(\cdot)$ and $g(\cdot)$ with respect to $\mathbf{H}$ is established in [58]. A proof of Theorem 4.1 can be found in [107, Theorem 2 (p. 187)].

Theorem 4.1 (Second-Order Sufficiency). Suppose that $f(\cdot)$ and $g(\cdot)$ are twice continuously Fréchet differentiable (so that $L_c(\cdot, \lambda)$ is). Assume that at $\eta^* \in \mathbf{H}$,

$$g(\eta^*) = 0 \,, \tag{4.3a}$$

and there exists a $\lambda^* \in \mathbb{R}^q$ and a scalar $\sigma_1 > 0$ such that

$$\nabla_\eta L(\eta^*, \lambda^*) = 0 \,, \tag{4.3b}$$

$$\langle \delta\eta, \nabla^2_{\eta\eta} L(\eta^*, \lambda^*)\delta\eta \rangle_{H_2} \geq \sigma_1 \|\delta\eta\|^2_{H_2} \tag{4.3c}$$

for all $\delta\eta \in H_2$ such that $g_\eta(\eta^*)\delta\eta = 0$. Then η^* is a local minimizer for $\mathbf{P}$, *i.e.*, there exists a $\sigma_2 > 0$ and a corresponding $\delta_2 > 0$ such that

$$f(\eta) - f(\eta^*) \geq \tfrac{1}{2}\,\sigma_2 \|\eta - \eta^*\|^2_{H_2} \tag{4.3d}$$

for all $\eta \in \mathbf{H}$ such that $\|\eta - \eta^*\|_{H_2} \leq \delta_2$. □

The proof of the following Proposition concerning the Hessian of the augmented Lagrangian is based on the finite-dimensional result given in [89, Lemma 1.25 (p. 68)].

Proposition 4.2. Suppose that $f(\cdot)$ and $g(\cdot)$ are twice continuously Fréchet differentiable and that $\eta^* \in \mathbf{H}$ is a point satisfying the second-order sufficiency conditions for problem $\mathbf{P}$ (without control constraints) in Theorem 4.1. Let $\lambda^* \in \mathbb{R}^q$ be such that (4.3b) holds. Then there exists scalars $\sigma > 0$ and $\bar{c} \in (0, \infty)$ such that for all $c \geq \bar{c}$

$$\langle \delta\eta, \nabla^2_{\eta\eta} L_c(\eta^*, \lambda^*)\delta\eta \rangle_2 \geq \sigma \|\delta\eta\|^2_{H_2} \,, \ \ \forall\, \delta\eta \in H_2 \,. \tag{4.4a}$$

Proof. Let $H \doteq \nabla^2_{\eta\eta} L(\eta^*, \lambda^*)$ and $G \doteq g_\eta(\eta^*)$. Since $g(\eta^*) = 0$, we have that for all $\delta\eta \in H_2$,

$$\langle \delta\eta, \nabla^2_{\eta\eta} L_c(\eta^*, \lambda^*)\delta\eta \rangle_{H_2} = \langle \delta\eta, H\delta\eta \rangle_{H_2} + c\|G\delta\eta\|^2_{H_2} \,. \tag{4.4b}$$

Suppose that there is no $\bar{c}$ such that (4.4a) holds. Then, for each $k \in \mathbb{N}$ and any $\sigma > 0$, there exists $\delta\eta_k \in H_2$ such that $\|\delta\eta_k\|_{H_2} = 1$ and

$$\langle \delta\eta_k, H\delta\eta_k \rangle_{H_2} + k\|G\delta\eta_k\|^2_{H_2} \leq \sigma \,. \tag{4.4c}$$

Choose $\sigma = \sigma_1/4$ where σ_1 is the positive scalar from Theorem 4.1. Taking the limit superior,

$$\overline{\lim_{k\to\infty}} \langle \delta\eta_k, H\delta\eta_k \rangle + k\|G\delta\eta_k\|^2_2 \leq \frac{\sigma_1}{4} \,. \tag{4.4d}$$

Since $\|G\delta\eta_k\| \geq 0$, (4.4d) implies that $\|G\delta\eta_k\| \to 0$ as $k \to \infty$. Next, since $L_c(\cdot, \lambda^*)$ is twice

Fréchet differentiable, $\nabla^2_{\eta\eta} L_c(\eta^*, \lambda^*)$ is a bounded bi-linear operator. Thus, it follows from (4.3c) and the fact that $\| G\delta\eta_k \| \to 0$ as $k \to \infty$ that there exist a $\bar{k}$ such that, for all $k \geq \bar{k}$, $\langle \delta\eta_k, H\delta\eta_k \rangle \geq \frac{\sigma_1}{2} \| \delta\eta_k \|^2 = \frac{\sigma_1}{2}$. But this contradicts (4.4d). □

We now proceed with a heuristic derivation of an estimate for $\| \eta_N - \eta^* \|_{H_2}$ assuming there are no control bounds. In what follows, all of the norms are H_2 norms. The following assumption is needed for our derivation.

Assumption 4.3.

(a) $L(\cdot, \lambda)$ is twice continuously differentiable.

(b) $\eta^* \in \mathbf{H}$ is a local minimizer for $\mathbf{P}$ and there exists scalars $c > 0$ and $\sigma > 0$ such that

$$\langle \delta\eta, \nabla^2_{\eta\eta} L_c(\eta^*, \lambda^*)\delta\eta \rangle_{H_2} \geq \sigma \| \delta\eta \|^2_{H_2} \ , \ \ \forall\, \delta\eta \in H_2 \ . \tag{4.5a}$$

(c) $\{ \eta_N \}_{N=0}^{\infty}$ is a sequence such that $\eta_N \in \mathbf{H}$ and $\eta_N \to \eta^*$ as $N \to \infty$. □

Note that that this assumption does *not* specify that $g(\eta_N) = 0$; generally, $g(\eta_N) \neq 0$ even if η_N is a solution of $\mathbf{P}_N$.

Now, let $\delta\eta_N \doteq \eta_N - \eta^*$. Expanding $\nabla_\eta L_c(\eta_N, \lambda^*)$ to first order around η^* and using the fact that $\nabla_\eta L_c(\eta^*, \lambda^*) = 0$, we get

$$\nabla_\eta L_c(\eta_N) = \int_0^1 \nabla^2_{\eta\eta} L_c(\eta^* + s\delta\eta_N, \lambda^*)\delta\eta_N \, ds \ . \tag{4.6}$$

For convenience, define

$$H_c(\delta\eta_N) \doteq \int_0^1 \nabla^2_{\eta\eta} L_c(\eta^* + s\delta\eta_N, \lambda^*) ds \tag{4.7a}$$

so that (4.6) can be written

$$\nabla_\eta L_c(\eta_N) = H_c(\delta\eta_N)\delta\eta_N \ . \tag{4.7b}$$

From (4.7b), we have for any $\delta\eta_N \in H_2$,

$$\langle \nabla_\eta L_c(\eta_N, \lambda^*), \delta\eta_N \rangle = \langle \delta\eta_N, H_c(\delta\eta_N)\delta\eta_N \rangle = \frac{1}{K_N} \| \delta\eta_N \|^2 \ , \tag{4.8a}$$

for some K_N. We will show that K_N is finite and bounded away from zero. From the continuity of $\nabla^2_{\eta\eta} L_c(\cdot, \lambda^*)$, the fact that $\delta\eta_N \to 0$ as $N \to \infty$, and Assumption 4.3*(b)*, there exists an $\bar{N}_1 < \infty$ such that for $N \geq \bar{N}_1$,

$$\langle \delta\eta_N, H_c(\delta\eta_N)\delta\eta_N \rangle \geq \tfrac{1}{2} \langle \delta\eta_N, \nabla^2_{\eta\eta} L_c(\eta^*, \lambda^*)\delta\eta_N \rangle \geq \frac{\sigma}{2} \| \delta\eta_N \|^2 \ . \tag{4.8b}$$

Comparing (4.8a) and (4.8b), we see that $K_N \leq \frac{2}{\sigma} < \infty$ (since $\sigma > 0$) for all $N \geq \bar{N}_1$. Also, since

$\nabla^2_{\eta\eta} L_c(\eta^*, \lambda^*)$ is a bounded bi-linear operator, there exists an $M < \infty$ such that

$$\langle \delta\eta, \nabla^2_{\eta\eta} L_c(\eta^*, \lambda^*) \delta\eta \rangle_2 \le M \|\delta\eta\|^2_{H_2} \ , \ \forall\, \delta\eta \in H_2 \ . \tag{4.8c}$$

Hence, there exists an $\bar{N}_2$ such that, for all $N \ge \bar{N}_2$, $\frac{1}{K_N} \le 2M$. Thus,

$$\frac{1}{2M} \le K_N \le \frac{2}{\sigma} \ , \ \forall\, N \ge \max \{ \bar{N}_1, \bar{N}_2 \} \ , \tag{4.8d}$$

and from (4.8a),

$$\|\delta\eta_N\|^2 = K_N \langle \nabla_\eta L_c(\eta_N), \delta\eta_N \rangle \le K_N \|\nabla_\eta L_c(\eta_N, \lambda^*)\| \|\delta\eta_N\| \ . \tag{4.8e}$$

This implies that, for $N \ge \max \{ \bar{N}_1, \bar{N}_2 \}$,

$$\|\delta\eta_N\| \le K_N \|\nabla_\eta L_c(\eta_N, \lambda^*)\| \ . \tag{4.9}$$

At this point, we could try to estimate M, use $1/2M$ in (4.8d) as a lower bound for K_N and use (4.9) as our estimate. However, this would lead to a very conservative estimate. Instead, we will attempt to estimate K_N directly using two solutions, η_{N_i} and $\eta_{N_{i+1}}$ computed for $N_i \ne N_{i+1}$. We have $\delta\eta_{N_{i+1}} = \eta_{N_{i+1}} - \eta_{N_i} + \delta\eta_{N_i}$ and $\delta\eta_{N_i} = \eta_{N_i} - \eta_{N_{i+1}} + \delta\eta_{N_{i+1}}$. Hence

$$\|\delta\eta_{N_{i+1}}\| - \|\delta\eta_{N_i}\| \le \|\eta_{N_{i+1}} - \eta_{N_i}\| \tag{4.10a}$$

and

$$\|\delta\eta_{N_i}\| - \|\delta\eta_{N_{i+1}}\| \le \|\eta_{N_{i+1}} - \eta_{N_i}\| \ . \tag{4.10b}$$

We conclude that

$$\big| \|\delta\eta_{N_{i+1}}\| - \|\delta\eta_{N_i}\| \big| \le \|\eta_{N_{i+1}} - \eta_{N_i}\| \ . \tag{4.10c}$$

Proceeding heuristically, we are going to assume that for N sufficiently large there is a $K < \infty$ such that we can replace (4.9) with

$$\|\delta\eta_N\| \approx K \|\nabla_\eta L_c(\eta_N, \lambda^*)\| \ . \tag{4.11a}$$

Then, from (4.10c) and (4.11a), we have

$$K \big| \|\nabla_\eta L_c(\eta_{N_{i+1}}, \lambda^*)\| - \|\nabla_\eta L_c(\eta_{N_i}, \lambda^*)\| \big| \le \|\eta_{N_{i+1}} - \eta_{N_i}\| \ , \tag{4.11b}$$

from which we estimate

$$K \le \frac{\|\eta_{N_{i+1}} - \eta_{N_i}\|}{\big| \|\nabla_\eta L_c(\eta_{N_{i+1}}, \lambda^*)\| - \|\nabla_\eta L_c(\eta_{N_i}, \lambda^*)\| \big|} \ . \tag{4.12}$$

Substituting this bound back into (4.11a) we get

$$\boxed{\|\eta_{N_{i+1}} - \eta^*\| \le \frac{\|\eta_{N_{i+1}} - \eta_{N_i}\| \|\nabla_\eta L_c(\eta_{N_{i+1}}, \lambda^*)\|}{\big|\, \|\nabla_\eta L_c(\eta_{N_{i+1}}, \lambda^*)\| - \|\nabla_\eta L_c(\eta_{N_i}, \lambda^*)\| \,\big|}} \ . \tag{4.13a}$$

In this result, neither the the Lagrange multipliers λ^* nor the minimum penalty parameter $\bar{c}$ are known *a priori*. So, for the purpose of implementation, we use $L_c(\eta_N, \lambda_N)$ in place of $L_c(\eta_N, \lambda^*)$ where λ_N is the vector of Lagrange multiplier estimates obtained from the solution of $\mathbf{P}_N$. Also, instead of a single penalty parameter c, we use separate penalties for each constraint. For the i-th constraint, we choose

$$c_i = |\lambda_N^i| \ , \ \ i = 1, \ldots, q \ . \tag{4.13b}$$

These values for c_i are used because, with this choice, the function $f(\eta) + \bar{c}|g_i(\eta)|$ has an unconstrained minimum at η^* if $\bar{c} \ge \sum_i c_i$ with c_i given in (4.13b), see. [89, Proposition 4.1] This, of course, does not imply that with this choice for c_i the augmented Lagrangian has an unconstrained minimum at η^*. But (4.13b) seemed to work well in our limited experiments. A rigorous choice for the c_i is an open question at this point.

Extension to Problems with Control Bounds. In the presence of control bounds, expression (4.13a) is not useful because *(i)* requiring Assumption 4.3*(b)* to hold may be too much to ask and *(ii)* the expansion in (4.6) is incorrect since $\nabla_{\eta\eta} L_c(\eta^*, \lambda^*) \ne 0$. However, we can easily produce a modification to handle control bounds. For simplicity of presentation, we will only deal with single input systems and problems with fixed initial conditions. The extension to problems with vector inputs and free initial conditions is straightforward.

Consider a problem of the form

$$\mathbf{P} \qquad \min_{u \in \mathbf{U}} \{ \, f(u) \mid g(u) = 0 \, \} \ ,$$

where $f\colon \mathbf{U} \to \mathbb{R}$, $g : \mathbf{U} \to \mathbb{R}^q$ are defined as

$$f(u) \doteq \zeta_o(x^u(1)) \ , \tag{4.14a}$$

$$g^\nu(u) \doteq \zeta_c^\nu(x^u(1)) \ , \ \ \nu = 1, \ldots, q \ , \tag{4.14b}$$

where $x^u(\cdot)$ is the solution of

$$\dot{x} = h(x, u) \ , \ \ x(0) = \xi \ , \ \ t \in [0, 1] \ , \tag{4.14c}$$

and the feasible control set is defined as

$$\mathbf{U} \doteq \{ \, u \in L^1_{\infty,2}[0,1] \ \mid \ b_l(t) \le u(t) \le b_u(t) \ , \ \ \forall \, t \in [0,1] \, \} \tag{4.14d}$$

where $b_l : [0,1] \to \mathbb{R}$ and $b_u : [0,1] \to \mathbb{R}$ are functions such that $-\infty < b_l(t) < b_u(t) < \infty$ for all $t \in [0, 1]$.

There are several versions of Kuhn-Tucker like sufficiency conditions for problems with control bounds which can be found in [108-113]. The results provided in [109] are the most useful for our purposes. The statement of the second-order sufficiency conditions requires the Hamiltonian which, for problem **P**, is defined as

$$H(x, p, u) \doteq p^T h(x, u) , \tag{4.15a}$$

where, with $u \in \mathbf{U}$ and $\lambda \in \mathbb{R}^q$ and the adjoint variable $p^{u,\lambda}(t)$, $t \in [0, 1]$, is the solution of

$$\frac{d}{dt} p(t) = -H_x(x^u(t), p(t), u(t))^T \quad ; \quad p(1) = \nabla \zeta_o(x^u(1)) + \lambda^T \nabla \zeta_c(x^u(1)) . \tag{4.15b}$$

With this definition, $\nabla_u L(u, \lambda)(t) = H_u(x^u(t), p^{u,\lambda}(t), u(t))$, $t \in [0, 1]$ (see [109, Note 4.1]). We also need to define the quantity

$$\mathcal{H}(\lambda, u\,; v, t) \doteq \langle \mathcal{H}_u(\chi(t), p(t), u(t)), v - u(t) \rangle$$

$$+ \tfrac{1}{2} \langle v - u(t), H_{uu}(xu(t), p^*(t), u(t))(v - u(t)) \rangle . \tag{4.15c}$$

Given a solution u^* of problem **P**, define the set

$$A \doteq \{\, t \in [0, 1] \;\big|\; u^*(t) = b_l(t) \;\text{ or }\; u^*(t) = b_u(t) \,\} . \tag{4.15d}$$

The following theorem is a special case of [109, Theorem 4.2].

Theorem 4.4 (Second-Order Sufficiency with Control Bounds). Suppose that $f(\cdot)$ and $g(\cdot)$ are twice continuously Fréchet differentiable (so that $L_c(\cdot, \lambda)$ is). Assume that the following conditions hold at $u^* \in \mathbf{U}$:

$$g(u^*) = 0 , \tag{4.16a}$$

there exists $\lambda^* \in \mathbb{R}^q$ such that

$$\nabla L(u^*, \lambda^*)(t) = 0 \;\text{ if }\; t \notin A , \tag{4.16b}$$

$$\nabla L(u^*, \lambda^*)(t) > 0 \;\text{ if }\; u^*(t) = b_l(t) , \tag{4.16c}$$

$$\nabla L(u^*, \lambda^*)(t) < 0 \;\text{ if }\; u^*(t) = b_u(t) , \tag{4.16d}$$

holds for $t \in [0, 1]$ a.e., and there exists scalars $\sigma_1 > 0$ and $\sigma_2 > 0$, such that

$$\langle \delta u, \nabla^2 L(u^*, \lambda^*) \delta u \rangle_2 \geq \sigma_1 \|\delta u\|_2^2 \tag{4.16e}$$

for all for all $\delta u \in L_{\infty,2}[0, 1]$ such that $\delta u(t) = 0$ for $t \in A$, and $g_u(u^*)\delta u = 0$, and, with $\mathcal{H}(\lambda^*, u^*\,; v)$ as defined in (4.15c),

$$\mathcal{H}(\lambda^*, u^*; v, t) \geq \tfrac{1}{2}\,\sigma_2 \|v - u^*(t)\|_2^2 \ , \ \ \forall\, b_l(t) \leq v \leq b_u(t) \tag{4.16f}$$

holds for $t \in [0,1]$ a.e.. Then u^* is a local minimizer for **P**, *i.e.*, there exists a $\sigma_3 > 0$ and a corresponding $\delta_3 > 0$ such that

$$f(u) - f(u^*) \geq \tfrac{1}{2}\,\sigma_3 \|u - u^*\|_2^2 \tag{4.16g}$$

for all $u \in \mathbf{U}$ such that $g(u) = 0$ and $\|u - u^*\|_2 \leq \delta_3$. □

Remark 4.5 In [109], Dunn presents a set of conditions which ensure that $f(\cdot)$ and $g(\cdot)$ are twice continuously Fréchet differentiable. One of those conditions,

$$\lim_{\|v-u\|_2 \to 0} \|S(v) - S(u)\|_\infty = 0 \ , \tag{4.17a}$$

where $S(u)(t) \doteq H_{uu}(x^u(t), p^{u,\lambda}(t), u(t))$, can only hold if the Hamiltonian is u-quadratic (*i.e.*, has no terms higher than quadratic in u). However, Fréchet differentiability relative to the set **U** was established in [58] without condition (4.3f). □

Remark 4.6. Condition (4.16f), which does not have a counterpart for finite-dimensional nonlinear programs is, locally, a strengthening of the Pontryagin Minimum principle,

$$H(x^*(t), p^*(t), u^*(t)) = \min_{u \in U} H(x^*(t), p^*(t), v) \ , \ \ \forall\, t \in [0,1] \ . \tag{4.17b}$$

Also, condition (4.16f) can be replaced by the locally stronger Legendre-Clebsch condition

$$\langle v, H_{uu}(x^*(t), p^*(t), u^*(t)) v \rangle \geq \sigma_2 \|v\|_2^2 \ , \ \ \forall\, v \in \mathbb{R}^m \ , \ \ a.e.\, t \in [0,1] \ . \tag{4.17c}$$

□

Proposition 4.7. Suppose that $f(\cdot)$ and $g(\cdot)$ are twice continuously , that $L_c(\cdot, \lambda)$ is twice continuously Fréchet differentiable and that $u^* \in \mathbf{U}$ is a point satisfying second-order sufficiency conditions for problem **P** with control constraints $b_l(t) \leq u(t) \leq b_u(t)$ for almost all $t \in [0,1]$. Then there exists $\lambda^* \in \mathbb{R}^q$ such that equations (4.16b,c,d) hold and there exists scalars $\sigma > 0$ and $\bar{c} \in (0, \infty)$ such that for all $c \geq \bar{c}$

$$\langle \delta u, \nabla^2 L_c(u^*, \lambda^*) \delta u \rangle_{L_2} \geq \sigma \|\delta u\|_{L_2}^2 \ , \ \ \forall\, \delta u \in L_{\infty,2}[0,1] \text{ with } \delta u(t) = 0 \ \ \forall\, t \in A \ . \tag{4.18}$$

□

We now assume that $\{ u_N \}$ is a sequence in **U** such that $u_N \to u^* \in \mathbf{U}$ where u^* is a solution of **P** for which there exists a $\sigma > 0$ and $\bar{c} < \infty$ such that (4.18) holds for all $c \geq \bar{c}$. Condition (4.18) allows us to make use of the first order expansion of $\nabla L_c(u_N, \lambda^*)$ around u^* with perturbations δu restricted such that $\delta u(t) = 0$ for all $t \in A$. Let $I \doteq \{ t \in [0,1] \mid t \notin A \}$. For any N, let $u_N \in \mathbf{U}$ be a solution (or approximate solution) to $\mathbf{P}_N$ and define

$$\delta u_N \doteq \begin{cases} u_N(t) - u^*(t) & \text{if } t \in I \\ 0 & \text{if } t \in A \end{cases} . \tag{4.19a}$$

Then,

$$\nabla L_c(u^* + \delta u_N, \lambda^*) = \int_0^1 \nabla^2 L_c(u^* + s\delta u_N, \lambda^*)\delta u_N \, ds . \tag{4.19b}$$

Now, using the same reasoning that led to equation (4.13), we have

$$\|\delta u_{N_{i+1}}\| \le \frac{\|u_{N_{i+1}} - u_{N_i}\|_I \|\nabla L_c(u_{N_{i+1}}, \lambda^*)\|_I}{\left| \|\nabla L_c(u_{N_{i+1}}, \lambda^*)\|_I - \|\nabla L_c(u_{N_i}, \lambda^*)\|_I \right|} , \tag{4.20}$$

where we have used the notation

$$\|u\|_I^2 = \int_I u(t)^2 dt .$$

Next, since $I \cup A = [0, 1]$, we have for any N

$$\|u_N - u^*\|^2 = \|u^* - u_N\|_I^2 + \|u^* - u_N\|_A^2 . \tag{4.21}$$

In expressions (4.20) and (4.21), the subset I and A are unknown. To proceed, we must compute an approximation $\hat{I}_N$ to I. We can obtain this approximation using the index set $I_N \doteq \{ k \in \{0, \ldots, N\} \mid b_l(t) < u_N(t_k) < b_u(t) \}$ corresponding to the unconstrained portion of the numerical solution u_N for the approximating problem $\mathbf{P}_N$. Let

$$\hat{I}_N = \bigcup_{k \in I_N} [t_{N,k-1}, t_{N,k+1}] , \tag{4.22}$$

where we treat $t_{N,-1} = t_{N,0}$ and $t_{N,N+1} = t_{N,N}$. Then let $\hat{A}_N = [0, 1] \sim \hat{I}_N$. With this construction, we can assume that

$$\|u_N - u^*\|_{\hat{A}_N} \approx 0 \tag{4.23a}$$

since, if N is large enough, the set $\hat{A}_N \approx A$, and hence, $u_N(t) = u^*(t)$ for $t \in \hat{A}_N$. Thus, with $N = N_{i+1}$, (4.21) becomes

$$\|u_{N_{i+1}} - u^*\|^2 \approx \|u^* - u_{N_{i+1}}\|_I^2 = \|\delta u_{N_{i+1}}\|^2 . \tag{4.23b}$$

where δu_{N_i+1} is defined by (4.19a). Finally, since $\hat{I}_N$ is only an estimate of I, instead of using (4.20), we estimate the total error with a more conservative estimate (and will replace u with η since this formula works in the more general case of $\mathbf{H} \doteq \mathbb{R}^n \times \mathbf{U}$) to get

$$\boxed{\|\eta_{N_{i+1}} - \eta^*\| \approx \|\eta_{N_{i+1}} - \eta^*\|_I \le \frac{\|\eta_{N_{i+1}} - \eta_{N_i}\| \|\nabla L_c(\eta_{N_{i+1}}, \lambda^*)\|_{\hat{I}_{N_{i+1}}}}{\left| \|\nabla L_c(\eta_{N_{i+1}}, \lambda^*)\|_{\hat{I}_{N_{i+1}}} - \|\nabla L_c(\eta_{N_i}, \lambda^*)\|_{\hat{I}_{N_i}} \right|}} \ , \tag{4.24}$$

where $\|\eta\|_I^2 \doteq \|\xi\|^2 + \|u\|_I^2$. As in (4.13a), we use $L_c(\eta_N, \lambda_N)$ in place of $L_c(\eta_N, \lambda^*)$, where λ_N is the vector of Lagrange multipliers obtained from the solution of $\mathbf{P}_N$. Also, instead of a single penalty parameter c, we use separate penalties for each constraint. For the i-th constraint, we choose $c_i = |\lambda_N^i|$ as in (4.13b).

Numerical Examples. To demonstrate the usefulness of formula (4.22), we have computed this error estimate for several optimal control problems using the numerical solutions from two different meshes, t_{N_1} and t_{N_2}. To compute the required function space norms we used a variable step-size integration algorithm with its local error tolerance set to

$$\varepsilon_{\text{local}} = \min \{ \max \{ E_{\mathbf{t}_N} / 1000, 10^{-12} \} , 10^{-4} \} \ ,$$

where $E_{\mathbf{t}_N}$ is the global error estimate given by (3.6d) for the fixed integration routine that was used to discretize the optimal control problem.

The following table compares the estimate of the error $\|\eta^* - \eta_{N_2}{}^*\|$ with the actual error. The norm used in these computations is defined for $\eta = (\xi, u)$ by

$$\|\eta\|^2 \doteq \|\xi\|_2^2 + \int_0^T \|u(t)\|_2^2 dt \ . \tag{4.25}$$

For each problem, we constructed the approximating problems using both *(i)* linear splines with RK method $\mathbf{A}_2$ and *(ii)* quadratic splines with RK method $\mathbf{A}_4$. For the first four problems, the meshes were uniformly spaced. For the last problem $\mathbf{t}_{N_1}$ was uniformly spaced and $\mathbf{t}_{N_2}$ was determined by redistributing t_{N_1} with mesh redistribution Strategy 1. The problems used in this table are described in Appendix B. The quantities in Table 4.1 marked with a * were computed using approximations of η^* obtained by solving the discretized problem with $N = 1000$. The last problem has control bounds so we did not solve it with quadratic splines (see discussion of spline order selection for constrained problems). The results in Table 4.1 show that the estimates provided by (4.22) are remarkably good.

Problem	N_1	N_2	$\mathbf{A}_2$, $\rho = 2$ Estimate	$\mathbf{A}_2$, $\rho = 2$ Actual	$\mathbf{A}_4$, $\rho = 3$ Estimate	$\mathbf{A}_4$, $\rho = 3$ Actual
LQR ($T = 1$)	10	20	3.6677e-4	3.6008e-4	1.5721e-6	1.5801e-6
Switch ($T = 1$)	40	80	0.0148	0.0322	0.01235	0.0229
Rayleigh ($T = 2.5$)	50	80	0.0410	0.0335*	0.0021	0.0020*
Constr. Rayleigh	50	80	0.0369	0.0355*	0.0023	0.0021*
Bang ($T = 30$)	20	20	0.4737	0.4653		

Table 4.1: A comparison of the estimate for the error $\|\eta^* - \eta_{N_2}{}^*\|$ produced by formula (4.22) and the actual error. N_1 and N_2 are the discretization levels of the two meshes for which solutions were computed. The estimates and actual error are given for solutions produced using RK method $\mathbf{A}_2$ with linear splines and RK method $\mathbf{A}_4$ with quadratic splines (except for Bang which has control constraints). For the last problem, Bang, $N_1 = N_2$ but the mesh $\mathbf{t}_{N_1}$ is a uniform mesh and $\mathbf{t}_{N_2}$ is a non-uniform redistribution of $\mathbf{t}_{N_1}$.

4.5 SINGULAR CONTROL PROBLEMS AND THE PIECEWISE DERIVATIVE VARIATION OF THE CONTROL

It is quite possible for an optimal control problem to not satisfy second order sufficiency conditions as needed in Section 4. A common practical situation in which this occurs are problems called singular control problems. For the purpose of illustration, consider the optimal control problem

$$\mathbf{P} \qquad \min_{u \in \mathbf{U}} \{ f(u) \mid g(u) = 0 \} \ ,$$

where $f(u) = \zeta_o(x^u(1)) \in \mathbb{R}$, $g(u) = \zeta_c(x^u(1)) \in \mathbb{R}^q$, $x^u(t)$, $t \in [0,1]$, is the solution of

$$\dot{x} = h(x,u) \quad ; \quad x(0) = \xi \ . \tag{5.1a}$$

and $\mathbf{U} \doteq \{ u \in L^m_{\infty,2}[0,1] \mid u(t) \in U \ , \ \forall t \in [0,1] \}$. We assume that $f(\cdot)$, $g(\cdot)$ and $h(\cdot)$ are twice continuously differentiable. For this problem, we define the Hamiltonian as

$$H(x,p,u) \doteq p^T h(x,u) \ . \tag{5.1b}$$

For $u \in \mathbf{U}$ and $\lambda \in \mathbb{R}^q$, let the adjoint variable $p^{u,\lambda}(t)$, $t \in [0,1]$, be the solution of

$$\frac{d}{dt}\, p(t) = -H_x(x^u(t), p(t), u(t))^T \quad ; \quad p(1) = \nabla\zeta_o(x^u(1)) + \lambda^T \nabla\zeta_c(x^u(1)) \ . \tag{5.1c}$$

If u^* is a local solution of problem $\mathbf{P}$, then the following condition (Pontryagin's minimum principle, see [1]) holds for some $\lambda^* \in \mathbb{R}^q$:

$$u^*(t) = \arg\min_{u \in U} H(x^*(t), p^*(t), u) \ , \ \forall\, t \in [0,1] \ , \tag{5.1d}$$

where $x^* \doteq x^{u^*}$ and $p^* \doteq p^{u^*,\lambda^*}$. At this point, we assume that the set $\mathbf{U}$ *does not include* control constraints, *i.e.*, $U = \{ v \in \mathbb{R}^m \mid \|v\| \le \rho_{\max} \}$ with $\rho_{\max}$ sufficiently large that the values of $u(t)$ always lie in the interior of U. Then, in addition to (5.1d), the *Legendre-Clebsch* condition must hold at a solution:

$$H_{uu}(x^*(t), p^*(t), u^*(t)) \ge 0 \ , \ t \in [0,1] \ . \tag{5.1e}$$

Definition 5.1. An extremal arc, (x^*, p^*, u^*), is a triple that satisfies the necessary conditions (5.1a,b,c,d,e) for optimality. An extremal arc for problem $\mathbf{P}$ without control constraints is said to be *singular* if $H_{uu}(x^*(t), p^*(t), u^*(t))$ is singular for any $t \in [0,1]$. Any interval of an extremal arc on which $H_{uu}(x^*(t), p^*(t), u^*(t))$ is singular is called a *singular sub-arc*. If (x^*, p^*, u^*) is singular, then u^* is called a *singular control*. An optimal control problem that has singular extremal arcs is called a *singular optimal control problem.* □

The contrapositive of the following proposition (which is proved in [111,Lemma 2] for a more general setting that includes control constraints; also see Notes 4.1 and 4.2 in [109]) indicates that for a singular control, the Hessian of the Lagrangian for problem $\mathbf{P}$ is not strongly positive.

Proposition 5.2. Let $L(u, \lambda) = f(u) + \lambda^T g(u)$ be the Lagrangian for problem $\mathbf{P}$. Let u^* be a stationary point for $\mathbf{P}$ and let $\lambda^* \in \mathbb{R}^q$ be the Lagrange multipliers associated with the constraint $g(u^*) = 0$. If there exists a $\sigma > 0$ such that

$$\langle \delta u, \nabla_u L(u^*, \lambda^*)\delta u \rangle_2 \ge \sigma \|\delta u\|_2^2 \tag{5.2a}$$

for all $\delta u \in L^m_{\infty,2}[0,1]$ such that $u^* + \delta u \in \mathbf{U}$ and $g_u(u^*)\delta u = 0$, then

$$H_{uu}(x^*(t), p^*(t), u^*(t)) \ge \sigma \ , \ \forall\, t \in [0,1] \ . \tag{5.2b}$$

□

Singular arcs occur most commonly when the Hamiltonian is linear in one or more of the components of u. Then u^* is always a singular control. Suppose that $H(x, p, u) = \tilde{H}_1(x, p) + \tilde{H}_2(x, p)u$. In this case, when there are no control bounds (or in regions where the control bounds are inactive), it follows from (5.1d) that a necessary condition for u^* to be extremal is

$$H_u(x^*(t), p^*(t), u^*(t)) = \tilde{H}_2(x^*(t), p^*(t)) = 0 \ , \ \forall\, t \in [0,1] \ . \tag{5.3}$$

It is clear that, in this case, not only is u^* a singular control, but Pontryagin's minimum principle (because of equation (5.3)) provides no information about the value of the optimal control. If u^* is to be obtained from the solution of the two-point boundary value problem defined by (5.1a), (5.1c) and (5.3), *i.e.* by the so-called indirect method [5,114,115], then additional conditions on u^* are needed. Such conditions are available as generalized necessary and sufficient conditions (in particular, generalizations of (5.1d,e) involving time derivatives of H_u), see . [3,116-118]

For problems that have control constraints, (5.1e) is not a necessary condition and therefore Definition 5.1 is not useful. In the case that $H(x, p, u) = \tilde{H}_1(x, p) + \tilde{H}_2(x, p)u$, the problem is singular if $\tilde{H}_2(x(t), p(t))$, the so-called switching function, is zero for any non-zero interval of time. On such an interval, it is clear that the Pontryagin minimum principle, equation (5.1d), gives no information about the optimal control since the Hamiltonian does not involve u.

If u^* is a singular control, we have from Proposition 5.2 that the Hessian of the Lagrangian at u^* projected onto the subspace $\{ \delta u \in L^m_{\infty,2}[0, 1] \mid g_u(u^*)\delta u = 0 \}$ is singular. Thus, from the Taylor expansion of the Lagrangian we see that, on singular arcs, small perturbations in the control have only fourth†, or higher, order (very small) effects on the projected Hessian of the Lagrangian. Singularity of problem **P** will manifest itself as singularity or near-singularity in the approximating problems $\mathbf{P}_N$ for N sufficiently large. From a computational point of view, singularity of the Hessian can inhibit superlinear convergence of mathematical programming algorithms that rely on second order information.

Our primary concern, however, is that we have observed, as have other authors (for instance [119]), that the numerical solutions $u_N(\cdot)$ of singular optimal control problems can exhibit spurious oscillations along singular sub-arcs that are artifacts of the numerical method rather than being an approximation to the solution of **P**. This seems to be especially true when trajectory constraints are active on singular sub-arcs. When spurious oscillations occur in the solutions u_N of the approximating problems, the sequence $\{ u_N \}$ may not have any accumulation points. In other words, if the oscillations persist as $N \to \infty$, the sequence of solutions will not converge. Besides preventing convergence, these oscillations also prevent useful estimates of $\|\eta_{N_{i+1}} - \eta^*\|$ obtained from (4.13) or (4.22). This is because the oscillations on singular sub-arcs are erratic and prevent the quantity $\|u_{N_i} - u_{N_{i+1}}\|$ from converging to zero. We believe that these spurious oscillations appear due to the accumulation of numerical errors which, on the singular sub-arcs, have very little effect on the Lagrangian because of the singularity of its Hessian. The reason this problem appears when trajectory constraints are active may have to do with the fact that the optimization algorithm chooses control iterates that cause the trajectory to follow the

† On an extremal arc, the third variation is necessarily zero with respect to any perturbation in the null-space of the Hessian. Otherwise, u^* would not be a local solution.

constraint over the active region. It may be "easier" to accomplish this using an oscillatory control.

Numerical Method for Solving Singular Control problems.

We propose here a modification to the approximating problems that reduces the numerical difficulties associated with singular control problems. Our modification involves adding to the objective function a penalty on the variation of the control derivative. The only other method (see [116,120]) that has been proposed for solving singular optimal control problems involves adding the term $\varepsilon \|u\|_Q^2 \doteq \varepsilon \int_0^1 u(t)^T Q u(t)\, dt$, where Q is positive definite and ε is a positive scalar, to the cost function. With this additional term, the Hamiltonian becomes, with H being the Hamiltonian of the unmodified problem,

$$H_\varepsilon(x(t), p(t), u(t)) = H(x(t), p(t), u(t)) + \varepsilon u(t)^T Q u(t)\ ,\ \ t \in [0,1]\ . \tag{5.4a}$$

Hence, assuming that the minimum eigenvalue of $H_{uu}(x(\cdot), p(\cdot), u(\cdot))$ is bounded below, there exists an $\varepsilon > 0$ such that

$$H_{\varepsilon\, uu}(x(t), p(t), u(t)) = H_{uu}(x(t), p(t), u(t)) + \varepsilon Q > 0\ . \tag{5.4b}$$

This method has some numerical drawbacks as discussed in [121]. One difficulty is that adding the term $\varepsilon \|u\|_Q^2$ is a brute-force way to eliminate the singularity of H_{uu}. In order to get a reasonable solution with this approach, ε must be driven to zero. But this causes the problem to become singular again.

The method we propose for handling singular optimal control problems is motivated as a direct approach to preventing the erratic oscillations that can appear in numerical solutions. It has the property that, as the discretization level $N \to \infty$, the solutions of the approximating problems converge to solutions of the original problem. At the same time, our method does not cause the approximating problems to become increasingly singular as $N \to \infty$. For simplicity of presentation, we will only discuss single-input systems but the ideas are easily extended to multiple-input systems. Also, the treatment is developed for second order splines ($\rho = 2$). The application of these results to first order splines is taken by formally setting $\rho = 1$ in the given formulas. We do not consider higher order splines because *(a)* we do not recommend using higher-order splines for control and trajectory constrained problems in general, and *(b)* the smoothness of higher-order splines tends to automatically prevent spurious oscillations in the numerical solution.

The total variation of a function $u : [0,1] \to \mathbb{R}$ is defined as

$$Var_{[0,1]}(u) \doteq \sup_{N} \sum_{k=0}^{N-1} |u(t_{k+1}) - u(t_k)| \,, \tag{5.5a}$$

where, for each N, the supremum is taken over all sequence $\{ t_k \}_{k=0}^{N}$ such that $0 = t_0 < t_1 < \cdots < t_{N-1} < t_N = 1$. If $Var_{[0,1]}(u) < \infty$, then u is said to be of *bounded variation*. The space of all functions u of bounded variations is denoted by BV. Note that, if $u \in L^{(2)}_{\mathbf{t}_N}$, then

$$Var_{[0,1]}(u) = \sum_{k=1}^{N} |u(t_{k+1}) - u(t_k)| \,. \tag{5.5b}$$

In order to prove epi-converge of the approximating problems, constructed below, to the original problem, the space of controls will have to be restricted to those of bounded variation. Hence, we define the original problem as

P
$$\min_{\eta \in \mathbf{H}} \{ \psi_o(\eta) \mid \psi_c(\eta) \le 0 \} \,, \tag{5.6a}$$

where $\mathbf{H} \doteq \mathbb{R}^n \times \mathbf{U}$ and

$$\mathbf{U} \doteq \{ u \in BV \mid u(t) \in U \,,\ \forall\, t \in [0,1] \} \tag{5.6b}$$

with $U \subset B(0, \rho_{\max})$ a compact, convex set. We note that $BV \subset L^1_{\infty,2}[0,1]$, so this new definition is a restriction on the set of controls that we had previously used.

If u is of bounded variation, then its derivative $\dot{u}(\cdot)$ exists almost everywhere. Unfortunately, the fact that the derivative does not exist at every point prevents us from being able to compute the total variation of $\dot{u}$ for all $u \in BV$. However, for $u \in L^{(\rho)}_{\mathbf{t}_N}$ we can define the *piecewise derivative variation* of the control as

$$\dot{V}ar_{\mathbf{t}_N}(u) \doteq \sum_{k=1}^{N+\rho-3} |s_{k+1} - s_k| \,, \tag{5.7a}$$

where

$$s_k = \frac{u(t_k) - u(t_{k-1})}{\Delta_{N,k}} \,,\ \ k = 1, \ldots, N + \rho - 2 \,, \tag{5.7b}$$

where $\Delta_{N,k} \doteq t_{N,k+1} - t_{N,k}$. Note that for second order splines ($\rho = 2$), s_k is the slope of $u(t)$ on the interval $t \in [t_k, t_{k+1}]$ and $u(t_k) = \alpha_{k+1}$, where $\{ \alpha_k \}_{k=1}^{N+1}$ are the spline coefficients of u. So, in what follows, we will use the notation $\bar{u}_k \doteq \alpha_k = u(t_{k-1})$ for $k = 1, \ldots, N+1$. Note that this is different than our notation in Chapter 2 where we had $\bar{u}_k = (u_{k,1}, \ldots, u_{k,r})$ for $k = 0, \ldots, N-1$.

To deal with the non-differentiability of $\dot{V}ar_{\mathbf{t}_N}(\cdot)$ we define the modified approximating problems as

$$\mathbf{P}_{N,c} \qquad \min_{\eta = (\xi,u) \in \mathbf{H}_N^{(r)}} \{ \psi_{o,N}(\eta) + c_N \dot{V}ar^2_{\mathbf{t}_N} u \mid \psi_{c,N}(\eta) \le 0 \} , \tag{5.8a}$$

where

$$\boxed{\dot{V}ar^2_{\mathbf{t}_N}(u) \doteq \frac{1}{2} \sum_{k=1}^{N+\rho-3} |s_{k+1} - s_k|^2} \tag{5.8b}$$

$$\boxed{c_N = \frac{c}{(N+\rho-3)N^2}} \; . \tag{5.8c}$$

In (5.8c), $c \ge 0$ is a small number supplied by the user, $N + \rho - 3$ is the number of terms in the summation in (5.8b) and the N^2 roughly cancels the $\Delta^2_{N,k}$ terms in the definition of s_k^2 (if the mesh is uniform then $\Delta^2_{N,k} = 1/N^2$). The parameter c_N goes to zero as $N \to \infty$ fast enough to ensure that $c_N \dot{V}ar^2_{\mathbf{t}_N}(u)$ stays bounded as $N \to \infty$ (as long as the meshes are quasi-uniform). To wit, we have from (5.7b) and (5.8b) that for any $u \in \mathbf{U}_N^{(2)} \subset L^{(2)}_{\mathbf{t}_N}$,

$$\begin{aligned}
c_N \dot{V}ar^2_{\mathbf{t}_N}(u) &= \frac{c}{2(N+\rho-3)N^2} \sum_{k=1}^{N+\rho-3} \left| \frac{\bar{u}_{k+2} - \bar{u}_{k+1}}{\Delta_{N,k+1}} - \frac{\bar{u}_{k+1} - \bar{u}_k}{\Delta_{N,k}} \right|^2 \\
&\le \frac{c(\bar{\delta}N)^2}{2(N+\rho-3)N^2} \sum_{k=1}^{N+\rho-3} |\bar{u}_{k+2} - \bar{u}_{k+1}|^2 + |\bar{u}_{k+1} - \bar{u}_k|^2 \\
&\le \frac{c\bar{\delta}^2}{2(N+\rho-3)} \sum_{k=1}^{N+\rho-3} 8\rho^2_{\max} \\
&= 4c(\bar{\delta}\rho_{\max})^2 ,
\end{aligned} \tag{5.9}$$

where we have used the fact that *(i)* $|\bar{u}_{k+1} - \bar{u}_k| \le 2\rho_{\max}$ since, by definition of $\mathbf{U}_N^{(2)}$, $\mathbf{U}_N^{(2)} \subset \mathbf{U}$ where $\mathbf{U}$ is given by (5.6b), *(ii)* $(a-b)^2 \le 2(a^2 + b^2)$ and *(iii)* the meshes are quasi-uniform with quasi-uniformity ratio $\bar{\delta}$, which by (1.2b), implies that $1/\Delta_{N,k} \le \bar{\delta}N$ for all k. On the other hand, the fact that the bound in (5.9) is independent of N (in particular, does not go to zero as $N \to \infty$) indicates that c_N stays large enough so that the penalty term will be effective in damping out unwanted control oscillations even as $N \to \infty$. Thus, c_N is the correct order in $1/N$; any smaller and the penalty term would not damp out unwanted oscillations and any larger and we would not be able to prove epi-convergence as we do below. Typically, the fixed parameter c will be chosen very small so that the penalty term does not affect the solution away from singular arcs. Even so, erratic control behavior on singular sub-arcs will be damped because even a small penalty effects control perturbations on singular sub-arcs that would otherwise have very little effect on the Lagrangian.

Our first task is to show that the modified problems $\mathbf{P}_{N,c}$ epi-converge to the original problem $\mathbf{P}$. We emphasize that Theorem 5.3 relies on the fact that $\mathbf{P}$ is defined over the set of controls with bounded variation.

Theorem 5.3 (Epi-convergence). Suppose $\{ \mathbf{P}_N \}$ is a sequence of approximating problems defined on quasi-uniform meshes $\{ \mathbf{t}_N \}$ with quasi-uniformity ratio $\bar{\delta}$ using second order splines ($\rho = 2$) and that $\{ \mathbf{P}_N \}$ epi-converges to problem $\mathbf{P}$ as $N \to \infty$. Also suppose that $\{ \mathbf{P}_{N,c} \}$ is a sequence of modified approximating problems formed by adding to the objective functions of $\mathbf{P}_N$ the term $c_N \dot{V}ar^2(u)$. Then $\{ \mathbf{P}_{N,c} \}$ epi-converges to $\mathbf{P}$ as $N \to \infty$.

Proof. Since $\{ \mathbf{P}_N \}$ epi-converges to $\mathbf{P}$ we merely need to show that the extra term, $c_N \dot{V}ar^2_{\mathbf{t}_N}(u)$, in $\mathbf{P}_{N,c}$ epi-converges to zero. Because $c_N \dot{V}ar^2(u_N) \geq 0$ for all $u_N \in L^{(2)}_{\mathbf{t}_N}$, it is clear that for any sequence $\{ u_N \}$ with $u_N \in L^{(2)}_{\mathbf{t}_N}$ such that $u_N \to u \in BV$ as $N \to \infty$, $\underline{\lim}\, c_N \dot{V}ar^2(u_N) \geq 0$. Thus, part *(b)* of Definition 2.2.1 is satisfied. We will now show that part *(a)* of Definition 2.2.1 is also satisfied. Let $u \in BV$. For each N, define the piecewise constant function

$$v_N(t) = \sum_{k=1}^{N} \bar{v}_{N,k} \Pi_{N,k}(t) \ , \ \ t \in [0,1] \ , \tag{5.10a}$$

where

$$\bar{v}_{N,k} = \int_{t_{k-1}}^{t_k} u(t)dt \ , \tag{5.10b}$$

$$\Pi_{N,k}(t) \doteq \begin{cases} 1 & \text{if } t \in [t_{k-1}, t_k) \\ 0 & \text{otherwise,} \end{cases} \quad k = 1, \ldots N-1 \tag{5.10c}$$

$$\Pi(t)_{N,N} \doteq \begin{cases} 1 & \text{if } t \in [t_{N-1}, t_N] \\ 0 & \text{otherwise.} \end{cases} \tag{5.10d}$$

We then construct $u_N \in L^{(2)}_{\mathbf{t}_N}$ as $u_N = S^{-1}_{\mathbf{t}_N}(\alpha_N)$ where $\alpha_{N\,k} = \bar{v}_{N,k}$ for $k = 1, \ldots, N$. We see that

$$u_N \to u \ \text{ as } \ N \to \infty \tag{5.11a}$$

since the space of piecewise constant function is dense in $L_2[0,1] \supset L^1_{\infty,2}[0,1] \supset BV$. Also, for all N,

$$Var_{[0,1]}(u_N) = Var_{[0,1]}(v_N) \leq Var_{[0,1]}(u) \ , \tag{5.11b}$$

where the equality in (5.11b) follows immediately from the definition of $Var_{[0,1]}(\cdot)$ and the inequality follows from Lemma 3.2 in [122]. Next, with our notation $\bar{u}_{N\,k} \doteq u_N(t_{k-1})$ for $k = 1, \ldots, N+1$, we have from (5.8b,c),

$$c_N \dot{V}ar^2(u_N) = \frac{c}{2(N+1)N^2} \sum_{k=1}^{N-1} \left| \frac{\bar{u}_{N\,k+2} - \bar{u}_{N\,k+1}}{\Delta_{N\,k+1}} - \frac{\bar{u}_{N\,k+1} - \bar{u}_{N\,k}}{\Delta_{N\,k}} \right|^2$$

$$\leq \frac{c}{2(N+1)N^2}\, 2 \sum_{k=1}^{N-1} \left| \frac{\bar{u}_{N\,k+2} - \bar{u}_{N\,k+1}}{\Delta_{N,k+1}} \right|^2 + \left| \frac{\bar{u}_{N\,k+1} - \bar{u}_{N\,k}}{\Delta_{N,k}} \right|^2 ,$$

where we have used the fact that $(a-b)^2 \leq 2(a^2+b^2)$

$$\leq \frac{2c}{2(N+1)N^2}\, 2 \sum_{k=1}^{N} \left| \frac{\bar{u}_{N\,k+1} - \bar{u}_{N\,k}}{\Delta_{N,k+1}} \right|^2$$

$$\leq \frac{2c}{(N+1)N^2}\, (\bar{\delta} N)^2 \sum_{k=1}^{N} \left| \bar{u}_{N\,k+1} - \bar{u}_{N\,k} \right|^2 ,$$

where we have used the quasi-uniformity condition $\Delta_{N,k} \geq 1 / \bar{\delta} N$,

$$\leq \frac{2c\bar{\delta}^2}{N+1} \left(\sum_{k=1}^{N} \left| \bar{u}_{N\,k+1} - \bar{u}_{N\,k} \right| \right)^2$$

$$= \frac{2c\bar{\delta}^2}{N+1} \Big(Var_{[0,1]}(u_N) \Big)^2 , \tag{5.12}$$

since $u_N \in L^{(2)}_{\mathbf{t}_N}$. Therefore, from (5.11b) and (5.12),

$$c_N \dot{V}ar^2(u_N) \leq \frac{2c\bar{\delta}^2}{N+1} \Big(Var_{[0,1]}(u) \Big)^2 \to 0 \text{ as } N \to \infty \tag{5.13}$$

since $u \in BV$ implies that $Var_{[0,1]}(u) < \infty$. This shows that, $\overline{\lim}\, c_N \dot{V}ar^2(u_N) \leq 0$. Hence, Definition 2.2.1*(a)* holds. □

We can gain insight into how the penalty term $c_N \dot{V}ar^2_{\mathbf{t}_N}(u)$ damps out spurious control oscillations by considering its Hessian. Given $u \in L^{(2)}_{\mathbf{t}_N}$, let $\alpha = S_{\mathbf{t}_N,2}(u)$. If we define the row vector

$$s(\alpha) \doteq [\, (s_1 - s_0) \quad (s_2 - s_1) \quad \cdots \quad (s_{N+\rho-2} - s_{N+\rho-3}) \,] , \tag{5.14}$$

with s_k as defined in (5.17b), then

$$\dot{V}ar^2_{\mathbf{t}_N}(u) = \frac{1}{2}\, s(\alpha) s(\alpha)^T . \tag{5.15}$$

Further, if we define

$$D_N \doteq \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & & \\ & & & -1 & 1 \end{bmatrix}_{N \times N+1} , \tag{5.16a}$$

$$\Delta \doteq \begin{bmatrix} \Delta_{N,0} & & \\ & \ddots & \\ & & \Delta_{N,N-1} \end{bmatrix}, \tag{5.16b}$$

then we see that

$$\frac{ds(\alpha)}{d\alpha} = s_\alpha \doteq D_{N-1} \Delta^{-1} D_N \ , \tag{5.16c}$$

and

$$s(\alpha) = \alpha s_\alpha^T \ . \tag{5.16d}$$

Hence, $\dot{V}ar^2_{\mathbf{t}_N}(u) = \frac{1}{2}\, \alpha s_\alpha^T s_\alpha \alpha^T$ and its second derivative is given by

$$\frac{d^2 \dot{V}ar^2_{\mathbf{t}_N}(u)}{d\alpha^2} = s_\alpha^T s_\alpha \ . \tag{5.17}$$

Proposition 5.4. For $u \in L^{(2)}_{\mathbf{t}_N}$, the matrix $\dfrac{d^2 \dot{V}ar^2_{\mathbf{t}_N}(u)}{d\alpha^2}$ is symmetric, positive semi-definite. The null-space of $\dfrac{d^2 \dot{V}ar^2_{\mathbf{t}_N}}{d\alpha^2}$ is the two-dimensional subspace

$$\{ \ u \in L^{(2)}_{\mathbf{t}_N} \mid u(t) = a + bt \ , \ \ a, b \in \mathbb{R} \ \} \ . \tag{5.18a}$$

Proof. That $\dfrac{d^2 \dot{V}ar^2_{\mathbf{t}_N}}{d\alpha^2}$ is symmetric, positive semi-definite is obvious. Let $u \in L^{(2)}_{\mathbf{t}_N}$ have coefficients $\alpha = S_{\mathbf{t}_N,2}(u)$. Then, from (5.17) and (5.16d),

$$\langle \alpha \, \frac{d^2 \dot{V}ar^2_{\mathbf{t}_N}}{d\alpha^2} \, , \alpha \rangle = \langle \alpha s_\alpha^T, \alpha s_\alpha^T \rangle = \langle s(\alpha), s(\alpha) \rangle \ . \tag{5.18b}$$

The result now follows by noting from (5.7b) and (5.14) that $s(\alpha) = 0$ if and only if $u = a + bt$ for some $a, b \in \mathbb{R}$. □

Thus, $\dfrac{d^2 \dot{V}ar^2_{\mathbf{t}_N}(u)}{d\alpha^2}$ is positive-definite except on the subspace of controls that have constant slopes. To see how this affects the solutions of the approximating problem consider a singular control $\mathbf{P}$ with approximating problems $\mathbf{P}_N$ and modified approximating problems $\mathbf{P}_{N,c}$. First, let $\{ \ \hat{u}_N \ \}$ be a sequence of solutions to $\{ \ \mathbf{P}_N \ \}$ such that $\hat{u}_N \to u^*$ where u^* is a singular optimal control. For each N, let $\hat{\lambda}_N$ be the Lagrange multipliers associated with the constraints for $\mathbf{P}_N$ at the solution $\hat{u}_N$. If N is large enough, the Hessian of the Lagrangian, $\nabla^2_{uu} L_N(\hat{u}_N, \hat{\lambda}_N)$, for $\mathbf{P}_N$ is singular or near singular. Let $\delta u_N \in L^{(2)}_{\mathbf{t}_N}$ be a vector in the null-space of $\nabla^2_{uu} L_N(\hat{u}_N)$. Then, expanding the Lagrangian around $\hat{u}_N$, we see that

$$L(\hat{u}_N + \delta u_N, \hat{\lambda}_N) = L(\hat{u}_N, \hat{\lambda}_N) + O(\|\delta u_N\|^4) \tag{5.19a}$$

since the first and third order variations with respect to δu_N are zero for a solution to $\mathbf{P}_N$. Thus, small perturbations in the null-space of $\nabla^2_u L_N(\hat{u}_N, \hat{\lambda}_N)$ have very little affect on the Lagrangian for $\mathbf{P}_N$. Next, consider the modified approximating problem $\mathbf{P}_{N,c}$. Let $\{ \hat{u}_N \}$ be a sequence of solutions to $\{ \mathbf{P}_{N,c} \}$ such that $\hat{u}_N \to u^*$. As before, let $\hat{\lambda}_N$ be the Lagrange multipliers and let δu_N be a vector in the null-space of $\nabla^2_u L_N(\hat{u}_N, \hat{\lambda}_N)$. Now when we expand the Lagrangian for the modified approximating problem we find that

$$L(\hat{u}_N + \delta u_N, \hat{\lambda}_N) = L(\hat{u}_N, \hat{\lambda}_N) + \frac{1}{2}\langle \delta u_N, \frac{d^2 \dot{V}ar^2_{\mathbf{t}_N}}{d\alpha^2}\, \delta u_N \rangle + O(\|\delta u_N\|^4)\ . \tag{5.19b}$$

Thus, according to Proposition 5.4, any perturbation, δu_N, that is not a straight line, *i.e.* contains oscillations, will lead to a second-order increase in the value of the Lagrangian. It is this property that tends to damp out control oscillations that may occur on singular arcs of the numerical solutions.

Remark 5.5. Any control $u \in L^{(2)}_{\mathbf{t}_N}$ can be written as $u = u_1 + u_2$ where $u_2 = u(0) + u(1)t$. Proposition 5.4 shows that u_2 does not contribute to the penalty term in $\mathbf{P}_{N,c}$. This is precisely the behavior we want because any spurious oscillation in u must be contained in u_1. Had we chosen to penalize the variation of the control, $Var_{[0,1]}(u)$, rather than the piecewise derivative variation of the control, the null-space of the Hessian of the penalty term would consist of only constant functions. Thus, even u_2 would contribute to the penalty term. That would be undesirable. □

Remark 5.6. Another possibility for solving singular optimal control is to use the proximal point method. In this approach, the approximating problems are defined as

$$\mathbf{P}_{N,c_N} \qquad \min_{\eta = (\xi,u) \in \mathbf{H}^{(r)}_N} \{ \psi_{o,N}(\eta) + c_N \|u - u_{N-1}{}^*\|^2 \ \mid\ \psi_{c,N}(\eta) \le 0 \}\ , \tag{5.20}$$

where $u_{-1}{}^*$ is chosen by the user and $u_{N-1}{}^*$ is the solution obtained for $\mathbf{P}_{N-1,c_{N-1}}$. Clearly, the additional term $\|u_{N-1}{}^* - u\|^2$ adds a positive definite matrix to the Hessian of the Lagrangian for $\mathbf{P}_{N,c_N}$. This is similar to the idea of adding the term $\varepsilon\|u\|^2_Q$ to the objective function. However, $\|u_{N-1}{}^* - u_N{}^*\| \to 0$ automatically if $u_N{}^* \to u^*$, even if $c_N = c$ is fixed. In this way, some of the numerical difficulties associated with the former method may be overcome. Convergence results for the proximal point method can be found in [123,124]. One advantage we see to our modification over both of these methods is that it has a minimal impact on the solutions of the approximating problems since the value of c in the definition (5.8c) of c_N can be chosen to be very small. Thus, a good numerical solution can be obtained even if only a single $P_{N,c}$ is solved using a moderate value of N. In the other methods, N must be allowed to grow large. □

Numerical Examples. Below we present the results of solving two trajectory constrained singular control problems with and without the addition of a penalty on the piecewise derivative variation of the control. We used second order splines and the fourth order RK method $\mathbf{A}_4$ and solved the discretized problems using NPSOL [125]. The first problem is the Obstacle problem and the second problem is the Goddard Rocket maximum ascent problem with a trajectory constraint on the dynamic air pressure (see Appendix B).

The first problem has two trajectory constraints, $l_1(t, x(t), u(t) \leq 0$ and $l_2(t, x(t), u(t)) \leq 0$. Two versions of this problem were solved. One version includes these trajectory constraints directly as part of the optimization problem. The second version of this problem has these trajectory constraints replaced with the endpoint constraint

$$g(\eta) \doteq \int_0^1 \max \{ (0, l_1(t, x(t), u(t))^2 \} + \max \{ (0, l_2(t, x(t), u(t))^2 \} \, dt = 0 \,. \tag{5.21}$$

Clearly, the trajectory constraint is satisfied if and only if $g(\eta) = 0$. However, this is not an equivalent representation of the constraint because $\nabla g(\eta) = 0$ for any feasible η. Hence, $g(\eta)$ does not satisfy the usual constraint qualification. No penalty on the piecewise derivative variation of the control was needed when using (5.21).

The solution of the first problem at discretization level $N = 100$ and a uniform mesh is shown in the first three plots. The first plot shows the solution with and without a penalty of $c = 10^{-3}$ on the piecewise derivative variation of the control. The next plot shows the solution for this problem with the trajectory constraints replaced by the endpoint constraint. We did not use a penalty on the piecewise derivative variation of the control to produce this solution which does not exhibit control oscillation. The final plot shows a phase plot of the system trajectory and shows how the trajectory constraints (depicted as dotted lines) are avoided.

The table below lists how many iterations and how much CPU time (in seconds on a Sun SparcStation 20) were required to solve the two formulations of this problem. Also listed, is the maximum constraint violation for the continuous-time systems (computed using a variable step-size integration method with the tolerance set to 10^{-12}). For each solution, the difference in the computed objective functions was negligible.

Constraint Type	Variation Penalty c	Iterations	CPU time	Constraint Violation
trajectory	0	37	29.4	2.62e-8
trajectory	1e-3	16	14.8	2.47e-8
endpoint	0	154	7.2	3.00e-5

Table 5.1: Work required to solve the Obstacle problem. The last row is for the problem reformulated with an endpoint constraint.

The reason the solution for the first two versions take much longer than the problem with the trajectory constraint converted to an endpoint constraint (even though the latter version requires many more iterations to solve) is that, at each iteration, a gradient is computed at each point, t_k, of the trajectory constraint. Currently, this is done with adjoint equations which is inefficient for trajectory constraints. The optimization procedure was not able to converge (although the solution was reasonable) with the endpoint constraint formulation. This is because the endpoint constraint does not satisfy the standard constraint qualification.

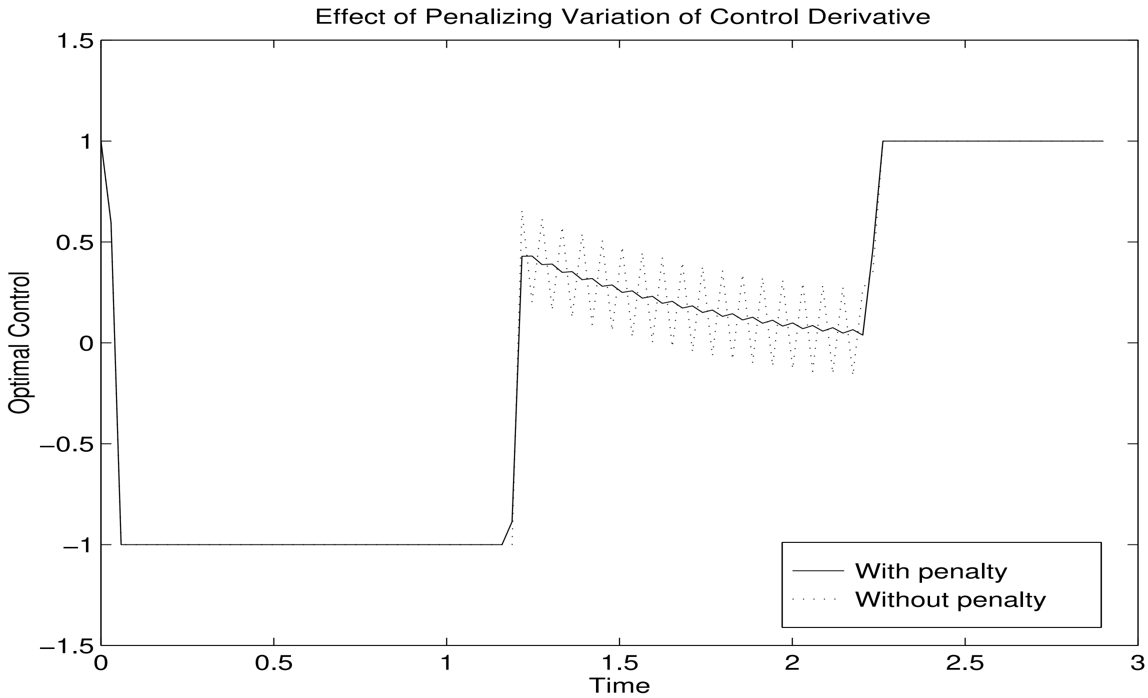


Fig. 5.2a: Solution of trajectory constrained, singular optimal control problem Obstacle. The oscillations are damped out by adding a small penalty on the piecewise derivative variation of the control to the objective function.

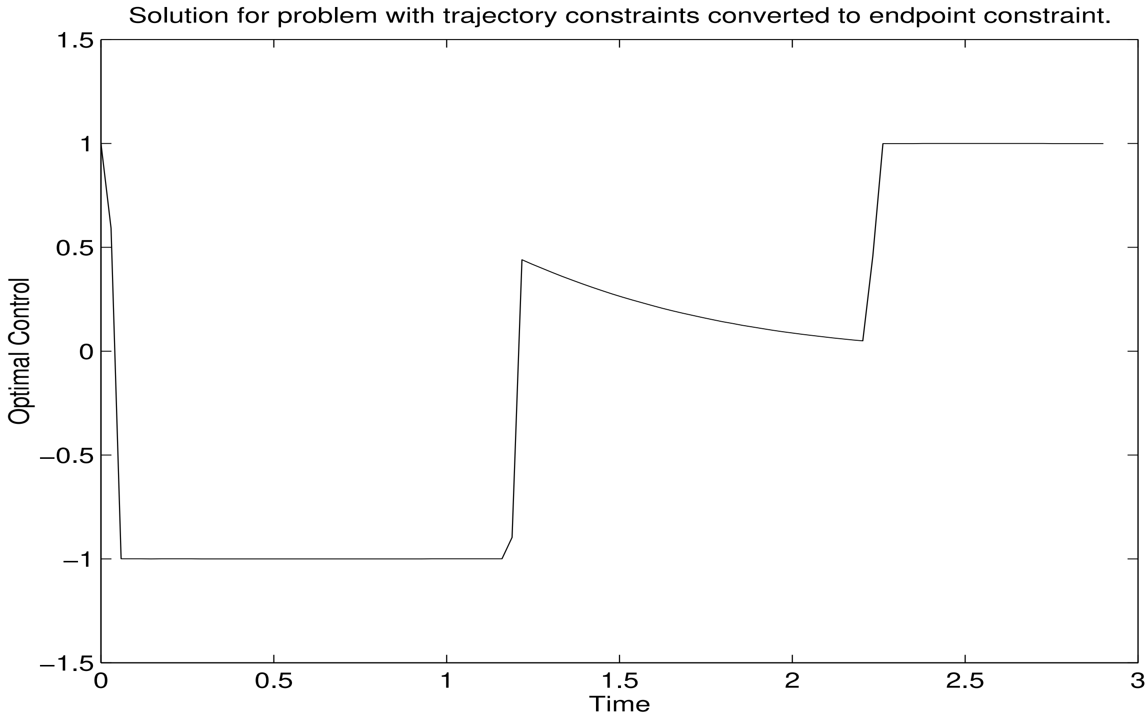


Fig. 5.2b: Solution of the same problem with the trajectory constraints converted into an endpoint constraint. Here there is no unwanted oscillation in the solution.

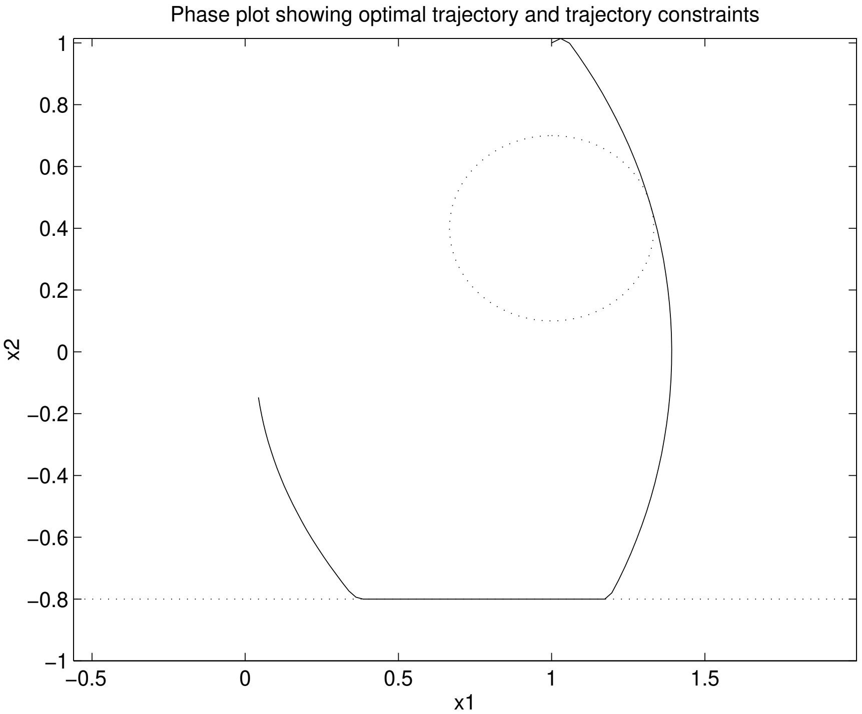


Fig. 5.2c: Phase plot of the optimal trajectory for problem Obstacle. The dotted lines represent the two trajectory constraints.

The next two plots show the solution for the Goddard rocket problem with the trajectory constraint $\max_k Aq(t_k) \le 10$ where $Aq(t)$ is the dynamic pressure given by

$$q(t) = \tfrac{1}{2}\,\rho_0 e^{\beta(1-r(t))} v^2(t) \tag{5.22}$$

with $A\rho_0 = 12,400$ and $\beta = 500$. We first solved this problem on a uniform grid with a discretization level of $N = 40$ using RK method $\mathbf{A}_4$ with second order splines. The solution exhibited oscillations. So, the mesh was redistributed using mesh redistribution Strategy 1 with $FAC = 50$. This produced a non-uniform grid with $N = 79$. We then solved this problem with and without a penalty on the piecewise derivative variation of control. This penalty was set at $c = 10^{-5}$. A plot of the control solutions for both cases is presented in Figure 5.4a. The plot includes marks on the time axis that show the locations of the mesh points. Figure 5.4b shows a plot of the dynamic pressure clearly indicating that the dynamic pressure constraint is satisfied. Table 5.3 shows how many iteration and how much CPU time (in seconds on a Sun SparcStation 20) was required to solve this problem with and without the penalty.

Variation Penalty	Iterations	CPU time
0	35	15.1
10^{-5}	20	9.2

Table 5.3: Work required to solve the trajectory constrained Goddard problem with and without a penalty on the piecewise derivative variation of the control. The difference in the objective values is negligible.

4.6 OTHER ISSUES

4.6.1 Fixed versus Variable step-size integration

The notion of using a fixed step-size integration routine to solve optimal control problems runs counter to the standard understanding of how to efficiently integrate differential equations. One could instead use variable step-size integration to approximately solve the differential equations and integrals in the statement of the optimal control problem. For optimal control problems, however, the overall error in the numerical solutions is not determined only by integration accuracy. Instead, as shown in equation (2.11a), the approximating capability of the finite-dimensional control representation also affects the solution error. Furthermore, many simulations will be required to solve an optimal control problem. So, having high integration accuracy in the early iterations will provide little, if any, benefit. The solution error can always be reduced in later iterations

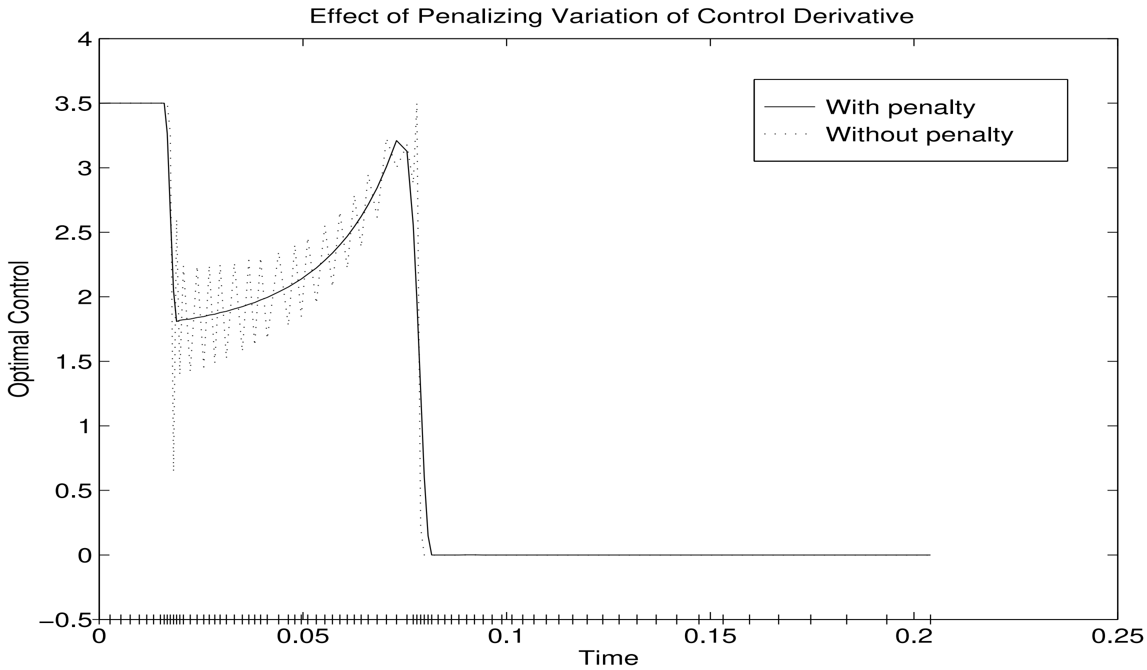


Fig. 5.4a: Solution of the trajectory constrained Goddard maximum ascent rocket problem showing how a small penalty on the piecewise derivative variation of the control can damp out spurious oscillations in the numerical solution. The tick marks indicate the locations of the mesh points.

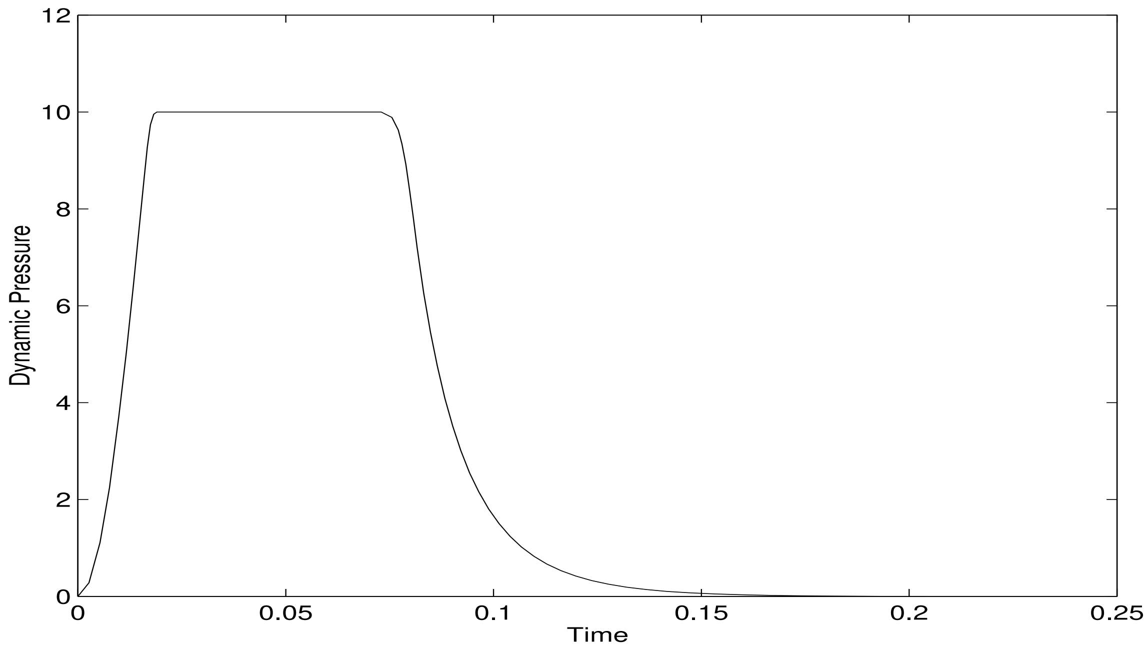


Fig. 5.4b: A plot of the dynamic pressure showing that the trajectory constraint $Aq(t) \leq 10$ is satisfied.

using the mesh refinement strategies discussed in Section 3.

On the other hand, the price of using a variable step-size algorithm is that they are much slower than fixed step-size algorithms. There are two main factors that contribute to this inefficiency. One problems is that the gradients computed for the objective and constraint functions cannot be computed exactly. Rather, they are computed as numerical approximations to the gradients of the continuous-time systems. This reduces the desirability of the search directions that depend on these gradients. Another problem occurs during line searches. Since the evaluation of function values depends on the integration mesh, part of the change in function values obtained from simulations with different step-lengths of the line search is due to the fact that the integration mesh changes from one simulation to the next when using a variable step-size method. This can cause line searches to fail if the integration tolerances are not tight enough.

In the following table we provide some experimental results that show that solutions obtained using a variable step-size integration routine[†] take much longer to compute and, yet, are no more accurate than the solutions obtained using the fourth order, fixed step-size Runge-Kutta integration method $\mathbf{A}_4$.

We used linear spline defined on a uniform mesh with $N = 100$ intervals for each of the problems. The problems were solved using NPSOL [125] (a sequential quadratic programming algorithm). Problems Switch and Goddard2 were solved with penalties of 10^{-3} and 10^{-4}, respectively, on the piecewise derivative variation of the control added to the objective function. Rayleigh 1 is the unconstrained Rayleigh problem and Rayleigh 2 includes the endpoint constraint $x_1(2.5) = 0$. Goddard 1 is the Goddard rocket maximum ascent problem, and Goddard 2 is the same problem with a trajectory constraint on the dynamic pressure, $Aq(t) \leq 10$, included. These problems are described in Appendix B.

The results are shown in Table 6.1. The CPU time is given in seconds on a Sun SparcStation 20. For the problems which took only a few iterations to solve, a substantial amount of the execution time was involved in computing the coordinate transformation. The optimization was terminated when all of the following termination criteria were met: *(i)*

$$\|Z^T \nabla_{\mathrm{FR}} f(\eta)\|_{H_2} \leq 10^{-6}(1 + \max\{\, 1 + |f(\eta)|, \|\nabla_{\mathrm{FR}} f(\eta)\|_{H_2} \,\})\,, \tag{6.1a}$$

where $\nabla_{\mathrm{FR}} f(\eta)$ is the gradient of the objective function with respect to free variables (variable not at their bounds) and $Z^T \nabla_{\mathrm{FR}} f(\eta)$ is the projection of the free gradient into the feasible region, *(ii)*

[†] We used LSODA which is a very efficient linear, multi-step Adams-Moulton method with variable step-size and variable order. The integration was reset at each mesh point because of discontinuities in $\dot{u}(t_k)$. Further details of the implementation can be found in the RIOTS user's manual.

$$\max_j |res_j| \le 10^{-4} \tag{6.1b}$$

where res_j is the violation of the j-th nonlinear constraint, and *(iii)*

$$\|\eta_{i+1} - \eta_i\|_{H_2} \le 10^{-6}(1 + \|\eta\|)_{H_2} \ . \tag{6.1c}$$

The relative and absolute integration tolerances for the variable step-size integration routine were both set to 10^{-8}. At lower tolerance the line searches often failed.

	RK method $\mathbf{A}_4$			Variable step-size integration		
Problem	Iterations	CPU time	$\|\eta^* - \eta^*_N\|$	Iterations	CPU time	$\|\eta^* - \eta^*_N\|$
LQR	5	1.0167	2.9087e-6	5	2.3500	3.0363e-6
Rayleigh 1	18	1.300	0.0014[2]	18	5.5333	0.0104[2]
Rayleigh 2	22	1.9833	0.0015[2]	22	10.4167	0.1402[2]
Bang	12	3.500	0.5471	12	9.1167	0.5471
Switch	1	2.500	0.0134	2	9.4833	0.0134
Obstacle	16	14.3000	0.0524[2]	15	282.0833	0.0524[2]
Goddard 1	103	4.2333	1.0062[2]	48[1]	22.8833[1]	
Goddard 2	27	24.4667	1.0789[2]	7[1]	90.7333[1]	

Table 6.1: Comparison of the amount of work required to solve problems using fixed step-size RK method $\mathbf{A}_4$ and a variable step-size integration method.

The superscript [1] indicates that the optimization algorithm failed to converge for the marked entry. The solutions returned in these two cases were unacceptably bad. The superscript [2] indicates that the quantity $\|\eta^* - \eta^*_N\|$ was estimated using (4.20).

We make the following observations from the data in Table 6.1:

- The execution time for the variable step-size method was about two to twenty times greater than the execution time for the fixed step-size RK method.
- The solution errors obtained with the variable step-size integration method was no better, and in some cases was worse, than the solution errors obtained with the fixed step-size RK method. The reason for this is that the gradients computed with the variable step-size routine are computed as approximations to the continuous-time gradients for the original problem. They are not exact gradients for the approximating problem. This prevents the optimization algorithm from being able to obtain very accurate solutions to the approximating problems.
- The optimization algorithm failed to converge for the Goddard Rocket problems when using the variable step-size integration routine.

These observations strongly suggest that, without a specific reason to the contrary, it is better to use a fixed step-size RK method than a variable step-size integration method. Cases where this may not be true arise when the differential equations describing the system dynamics are stiff or very difficult to numerically integrate.

4.6.2 Problems with equality constraints and constraints that do not satisfy the Slater Condition.

Although we did not consider equality constraints in our convergence theory in Chapter 2, our discretization scheme can still be formally applied to optimal control problems with state end-point equality constraints. If any constraints, equality or inequality, in **P** do not satisfy the Slater conditions, Assumption 2.4.11, then the approximating problems $\mathbf{P}_N$ may have no solution even though there is a solution to the original problem **P** (see [44,51,126]). However, from a practical point of view, the outcome is that the algorithm which is used to solve $\mathbf{P}_N$ will simply terminate without being able to satisfy the constraints beyond a certain accuracy. It is, therefore, a good idea to use large constraint violation tolerances when the discretization level is low to avoid any unnecessary iterations that occur trying to satisfy the constraints. The achievable accuracy will, however, increase as the discretization level N increases.

Chapter 5

USER'S MANUAL FOR RIOTS

1. INTRODUCTION

This chapter describes the implementation of a Matlab† toolbox called RIOTS for solving optimal control problems. The name RIOTS stands for "Recursive‡ Integration Optimal Trajectory Solver." This name highlights the fact that the function values and gradients needed to find the optimal solutions are computed by forward and backward integration of certain differential equations.

RIOTS is a collection of programs that are callable from the mathematical simulation program Matlab. Most of these programs are written in either C (and linked into Matlab using Matlab's MEX facility) or Matlab's M-script language. All of Matlab's functionality, including command line execution and data entry and data plotting, are available to the user. The following is a list of some of the main features of RIOTS.

- Solves a very large class of finite-time optimal controls problems that includes: trajectory and endpoint constraints, control bounds, variable initial conditions (free final time problems), and problems with integral and/or endpoint cost functions.
- System functions can be supplied by the user as either object code or M-files.
- System dynamics can be integrated with fixed step-size Runge-Kutta integration, a discrete-time solver or a variable step-size method. The software automatically computes gradients for all functions with respect to the controls and any free initial conditions. These gradients are computed exactly for the fixed step-size routines.
- The controls are represented as splines. This allows for a high degree of function approximation accuracy without requiring a large number of control parameters.

---

† Matlab is a registered trademark of Mathworks, Inc. Matlab version 4.2c with the Spline toolbox is required.
‡ Iterative is more accurate but would not lead to a nice acronym.

- The optimization routines use a coordinate transformation that creates an orthonormal basis for the spline subspace of controls. The use of an orthogonal basis can results in a significant reduction in the number of iterations required to solve a problem and an increase in the solution accuracy. It also makes the termination tests independent of the discretization level.
- There are three main optimization routines, each suited for different levels of generality of the optimal control problem. The most general is based on sequential quadratic programming methods. The most restrictive, but most efficient for large discretization levels, is based on the projected descent method. A third algorithm uses the projected descent method in conjunction with an augmented Lagrangian formulation.
- There are programs that provide estimates of the integration error for the fixed step-size Runge-Kutta methods and estimates of the error of the numerically obtained optimal control.
- The main optimization routine includes a special feature for dealing with singular optimal control problems.
- The algorithms are all founded on rigorous convergence theory.

In addition to being able to accurately and efficiently solve a broad class of optimal control problems, RIOTS is designed in a modular, toolbox fashion that allows the user to experiment with the optimal control algorithms and construct new algorithms. The programs **outer** and **aug_lagrng**, described later, are examples of this toolbox approach to constructing algorithms.

RIOTS is a collection of several different programs (including a program which is, itself, called **riots**) that fall into roughly three categories: integration/simulation routines, optimization routines, and utility programs. Of these programs, the ones available to the user are listed in the following table,

Simulation Routines	Optimization Routines	Utility Programs
simulate	**riots**	**control_error**
check_deriv	**pdmin**	**distribute**
check_grad	**aug_lagrng**	**est_error**
eval_fnc	**outer**	**make_spline**
		transform

Several of the programs in RIOTS require functions that are available in the Matlab Spline toolbox. In addition to these programs, the user must also supply a set of routines that describe the optimal control problem which must be solved. Several example optimal control problems come supplied with RIOTS. Finally, there is a Matlab script called **RIOTS_demo** which provides a demonstration of some of the main features of RIOTS. To use the demonstration, perform the following steps:

Step 1: Follow the directions in §8 on compiling and linking RIOTS. Also, compile the sample systems rayleigh.c, bang.c and goddard.c that come supplied with RIOTS.

Step 2: Start Matlab from within the 'RIOTS/systems' directory.

Step 3: Add the RIOTS directory to Matlab's path by typing at the Matlab prompt,

```
>> path(path,'full_path_name_for_RIOTS')
>> RIOTS_demo
```

Limitations. This is the first version of RIOTS. As it stands, there are a few significant limitations on the type of problems which can be solved by RIOTS:

1. Problems with inequality state constraints that require a very high level of discretization cannot be solved by RIOTS. Also, the computation of gradients for trajectory constraints is not handled as efficiently as it could be.
2. Problems that have highly unstable, nonlinear dynamics may require a very good initial guess for the solution in order to be solved by RIOTS.
3. General constraints on the controls that do not involve state variables are not handled efficiently: adjoints are computed but not used.
4. RIOTS does not allow delays in the systems dynamics (although Padé approximations can be used).
5. Numerical methods for solving optimal control problems have not reached the stage that, say, methods for solving differential equations have reached. Solving an optimal control problem can, depending on the difficulty of the problem, require significant user involvement in the solution process. This sometimes requires the user to understand the theory of optimal control, optimization and/or numerical approximation methods.

Conventions. This manual assumes familiarity with Matlab. The following conventions are used throughout this manual.

- Program names and computer commands are indicated in **bold** typeface.
- User input is indicated in `Courier` typeface.
- Optional program arguments are listed in brackets. The default value for any optional argument can be specified using `[]`.
- Optional program arguments at the end of an argument list can be omitted in which case

these arguments take on their default values.

- Typing a function's name without arguments shows the calling syntax for that function. Help can be obtained for M-file programs by typing `help` followed by the function name at Matlab's prompt. Typing `help RIOTS` produces a list of the programs in RIOTS.
- The machine precision is denoted by $\varepsilon_{\mathrm{mach}}$.

2. PROBLEM DESCRIPTION

RIOTS is designed to solve optimal control problems of the form[†]

$$\textbf{OCP} \qquad \underset{(u,\xi)\,\in\, L^m_{\infty,2}[a,b]\times\mathbb{R}^n}{\text{minimize}} \left\{ \max_{\nu\,\in\,\mathbf{q}_o} \left\{ f^\nu(u,\xi) \doteq g^\nu_o(\xi, x(b)) + \int_a^b l^\nu_o(t,x,u)dt \right\} \right\}$$

$$\text{subject to:} \quad \dot{x} = h(t,x,u)\ ,\ \ x(a) = \xi\ ,\ t \in [a,b]\ ,$$

$$u^j_{\min}(t) \le u^j(t) \le u^j_{\max}(t)\ ,\ \ j = 1,\dots,m\ ,\ \ t \in [a,b]\ ,$$

$$\xi^j_{\min} \le \xi^j \le \xi^j_{\max}\ ,\ \ j = 1,\dots,n\ ,$$

$$l^\nu_{ti}(t, x(t), u(t)) \le 0\ ,\ \ \nu \in \mathbf{q}_{ti}\ ,\ t \in [a,b]\ ,$$

$$g^\nu_{ei}(\xi, x(b)) \le 0\ ,\ \ \nu \in \mathbf{q}_{ei}\ ,$$

$$g^\nu_{ee}(\xi, x(b)) = 0\ ,\ \ \nu \in \mathbf{q}_{ee}\ ,$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, $g : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $l : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $h : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and we have used the notation $\mathbf{q} \doteq \{ 1, \dots, q \}$. Only with the optimization program **riots** linked with CFSQP (see description of **riots**) can $q_o > 1$. The functions in **OCP** can also depend upon parameters which are passed from Matlab at execution time using **get_flags** (described in §4).

The subscripts o, ti, ei, and ee on the functions $g(\cdot,\cdot)$ and $l(\cdot,\cdot,\cdot)$ stand for, respectively, "objective function", "trajectory constraint", "endpoint inequality constraint" and "endpoint equality constraint". The subscripts for $g(\cdot,\cdot)$ and $l(\cdot,\cdot,\cdot)$ are omitted when all functions are being considered without regard to the subscript. The functions in the description of problem

[†] Not all of the optimization routines in RIOTS can handle the full generality of problem **OCP**.

OCP, and the derivatives of these functions[‡], must be supplied by the user as either object code or as M-files. The bounds on the components of ξ and u are specified on the Matlab command line.

The optimal control problem **OCP** allows optimization over both the control u and one or more of the initial states ξ. To be concise, we will define the variable

$$\eta = (u, \xi) \in H_2 \doteq L^m_{\infty,2}[a, b] \times \mathbb{R}^n \ .$$

Note that the order of u and ξ is reversed of the order used in the rest of this thesis because of programming considerations. With this notation, we can write, for example, $f(\eta)$ instead of $f(\xi, u)$. The inner product on H_2 is given by

$$\langle \eta_1, \eta_2 \rangle_{H_2} \doteq \langle u_1, u_2 \rangle_{L_2} + \langle \xi_1, \xi_2 \rangle \ .$$

The norm corresponding to this inner product is given by

$$\| \eta \|_{H_2} = \langle \eta, \eta \rangle^{1/2}_{H_2} \ .$$

Transcription for Free Final Time Problems.

Problem **OCP** is a fixed final time optimal control problem. However, free final time problems are easily incorporated into the form of **OCP** by augmenting the system dynamics with two additional states (one additional state for autonomous problems). The idea is to specify a nominal time interval, $[a, b]$, for the problem and to use a scale factor, adjustable by the optimization procedure, to scale the system dynamics and hence, in effect, scale the duration of the time interval. This scale factor, and the scaled time, are represented by the extra states. Then RIOTS can minimize over the initial value of the extra states to adjust the scaling. For example, the free final time optimal control problem

$$\min_{u,T} \ \tilde{g}(T, y(T)) + \int_a^{a+T} \tilde{l}(t, y, u) dt$$

$$\text{subject to } \dot{y} = \tilde{h}(t, y, u) \ , \ y(a) = \zeta \ , \ t \in [a, a+T] \ ,$$

can, with an augmented state vector $x \doteq (y, x^{n-1}, x^n)$, be converted into the equivalent fixed final time optimal control problem

[‡]If the user does not supply derivatives, the problem can still be solved using **riots** with finite-difference computation of the gradients.

$$\min_{u,\xi^n} \; g(\xi, x(b)) + \int_a^b l(t,x,u)dt$$

$$\text{subject to } \dot{x} = h(t,x,u) \doteq \begin{pmatrix} x^n \tilde{h}(x^{n-1}, y, u) \\ x^n \\ 0 \end{pmatrix}, \; x(a) = \xi = \begin{pmatrix} \zeta \\ a \\ \xi^n \end{pmatrix}, \; t \in [a,b] \,,$$

where y is the first $n-2$ components of x, $g(\xi, x(b)) \doteq \tilde{g}(a + T\xi^n, y(b))$, $l(t,x,u) \doteq \tilde{l}(x^{n-1}, y, u)$ and $b = a + T$. Endpoint and trajectory constraints can be handled in the same way. The quantity $T = b - a$ is the nominal duration of the trajectories. In this transcription, x^{n-1} plays the role of time and ξ^n is the *duration scale factor*, so named because $T\xi^n$ is the effective duration of the trajectories for the scaled dynamics. Thus, for any $t \in [a,b]$, $x^n(t) = \xi^n$, $x^{n-1}(t) = a + (t-a)\xi^n$ and the solution, t_f, for the final time is $t_f = x^{n-1}(b) = a + (b-a)\xi^n$. Thus, the optimal duration is $T^* = t_f - a = (b-a)\xi^n = T\xi^n$. If $a = 0$ and $b = 1$, then $t_f = T^* = \xi^n$. The main disadvantage to this transcription is that it converts linear systems into nonlinear systems.

For autonomous systems, the extra variable x^{n-1} is not needed. Note that, it is possible, even for non-autonomous systems, to transcribe minimum time problems into the form of **OCP** using only one extra state variable. However, this would require functions like $h(t,x,u) = \tilde{h}(tx^n, y, u)$. Since RIOTS does not expect the user to supply derivatives with respect to the t argument it can not properly compute derivatives for such functions. Hence, in the current implementation of RIOTS, the extra variable x^{n-1} is needed when transcribing non-autonomous, free final time problems.

Trajectory constraints.

The definition of problem **OCP** allows trajectory constraints of the form $l_{ti}(t,x,u) \le 0$ to be handled directly. However, constraints of this form are quite burdensome computationally. This is mainly due to the fact that a separate gradient calculation must be performed for each point at which the trajectory constraint is evaluated.

At the expense of increased constraint violation, reduced solution accuracy and an increase in the number of iterations required to obtain solutions, trajectory constraints can be converted into endpoint constraints which are computationally much easier to handle. This is accomplished as follows. The system is augmented with an extra state variable x^{n+1} with

$$\dot{x}^{n+1}(t) = \mu \max \{\, 0, l_{ti}(t, x(t), u(t)) \,\}^2 \,, \;\; x^{n+1}(a) = 0 \,,$$

where $\mu > 0$ is a positive scalar. The right-hand side is squared so that it is differentiable with respect to x and u. Then it is clear that either of the endpoint constraints

$$g_{ei}(\xi, x(b)) \doteq x^{n+1}(b) \le 0$$

or

$$g_{ee}(\xi, x(b)) \doteq x^{n+1}(b) = 0$$

is satisfied if and only if the original trajectory constraint is satisfied. In practice, the accuracy to which **OCP** can be solved with these endpoint constraints is quite limited because these endpoint constraints do not satisfy the standard constraint qualification (described in the next section). This difficulty can be circumvented by eliminating the constraints altogether and, instead, adding to the objective function the penalty term $g_o(\xi, x(b)) \doteq x^{n+1}(b)$ where now μ serves as a penalty parameter. However, in this approach, μ must now be a large positive number and this will adversely affect the conditioning of the problem. Each of these possibilities is implemented in 'obstacle.c' for problem Obstacle (see Appendix B).

Continuum Objective Functions.

Objective functions of the form

$$\min_{u} \max_{t \in [a,b]} l(t, x(t), u(t))$$

can be converted into the form of problem **OCP** by augmenting the state vector with an additional state, w, such that

$$\dot{w} = 0 \ ; \quad w(0) = \xi^{n+1}$$

and forming the equivalent, trajectory constrained problem

$$\min_{(u, \xi^{n+1})} \xi^{n+1}$$

subject to

$$l(t, x(t), u(t)) - \xi^{n+1} \le 0 \ , \ \ t \in [a, b] \ .$$

This transcription also works for standard min-max objective functions (which are only supported for problem **OCP** when **riots** is linked with CFSQP) of the form

$$\min_{u} \max_{\nu \in \mathbf{q}_o} g^{\nu}(u, \xi) + \int_a^b l^{\nu}(t, x(t), u(t)dt \ .$$

In this case, an equivalent endpoint constrained problem with a single objective function,

$$\min_{u, \xi^{n+1}} \xi^{n+1}$$

subject to

$$\tilde{g}^{\,\nu}(u,\xi) - \xi^{n+1} \le 0\ ,\ \ \nu \in \mathbf{q}_o$$

is formed by using the augmented state vector (x, w, z) wtih

$$\dot{w} = 0\ ,\ \ w(0) = \xi^{n+1}$$

$$\dot{z}^{\nu} = l^{\nu}(t, x(t), u(t))\ ,\ \ z^{\nu}(0) = 0\ ,\ \ \nu \in \mathbf{q}_o\ ,$$

and defining

$$\tilde{g}^{\,\nu}(u,\xi) \doteq g^{\nu}(u,\xi) + z^{\nu}(b)\ .$$

3. USING RIOTS

This section provides some examples of how to simulate systems and solve optimal control problems with the RIOTS toolbox. Detailed descriptions of all required user-functions, simulation routines, optimization programs and utility programs are given in subsequent sections. These programs are all callable from within Matlab once Matlab's path is set to include the directory containing RIOTS. The Matlab command

```
>> path(path,'full_path_name_for_RIOTS')
>> RIOTS_demo
```

should be used for this purpose. Refer to the §8, ‘‘Compiling and Linking RIOTS’’, for details on how to install RIOTS.

RIOTS provides approximate solutions of continuous time optimal control problems by solving discretized ‘‘approximating’’ problems. These approximating problems are obtained by *(i)* numerically integrating the continuous time system dynamics with one of four Runge-Kutta integration methods† and *(ii)* restricting the space of allowable controls to finite-dimensional subspaces of splines. In this way, the approximating problems can by solved using standard mathematical programming techniques to optimize over the spline coefficients and any free initial conditions. It is not important for the user of RIOTS to understand the discretization procedure or splines.

The accuracy of the solutions obtained in this manner depends on several factors which include:

†RIOTS also includes a discrete-time system solver and a variable step-size integration routine.

(1) The accuracy of the integration scheme (which depends on the order of the integration scheme and the selection of the integration mesh).

(2) How well elements of the spline subspace can approximate solutions of the original, infinite-dimensional problem (this depends on the order and knot sequence of the splines and on the smoothness of the optimal control).

(3) How accurately the approximating problems are solved by the underlying mathematical programming algorithm.

The allowable spline orders are related to the particular integration method used (see description of **simulate** in §5). For problems that have smooth optimal controls, higher order splines will provide solutions with higher accuracy. Smoothness is not, however, typical of optimal controls for problems with control and/or trajectory constraints. In general, the spline knot sequence is constructed from the integration mesh

$$\mathbf{t}_N \doteq \{ t_k \}_{k=1}^{N+1} .$$

We start our indexing from $k = 1$ rather than $k = 0$, as we did in previous chapters, because Matlab's indexing begins with one. This integration mesh also represents the breakpoints for the control splines. The subscript N, referred to as the discretization level, indicates that there are N integration steps and $N + 1$ spline breakpoints. Each spline is determined from the knot sequence and its coefficients. For a spline of order ρ, each control input requires $N + \rho - 1$ coefficients and these coefficients are stored as *row* vectors. Thus, a system with m inputs will be stored in a "short-fat" matrix with m rows and $N + \rho - 1$ columns. More details about splines are given in the next section.

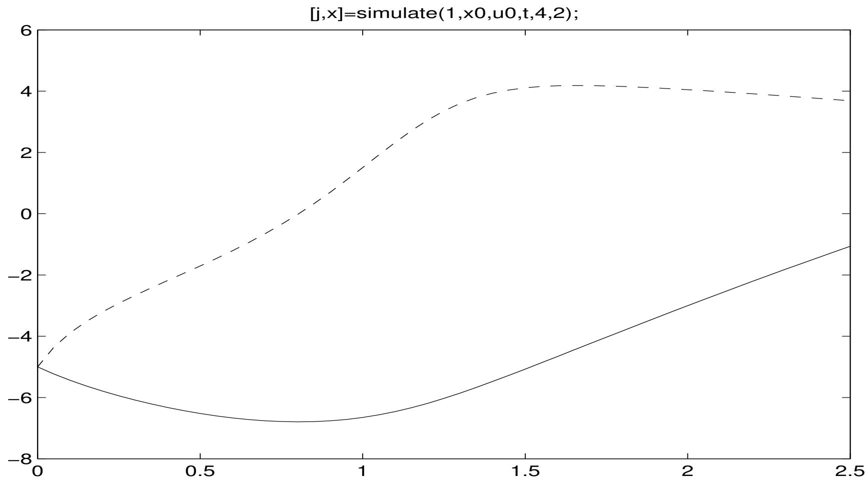
Typically, we use the Matlab variable `u` to store the spline coefficients. The system trajectories computed by integrating the system dynamics are stored in the variable `x`. Like `u`, `x` is a "short-fat" matrix with n rows and $N + 1$ columns. Thus, for example, `x(:,k)` is the computed value of $x(t_k)$. Other quantities, such as gradients and adjoints, are also stored as "short-fat" matrices.

The following sample sessions with RIOTS solve a few of the sample optimal control problems that are supplied with RIOTS as examples. Appendix B provides a description of these problems and the C-code implementations are included in the 'RIOTS/systems' sub-directory.

Session 1 (unconstrained problem). In this session we compute a solution to the unconstrained nonlinear Problem Rayleigh. This system has two states and one input. We start by defining the initial conditions and a uniform integration mesh over the time interval $[0, 2.5]$ with with a discretization level of $N = 50$ intervals.

We can take a look at the solution trajectories by simulating this system with some initial control. We will specify an arbitrary piecewise linear (order $\rho = 2$) spline by using $N + \rho - 1 = N + 1$ coefficients and perform a simulation by calling **simulate**.

```
>> N=50;
>> x0=[-5;-5];                              % Initial conditions
>> t=[0:2.5/50:2.5];                        % Uniform integration mesh
>> u0=zeros(1,N+1);                         % Spline with all coeff's zero.
>> [j,x]=simulate(1,x0,u0,t,4,2);
>> plot(t,x)
```



Next, we find an approximate solution to the Problem Rayleigh, which will be the same type of spline as `u0`, by using either **riots** or **pdmin**.

```
>> [u1,x1,f1]=riots(x0,u0,t,[],[],[],100,4);
>> [u1,x1,f1]=pdmin(x0,u0,t,[],[],[],100,4);
```

The first three input arguments are the initial conditions, initial guess for the optimal control, and the integration mesh. The next three inputs are empty brackets indicating default values which, in this case, specify that there are no control lower bounds, no control upper bounds, and no systems parameters. The last two inputs specify that a maximum of 100 iterations are to be allowed and that integration routine 4 (which is a fourth order Runge-Kutta method) should be used. The outputs are the control solution, the trajectory solution, and the value of the objective function.

The displayed output for **pdmin** is shown below. The displayed output for **riots** depends on the mathematical programming algorithm with which it is linked (see description of **riots** in §6).

```
This is a nonlinear system with 2 states, 1 inputs and 0 parameters,
     1 objective function,
     0 nonlinear and 0 linear trajectory constraints,
     0 nonlinear and 0 linear endpoint inequality constraints,
     0 nonlinear and 0 linear endpoint equality constraints.
Initial Scale factor = 0.02937
Method = L-BFGS.
Quadratic fitting off.
Completed 1    pdmin iter ; step = +1.67e+00 (k= -1), ||free_grad|| = 1.47e-01, FFF, cost = 34.40807327949193
Completed 2    pdmin iters; step = +4.63e+00 (k= -3), ||free_grad|| = 1.01e-01, FFF, cost = 31.33402612711411
Completed 3    pdmin iters; step = +2.78e+00 (k= -2), ||free_grad|| = 5.26e-02, FFF, cost = 29.78609937166251
Completed 4    pdmin iters; step = +1.67e+00 (k= -1), ||free_grad|| = 2.25e-02, FFF, cost = 29.30022802876513
Completed 5    pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 9.03e-03, FFF, cost = 29.22362561134763
Completed 6    pdmin iters; step = +1.67e+00 (k= -1), ||free_grad|| = 2.61e-03, FFF, cost = 29.20263210973429
Completed 7    pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 5.06e-04, FFF, cost = 29.20066785222028
Completed 8    pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 1.80e-04, FFF, cost = 29.20060360626269
Completed 9    pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 1.86e-05, FFF, cost = 29.20059986273411
Completed 10   pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 5.94e-06, FFF, cost = 29.20059981048738
Completed 11   pdmin iters; step = +1.67e+00 (k= -1), ||free_grad|| = 2.07e-06, FFF, cost = 29.20059980021174
Completed 12   pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 1.57e-07, FFF, cost = 29.20059979946436
Completed 13   pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 5.18e-08, FFF, cost = 29.20059979945842
Completed 14   pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 1.16e-08, FFF, cost = 29.20059979945757
Completed 15   pdmin iters; step = +1.00e+00 (k= +0), ||free_grad|| = 3.20e-10, TTF, cost = 29.20059979945753
Completed 16   pdmin iters; step = +6.00e-01 (k= +1), ||free_grad|| = 1.66e-10, TTT, cost = 29.20059979945752

Finished pdmin loop on the 16-th iteration.
Normal termination test satisfied.
```

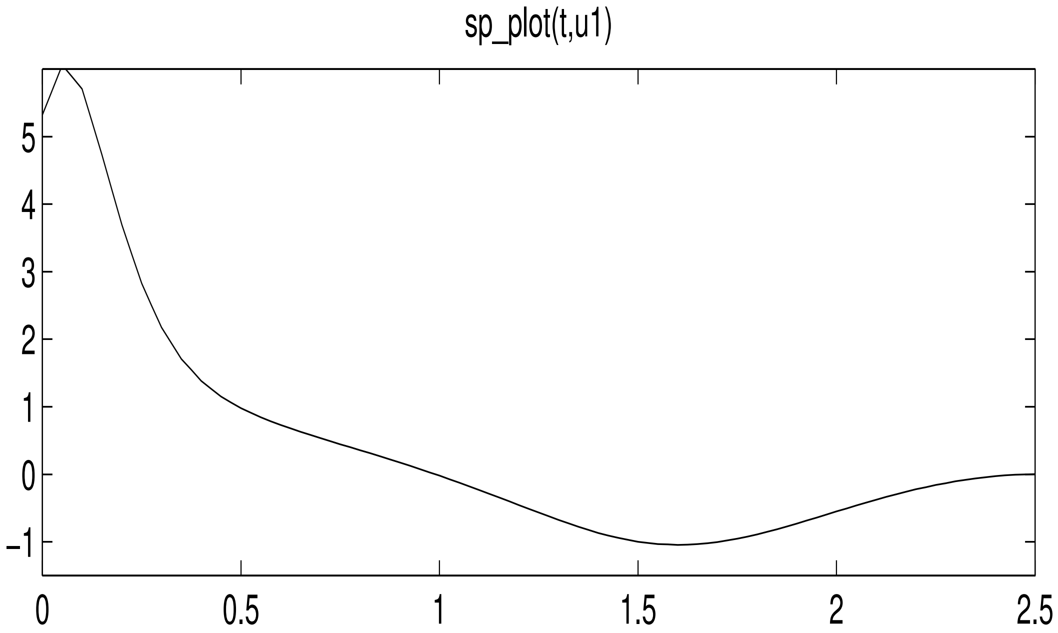
The column labeled `||free_grad||` gives the value of $\|\nabla f(\eta)\|_{H_2}$. For problems with bounds on the free initial conditions and/or controls, this norm is restricted to the subspace where the bounds are not active. The column with three letters, each a `T` or `F`, indicates which of the three normal termination criterion (see description of **pdmin** in §6) are satisfied. For problems with control or initial condition bounds there are four termination criteria.

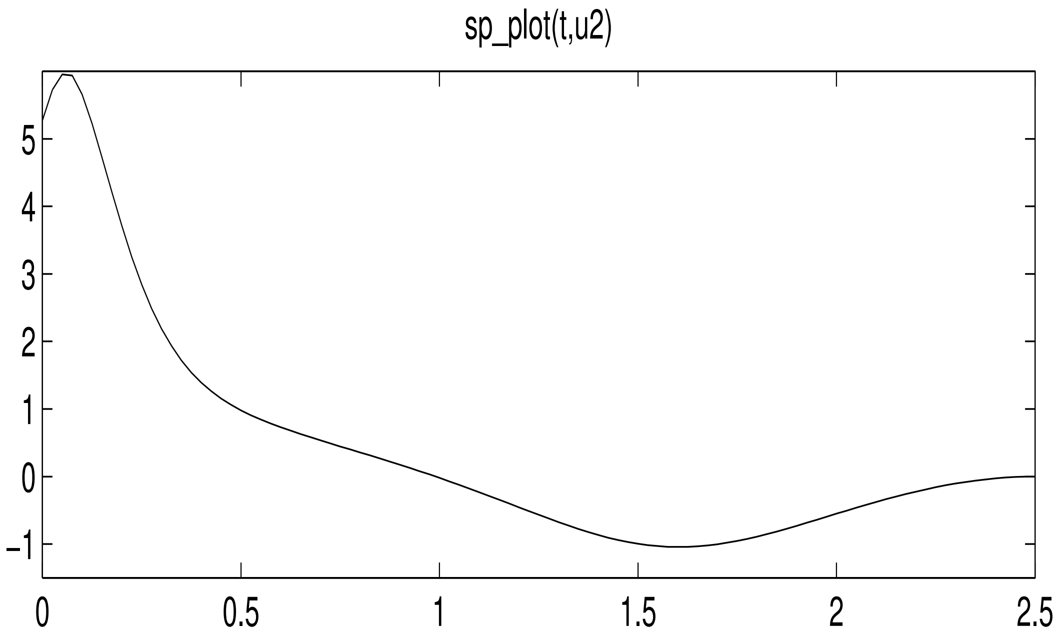
We can also solve this problem with quadratic splines ($\rho = 3$) by using $N + \rho - 1 = N + 2$ spline coefficients.

```
>> u0=zeros(1,N+2);
>> [u2,x2,f2]=pdmin(x0,u0,t,[],[],[],100,4);
```

We can view the control solutions using **sp_plot** which plots spline functions. The trajectory solutions can be viewed using **plot** or **sp_plot**.

```
>> sp_plot(t,u1)                    % Plot linear spline solution
>> sp_plot(t,u2)                    % Plot quadratic spline solution
```





Session 2 (problem with endpoint constraint). The user-defined functions for Problem Rayleigh, solved in session 1, are written so that it will include the endpoint constraint $x_1(2.5) = 0$ if there is a global Matlab variable called FLAGS set to the value of 1 (see **get_flags** in §4). To solve this problem with the endpoint constraint we can use either **riots** or **aug_lagrng**. We must clear **simulate** before re-solving so that the variable FLAGS gets read.

```
>> global FLAGS
>> FLAGS = 1;
>> clear simulate             % Reset simulate so the it will check for FLAGS
>> simulate(0,[]);            % Initialize
Loaded 1 flag.
Rayleigh
This is a nonlinear system with 2 states, 1 inputs and 0 parameters,
     1 objective function,
     0 nonlinear and 0 linear trajectory constraints,
     0 nonlinear and 0 linear endpoint inequality constraints,
     0 nonlinear and 1 linear endpoint equality constraints.
```

The output displayed above shows that one flag has been read from the Matlab workspace. The next two lines are messages produced by the user-supplied routines. The last set of data shows the value of the system information (see discussion of neq[] in the description of **init**, §4, and also **simulate**, §5). Since this problem has a state constraint, we can use either **aug_lagrng** or **riots** to solve it.

```
>> x0=[-5;-5];
>> u0=zeros(1,51);
>> t=[0:2.5/50:2.5];
>> u=aug_lagrng(x0,u0,t,[],[],[],100,5,4);

Finished pdmin loop on the 2-nd iteration.
Step size too small.

Completed 1  Outer loop iterations.
Multipliers         : -2.81973
Penalties           : 10
Constraint Violations: 1.90255
Norm of unconstrained portion of Lagrangian gradient = 0.00646352
Rayleigh

Finished pdmin loop on the 15-th iteration.
Normal termination test satisfied.

Completed 2  Outer loop iterations.
Multipliers         : -0.658243
Penalties           : 10
Constraint Violations: 0.000483281
Norm of unconstrained portion of Lagrangian gradient = 0.000206008
```

```
Rayleigh

Finished pdmin loop on the 8-th iteration.
Normal termination test satisfied.

Completed 3  Outer loop iterations.
Multipliers          : -0.653453
Penalties            : 10
Constraint Violations: -7.91394e-06
Norm of unconstrained portion of Lagrangian gradient = 1.37231e-06
Rayleigh

Finished pdmin loop on the 7-th iteration.
Normal termination test satisfied.

Completed 4  Outer loop iterations.
Multipliers          : -0.653431
Penalties            : 10
Constraint Violations: -8.6292e-07
Norm of unconstrained portion of Lagrangian gradient = 2.19012e-07
Objective Value      : 29.8635
Normal termination of outer loop.
```

The displayed output reports that, at the current solution, the objective value is 29.8635 and the endpoint constraint is being violated by -8.63×10^{-6}. There is some error in these values due to the integration error of the fixed step-size integration routines. We can get a more accurate measure by using the variable step-size integration routine to simulate the system with the control solution `u`:

```
>> simulate(1,x0,u,t,5,0);          % Simulate system using LSODA
>> simulate(2,1,1)                  % Evaluate the objective function

ans =

   29.8648

>> simulate(2,2,1)                  % Evaluate the endpoint constraint

ans =

   5.3852e-06
```

So the reported values are fairly accurate.

Session 3 (Problem with control bounds and free final time). This session demonstrates the transcription, explained in §2, of a free final time problem into a fixed final time problem. The transcribed problem has bounds on the control and free initial states. Also, **distribute** (see §7) is used to improve integration mesh after an initial solution is found. A more accurate solution will then be computed by re-solving the problem on the new mesh.

The original problem, Problem Bang, is a minimum-time problem with three states and one input. This problem is converted into a fixed final time problem using the transcription described in §2. Only one extra state variable was needed since the problem has time-independent (autonomous) dynamics. The augmented problem is implemented in the file 'bang.c'. First we will define the integration mesh and then the initial conditions.

```
>> N = 20;                          % Discretization level
>> T = 10;                          % Nominal final time
>> t=[0:T/N:T];                     % Nominal time interval for maneuver
```

The nominal time interval is of duration `T`. Next, we specify a value for ξ^3, the duration scale factor, which is the initial condition for the augmented state. The quantity $T\xi^3$ represents our guess for the optimal duration of the maneuver.

```
>> x0=[0 0 1]';                     % Initial conditions for augmented system
>> fixed=[1 1 0]';                  % Which initial conditions are fixed
>> x0_lower=[0 0 0.1]';             % Lower bound for free initial condition
>> x0_upper=[0 0 10]';              % Upper bound for free initial condition

>> X0=[x0,fixed,x0_lower,x0_upper]

X0 =

        0    1.0000         0         0
        0    1.0000         0         0
   1.0000         0    0.1000   10.0000
```

The first column of `X0` is the initial conditions for the problem; there are three states including the augmented state. The initial conditions for the original problem were $x(0) = (0, 0)^T$. The initial condition for the augmented state is set to `x0(3)` $= \xi^3 = 1$ to indicate that our initial guess for the optimal final time is one times the nominal final time of $T = 10$, *i.e.*, $\xi^3 T$. The second column of `X0` indicates which initial conditions are to be considered fixed and which are to be treated as free variables for the optimization program to adjust. A one indicates fixed and a zero indicates free. The third and fourth columns provide lower an upper bound for the free initial conditions.

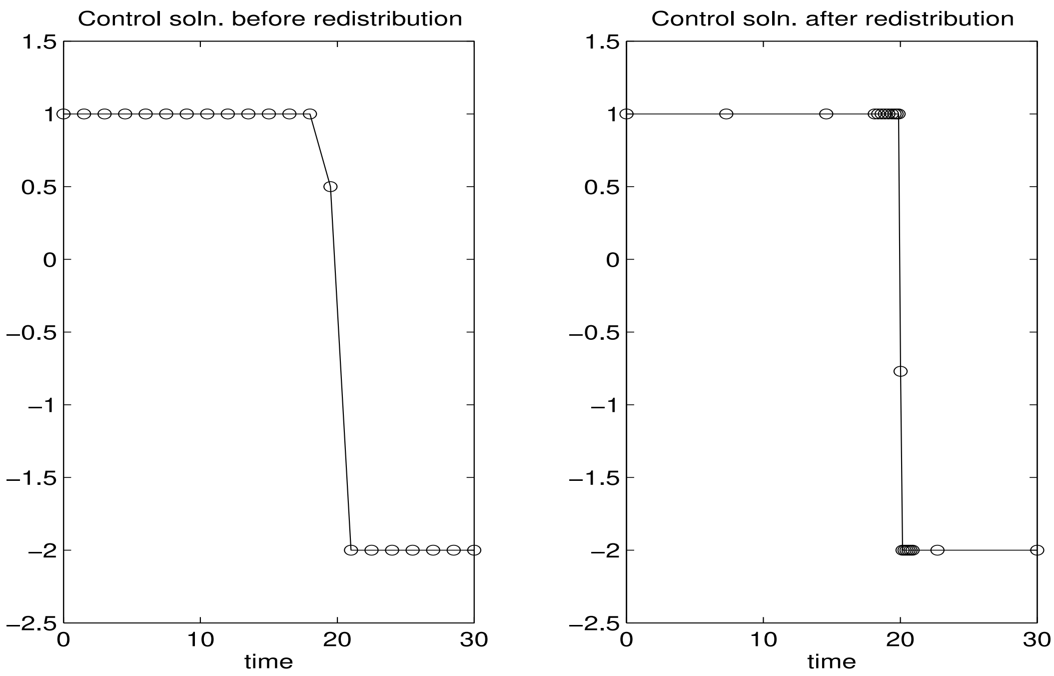
```
>> u0=zeros(1,N+1);
>> [u,x,f]=riots(X0,u0,t,-2,1,[],100,2);        % Solve problem; f=x(3,1)=x0(3)
>> f*T                                          % Show the final time.

ans =

   29.9813
```

In this call to **riots**, we have also specified a lower bound of -2 and an upper bound of 1 for all of the control spline coefficients. Since we are using second order splines, this is equivalent to specifying bounds on the value of the control at the spline breakpoints, *i.e.* bounds on $u(t_k)$. We also specify that the second order Runge-Kutta integration routine should be used. The objective value `f` $= \xi^3$ is the duration scale factor. The final time is given by $a + (b - a)\xi^3 = T\xi^3 =$ `10f`. Here we see that the final time is 29.9813. A plot of the control solution indicates a fairly broad transition region whereas we expect a bang-bang solution. We can try to improve the solution by redistributing the integration mesh. We can then re-solve the problem using the new mesh and starting from the previous solution interpolated onto the new mesh. This new mesh is stored in `new_t`, and `new_u` contains the control solution interpolated onto this new mesh.

```
>> [new_t,new_u]=distribute(t,u,x,2,[],1,1);           % Re-distribute mesh

redistribute_factor = 7.0711

Redistributing mesh.
>> X0(:,1) = x(:,1);
>> [u,x,f]=riots(X0,new_u,new_t,-2,1,[],100,2);
>> f*10

ans =

   30.0000
```

Notice that before calling **riots** the second time, we set the initial conditions (the first column of `X0`) to `x(:,1)`, the first column of the trajectory solution returned from the preceding call to **riots**. Because ξ^3 is a free variable in the optimization, `x(3,1)` is different than what was initially specified for `x0(3)`. Since `x(3,1)` is likely to be closer to the optimal value for ξ^3 than our original guess we set the current guess for `X0(3,1)` to `x(3,1)`.

We can see the improvement in the control solution and the solution for the final time. The reported final time solution is 30 and this happens to be the exact answer. The plot of the control solution before and after the mesh redistribution is shown below. The circles indicate where the mesh points are located. The improved solution does appear to be a bang-bang solution.


Control soln. before redistribution
Control soln. after redistribution
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2.5
0
10
20
30
time

Session 4 (Example using outer). This example demonstrates the experimental program **outer** which repeatedly adjusts the integration mesh between calls to **riots** in order to achieve a desired solution accuracy. We use **outer** to solve the Goddard rocket ascent problem implemented in the file 'goddard.c'. The Goddard rocket problem is a free-time problem whose objective is to maximize the rocket's altitude subject to having a fixed amount of fuel. This problem is particularly difficult because its solution contains a singular sub-arc. We use an initial guess of $u(t) = 1$ for all t so that the rocket starts out climbing and does not fall into the ground. We will use a second order spline representation and start with a discretization level of $N = 50$. Also, since this is a minimum-time problem, we augmented the system dynamics with a fourth state that represents the duration scale factor. We start by guessing a duration scale factor of 0.1 by setting $\xi^4 = 0.1$ and we specify $[0, 1]$ for the nominal time interval. Thus the nominal final time is $T\xi^4 = 0.1$.

```
>> x0=[0 1 1 0.1]';
>> fixed=[1 1 1 0]';
>> t=[0:1/50:1];
>> u0=ones(1,51);
```

Now **outer** is called with lower and upper control bounds of 0 and 3.5, respectively; no systems parameters; a maximum of 300 iterations for each inner loop; a maximum of 10 outer loop iteration with a maximum discretization level of $N = 500$; default termination tolerances; integration algorithm 4 (RK4); and mesh redistribution strategy 2.

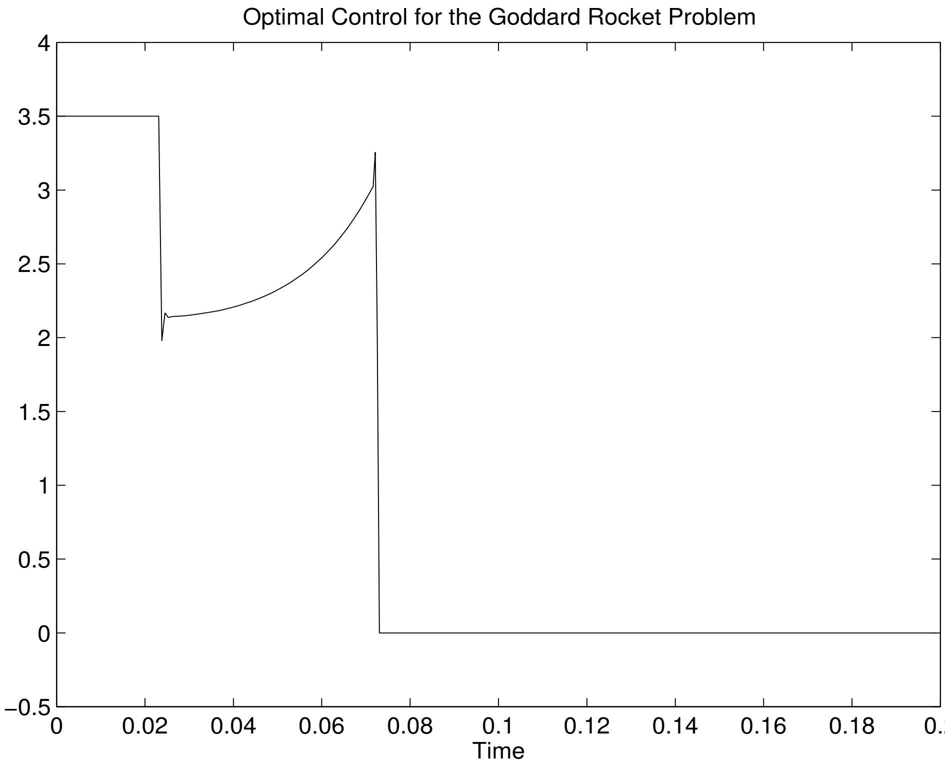
```
>> [new_t,u,x]=outer([x0,fixed],u0,t,0,3.5,[],500,[10;500],4,[],2);
Goddard

Completed 70  riots  iterations.  Normal Termination.
Doubling mesh.
========Completed  1  OUTER  iteration=========
Norm of Lagrangian gradient = 3.43882e-05
Sum of constraint errors    = 4.57119e-09
Objective function value    = -1.01284
Integration error           = 1.49993e-06
===============================================
Goddard

Completed 114  riots  iterations.  Kuhn-Tucker conditions satisfied but sequence did not converge.

========Completed   2 OUTER  iterations========
Norm of Lagrangian gradient = 4.64618e-06
Sum of constraint errors    = 4.41294e-10
Objective function value    = -1.01284
Integration error           = 2.01538e-07
Change in solutions         = 0.128447
Control error estimate      = 0.0200655
```

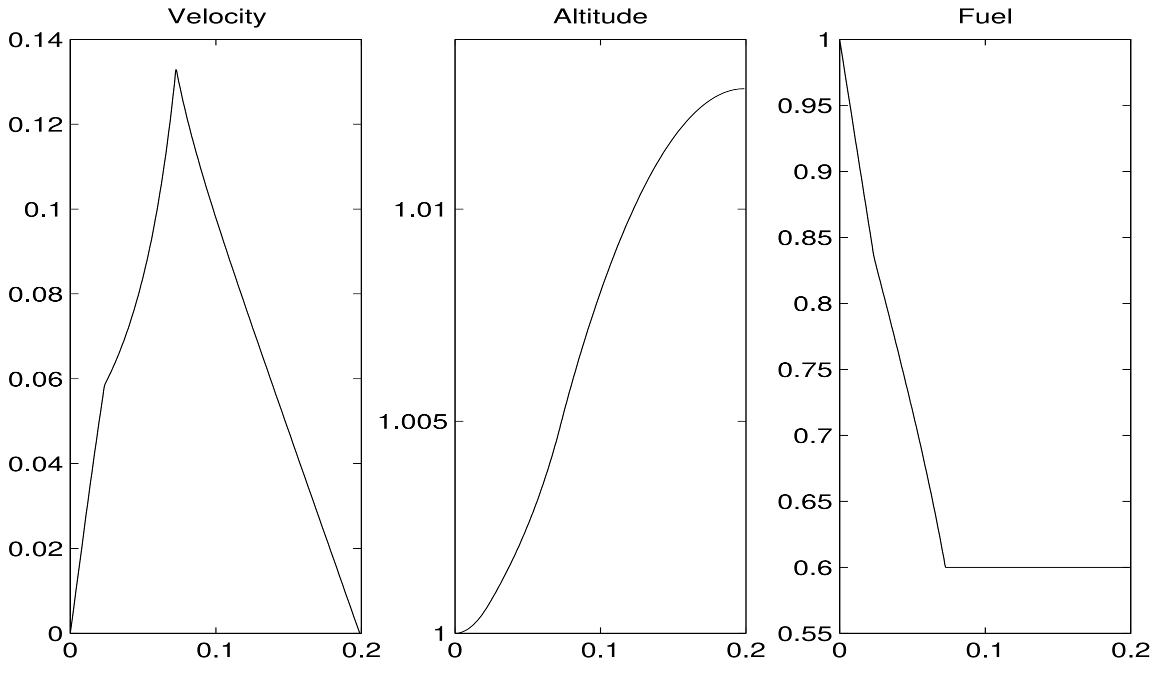
```
==============================================

Redistribution factor = 2.07904
Redistributing mesh.
New mesh contains 146 intervals.  Old mesh contained 100 intervals.
Goddard

Completed 206  riots  iterations.  Kuhn-Tucker conditions satisfied but sequence did not converge.

========Completed   3 OUTER  iterations========
Norm of Lagrangian gradient = 2.38445e-08
Sum of constraint errors    = 8.49733e-11
Objective function value    = -1.01284
Integration error           = 4.67382e-09
Change in solutions         = 0.0878133
Control error estimate      = 0.000452989
==============================================
Normal Termination.
CPU time = 26.9167 seconds.
```

The message stating that the Kuhn-Tucker conditions are satisfied but that the sequence did not converge is a message from NPSOL which is the nonlinear programming algorithm linked with **riots** in this example. This message indicates that, although first order optimality conditions for optimality are satisfied (the norm of the gradient of the Lagrangian is sufficiently small), the control functions from one iteration of **riots** to the next have not stopped changing completely. The sources of this problem are *(i)* the Goddard problem is a singular optimal control problem; this means that small changes in the controls over some portions of the time interval have very little effect on the objective function and *(ii)* **outer** calls **riots** with very tight convergence tolerances. Because of this, the calls to **riots** probably performed many more iterations than were useful for the level of accuracy achieved. Choosing better convergence tolerances is a subject for future research.

The optimal control and optimal state trajectories are shown on the next page. Notice that to plot the optimal control we multiply the time vector `new_t` by `x(4,1)` which contains the duration scale factor. The optimal final time for this problem, since $a = 0$ and $b = 1$, is just `x(4,1)=0.1989`. Note that the final mass of the rocket is 0.6. This is the weight of the rocket without any fuel. The maximum height is the negative of the objective function, $h^{*}(t) \approx 1.01284$.

```
>> sp_plot(new_t*x(4,1),u)
>> plot(new_t*x(4,1),x(1,:))
>> plot(new_t*x(4,1),x(2,:))
>> plot(new_t*x(4,1),x(3,:))
```


Optimal Control for the Goddard Rocket Problem
Time


Velocity
Altitude
Fuel

4. USER SUPPLIED SYSTEM SUBROUTINES

All of the functions in the description of **OCP** in §2 are computed from the user functions **h**, **l** and **g**; the derivatives of these functions are computed from the user functions **Dh**, **Dl** and **Dg**. Two other user functions, **activate** and **init**, are required for the purpose of passing information to and from RIOTS.

Smoothness Requirements. The user-supplied functions must have a certain degree of smoothness. The smoothness requirement comes about for three reasons. First, the theory of differential equations requires, in general, that $h(t, x, u)$ be piecewise continuous with respect to t, Lipschitz continuous with respect to x and u and that $u(\cdot)$ be continuous, in order to ensure the existence and uniqueness of a solution satisfying the system of differential equations. A finite number of discontinuities in $h(\cdot, x, u)$ and $u(\cdot)$ are allowable. Second, the optimization routines needs at least one continuous derivative of the objective and constraint functions $g(\cdot,\cdot)$ and $l(t,\cdot,\cdot)$. Two continuous derivatives are needed in order for there to be a chance of superlinear convergence. The third reason is that the accuracy of numerical integration of differential equations depends on the smoothness of $h(\cdot,\cdot,\cdot)$ and $l(\cdot,\cdot,\cdot)$. For a fixed step-size methods with order s, $\partial^{(s)}h(t, x, u)/\partial x^s$ and $\partial^{(s)}h(t, x, u)/\partial u^s$ should be continuous (or the $(r-1)$-th partial should be Lipschitz continuous). Furthermore, any discontinuities in $h(\cdot, x, u(\cdot))$ or its derivatives should occur only at integration breakpoints†. Conversely, the user should place integration breakpoints wherever such discontinuities occur. The same considerations also hold for the function $l(t, x, u)$. For variable step-size integration, $h(t, x, u)$ and $l(t, x, u)$ should have at least continuous partial derivatives of order one with respect to x and u. Again, any discontinuities in $h(\cdot, x, u(\cdot))$ and $l(\cdot, x, u(\cdot))$ or its derivatives should only occur at integration break points.

Constraint Qualifications. A common requirement of mathematical programming algorithms is linear independence of the active constraints gradients at a solution. It is easy to mathematically specify a valid constraint in such a way that this condition is violated. For example, consider a scalar constraint of the form $g(u) = 0$. This constraint can be specified as

$$g(u)^2 = 0 \ .$$

However, $\frac{d}{du}(g(u)^2) = 2g(u)\frac{dg}{du}$. Thus, if this constraint is active at the solution u^*, *i.e.*, $g(u^*) = 0$, then the gradient of this constraint is zero. So this specification for the constraint violates the constraint qualification. However, if the constraint is specified simply as

† Note that discontinuities in $u(t)$ can only occur at the spline breakpoints, t_k.

$$g(u) = 0\ ,$$

then the constraint qualification is not violated.

The user functions can be supplied as object code or as M-files. The C-syntax and M-file syntax for these functions are given below. Because all arguments to the object code versions of the functions are passed by reference, the object code format is compatible with Fortran. A template for these functions can be found in the file `systems/template.c`. There are also several example problems in the `systems` directory. In addition to the user-supplied routines, this section also describes two other functions, **get_flags** and **time_fnc**, that are callable by user object code.

There are three main differences between object code and M-file versions of the user functions:

- The programs in RIOTS execute much faster when object code is given.
- Object code versions of the user functions do not need to assign zero values to array components which are always zero. M-file versions must set all array values (with the exception of **sys_init**).
- There must be a separate M-file for each function with the same name as that function. The names begin with `sys_` followed by the name of the function. For example, sys_Dh.m is the M-file for the user function **sys_Dh**. The directory in which these M-files are located must be in Matlab's search path.
- **Important:** Arrays in Matlab are indexed starting from 1 whereas in C arrays are indexed starting from 0. For example, `neq[4]` in C code has an M-file equivalent of `neq(5)`.

activate, sys_activate

Purpose

This function is always called once before any of the other user-supplied functions. It allows the user to perform any preliminary setup needed, for example, loading a data array from a file.

C Syntax

```
void activate(message)
     char **message;
{
  *message = "";
  /* Any setup routines go here. */
}
```

M-file Syntax

```
function message = sys_activate

message = '';
```

Description

If the message string is set, that string will be printed out whenever **simulate** (form 0) or an optimization routine is called. It is useful to include the name of the optimal control problem as the message.

See Also: **get_flags**.

init, sys_init

Purpose

This function serves two purposes. First, it provides information about the optimal control problem to RIOTS. Second, it allows system parameters to be passed from Matlab to the user-defined functions at run-time. These system parameters can be used, for instance, to specify constraint levels. Unlike **activate**, **init** may be called multiple times. The array `neq[]` is explained after the syntax.

C Syntax

```
void init(neq,params)
int neq[];
double *params;
{
  if ( params == NULL ) {
    /* Set values in the neq[] array. */
  }
  else {
    /* Read in runtime system parameters. */
  }
}
```

M-file Syntax

```
function neq = sys_init(params)

% if params is NULL then setup neq.  Otherwise read system
% parameters in params.  In Matlab, arrays are indexed
% starting from 1, so neq(i) corresponds to the C statement
% neq[i-1].

if params == [],
   % Each row of neq consists of two columns.  The value in
   % the first column specifies which piece of system
   % information to set.  The value in the second column is
   % the information.  For example, to indicate that the
   % system has 5 system parameters, one row in neq should be
   % [3 5] since neq(3) stores the number of system
   % parameters.

   % Here we set nstates = 2; ninputs = 1; 1 nonlinear
   % endpoint constr.
   neq = [1 2 ; 2 1 ; 12 1 ];
else
   % Read in systems parameters from params and store them in
```

```
    % the global variable sys_params which will be accessible
    % to other systems M-files.
    global sys_params
    sys_params = params;
end
```

Description

When this functions is called, the variable `params` will be set to 0 (`NULL`) if `init()` is expected to return information about the optimal control problem via the `neq[]` array. Otherwise, `params` is a vector of system parameters being passed from Matlab to the user's program. When `params==0`, the values in `neq[]` should be set to indicate the following:

```
neq[0]  ---  Number of state variables.
neq[1]  ---  Number of inputs.
neq[2]  ---  Number of system parameters.
neq[3]  ---  Not used on calls to init().  Contains time index.
neq[4]  ---  Not used on calls to init().  Used to indicate which function to evaluate.
neq[5]  ---  Number of objective functions.
neq[6]  ---  Number of general nonlinear trajectory inequality constraints.
neq[7]  ---  Number of general linear trajectory inequality constraints.
neq[8]  ---  Number of general nonlinear endpoint inequality constraints.
neq[9]  ---  Number of general linear endpoint inequality constraints.
neq[10] ---  Number of general nonlinear endpoint equality constraints.
neq[11] ---  Number of general nonlinear endpoint equality constraints.
neq[12] ---  Indicates type of system dynamics and cost functions:
                0 --> nonlinear system and cost,
                1 --> linear system,
                2 --> linear and time-invariant system,
                3 --> linear system with quadratic cost,
                4 --> linear and time-invariant with quadratic cost.
```

Remember that, for M-files, `neq(i)` is equivalent to the C-code statement `neq[i-1]`. The values of `neq[]` all default to zero except `neq[5]` which defaults to 1. The relationship between the values in `neq[]` and the general problem description of **OCP** given in §2 is as follows: n = `neq[0]`, m = `neq[1]`, p = `neq[2]`, q_o = `neq[5]`, q_{ti} = `neq[6]+neq[7]`, q_{ei} = `neq[8]+neq[9]` and q_{ee} = `neq[10]+neq[11]`. The locations `neq[3]` and `neq[4]` are used in calls to the other user-defined functions.

If **init** sets `neq[2]>0`, then **init** will be called again with `params` pointing to an array of system parameters which are provided by the user at run-time. These parameters can be stored in global variables for use at other times by any of the other user-defined functions. Some examples of useful system parameters include physical coefficients and penalty function parameters. These parameters are fixed and will not be adjusted during optimization. Parameters that are to be used as decision variables must be specified as initial conditions to augmented states π with $\dot{\pi} = 0$.

Notes

1. Control bounds should be indicated separately when calling the optimization routines. Do not include any simple bound constraints in the general constraints. Similarly, simple bounds on free initial conditions should be specified on the command line.

2. For nonlinear systems, all constraints involving a state variable are nonlinear functions of the control. Thus, the constraint $g(\xi, x(b)) = x(b) = 0$, while linear in its arguments, is nonlinear with respect to u. The user does not need to account for this situation, however, and should indicate that g is a linear constraint. RIOTS automatically treats all general constraints for nonlinear systems as nonlinear.

h, sys_h

Purpose

This function serves only one purpose, to compute $h(t, x, u)$, the right hand side of the differential equations describing the system dynamics.

C Syntax

```
void h(neq,t,x,u,xdot)
     int neq[];
     double *t,x[NSTATES],u[NINPUTS],xdot[NSTATES];
{
  /* Compute xdot(t) = h(t,x(t),u(t)). */
}
```

M-file Syntax

```
function xdot = sys_h(neq,t,x,u)

global sys_params

% xdot must be a column vector with n rows.
```

Description

On entrance, `t` is the current time, `x` is the current state vector and `u` is the current control vector. Also, `neq[3]` is set to the current discrete-time index, $k - 1$, such that $t_k \le t < t_{k+1}$[†].

On exit, the array `xdot[]` should contain the computed value of $h(t, x, u)$. The values of `xdot[]` default to zero for the object code version. Note that for free final time problems the variable `t` should not be used because derivatives of the system functions with respect to `t` are not computed. In the case of non-autonomous systems, the user should augment the state variable with an extra state representing time (see transcription for free final time problems in §2).

See Also: **time_fnc**.

[†] The index is $k - 1$ since indexing for C code starts at zero. For M-files, `neq(4) = k`.

l, sys_l

Purpose

This function serves two purposes. It is used to compute values for the integrands of cost functions, $l_o(t,x,u)$, and the values of state trajectory constraints, $l_{ti}(t,x,u)$.

C Syntax

```
double l(neq,t,x,u)
     int neq[];
     double *t,x[NSTATES],u[NINPUTS];
{
  int F_num, constraint_num;
  double z;

  F_num = neq[4];
  NFUNS = neq[5];
  if ( F_num <= NFUNS ) {
    /* Compute z = l(t,x(t),u(t) for the F_num integrand.  */
    /* If this integrand is identically zero,              */
    /* set z = 0 and neq[3] = -1.                          */
  }
  else {
    constraint_num = F_num - NFUNS;
    /* Compute z = l(t,x(t),u(t) for the     */
    /* constraint_num trajectory constraint. */
  }
  return z;
}
```

M-file Syntax

```
function z = sys_l(neq,t,x,u)
% z is a scalar.

global sys_params
F_NUM = neq(5);
NFUNS = neq(6);

if F_NUM <= NFUNS
  % Compute z = l(t,x(t),u(t)) for the F_num integrand.
else
  constraint_num = F_num - NFUNS;
  % Compute z = l(t,x(t),u(t)) for the constraint_num
  % traj. constraint.
end
```

Description

On entrance, `t` is the current time, `x` is the current state vector and `u` is the current control vector. Also, `neq[3]` is set to the current discrete-time index $k-1$ such that $t_k \le t < t_{k+1}$ (see footnote for **h**) and `neq[4]` is used to indicate which integrand or trajectory constraint is to be evaluated. Note that, for free final time problems, the variable `t` should not be used because derivatives of the system functions with respect to `t` are not computed. In this case, the user should augment the state variable with an extra time state and an extra final-time state as described in §2.

If $1 \le neq[4] \le q_o$, then `z` should be set to $l_o^{neq[4]}(t, x, u)$. If $l_o^{neq[4]}(\cdot, \cdot, \cdot) = 0$ then, besides returning 0, **l** (in object code versions) can set $neq[3] = -1$ to indicate that the function is identically zero. The latter increases efficiency because it tells RIOTS that there is no integral cost. Only the function **l** is allowed to modify $neq[3]$. Regardless of how $neq[3]$ is set, **l** *must* always return a value even if the returned value is zero.

If $neq[4] > q_o$, then z should be set to $l_{ti}^{neq[4]-q_o}(t, x, u)$. If there are both linear and nonlinear trajectory constraints, the nonlinear constraints must precede those that are linear. The ordering of the functions computed by **l** is summarized in the following table:

	ν	function to compute
$1 \le neq[4] \le q_o$	$neq[4]$	$l_o^\nu(t, x, u)$
$neq[4] > q_o$	$neq[4] - q_o$	$l_{ti}^\nu(t, x, u)$, nonlinear $l_{ti}^\nu(t, x, u)$, linear

g, sys_g

Purpose

This function serves two purposes. It is used to compute the endpoint cost function $g_o(\xi, x(b))$ and the endpoint inequality and equality constraints $g_{ei}(\xi, x(b))$ and $g_{ee}(\xi, x(b))$. The syntax for this function includes an input for the time variable t for consideration of future implementations and should not be used. Problems involving a cost on the final time T should use the transcription for free final time problems described in §2.

C Syntax

```
double g(neq,t,x0,xf)
     int neq[];
     double *t,x0[NSTATES],xf[NSTATES];
{
  int F_num, constraint_num;
  double value;

  F_num = neq[4];
  NFUNS = neq[5];
  if ( F_num <= NFUNS ) {
    /* Compute value of g(t,x0,xf) for the */
    /* F_num cost function.                */
  }
  else {
    constraint_num = F_num - NFUNS;
    /* Compute value g(t,x0,xf) for the    */
    /* constraint_num endpoint constraint. */
  }
  return value;
}
```

M-file Syntax

```
function J = g(neq,t,x0,xf)

% J is a scalar.

global sys_params
F_NUM = neq(5);
if F_NUM <= sys_params(6)
   % Compute g(t,x0,xf) for cost function.
elseif F_NUM == 2
   % Compute g(t,x0,xf) for endpoint constraints.
end
```

Description

On entrance, `x0` is the initial state vector and `xf` is the final state vector. The value `neq[4]` is used to indicate which cost function or endpoint constraint is to be evaluated. Nonlinear constraints must precede linear constraints. The order of functions to be computed is summarized in the following table:

	ν	function to compute
$1 \le neq[4] \le q_o$	$neq[4]$	$g_o^\nu(\xi, x(b))$
$q_o < neq[4] \le q_o + q_{ei}$	$neq[4] - q_o$	$g_{ie}^\nu(\xi, x(b))$, nonlinear $g_{ie}^\nu(\xi, x(b))$, linear
$q_o + q_{ei} < neq[4] \le q_o + q_{ei} + q_{ee}$	$neq[4] - q_o - q_{ei}$	$g_{ee}^\nu(\xi, x(b))$, nonlinear $g_{ee}^\nu(\xi, x(b))$, linear

See Also: **time_fnc**.

Dh, sys_Dh
Dl, sys_Dl
Dg, sys_Dg

Purpose

These functions provide the derivatives of the user-supplied function with respect to the arguments x and u. The programs **riots** (see §6) can be used without providing these derivatives by selecting the finite-difference option. In this case, dummy functions must be supplied for **Dh**, **Dl** and **Dg**.

C Syntax

```
void Dh(neq,t,x,u,A,B)
     int neq[];
     double *t,x[NSTATES],u[NINPUTS];
     double A[NSTATES][NSTATES],B[NSTATES][NINPUTS];
{
  /* The A matrix should contain dh(t,x,u)/dx. */
  /* The B matrix should contain dh(t,x,u)/du. */
}

double Dl(neq,t,x,u,l_x,l_u)
  int neq[];
  double *t,x[NSTATES],u[NINPUTS],l_x[NSTATES],l_u[NINPUTS];
{
  /* l_x[] should contain dl(t,x,u)/dx         */
  /* l_u[] should contain dl(t,x,u)/du         */
  /* according to the value of neq[4].         */
  /* The return value is dl(t,x0,xf)/dt which */
  /*   is not currently used by RIOTS.         */
  return 0.0;
}

double Dg(neq,t,x0,xf,g_x0,g_xf)
     int neq[];
     double *t,x0[NSTATES],xf[NSTATES],J_xf[NSTATES];
{
  /* g_x0[] should contain dg(t,x0,xf)/dx0.   */
  /* g_xf[] should contain dg(t,x0,xf)/dxf.   */
  /*   according to the value of neq[4].      */
  /* The return value is dg(t,x0,xf)/dt which */
  /*   is not currently used by RIOTS.        */
  return 0.0;
}
```

M-file Syntax

```
function [A,B] = sys_Dh(neq,t,x,u)
global sys_params
% A must be an n by n matrix.
% B must be an n by m matrix.

function [l_x,l_u,l_t] = sys_Dl(neq,t,x,u)
global sys_params
% l_x should be a row vector of length n.
% l_u should be a row vector of length m.
% l_t is a scalar---not currently used.

function [g_x0,g_xf,g_t] = sys_cost(neq,t,x0,xf)
global sys_params
% g_x0 and g_xf are row vectors of length n.
% g_t is a scalar---not currently used.
```

Description

The input variables and the ordering of objectives and constraints are the same for these derivative functions as they are for the corresponding functions **h, l,** and **g**. The derivatives with respect to t are not used in the current implementation of RIOTS and can be set to zero. The derivatives should be stored in the arrays as follows:

Function	First output	index range	Second output	index range
Dh	$A[i][j] = \left[\frac{dh(t,x,u)}{dx}\right]_{i+1,j+1}$	$i = 0: n-1$ $j = 0: n-1$	$B[i][j] = \left[\frac{dh(t,x,u)}{du}\right]_{i+1,j+1}$	$i = 0: n-1$ $j = 0: m-1$
Dl	$l\_x[i] = \left[\frac{dl(t,x,u)}{dx}\right]_{i+1}$	$i = 0: n-1$	$l\_u[i] = \left[\frac{dl(t,x,u)}{du}\right]_{i+1}$	$i = 0: m-1$
Dg	$g\_x0[i] = \left[\frac{dg(t,x0,xf)}{dx0}\right]_{i+1}$	$i = 0: n-1$	$g\_xf[i] = \left[\frac{dg(t,x0,xf)}{dxf}\right]_{i+1}$	$i = 0: m-1$
sys_Dh	$A(i,j) = \left[\frac{dh(t,x,u)}{dx}\right]_{i,j}$	$i = 1: n$ $j = 1: n$	$B(i,j) = \left[\frac{dh(t,x,u)}{du}\right]_{i,j}$	$i = 1: n$ $j = 1: m$
sys_Dl	$l\_x(i) = \left[\frac{dl(t,x,u)}{dx}\right]_{i}$	$i = 1: n$	$l\_u(i) = \left[\frac{dl(t,x,u)}{du}\right]_{i}$	$i = 1: m$
sys_Dg	$g\_x0(i) = \left[\frac{dg(t,x0,xf)}{dx0}\right]_{i}$	$i = 1: n$	$g\_xf(i) = \left[\frac{dg(t,x0,xf)}{dxf}\right]_{i}$	$i = 1: m$

Note that, for **sys_Dh**, RIOTS automatically accounts for the fact that Matlab stores matrices transposed relative to how they are stored in C.

get_flags

Purpose

This function allows user-supplied object code to read a vector of integers from Matlab's workspace.

C Syntax

```
int get_flags(flags,n)
     int flags[],*n;
```

Description

A call to **get_flags** causes `flags[]` to be loaded with up to `n` integers from the array `FLAGS` if `FLAGS` exists in Matlab's workspace. It is the user's responsibility to allocate enough memory in `flags[]` to store `n` integers. The value returned by **get_flags** indicates the number of integers read into `flags[]`.

The main purpose of **get_flags** is to allow a single system program to be able to represent more than one problem configuration. The call to **get_flags** usually takes place within the user-function **activate**. In the example below, **get_flags** is used to read in the number of constraints to use for the optimal control problem.

Example

```
extern int get_flags();
static int Constraints;

void activate(message)
     char **message;
{
  int n,flags[1];

  *message = "Use FLAGS to specify number of constraints.";
  n = 1;
  if ( get_flags(flags,&n) > 0 );
    Constraints = flags[0];
  else
    Constraints = 0;
}
```

Notes

1. It is best to define `FLAGS` as a global variable in case **simulate** gets called from within an M-file. This is accomplished by typing

```
>> global FLAGS
```

At the Matlab prompt. To clear `FLAGS` use the Matlab command

```
>> clear global FLAGS
```

2. Since **activate** is called once only, you must clear **simulate** if you want to re-read the values in `FLAGS`. To clear **simulate**, type

```
>> clear simulate
```

at the Matlab prompt.

3. For M-files, any global variable can be read directly from Matlab's workspace so an M-file version of **get_flags** is not needed.

time_fnc

Purpose

This function allows user-supplied object code to make calls back to a user-supplied Matlab m-function called **sys_time_fnc.m** which can be used to compute a function of time. Call-backs to Matlab are very slow. Since this function can be called thousand of times during the course of a single system simulation it is best to provide the time function as part of the object code if possible.

C Syntax

```
void time_fnc(t,index,flag,result)
     int *index,*flag;
     double *t,result[];
```

Syntax of **sys_time_fnc.m**

```
function f = sys_time_fnc(tvec)

% tvec = [time;index;flag]
% Compute f(time,index,flag).
```

Description

If **time_fnc** is to called by one of the user-functions, then the user must supply an m-function named **sys_time_fnc**. The inputs `tvec(1)=time` and `tvec(2)=index` to **sys_time_fnc** are related by $t_{index} \le \texttt{time} \le t_{index+1}$. The value of `index` passed to **sys_time_fnc** is one greater than the value passed from **time_fnc** to compensate for the fact the Matlab indices start from 1 whereas C indices start from 0. The input `flag` is an integer that can be used to select from among different time functions. Even if `flag` is not used, it *must* be set to some integer value.

The values in the vector `f` returned from **sys_time_fnc** are stored in `result` which must have enough memory allocated for it to store these values.

Example

Suppose we want **l** to compute $f(t)x^1(t)$ where $f(t) = sin(t) + y_d(t)$ with $y_d(t)$ is some pre-computed global variable in the Matlab workspace. Then we can use **time_fnc** to compute $f(t)$ and use this value to multiply `x[0]`:

```
extern void time_fnc();
double l(neq,t,x,u)
     int neq[];
     double *t,x[NSTATES],u[NINPUTS];
{
  int i,zero;
  double result;

  i = neq[3];                        /* Discrete-time index. */
  zero = 0;
  time_fnc(t,&i,&zero,&result);      /* Call time_fnc with flag=0. */
  return result*x[0];                /* Return f(t)*x1(t).   */
}
```

Here is the function that computes $f(t)$. It computes different functions depending on the value of `flag=t(3)`. In our example, it is only called with `flag=0`.

```
function f = sys_time_fnc(t)

global yd             % Suppose yd is a pre-computed, global variable.
time = t(1);
index = t(2);
flag = t(3);

if flag == 0
  f = yd(time) + sin(time);
else
  f = another_fnc(time);
end
```

5. SIMULATION ROUTINES

This section describes the central program in RIOTS, **simulate**. All of the optimization programs in RIOTS are built around **simulate** which is responsible for computing all function values and gradients and serves as an interface between the user's routines and Matlab.

The computation of function values and gradients is performed on the integration mesh

$$\mathbf{t}_N \doteq \{ t_{N,k} \}_{k=1}^{N+1} .$$

Note that the indexing starts from $k = 1$ (rather than $k = 0$ as in earlier chapters) to conform with Matlab's indexing convention. For any mesh $\mathbf{t}_N$ we define

$$\Delta_{N,k} \doteq t_{N,k+1} - t_{N,k} .$$

This mesh also specifies the breakpoints of the control splines. The values of the trajectories computed by **simulate** are given at the times $t_{N,k}$ and are denoted, $x_{N,k}$, $k = 1, \ldots, N+1$. Thus, $x_{N,k}$ represents the computed approximation to the solution $x(t_{N,k})$ of the differential equation $\dot{x} = h(t, x, u)$, $x(a) = \xi$. The subscript N is omitted when its its presence is clear from context.

Spline Representation of controls. The controls u are represented as splines given by

$$u(t) = \sum_{k=1}^{N+\rho-1} \alpha_k \phi_{\mathbf{t}_N,\rho,k}(t)$$

where $\alpha_k \in \mathbb{R}^m$ and $\phi_{\mathbf{t}_N,\rho,k}(\cdot)$ is the k-th B-spline basis element of order ρ, defined on the knot sequence formed from $\mathbf{t}_N$ by repeating its endpoints ρ times. Currently, RIOTS does not allow repeated interior knots. We will denote the collection of spline coefficients by

$$\alpha \doteq \{ \alpha_k \}_{k=1}^{N+\rho-1} .$$

For single input systems, α is a row vector. Those interested in more details about splines are referred to the excellent reference [63]. The times t_k, $k = 1, \ldots, N$, define the spline breakpoints. On each interval $[t_k, t_{k+1}]$, the spline coincides with an ρ-th order polynomial. Thus, fourth order splines are made up of piecewise cubic polynomials and are called cubic splines. Similarly, third order splines are piecewise quadratic, second order splines are piecewise linear and first order splines are piecewise constant. For first and second order splines, $\alpha_k = u(t_k)$. For higher-order splines, the B-spline basis elements are evaluated using the recursion formula in (A2.2a).

The following pages describe **simulate**. First, the syntax and functionality of **simulate** is presented. This is followed by a description of the methods used by **simulate** to compute function values and gradients. Finally, two functions, **check_deriv** and **check_grad**, for checking user-supplied derivative information, and the function **eval_fnc** are described.

simulate

Purpose

This is the central program in RIOTS. The primary purpose of **simulate** is to provide function values and gradients of the objectives and constraints using one of six integration algorithms. The optimization routines in RIOTS are built around **simulate**. This program also serves as a general interface to the user-supplied functions and provides some statistical information.

There are currently seven different forms in which **simulate** can be called. Form 1 and form 2 (which is more conveniently accessed using **eval_fnc**) are the most useful for the user. The other forms are used primarily by other programs in RIOTS. The form is indicated by the first argument to simulate.

Form 0

```
[info,simed] = simulate(0,{params})
```

Form 1

```
[f,x,du,dz,p] = simulate(1,x0,u,t,ialg,action)
```

Form 2

```
f=simulate(2,f_number,1)
[du,dz,p] = simulate(2,f_number,action)
```

Form 3

```
[xdot,zdot] = simulate(3,x,u,t,{f_num,{k}})
[xdot,zdot,pdot] = simulate(3,x,u,t,p,{k})
```

Form 4

```
[h_x,h_u,l_x,l_u] = simulate(4,x,u,t,{f_num,{k}})
```

Form 5

```
[g,g_x0,g_xf] = simulate(5,x0,xf,tf,{f_num})
```

Form 6

```
stats = simulate(6)
```

Form 7

```
lte = simulate(7)
```

Description of Inputs and Outputs

The following table describes the inputs that are required by the various forms of **simulate**.

Table S1

Input	number of rows	number of columns	description
`x0`	n	1†	initial state
`xf`	n	1	final state
`u`	m	$N+\rho-1$	control vector
`t`	1	$N+1$	time vector
`tf`	1 to 4	1	final time
`ialg`	1	1	integration algorithm
`action`	1	1	(see below)
`f_num`	1	1	(see below)
`params`	(see below)	(see below)	system parameters

The following table describes the outputs that are returned by the various forms of **simulate**.

Table S2

Output	number of rows	number of columns	description
`f`	1	1	objective or constraint value
`x`	n	$N+1$	state trajectory
`p`	n	$N+1$	adjoint trajectory
`du`	m	$N+\rho-1$	control gradient
`dx0`	n	1	gradient of initial conditions
`lte`	$n+1$	$N+1$	local integration error
`xdot`	n	$N+1$	$h(t,x,u)$
`zdot`	1	$N+1$	$l(t,x,u)$
`h_x`	n	n	$\partial h/\partial x$
`h_u`	n	m	$\partial h/\partial u$
`l_x`	1	n	$\partial l/\partial x$
`l_u`	1	m	$\partial l/\partial u$
`g_x0`	1	n	$\partial g/\partial x_0$
`g_xf`	1	n	$\partial g/\partial x_f$

If a division by zero occurs during a simulation, **simulate** returns the Matlab variable `NaN`, which stands for "Not a Number", in the first component of each output. This can be detected, if desired, using the Matlab function `isnan()`.

† `x0` can be a matrix but only the first column is used.

Note: The length of the control vector depends on the control representation. Currently, all of the integration routines are setup to work with splines of order ρ defined on the knot sequence constructed from $\mathbf{t}_N$. The current implementation of RIOTS does not allow repeated interior knots. The length (number of columns) of `u` and `du` is equal to $N + \rho - 1$ where `N=length(t)-1` is the number of intervals in the integration mesh. The allowable spline orders depends on the integration algorithm, `ialg`, according to the following table:

Table S3

IALG	Order of spline representation
0 (discrete)	discrete-time controls
1 (Euler)	$\rho = 1$
2 (RK2)	$\rho = 2$
3 (RK3)	$\rho = 2$
4 (RK4)	$\rho = 2, 3, 4$
5 (LSODA)	$\rho = 1, 2, 3, 4$ †
6 (LSODA w/0 Jacobians)	$\rho = 1, 2, 3, 4$ †

When more than one spline order is possible, the integration determines the order of the spline representation by comparing the length of the control input `u` to the length of the time input `t`. If LSODA is called with `ialg=5`, the user must supply $\frac{dh}{dx}$ and $\frac{dl}{dx}$ in the user-functions **Dh** and **Dl**. If the user has not programmed these Jacobians, LSODA must be called with `ialg=6` so that, if needed, these Jacobians will be computed by finite-differences. The different integration methods are discussed in detail following the description of the various forms in which **simulate** can be called.

Bugs

1. There may be a problem with computation of gradients for the variable step-size integration algorithm (`ialg=5,6`) if the number of interior knots n_{knots} is different than one (see description of form 1 and gradient computations for LSODA below).

See Also: **eval_fnc**

† The maximum spline order allowed by **simulate** when using LSODA can be increased by changing the pre-compiler define symbol `MAX_ORDER` in adams.c.

Description of Different Forms

`[info,simed] = simulate(0,{params})`

This form is used to load system parameters and to return system information. If `params` is supplied, **simulate** will make a call to **init** so that the user's code can read in these parameters. Normally `params` is a vector. It can be a matrix in which case the user should keep in mind that Matlab stores matrices column-wise (Fortran style). If the system has no parameters then either omit `params` or set `params=[]`. If no output variables are present in this call to **simulate** the system message loaded on the call to **activate** and other information about the system will be displayed.

The following is a list of the different values in `info` returned by **simulate**:

`info(1)`	number of states
`info(2)`	number of inputs
`info(3)`	number of system parameters
`info(4)`	(reserved)
`info(5)`	(reserved)
`info(6)`	number of objective functions
`info(7)`	number of nonlinear trajectory inequality constraints
`info(8)`	number of linear trajectory inequality constraints
`info(9)`	number of nonlinear endpoint inequality constraints
`info(10)`	number of linear endpoint inequality constraints
`info(11)`	number of nonlinear endpoint equality constraints
`info(12)`	number of linear endpoint equality constraints
`info(13)`	type of system (0 through 4) 0: nonlinear dynamics and objective 1: linear dynamics; nonlinear objective 2: linear, time-invariant dynamics; nonlinear objective 3: linear dynamics; quadratic objective 4: linear, time-invariant dynamics; quadratic objective
`info(14)`	number of mesh points used in the most recent simulation

The scalar output `simed` is used to indicate whether a call to **simulate**, form 1, has been made. If `simed=1` then a simulation of the system has occurred. Otherwise `simed=0`.

`[f,x,du,dx0,p] = simulate(1,x0,u,t,ialg,action)`

This form causes the system dynamics, $\dot{x} = h(t, u, x)$ with $x(a) = x0$, to be integrated using the integration method specified by `ialg` (*cf.* Table S3). Also, the value `f` of the first objective function, and possibly its gradients, `du` and `dx0` and the adjoint `p`, can be evaluated. Only the first column of `x0` is read. The strictly increasing time vector `t` of length $N + 1$ specifies the integration mesh on $[a, b]$ with `t`$(1) = a$ and `t`$(N + 1) = b$. The control `u` is composed of m rows of spline coefficients.

The calculations performed by **simulate**, form 2, depend on the value of `action`. These actions are listed in the following table:

Table S4

Action	Return Values
0	no return values
1	function value f
2	f and system trajectory x
3	f, x and control and initial condition gradients du and dz
4	f, x, du, dz and the adjoint trajectory p.

When using the variable step-size method LSODA (`ialg = 5,6`), the argument `ialg` can include three additional pieces of data:

	Setting	Default Value
`ialg(2)`	Number of internal knots used during gradient computation.	1
`ialg(3)`	Relative integration tolerance.	1e-8
`ialg(4)`	Absolute integration tolerance.	1e-8

The meaning of “internal knots” is discussed below in the description of gradient computation with LSODA.

Example

The following commands, typed at the Matlab prompt, will simulate a three state system with two inputs using integration algorithm RK4 and quadratic splines. The simulation time is from $a = 0$ until $b = 2.5$ and there are $N = 100$ intervals in the integration mesh.

```
>> N=100;
>> t = [0:2.5/N:2.5];
>> x0 = [1;0;3.5];
```

```
>> u0 = ones(2,N+2);                    % u0(t)=[1;1];
>> [j,x] = simulate(1,x0,u0,t,4,2);
```

`j = simulate(2,f_number,1)`
`[du,dx0,p] = simulate(2,f_number,action)`

This form allows function values and gradients to be computed without re-simulating the system. A call to this form must be proceeded by a call to **simulate**, form 1. The results are computed from the most recent inputs (`x0,u,t,ialg`) for the call to **simulate**, form 1. The following table shows the relationship between the value of `f_number`, and the function value or gradient which is computed.

Table S5

`f_number` range	Function	Function to be evaluated
$1 \le$ `f_number` $\le n_1$	$g_o^\nu(\xi, x(b)) + \int_a^b l_o^\nu(t, x, u)dt$	$\nu =$ `f_number`
$n_1 <$ `f_number` $\le n_2$	$l_{ti}^\nu(t, x(t), u(t))$	$\nu = n\%(N+1)+1$, $t = t_k$ where $n =$ `f_number` $- n_1 - 1$ and $k =$ `f_number` $- n_1 - \nu(N+1)$.
$n_2 <$ `f_number` $\le n_3$	$g_{ei}^\nu(\xi, x(b))$	$\nu =$ `f_number` $- n_2$
$n_3 <$ `f_number` $\le n_4$	$g_{ee}^\nu(\xi, x(b))$	$\nu =$ `f_number` $- n_3$

where $n_1 = q_o$ is the number of objective functions, $n_2 = n_1 + q_{ti}(N+1)$ with q_{ti} the number of trajectory constraints, $n_3 = n_2 + q_{ei}$ with q_{ei} the number of endpoint inequality constraints, and $n_4 = n_3 + q_{ee}$ with q_{ee} the number of endpoint equality constraints. The notation $n\%m$ means the remainder after division of n by m (n modulo m). Thus, for trajectory constraints, the ν-th constraint (with $\nu = n\%(N+1)+1$) is evaluated at time t_k.

If `action=1`, only `du` and `dx0` are returned. If `action=2`, `du`, `dx0` and `p` are returned. The function, **eval_fnc**, provides a convenient interface to this form.

`[xdot,zdot] = simulate(3,x,u,t,{f_num,{k}})`
`[xdot,zdot,pdot] = simulate(3,x,u,t,p,{k})`

This form evaluates (as opposed to integrates) the following quantities: $\dot{x} = h(t, x, u)$, $\dot{z} = l_o^\nu(t, x, u)$, and $\dot{p} = -(\frac{\partial h(t,x,u)^T}{\partial x} p + \frac{\partial l(t,x,u)^T}{\partial x})$ at the times specified by `t`. These functions are evaluated at the points in `t`. If `f_num` is specified, $\nu =$ `f_num`, otherwise $\nu = 1$. The function $l^\nu(\cdot,\cdot,\cdot)$ is evaluated according to Table S5 above. The last input, `k`, can only be supplied if `t` is a single time point. It is used to indicate the discrete-time interval containing `t`. That is, `k` is such that $t_k \le$ `t` $< t_{k+1}$. If `k` is given, **l** is called with `neq[3]` $= k - 1$. In this call, the values in `u` represent pointwise values of $u(t)$, not its spline coefficients. The inputs `x` and `u` must have the same number of columns as `t`.

[h_x,h_u,l_x,l_u] = simulate(4,x,u,t,{f_num,{k}})

This form evaluates $\frac{\partial h(t,x,u)}{\partial x}$, $\frac{\partial h(t,x,u)}{\partial u}$, $\frac{\partial l^\nu(t,x,u)}{\partial x}$, and $\frac{\partial l^\nu(t,x,u)}{\partial x}$. In this call, `t` must be a single time point. If `f_num` is specified, $\nu =$ `f_num`, otherwise $\nu = 1$. The function $l^\nu(\cdot,\cdot,\cdot)$ is evaluated according to Table S5 above. The last input, `k`, indicates the discrete-time interval containing `t`. That is, `k` is such that $t_k \le$ `t` $< t_{k+1}$. If `k` is given, **l** is called with `neq[3]` $= k - 1$. In this call, the values in `u` represent pointwise values of $u(t)$, not its spline coefficients.

[g,g_x0,g_xf] = simulate(5,x0,xf,tf,{f_num})

This form evaluates $g^\nu(\texttt{x0},\texttt{xf})$, $\frac{\partial g^\nu(\texttt{x0},\texttt{xf})}{\partial \texttt{x0}}$, and $\frac{\partial g^\nu(\texttt{x0},\texttt{xf})}{\partial \texttt{xf}}$. If `f_num` is specified, $\nu =$ `f_num`. Otherwise $\nu = 1$. The input `tf` gets passed to the user functions **g** and **Dg** (see descriptions in §2) for compatibility with future releases of RIOTS.

stats = simulate(6)

This form provides statistics on how many times the functions **h** and **Dh** have been evaluated, how many times the system has been simulated to produce the trajectory x, and how many times functions or the gradients of $f^\nu(\cdot,\cdot)$, $g^\nu(\cdot,\cdot)$ or $l^\nu_{ti}(\cdot,\cdot,\cdot)$ have been computed. The following table indicates what the components of `stats` represent:

Table S6

Component	Meaning
`stats(1)`	Number of calls to **h**.
`stats(2)`	Number of calls to **Dh**.
`stats(3)`	Number of simulations.
`stats(4)`	Number of function evaluations.
`stats(5)`	Number of gradient evaluations.

`lte = simulate(7)`

This form, which must be preceded by a call to **simulate**, form 1 with `ialg=1,2,3,4`, returns estimates of the local truncation error for the fixed step-size Runge-Kutta integration routines. The local truncation error is given, for $k = 1, \ldots, N$, by

$$lte_k = \begin{pmatrix} x_k(t_{k+1}) - x_{N,k+1} \\ z_k(t_{k+1}) - z_{N,k+1} \end{pmatrix},$$

where $x_k(t_{k+1})$ and $z_k(t_{k+1})$ are the solutions of

$$\begin{pmatrix} \dot{x} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} h(x,u) \\ l_o^1(t,x,u) \end{pmatrix}, \quad \begin{matrix} x(t_k) = x_{N,k} \\ z(t_k) = 0 \end{matrix}, \quad t \in [t_k, t_{k+1}] \, .$$

and $x_{N,k+1}$ and $z_{N,k+1}$ are the quantities computed by one Runge-Kutta step from $x_{N,k}$ and 0, respectively. These local truncations errors are estimated by taking double integration steps as described in Section 4.3.1. The local truncation error estimates are used by **distribute** (see description in §7) to redistribute the integration mesh points in order to increase integration accuracy.

IMPLEMENTATION OF THE INTEGRATION ROUTINES

Here we discuss some of the implementation details of the different integration routines built into **simulate**.

System Simulation

System simulation is accomplished by forward integration of the differential equations used to describe the system. There are four fixed step-size Runge-Kutta integrators, one variable step-size integrator (LSODA), and one discrete-time solver. The RK integrators and LSODA produce approximate solutions to the system of differential equation

$$\dot{x} = h(t, x, u)\ ,\ \ x(a) = \xi$$

$$\dot{z} = l(t, x, u)\ ,\ \ z(a) = 0$$

on the interval $t \in [a, b]$. The Runge-Kutta integrators, described by the Butcher arrays $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$ and $\mathbf{A}_4$ given in Chapter 4.2, are of order 1, 2, 3 and 4 respectively. The discrete-time integrator solves

$$x_{k+1} = h(t_k, x_k, u_k)\ ,\ \ x_0 = \xi$$

$$z_{k+1} = l(t_k, x_k, u_k)\ ,\ \ z_0 = 0$$

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

The variable step-size integrator is a program called LSODA [127,128]. LSODA can solve both stiff and non-stiff differential equations. In the non-stiff mode, LSODA operates as an Adams-Moulton linear, multi-step method. If LSODA detects stiffness, it switches to backwards difference formulae. When operating in stiff mode, LSODA requires the system Jacobians $\frac{\partial h(t,x,u)}{\partial x}$ and $\frac{\partial l(t,x,u)}{\partial x}$. If the user has not supplied these functions, LSODA must be called using `ialg=6` so that these quantities will be computed using finite-difference approximations. Otherwise, LSODA should be called using `ialg=5` so that the analytic expressions for these quantities will be used.

The integration precision of LSODA is controlled by a relative tolerance and an absolute tolerance. These both default to $1e-8$ but can be specified in `ialg(3:4)` respectively (see description of **simulate**, form 1). The only non-standard aspect of the operation of LSODA by **simulate** is that the integration is restarted at every mesh point t_k due to discontinuities in the control spline $u(\cdot)$, or its derivatives, at these points.

Gradient Evaluation

In this section we discuss the computation of the gradients of the objective and constraint functions of problem **OCP** with respect to the controls and free initial conditions. These gradients are computed via backwards integration of the adjoint equations associated with each function.

Discrete-time Integrator. For the discrete-time integrator, the adjoint equations and gradients are given by the following equations. For the objective functions, $\nu \in \mathbf{q}_o$, $k = N, \ldots, 1$,

$$p_k = h_x(t_k, x_k, u_k)^T p_{k+1} + l_x^\nu(t_k, x_k, u_k)^T \;\; ; \;\; p_{N+1} = \frac{\partial g^\nu(\xi, x_{N+1})^T}{\partial x_{N+1}}$$

$$\left[\frac{df^\nu(\xi, u)}{du} \right]_k^T = h_u(t_k, x_k, u_k)^T p_{k+1} + l_u^\nu(t_k, x_k, u_k)^T$$

$$\frac{df^\nu(\xi, u)^T}{d\xi} = \frac{\partial g^\nu(\xi, x_{N+1})^T}{\partial \xi} + p_0 \; .$$

For the endpoint constraints, $\nu \in \mathbf{q}_{ei} \cap \mathbf{q}_{ee}$, $k = N, \ldots, 1$,

$$p_k = h_x(t_k, x_k, u_k)^T p_{k+1} \;\; ; \;\; p_{N+1} = \frac{\partial g^\nu(\xi, x_{N+1})^T}{\partial x_{N+1}}$$

$$\left[\frac{dg^\nu(\xi, x_{N+1})}{du} \right]_k^T = h_u(t_k, x_k, u_k)^T p_{k+1}$$

$$\frac{dg^\nu(\xi, x_{N+1})^T}{d\xi} = \frac{\partial g^\nu(\xi, x_{N+1})^T}{\partial \xi} + p_1 \; .$$

For the trajectory constraints, $\nu \in \mathbf{q}_{ti}$, evaluated at the discrete-time index $l \in \{ 1, \ldots, N+1 \}$,

$$p_k = h_x(t_k, x_k, u_k)^T p_{k+1} \; , \;\; k = l-1, \ldots, 1 \;\; ; \;\; p_l = l_x^\nu(t_l, x_l, u_l)^T$$

$$\left[\frac{dl^\nu(t_k, x_k, u_k)}{du} \right]_k^T = \begin{cases} h_u(t_k, x_k, u_k)^T p_{k+1} & k = 1, \ldots, l-1 \\ l_u^\nu(t_k, x_k, u_k)^T & k = l \\ 0 & k = l+1, \ldots, N \end{cases}$$

$$\frac{dl^\nu(t_l, x_l, u_l)^T}{d\xi} = p_1 \; .$$

Runge-Kutta Integrators. For convenience, we introduce the notation

$$u_{k,j} = u(\tau_{k,i_j}) \ , \ \ k = 1, \ldots, N \ , \ \ j = 1, .., r \ ,$$

where

$$\tau_{k,j} \doteq t_k + c_j \Delta_k$$

and $c_i \in [0, 1]$ are parameters of the Runge-Kutta integration method. For RK1, RK2 and RK3, $r = s$ where s is the number of stages and $i_j = j$. However, because RK4 has a repeated control sample (*cf.* Chap 2.4), we have $r = 3$, $i_1 = 1$, $i_2 = 2$ and $i_3 = 4$.

The computation of the control gradients is a two-step process. First, the gradient of $f(\xi, u)$ with respect to the control samples $u_{k,j}$, $k = 1, \ldots, N$, $j = 1, \ldots, r$, where r is the number of control samples per integration interval, and with respect to ξ is computed. Second, the gradient with respect to the spline coefficients, α_k, of $u(t)$ is computed using the chain-rule as follows,

$$\frac{df(\xi, u)}{d\alpha_k} = \sum_{i=1}^{N} \sum_{j=1}^{r} \frac{df(\xi, u)}{du_{i,j}} \frac{du_{i,j}}{d\alpha_k} \ , \ \ k = 1, \ldots, N + \rho - 1 \ ,$$

where ρ is the order of the spline representation. Most of the terms in the outer summation are zero because the spline basis elements have local support. The quantity

$$\frac{du_{i,j}}{d\alpha_k} = \phi_{\mathbf{t}_N, r, k}(\tau_{i,j})$$

is easily determined from the recurrence relation (2.7.2a) for the B-spline basis.

A general formula for $df/du_{i,j}$ is given in Theorem 2.5.1. However, due to the special structure of the specific RK methods used by **simulate** there is a much more efficient formula, discovered by Hager [42]. We have extended Hager's formula to deal with the various constraints and the possibility of repeated control samples (see Chapter 2.4). To describe this formula, we use the notation for $k = 1, \ldots, N - 1$ and $j = 1, \ldots, s$,

$$A_{k,j} \doteq h_x(\tau_{k,j}, y_{k,j}, u(\tau_{k,j}))^T \ ,$$

$$B_{k,j} \doteq h_u(\tau_{k,j}, y_{k,j}, u(\tau_{k,j}))^T \ ,$$

$$lx^{\nu}_{k,j} \doteq l^{\nu}_x(\tau_{k,j}, y_{k,j}, u(\tau_{k,j}))^T \ ,$$

and

$$lu^{\nu}_{k,j} \doteq l^{\nu}_u(\tau_{k,j}, y_{k,j}, u(\tau_{k,j}))^T \ .$$

where, with $a_{k,j}$ parameters of the Runge-Kutta method,

$$y_{k,1} \doteq x_k \ , \quad y_{k,j} \doteq x_k + \Delta_k \sum_{m=1}^{j-1} a_{j,m} h(y_{k,m}, u(\tau_{k,m})) \ , \ j = 2, \ldots, s \ .$$

The quantities $y_{k,j}$ are estimates of $x(\tau_{k,j})$, see equation (2.4.3d).

The gradients of the objective and constraint functions with respect to the controls $u_{k,j}$ and the initial conditions ξ are given as follows. In what follows $a_{s+1,j} \doteq b_j$, $q_{k,s} = q_{k+1,0}$ and the standard adjoint variables, p_k, are given by $p_k = q_{k,0}$. For the objective functions, we have for $\nu \in \mathbf{q}_o$, $k = N, \ldots, 1$,

$$q_{N+1,0} = \frac{\partial g^\nu(\xi, x_{N+1})^T}{\partial x_{N+1}} \ ,$$

$$q_{k,j} = q_{k+1,0} + \Delta_k \sum_{m=j+1}^{s} a_{s-j+1,s-m+1}(A_{k,m} q_{k,m} + l x^\nu_{k,m+1}) \ , \ \ j = s-1, s-2, \ldots, 0$$

$$\left[\frac{df^\nu(\xi, u)}{du} \right]_{k,j}^T = b_j \Delta_k \left(B_{k,j} q_{k,j} + l^\nu_{k,j} \right), \ \ j = 1, \ldots, s \ ,$$

$$\frac{df^\nu(\xi, u)^T}{d\xi} = \frac{\partial g^\nu(\xi, x_N)^T}{\partial \xi} + q_1^0 \ ,$$

For the endpoint constraints, we have for $\nu \in \mathbf{q}_{ei} \cap \mathbf{q}_{ee}$, $k = N, \ldots, 1$,

$$q_{k,j} = q_{k+1,0} + \Delta_k \sum_{m=j+1}^{s} a_{s-j+1,s-m+1} A_{k,m} q_{k,m} \ , \ \ j = s-1, s-2, \ldots, 0 \ ; \ \ q_{N+1,0} = \frac{\partial g^\nu(\xi, x_{N+1})^T}{\partial x_{N+1}} \ ,$$

$$\left[\frac{dg^\nu(\xi, x_{N+1})}{du} \right]_{k,j}^T = b_j \Delta_k B_{k,j} q_{k,j} \ , \ \ j = 1, \ldots, s$$

$$\frac{dg^\nu(\xi, x_{N+1})^T}{d\xi} = \frac{\partial g^\nu(\xi, x_{N+1})^T}{\partial \xi} + q_1^0 \ ,$$

For the trajectory constraints, $\nu \in \mathbf{q}_{ti}$, evaluated at the the discrete-time index $l \in \{ 1, \ldots, N+1 \}$,

$$q_l^0 = l_x^\nu(t_l, x_l, u(\tau_{l,s}))^T$$

$$q_{k,j} = q_{k+1,0} + \Delta_k \sum_{m=j+1}^{s} a_{s-j+1,s-m+1} A_{k,m} q_{k,m} \ , \ \ k = l-1, .., 1 \ , \ \ j = s-1, s-2, \ldots, 0 \ ;$$

$$\left[\frac{dl^\nu(t_k, x_k, u(t_k))}{du}\right]^T_{k,j} = \begin{cases} b_j \Delta_k B_{k,j} q_{k,j} & k = 1, \ldots, l-1\ ,\ \ j = 1, \ldots, s \\ l^\nu_u(t_k, x_k, u(\tau_{k,j})) & k = l\ \ ;\ \ j = 0 \text{ if } l \le N, \text{ else } j = s \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{dl^\nu(t_l, x_l, u(t_l))^T}{d\xi} = q^0_1\ .$$

For method RK4, we have the special situation that $\tau_{k,2} = \tau_{k,3}$ for all k because $c_2 = c_3 = 1/2$. Hence, there is a repeated control sample: $u_{k,2} = u(\tau_{k,2}) = u(\tau_{k,3})$. Thus, for any function f, the derivatives with respect to $u_{k,1}$, $u_{k,2}$ and $u_{k,3}$ are given by the expressions,

$$\frac{df}{du_{k,1}} = \left[\frac{df}{du}\right]_{k,1}\ ,\quad \frac{df}{du_{k,2}} = \left[\frac{df}{du}\right]_{k,2} + \left[\frac{df}{du}\right]_{k,3}\ ,\quad \frac{df}{du_{k,3}} = \left[\frac{df}{du}\right]_{k,4}\ .$$

Variable Step-Size Integrator (LSODA). For the variable step-size integrator, LSODA, the adjoint equations and gradients are given by the equations below which require knowledge of $x(t)$ for all $t \in [a, b]$. As in [25], $x(t)$ is stored at the internal knots $\{\ t_k + \frac{i}{n_{knots}+1}\Delta_k\ \}_{i=0,k=1}^{n_{knots}+1,N+1}$ during the forward system integration. By default, $n_{knots} = 1$, but the user can specify $n_{knots} \ge 1$ by setting `ialg(2)` = n_{knots} (see description of **simulate**, form 1). Then, during the computation of the adjoints and gradients, $x(t)$ is determined by evaluating the quintic† Hermite polynomial which interpolates $(t, x(t), \dot{x}(t))$ at the nearest three internal knots within the current time interval $[t_k, t_{k+1}]$. Usually $n_{\text{knots}} = 1$ is quite sufficient.

We now give the formulae for the adjoints and the gradients. It is important to note that, unlike the fixed step-size integrators, the gradients produced by LSODA are not exact. Rather, they are numerical approximations to the continuous-time gradients for the original optimal control problem. The accuracy of the gradients is affected by the integration tolerance and the number of internal knots used to store values of $x(t)$. Under normal circumstances, the gradients will be less accurate than the integration tolerance. For the objective functions, $\nu \in \mathbf{q}_o$,

$$\dot{p} = -\,(h_x(t, x, u)^T p + l^\nu_x(t, x, u)^T)\ ,\ \ t \in [a, b]\ \ ;\ \ p(b) = \frac{\partial g^\nu(\xi, x(b))^T}{\partial x(b)}$$

† The order of the Hermite polynomial can be changed by setting the define'd symbol ORDER in the code adams.c. If the trajectories are not at least five time differentiable between breakpoints, then it may be helpful to reduce the ORDER of the Hermite polynomials and increase n_{knots}.

$$\left[\frac{df^{\nu}(\xi,u)}{d\alpha_k}\right]^T = \int_a^b (h_u(t,x,u)^T p(t) + l_u^{\nu}(t,x,u)^T)\phi_{\mathbf{t}_N,\rho,k}(t)dt\ ,\ \ k = 1,\ldots,N+\rho-1$$

$$\frac{df^{\nu}(\xi,u)^T}{d\xi} = \frac{\partial g^{\nu}(\xi,x(b))^T}{\partial\xi} + p(a)\ .$$

For the endpoint constraints, $\nu \in \mathbf{q}_{ei} \cap \mathbf{q}_{ee}$,

$$\dot{p} = -h_x(t,x,u)^T p\ ,\ \ t \in [a,b]\ \ ;\ \ p(b) = \frac{\partial g^{\nu}(\xi,x(b))^T}{\partial x(b)}$$

$$\left[\frac{dg^{\nu}(\xi,u)}{d\alpha_k}\right]^T = \int_a^b h_u(t,x,u)^T p(t)\,\phi_{\mathbf{t}_N,\rho,k}(t)dt\ ,\ \ k = 1,\ldots,N+\rho-1$$

$$\frac{dg^{\nu}(\xi,u)^T}{d\xi} = \frac{\partial g^{\nu}(\xi,x(b))^T}{\partial\xi} + p(a)\ .$$

For the trajectory constraints, $\nu \in \mathbf{q}_{ti}$, evaluated at time $t = t_l$, $l \in \{1,\ldots,N+1\}$,

$$\dot{p} = -h_x(t,x,u)^T p\ ,\ \ t \in [a,t_l]\ \ ;\ \ p(t_l) = l_x^{\nu}(t_l,x(t_l),u(t_l))^T$$

$$\left[\frac{dl^{\nu}(t_l,x_l,u(t_l))}{d\alpha_k}\right]^T = \int_a^{t_l} h_u(t,x,u)^T p(t)\,\phi_{\mathbf{t}_N,\rho,k}(t)dt\ ,\ \ k = 1,\ldots,N+\rho-1$$

$$\frac{dl^{\nu}(t_l,x_l,u(t_l))^T}{d\xi} = p(a)\ .$$

The numerical evaluation of the integrals in these expressions is organized in such a way that they are computed during the backwards integration of $\dot{p}(t)$. Also, the computation takes advantage of the fact that the integrands are zero outside the local support of the spline basis elements $\phi_{\mathbf{t}_N,\rho,k}(t)$.

check_deriv

Purpose

This function provides a check for the accuracy of the user-supplied derivatives **Dh**, **Dl** and **Dg** by comparing these functions to derivative approximations obtained by applying forward or central finite-differences to the corresponding user-supplied function **h**, **l** and **g**.

Calling Syntax

```
[errorA,errorB,max_error] = check_deriv(x,u,t,{params},{index},
                                         {central},{DISP})
```

Description

The inputs $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ and $t \in \mathbb{R}$ give the nominal point about which to evaluate the derivatives $h_x(t,x,u)$, $h_u(t,x,u)$, $l_x^\nu(t,x,u)$, $l_u^\nu(t,x,u)$, $g_x^\nu(t,x,u)$ and $g_u^\nu(t,x,u)$. If there are system parameters (see description of **init** in §3), they must supplied by the input `params`. If specified, `index` indicates the discrete-time index for which `t(index)` ≤ `t` ≤ `t(index+1)`. This is only needed if one of the user-supplied system functions uses the discrete-time index passed in `neq[3]`.

The error in each derivative is estimated as the difference between the user-supplied derivative and its finite-difference approximation. For a generic function $f(x)$, this error is computed, with e_i the i-th unit vector and δ_i a scalar, as

$$E = \frac{f(x) - f(x+\delta_i e_i)}{\delta_i} - \frac{df(x)}{dx}\, e_i \ ,$$

if forward differences are used, or

$$E = \frac{f(x-\delta_i e_i) - f(x+\delta_i e_i)}{2\delta_i} - \frac{df(x)}{dx}\, e_i \ ,$$

if central differences are used. The perturbation size is $\delta_i = \varepsilon_{\mathrm{mach}}^{1/3} \max\{\, 1, |x_i| \,\}$. Central difference approximations are selected by setting the optional argument `central` to a non-zero value. Otherwise, forward difference approximations will be used.

The first term in the Taylor expansion of E with respect to δ_i is of order is $O(\delta_i^2)$ for central differences and $O(\delta_i)$ for forward differences. More details can be found in [129, Sec. 4.1.1]. Thus, it is sometimes useful to perform both forward and central difference approximations to decide whether a large difference between the derivative and its finite-difference approximations is due merely a result of scaling or if it is actually due to an error in the implementation of the user-supplied derivative. If the derivative is is correct then E should decrease substantially when central differences are used.

If `DISP=0`, only the maximum error is displayed.

The outputs `errorA` and `errorB` return the errors for $h_x(t, x, u)$ and $h_u(t, x, u)$ respectively. The output `max_error` is the maximum error detected for all of the derivatives.

Example

The following example compares the output from **check_deriv** using forward and central finite-differences. The derivatives appear to be correct since the errors are much smaller when central differences are used. First forward differences are used, then central differences.

```
>> check_deriv([-5;-5],0,0);
=========================================================================
              System matrices:
Error in h_x =
   1.0e-04 *

         0   -0.0000
   -0.0000   -0.6358

Error in h_u =
   1.0e-10 *

         0
    0.9421

              For function 1:
Error in l_x =    1.0e-04 *

   -0.3028         0

Error in l_u =    6.0553e-06

              For function 1:
Error in g_x0 =      0     0

Error in g_xf =      0     0

Maximum error reported is 6.35823e-05.
=========================================================================
```

```
>> check_deriv([-5;-5],0,0,[],0,1);
========================================================================
              System matrices:
Error in h_x =
   1.0e-10 *

         0   -0.0578
   -0.2355   -0.3833

Error in h_u =
   1.0e-10 *

         0
    0.9421

              For function 1:
Error in l_x =    1.0e-10 *

    0.5782         0

Error in l_u =      0

              For function 1:
Error in g_x0 =      0     0

Error in g_xf =      0     0

Maximum error reported is 9.42135e-11.
========================================================================
```

See Also: **check_grad**

check_grad

Purpose

This function checks the accuracy of gradients of the objective and constraint functions, with respect to controls and initial conditions, as computed by **simulate**, forms 1 and 2. It also provides a means to indirectly check the validity of the user-supplied derivative **Dh**, **Dl** and **Dg**.

Calling Syntax

```
max_error = check_grad(i,j,k,x0,u,t,ialg,{params},{central},
                          {DISP})
```

Description

The input `x0`, `u`, `t` and `ialg` specify the inputs to the nominal simulation **simulate(1,x0,u,t,ialg,0)** prior to the computation of the gradients. The gradients are tested at the discrete-time indices as specified in the following table:

Index	Purpose
i	Spline coefficient control u that will be perturbed. If `i=0`, the gradients with respect to u will not be checked.
j	Index of initial state vector, ξ, that will be perturbed. If `j=0`, the gradients with respect to the ξ will not be checked.
k	For each trajectory constraints, `k` indicates the discrete-time index, starting with `k=1`, at which the trajectory constraints will be evaluated. If `k=0`, the trajectory constraint gradients will not be checked.

The finite-difference computations are the same as described for **check_deriv**.

If there are system parameters (see description of **init**, § 3), these must be given by the input `params`. Central difference approximations will be used if a non-zero value for `central` is specified; otherwise forward differences will be used. If `DISP=0`, only the maximum error is displayed. This is particularly useful if **check_deriv** is used in a loop on any of the indices `i,j,k`. The output `max_error` is the maximum error detected in the gradients.

Example

The following example checks the tenth component of the control gradient and the second component of initial condition gradient as computed by RK2 using central differences. The integration is performed on the time interval $t \in [0, 2.5]$ with $N = 50$ intervals. The gradients are evaluated for the second order spline control $u(t) = 1$ for all t (*i.e.*, $\alpha_k = 1$, $k = 1, \ldots, N+1$).

```
>> t = [0:2.5/50:2.5];
>> u = ones(1,51);
>> x0 = [-5;-5];
>> check_grad(10,2,0,x0,u,t,2,[],1);
============================================================================
Using perturbation size of 6.05545e-06

                    Evaluating function 1.
error_u  = 1.84329e-09
error_x0 = -4.88427e-11
Relative error in control gradient = 2.52821e-07%
Gradient OK

Relative error in x0 gradient = 1.14842e-09%
Gradient OK

                    Evaluating endpoint constraint 1.
error_u  = -5.46737e-11
error_x0 = -5.98271e-12
Relative error in control gradient = 6.04337e-08%
Gradient OK

Relative error in x0 gradient = 1.87846e-09%
Gradient OK
Maximum error reported is 1.84329e-09.
============================================================================
```

See Also: **check_deriv**

eval_fnc

Purpose

This function provides a convenient interface to **simulate**, form 2, for computing function and gradient values. A system simulation must already have been performed for this function to work.

Calling Syntax

```
[f,du,dx0,p] = eval_fnc(type,num,k)
```

Description of Inputs

`type` A string that specifies the type of function to be evaluated. The choices are

`'obj'`	Objective function
`'ei'`	Endpoint inequality constraint
`'ee'`	Endpoint equality constraint
`'traj'`	Trajectory constraint

`num` Specifies ν for the function of the type specified by `type` is to be evaluated.

`k` For trajectory constraints only. Specifies the index for the time, t_k, in the current integration mesh at which to evaluate the trajectory constraint. If `k` is a vector, the trajectory constraint will be evaluated at the times specified by each mesh point index in `k`.

Description of Outputs

`f` The function value.

`du` The gradient with repect to u. Not computed for trajectory constraints if `index` is a vector.

`dx0` The derivative of the function with respect to initial conditions, ξ. Not computed for trajectory constraints if `index` is a vector.

`p` The adjoint trajectory. Not computed for trajectory constraints if `index` is a vector.

Examples

The following examples assume that a simulation has already been performed on a system that has at least two endpoint equality constraints and a trajectory constraint. The first call to **eval_fnc** evaluates the second endpoint equality constraint.

```
>> f=eval_fnc('ee',2)

f =

    0.2424
```

Since equality constraints should evaluate to zero, this constraint is violated. This next call evaluates the first trajectory constraint at the times t_k, $k = 5, \ldots, 15$, in the current integration mesh.

```
>> eval_fnc('traj',1,5:15)

ans =

  Columns 1 through 7

   -1.0182   -1.0222   -1.0258   -1.0288   -1.0311   -1.0327   -1.0338

  Columns 8 through 11

   -1.0335   -1.0318   -1.0295   -1.0265
```

Since inequality constraints are satisfied if less than or equal to zero, this trajectory constraint is satisfied at these specified points.

6. OPTIMIZATION PROGRAMS

This section describes the suite of optimization programs that can be used to solve various cases of the optimal control problem **OCP**. These programs seek local minimizers to the discretized problem. The most general program is **riots** which converts **OCP** into a mathematical program which is solved using standard nonlinear programming techniques. Currently, **riots** can be linked with one of two sequential quadratic programming (SQP) algorithm as described later. Besides being able to solve the largest class of optimal control problems, **riots** is also the most robust algorithm amongst the optimization programs available in RIOTS. However, it can only handle medium size problems. The size of a problem, the number of decision variables, is primarily determined by the number of control inputs and the discretization level. What is meant by medium size problems is discussed in the description of **riots**.

The most restrictive program is **pdmin** which can solve optimal control problems with constraints consisting of only simple bounds on ξ and u. State constraints are not allowed. The algorithm used by **pdmin** is the projected descent method described in Chapter 3. Because of the efficiency of the projected descent method, **pdmin** can solve large problems.

Problems that have, in addition to simple bounds on u and ξ, endpoint equality constraints can be solved by **aug_lagrng**. The algorithm is a multiplier method which relies upon **pdmin** to solve a sequence of problems with only simple bound constraints. This program provides a good example of how the toolbox style of RIOTS can be used to create a complex algorithm from a simpler one. Currently, the implementation of **aug_lagrng** is fairly naive and has a great deal of room left for improvement. Also, it would be relatively straightforward to add an active set strategy to **aug_lagrng** in order to allow it to handle inequality constraints.

Finally, the program **outer** is an experimental outer loop which repeatedly calls **riots** to solve a sequence of increasingly accurate discretizations (obtained by calls to **distribute**) of **OCP** in order to efficiently compute the optimal control to a specified accuracy.

Choice of Integration and Spline Orders.

Each of these optimization programs requires the user to select an integration routine and the order of the spline representation for the controls. There are several factors involved in these selections. Some of these factors are discussed below and summarized in the Table O2 that follows. Consult Chapter 4.2 for a more in-depth discussion.

Fixed step-size integration. The first consideration is that, for each of the fixed step-size Runge-Kutta methods, there is a limit to how much accuracy can be obtained in the control solutions at certain discrete time points. The accuracy, $\|u_N^* - u^*\|$, of the control splines can not be greater than the solutions at these time points. The order of the accuracy of spline solutions with respect to the discretization level for *unconstrained* problems is given in the following table. The quantity Δ used in this table is defined as $\Delta \doteq \max_k t_{N,k+1} - t_{N,k}$. The third column is a reminder of the spline orders that are allowed by **simulate** for each RK method.

Table O1

RK Method	Order of Accuracy	Allowable Spline Orders
1	$O(\Delta^1)$	1
2	$O(\Delta^2)$	2
3	$O(\Delta^2)$	2
4	$O(\Delta^3)$	2, 3, 4

While it is possible with some optimal control problems to achieve higher order accuracies, this is a non-generic situation. The order of spline representation should therefore not exceed the accuracies listed in the second column of this table. Thus, for RK4, even though cubic splines are allowed there is usually no reason to use higher than quadratic splines ($\rho = 3$).

The orders listed in the above table are usually only achieved for unconstrained problems. For problems with control constraints it is typically impossible to achieve better than first order accuracy. This is even true if the discontinuities in the optimal control are known *a priori* since the locations of these discontinuities will not coincide with the discontinuities of the discretized problems. For problems with state constraints, the issue is more complicated. In general, we recommend using second order splines (except for Euler's method) for problems with control and/or trajectory constraints. Even if first order accuracy is all that can be achieved, there is almost no extra work involved in using second order splines. Furthermore, second order splines will often give somewhat better results than first order splines even if the accuracy is asymptotically limited to first order.

A second consideration is that the overall solution error is due to both the integration error and the error caused by approximating an infinite dimensional function, the optimal control, with a finite dimensional spline. Because of the interaction of these two sources of error and the fact that the accuracy of the spline representations is limited to the above table, improving the integration accuracy by using a higher order method does not necessarily imply that the accuracy of the solution to the approximating problem will improve. However, even if the spline accuracy is limited to first order, it is often the case that the integration error, which is of order $O(\Delta^s)$, where s is the order of the RK method, still has a significantly greater effect on the overall error than the spline error (especially at low discretization levels). This is partly due to the fact that errors in the control are integrated out by the system dynamics. Thus, it is often advantageous to use higher-order integration methods even though the solution error is asymptotically limited to first order by the spline approximation error.

The importance of the RK order, in terms of reducing the overall amount of computational work required to achieve a certain accuracy, depends on the optimization program being used. Each iterations of **riots** requires the solution of one or more dense quadratic program. The dimension of these quadratic programs is equal to the number of decision parameters (which is $m(N + \rho - 1)$ plus the number of free initial conditions). Because the work required to solve a dense quadratic program goes up at least cubically with the number of decision variables, at a certain discretization level most of the work at each iteration will be spent solving the quadratic program. Thus, it is usually best to use the fourth order RK method to achieve as much accuracy as possible for a given discretization level. An exception to this rule occurs when problem **OCP** includes trajectory constraints. Because a separate gradient calculation is performed at each mesh point for each trajectory constraint, the amount of work increases significantly as the integration order is increased. Thus, it may be beneficial to use a RK3 or even RK2 depending on the problem.

On the other hand, for the optimization programs **pdmin** and **aug_lagrng** (which is based on **pdmin**) the amount of work required to solve the discretized problem is roughly linear in the number of decision variables which is basically proportional to the discretization level N. The amount of work required to integrate the differential equations is linearly proportional to Ns where s the order of the Runge-Kutta method. Since the integration error is proportional to $1 / N^s$, if not for the error for the spline approximation it would always be best to use RK4. However, because there is error from the finite dimensional spline representation, it does not always pay to use the highest order RK method. If, roughly speaking, the error from the control representation contributes to the overall error in the numerical solution to larger extent than the integration error (note that the spline error and the integration error are in different units) then it is wasteful to use a higher order RK method. This usually happens only at high discretization levels.

The relative effect of the spline error versus the integration error depends on the nature of the system dynamics and the smoothness of the optimal control. Unfortunately, this is hard to predict in advance. But a sense of the balance of these errors can be obtained by solving, if possible, the problem at a low discretization level and viewing the solution using **sp_plot** and using **simulate** (form 7) or **est_errors** to obtain an estimate of the integration error.

There is a third consideration for selecting the integration order. For some problems with particularly nonlinear dynamics, in may not be possible integrate the differential equation if the discretization level is too small. In these cases, the minimum discretization level needed to produce a solution is smallest when using RK4. For some problems, it may not be possible to achieve an accurate solution of the differential equation at any reasonable discretization level. For these problems, the variable step-size integration method, discussed next, will have to be used.

Regardless of the integration method used, higher order splines ($\rho > 2$) should not be used unless the optimal control is sufficiently smooth. Of course, the optimal control is not known in advanced. Generally, though, when solving control and/or trajectory constrained problems, second order splines should be used (except with Euler's method which can only use first order splines) as per the discussion above. For other problems being integrated with RK4, it may be advantageous to use quadratic splines.

The following table provides a set of basic guidelines for the selection of the integration method and the spline order for solving different classes of problems. These choices may not be ideal for any specific problem but they are generally acceptable for most problems.

Table O2

type of problem	optimization program	RK order (`ialg`)	spline order (ρ)
no control nor trajectory constraints	**pdmin/aug_lagrng**	4 (N small) 2 (N large)	3 (N small) 2 (N large)
	riots	4	3
control constraints	**pdmin/aug_lagrng**	4 (N small) 2 (N large)	2
	riots	4	
trajectory constraints	**riots**	2[†]	2

Variable step-size integration. From the point of view of integrating differential equations, it is much more efficient to use a variable step-size integration routine than a fixed step-size method. However, this is usually not the case when solving optimal control problems. There are three basic reasons for this. First, the overall solution accuracy cannot exceed the accuracy with which splines can approximate the optimal control. Thus, it is quite conceivable that a great deal of work will be spent to achieve a very accurate integration but this effort will be wasted on a relatively inaccurate solution. Second, the solution of the discretized problem can easily involve

[†] Sometimes a higher-order method must be used to provide a reasonable solution to the system differential equations.

hundreds of simulations. The integration accuracy during most of the simulations will have very little affect on the accuracy of the final solution. Therefore, it is usually much more efficient to solve a sequence of discretized problems, each with a more accurate integration mesh, using a fast, fixed step-size integration method. Third, the gradients produced for the variable step-size method are approximations to the actual, continuous-time gradients for the original problem **OCP**; they are not exact gradients for the discretized problems. Thus, the solution of the discretized problem will usually require more iterations and will be less accurate (relative to the actual solution of the discretized problem) when using the variable step-size method than when using one of the fixed step-size integration routines.

There are, however, situations in which it is best to use the variable step-size integration method. The first situation is when the system dynamics are very difficult to integrate. In this case, or any other case in which the the integration error greatly exceeds the spline approximation error, it is more efficient to use the variable step-size method. In some cases, the integration has to be performed using the variable step-size method. This can occur if the system is described by stiff differential equations or if the system contains highly unstable dynamics.

Another situation in which it can be advantageous to use the variable step-size integration method is if the location of discontinuities in the optimal control, or discontinuities in the derivatives of the optimal control, are known *a priori*. In this case, it may be possible to increase the solution accuracy by placing breakpoints in the discretization mesh where these discontinuities occur and then using a spline of order one greater than the overall smoothness of the optimal control[†]. The location of the discontinuity for the discretized problem will be very close to the discontinuity in the optimal control if the integration tolerance is small and the optimal control is well-approximated by the spline away from the discontinuity. Hence, the overall accuracy will not be limited by the discontinuity.

The variable step-size integration routine can use first, second, third, or fourth order splines. For unconstrained problems, or problem with endpoint constraints, it is best to use fourth order splines so that the spline approximation error is as small as possible. For problems with control and/or trajectory constraints, first or second order splines are recommended.

[†] A spline of higher order would be too smooth since RIOTS currently does not allow splines with repeated interior knots.

Coordinate Transformation

All of the optimization programs in RIOTS solve finite-dimensional approximations to **OCP** obtained by the discretization procedure described in the introduction of Section 5. Additionally, a change of basis is performed for the spline control subspaces. The new basis is orthonormal. This change of basis is accomplished by computing the matrix $\mathbf{M}_\alpha$ with the property that for any two splines $u(\cdot)$ and $v(\cdot)$ with coefficients α and β,

$$\langle u, v \rangle_{L_2} = \langle \alpha \mathbf{M}_\alpha, \beta \rangle ,$$

(recall that α and β are row vectors. The splines coefficients in the transformed basis are given by $\widetilde{\alpha} = \alpha \mathbf{M}_\alpha^{1/2}$ and $\widetilde{\beta} = \beta \mathbf{M}_\alpha^{1/2}$ (see Section 6 and Remark A.2.8). In the new coordinates,

$$\langle u, v \rangle_{L_2} = \langle \widetilde{\alpha} , \widetilde{\beta} \rangle .$$

In words, the L_2-inner product of any two splines is equal to the Euclidean inner product of their coefficients in the new basis. The matrix $\mathbf{M}_\alpha$ is referred to as the *transform matrix* and the change of basis is referred to as the *coordinate transformation*.

By performing this transformation, the standard inner-product of decision variables (spline coefficients) used by off-the-shelf programs that solve mathematical programs is equal to the function space inner product of the corresponding splines. Also, because of the orthonormality of the new basis, the conditioning of the discretized problems is no worse than the conditioning of the original optimal control problem **OCP**. In practice, this leads to solutions of the discretized problems that are more accurate and that are obtained in fewer iterations than without the coordinate transformation. Also, any termination criteria specified with an inner product become independent of the discretization level in the new basis.

In effect, the coordinate transformation provides a natural column scaling for each row of control coefficients. It is recommended that, if possible, the user attempt to specify units for the control inputs so that the control solutions have magnitude of order one. Choosing the control units in this way is, in effect, a row-wise scaling of the control inputs.

One drawback to this coordinate transformation is that for splines of order two and higher the matrix $\mathbf{M}_\alpha^{-1/2}$ is dense. A diagonal matrix would be preferable for two reasons. First, computing $\mathbf{M}_\alpha^{-1/2}$ is computationally intensive for large N. Second, there would be much less work involved in transforming between bases: each time a new iterate is produced by the mathematical programming software, it has to be un-transformed to the original basis. Also, every gradient computation involves an inverse transformation. Third, simple control bounds are converted into general linear constraints by the coordinate transformation. This point is discussed next.

Control bounds under the coordinate transformation. Simple bounds on the spline coefficients takes the form $a_k \leq \alpha_k \leq b_k$, $k = 1, .., N + \rho - 1$. If a_k and b_k are in fact constants, a and b, then for all t, $a \leq u(t) \leq b$. Now, under the coordinate transformation, simple bounds of this form become

$$(a_1 , \ldots , \alpha_{N+\rho-1}) \leq \tilde{\alpha}\, \mathbf{M}_\alpha^{-1/2} \leq (b_1 , \ldots , b_{N+\rho-1}) .$$

Thus, because of the coordinate transformation, the simple bounds are converted into general linear bounds. Since this is undesirable from an efficiency point of view, RIOTS instead replaces the bounds with

$$(a_1 , \ldots , \alpha_{N+\rho-1})\mathbf{M}_\alpha^{1/2} \leq \tilde{\alpha} \leq (b_1 , \ldots , b_{N+\rho-1})\mathbf{M}_\alpha^{1/2} .$$

For first order splines, these bounds are equivalent to the actual bounds since $\mathbf{M}_\alpha^{1/2}$ is diagonal. For higher order splines, these bounds are not equivalent. They are, however, approximately correct since the entries of the matrix $\mathbf{M}_\alpha$ fall off rapidly to zero away from the diagonal.

It turns out that the problems enumerated above can be avoided when using second order splines which are, in any case, the recommended splines for solving problems with control bounds. Instead of using $\mathbf{M}_\alpha$ in the coordinate transformation, the diagonal matrix

$$\mathbf{M} = \begin{bmatrix} \Delta_1 & & & & & \\ & \frac{\Delta_1 + \Delta_2}{2} & & & & \\ & & \frac{\Delta_2 + \Delta_3}{2} & & & \\ & & & \ddots & & \\ & & & & \frac{\Delta_{N-1} + \Delta_N}{2} & \\ & & & & & \Delta_N \end{bmatrix},$$

with $\Delta_k \doteq t_{N,k+1} - t_{N,k}$, is used. This transformation matrix is derived in (2.7.19c) and retains the important attributes of the transformation given by $\mathbf{M}_\alpha$. In **riots** and **pdmin**, $\mathbf{M}$ is used for the coordinate transformation, instead of $\mathbf{M}_\alpha$, when second order splines are used if *(i)* problem **OCP** has control bounds or if *(ii)* RK2 is being used as the integration method.

If higher than second order splines are to be used with control bounds and exact bound satisfaction is required, then the transform mechanism should be disabled by setting `TRANSFORM=0` in **riots** and/or **pdmin**†. Finally, when N is large, $\rho \geq 1$ and RK3 or RK4 is being used, the computation of the square-root of $\mathbf{M}_\alpha$ can take a very long time. In this case if the discretization level N is greater than about 300, the transform mechanism should also be disabled.

† In **pdmin**, setting TRANSFORM=0 causes the transform mechanism to be disabled for splines of greater than two. For second order splines, $\mathbf{M}$ is used regardless of the RK method and for first order (piecewise constant) splines, the usual, diagonal, transformation is used.

Description of the Optimization Programs

The first six inputs are the same for all of the optimization programs; they are listed in the following table. Default values for vectors apply to each component of that vector. Specifying `[]` for an input causes that input to be set to its default value. In the following, N is the discretization level and ρ is the order of the control splines.

Table O3

Input	Rows	Columns	Description
`x0`	n	1, 2 or 4	`x0=[x0,{fixed,{x0min,x0max}}]` where `x0` is the nominal value of the initial state ξ. `fixed` For each i such that `fixed(i)=0`, the corresponding initial state value ξ^i is treated as a free decision variable. Default: 1 `x0min` Specifies lower bound for each free initial condition ξ^i. Default: $-\infty$ `x0max` Specifies upper bound for each free initial condition ξ^i. Default: ∞
`u0`	m	$N+\rho-1$	Initial guess for the spline coefficients of the control u.
`t`	1	$N+1$	The integration mesh points/spline breakpoints.
`Umin`	m	$N+\rho-1$ or 1	Lower bounds on the spline coefficients for u. If `Umin` is specified as a single column, its values will apply as a lower bound on all of the spline coefficients. Default: $-\infty$
`Umax`	m	$N+\rho-1$ or 1	Upper bounds on the spline coefficients for u. If `Umax` is specified as a single column, its values will apply as an upper bound on all of the spline coefficients. Default: ∞
`params`	p	1	Provides the system parameters if required.

The first two outputs are the same for all of the optimization programs; they are listed in the following table:

Table O4

Output	Rows	Columns	Description
`u`	m	$N+\rho-1$	The optimal control solution.
`x`	n	$N+1$	The optimal state trajectory solution.

aug_lagrng

Purpose

This function uses **pdmin** as an inner loop for an augmented Lagrangian algorithm the solves optimal control problem with, in addition to simple bounds on ξ and u, endpoint equality constraints. Only one objective function is allowed.

Calling Syntax

```
[u,x,f,lambda,I_i] = aug_lagrng([x0,{fixed,{x0min,x0max}}],u0,t,
                                Umin,Umax,params,N_inner,N_outer,
                                ialg,{method},{[tol1,tol2]},{Disp})
```

Description of the Inputs

The first six inputs are described in Table O3.

`N_inner` Maximum number of iterations for each inner loop call to **pdmin**.

`N_outer` Maximum number of outer iterations.

`ialg` Specifies the integration algorithm used by **simulate**.

`method` Specifies the method for computing descent directions in the unconstrained subspace. The choices are explained in the description of **pdmin**. Default: `'vm'`.

`tol1,tol2` Optimality tolerances. Default: [$\varepsilon_{\mathrm{mach}}^{1/2}$, $\varepsilon_{\mathrm{mach}}^{1/3}$]. The outer loop terminates if

$$\|\nabla f(\eta) - \sum_{\nu=1}^{q_{ee}} \lambda_\nu \nabla g_{ee}^{\nu}(\eta)\| \le \mathtt{tol1}(1 + |f(\eta)|)$$

and

$$\max_{\nu \in \mathbf{q}_{ee}} |g_{ee}^{\nu}(\eta)| \le \mathtt{tol2}\,.$$

`Disp` Passed on to **pdmin** to control amount of displayed output. Default: 0.

Description of the Outputs

The first two outputs are described in Table O4.

`f` The objective value at the obtained solution.

`I_i` Index set of elements of $[u(:)\,;\,\xi]$ that are not at their bounds.

`lambda` Vector of Lagrange multipliers associated with the endpoint equality constraints.

Description of the Algorithm

This program calls **pdmin** to minimize a sequence of augmented Lagrangian functions of the form

$$L_{c,\lambda}(\eta) = f(\eta) - \sum_{\nu=1}^{q_{ee}} \lambda_\nu g_{ee}^\nu(\eta) + \frac{1}{2}\sum_{\nu=1}^{q_{ee}} c_\nu g_{ee}^\nu(\eta)^2$$

subject to simple bounds on ξ and u. The value of the augmented Lagrangian and its gradient are supplied to **pdmin** by **a_lagrng_fnc** via extension 1 (see description of **pdmin**).

The values of the Lagrange multiplier estimates λ_ν, $\nu = 1, \ldots, q_{ee}$, are determined in one of two ways depending on the setting of the internal variable `METHOD` in aug_lagrng.m. Initially $\lambda_\nu = 0$, $\nu = 1, \ldots, q_{ee}$.

Multiplier Update Method 1. This method adjusts the multipliers at the end of each iteration of **pdmin** by solving the least-squares problem

$$\lambda = \min_{\lambda \in \mathbb{R}^{q_{ee}}} \|\nabla f(\eta) - \sum_{\nu=1}^{q_{ee}} \lambda_\nu \nabla g_{ee}^\nu(\eta)\|_{I\_i}^2 ,$$

where the norm is taken only on the uncstrained subspace of decision variables which is indicated by the index set `I_i`. This update is performed by **multiplier_update** which is called by **pdmin** via extension 2. If update method 1 is used, the tolerance requested for the inner loop is decreased by a factor of ten on each outer iteration starting from $10^{\min\{6,\texttt{N\_outer}\}} \varepsilon_{\text{mach}}^{1/2}$ until the tolerance is $\varepsilon_{\text{mach}}^{1/2}$.

Multiplier Update Method 2. This method is the standard "method of multipliers" which solves the inner loop completely and then uses the first order multiplier update

$$\lambda_\nu \leftarrow \lambda_\nu - c_\nu g_{ee}^\nu(\eta) \quad , \forall \nu \in I_\nu$$

where

$$I_\nu \doteq \{ \nu \in \mathbf{q}_{ee} \mid |g_{ee}^\nu(\eta)| \le \tfrac{1}{4} |g_{ee}^\nu(\eta_{\text{previous}})| \text{ or } |g_{ee}^\nu(\eta)| \le \texttt{tol2} \} .$$

If update method 2 is used, the tolerance requested for the inner loop is fixed at $\varepsilon_{\text{mach}}^{1/2}$.

Penalty Update. The initial values for the constraint violation penalties are $c_\nu = 1$, $\nu = 1, \ldots, q_{ee}$. It may be helpful to use larger initial values for highly nonlinear problems. The penalties are updated at the end of each outer iteration according to the rule

$$c_\nu \leftarrow 10 c_\nu \quad , \forall \nu \notin I_\nu \ ,$$

where I_ν is as defined above.

Note that this algorithm is implemented mainly to demonstrate the extensible features of **pdmin** and is missing features like, *(i)* constraint scaling, *(ii)* an active set method for handling inequality endpoint constraints, *(iii)* a mechanism for decreasing constraint violation penalties when possible and, most importantly, *(iv)* an automatic mechanism for setting the termination tolerance for each call to **pdmin**.

Notes

1. On return from a call to **aug_lagrng**, the variable `opt_program` will be defined in the Matlab workspace. It will contain the string `'aug_lagrng'`.

See Also: **pdmin**, a_lagrng_fnc.m, multiplier_update.m.

outer

Purpose

This program calls **riots** to solve problems defined on a sequence of different integration meshes, each of which result in a more accurate approximation to **OCP** than the previous mesh. The solution obtained for one mesh is used as the starting guess for the next mesh.

Calling Syntax

```
[new_t,u,x,J,G,E] = outer([x0,{fixed,{x0min,x0max}}],u0,t,
                          Umin,Umax,params,N_inner,[N_outer,{max_N}]
                          ialg,{[tol1,tol2,tol3]},{strategy},{Disp})
```

Description of the Inputs

The first six inputs are described in Table O3.

`N_inner` Maximum number of iterations for each inner loop of **riots**.

`N_outer` Maximum number of outer iterations.

`max_N` The maximum discretization level; **outer** will terminate if the discretization level exceeds `max_N`. Default: ∞

`ialg` Specifies the integration algorithm used by **simulate**.

`tol1,tol2,tol3`

Optimality tolerances. The outer loop terminates if

$$\|\nabla L(\eta)\| \leq \texttt{tol1}(1 + |f(\eta)|) ,$$

where $\|\nabla L(\eta)\|_I$ is the H_2-norm of the free portion of $\nabla L(\eta)$,

$$\max_{\nu \in \mathbf{q}_{ee}} |g^{\nu}_{ee}(\eta)| \leq \texttt{tol2} ,$$

and

$$\|u_N - u^*\| \leq \texttt{tol3}(1 + \|u_N\|_\infty) b ,$$

where b is the nominal final time. The default values for these tolerances factors are $[\varepsilon_{\text{mach}}^{1/3}, \varepsilon_{\text{mach}}^{1/4}, \varepsilon_{\text{mach}}^{1/6}]$.

`strategy` `Passed on to` **distribute** to select the mesh redistribution strategy. Default = 3.

`Disp` Passed on to **riots** to control amount of displayed output. Default = 1.

Description of the Outputs

The first two outputs are described in Table O4.

`new_t` The final integration mesh obtained from the final mesh redistribution.

`u` The optimal control solution defined on the final mesh `new_t`.

`x` The optimal trajectory solution.

`J` A row vector whose i-th component is the value of the objective function, computed using LSODA, after the i-th call to **riots**.

`G` A row vector whose i-th component is the sum of the constraint violations, computed using LSODA, after the i-th call to **riots**.

`E` A row vector whose i-th component is an estimate of $\|\eta_N - \eta^*\|_{H_2}$ after the $(i+1)$-th iteration. With $\eta = (u, \xi)$, $\|\eta\|_{H_2}$ is defined by

$$\|\eta\|_{H_2} \doteq \left[\|\xi\|_2 + \int_a^b \|u(t)\|_2^2 dt \right]^{1/2} .$$

Description of Algorithm

outer is an outer loop for **riots**. During each iteration, **riots** is called to solve the discretized problem on the current mesh starting from the solution of the previous call to **riots** interpolated onto the new mesh. After **riots** returns a solution, **est_errors** and **control_error** are called to provide estimates of certain quantities that are used to determine whether **outer** should terminate or if it should refine the mesh. If necessary, the mesh is refined by **distribute**, with `FAC=10`, according to `strategy` except following the first iteration. After the first iteration, the mesh is always doubled.

After each iteration, the following information is displayed: the H_2-norm of the free portion of the gradient of the Lagrangian, the sum of constraint errors, objective function value, and integration error of the integration algorithm `ialg` at the current solution. All of these quantities are computed by **est_errors**. The first three values are estimates obtained using LSODA with a tolerance set to about one thousandth of the integration error estimate. The control solution is plotted after each iteration (although the time axis is not scaled correctly for free final time problems).

Additionally, following all but the first iteration, the change in the control solution from the previous iteration and an estimate of the current solution error, $\|\eta_N^* - \eta^*\|_{H_2}$, are display.

Notes

1. If solutions exhibit rapid oscillations it may be helpful to add a penalty on the piecewise derivative variation of the control by setting the variable VAR in outer.m to a small positive value.

2. The factor by which **distribute** is requested to increase the integration accuracy after each iteration can be changed by setting the variable FAC in outer.m.

3. An example using **outer** is given in Session 4 (§3).

See Also: **riots**, **distribute**, **est_errors**, **control_error**.

pdmin

Purpose

This is an optimization method based on the projected descent method. It is highly efficient but does not solve problems with general constraints or more than one objective function.

The user is urged to check the validity of the user-supplied derivatives with the utility program **check_deriv** before attempting to use **pdmin**.

Calling Syntax

```
[u,x,J,inform,I_a,I_i,M] = pdmin([x0,{fixed,{x0min,x0max}}],u0,t,
                                   Umin,Umax,params,[miter,{tol}],
                                   ialg,{method},{[k;{scale}]},{Disp})
```

Description of Inputs

The first six inputs are described in Table O3. The remainder are described here.

`miter` The maximum number of iterations allowed.

`tol` Specifies the tolerance for the following stopping criteria

$$\|g_k\|_{I_k} / |I_k| < \mathtt{tol}^{2/3}(1 + |\mathrm{f}(\eta_\mathrm{k})|) \ ,$$

$$f(u_k) - f(u_{k-1}) < 100\mathtt{tol}(1 + |\mathrm{f}(\mathrm{u_k})|) \ ,$$

$$\|u_k - u_{k-1}\|_\infty < \mathtt{tol}^{1/2}(1 + \|u_k\|_\infty) \ ,$$

$$x_k^i = 0 \ \ , \forall\, i \in A_k \ ,$$

where g_k is the k-th component of the derivative of $f(\cdot)$ in transformed coordinates, I_k is set of inactive bound indices and A_k is set of active bound indices. Default: $\varepsilon_{\mathrm{mach}}^{1/2}$.

`ialg` Specifies the integration algorithm used by **simulate**.

`method` A string that specifies the method for computing descent directions in the unconstrained subspace. The choices are:

`''`	limited memory quasi-Newton (L-BFGS)
`'steepest'`	steepest descent
`'conjgr'`	Polak-Ribière conjugate gradient method
`'vm'`	limited memory quasi-Newton (L-BFGS)

The default method is the L-BFGS method.

`k` This value is used to determine a perturbation with which to compute an initial scaling for the objective function. Typically, `k` is supplied from a previous call to **pdmin** or not at all.

`scale` This value is used to determine a perturbation with which to compute an initial function scaling. Typically, `scale` is supplied from a previous call to **pdmin** or not at all.

`Disp` `Disp = 0,1,2` controls the amount of displayed output with 0 being minimal output and 2 being full output. Default: 2.

Description of Outputs

The first two outputs are described in Table O4.

`J` A row vector whose $(i+1)$-th component is the value of the objective function at the end of the i-th iteration. The last component of `J` is the value of the objective function at the obtained solution.

`I_a` Index set of elements of $[u(:)\,;\,\xi]$ that are actively constrained by bounds.

`I_i` Index set of elements of $[u(:)\,;\,\xi]$ that are not constrained by bounds.

`inform` This is a vector with four components:

`inform(1)`	Reason for termination (see next table).
`inform(2)`	Function space norm of the free portion of $\nabla f(\eta)$, $\eta = (u, \xi)$.
`inform(3)`	Final step-size $k = \log \lambda / \log \beta$ where λ is the Armijo steplength and $\beta = 3/5$.
`inform(4)`	The value of the objective function scaling.

The possible termination reasons are:

`inform(1)`	Cause of Termination.
-1	Simulation produced NaN or Inf.
0	Normal termination tests satisfied.
1	Completed maximum number of iterations.
2	Search direction vector too small.
3	All variables at their bounds and going to stay that way.
4	Gradient too small.
5	Step=size too small.
6	User test satisified (user test returned 2).

Description of Displayed Output

Depending on the setting of `Disp`, **pdmin** displays a certain amount of information at each iteration. This information is displayed in columns. In the first column is the number of iterations completed; next is the step-size, $\lambda = \beta^k$, with k shown in parenthesis; next is $\|\nabla f(\eta)\|_{I_k}$ which is the norm of the gradient with respect to those decision variables that are not at their bounds; next is a four (three if there are no upper or lower bounds) letter sequence of `T`'s and `F`'s where a `T` indicates that the corresponding termination test, described above, is satisfied; next is the value of the objective function; and in the last column, an asterix appears if the set of indices corresponding to constrained variables changed from the previous iteration.

Extensible Features

Because **pdmin** is designed to be callable by other optimization programs, it includes three extensions that allow the user to customize its behavior. These extensions are function calls that are made to user supplied subroutines at certain points during each iteration. They allow the user to *(i)* construct the objective function and its gradients, *(ii)* specify termination criteria and perform computations at the end of each **pdmin** iteration, and *(iii)* add additional tests to the step-size selection procedure. The use of the first two of these extensions is demonstrated in the program **aug_lagrng**.

Extension 1. If the global variable `USER_FUNCTION_NAME` is defined in Matlab's workspace and is a string containing the name of a valid m-file, **pdmin** will call that m-file, instead of **simulate**, to evaluate the system functions and gradients. This can be used to construct a composite function from several different calls to **simulate**. For instance, a penalty function can be formed to convert a constrained problem into an unconstrained problem. The syntax for the user function is

```
[f0,x,grad_u,grad_x0] = USER_FUNCTION_NAME(x0,u,t,ialg,action)
```

where the input and output variables are the same as for calls to **simulate**. See a_lagrng_fnc.m for an example.

Extension 2. If the global variable `USER_TEST_NAME` is defined in Matlab's workspace and is a string containing the name of a valid m-file, **pdmin** will call that m-file at the end of each iteration. The syntax for the user function is

```
user_terminate = USER_TEST_NAME(f0,x,u,grad_u,grad_x0,I_i,free_x0)
```

where `I_i` is a column vector indexing all elements of $[u(:);\xi]$ that are not actively constrained by bounds and `free_x0` is the index set of free initial conditions. If the user test returns `user_terminate=1` and the other termination conditions are satisfied, then **pdmin** will terminate. If `user_terminate=2`, then **pdmin** will terminate without regard to the other termination tests. This function can be used solely for the purpose of performing some operations at the end of each iteration by always returning 1. See multiplier_update.m for an example.

Extension 3. If the global variable `ARMIJO_USER_TEST` is defined in Matlab's workspace and is a string containing the name of a valid m-file, the function **armijo**, which is called by **pdmin** to compute the Armijo step-length, will call that m-file in order to guarantee that the step-length satisfies

```
ARMIJO_USER_TEST(j,x,x0,u,t,ialg,I_i,free_x0) <= 0
```

where x and u are evaluated at the current trial step-length and `I_i` and `free_x0` have the same meaning as for Extension 2. This extension can be used, for instance, in a barrier function algorithm to prevent trial step-lengths that are outside the region of definition of the barrier function.

Notes

The Armijo line search is discussed in Chapter 3. The following additional features are used in the current implementation of **pdmin**.

1. A scaling for the objective function is computed using the objective scaling 2 described for **riots**. The primary purpose of this scaling is to prevent an excessive number of function evaluations during the first line search.

2. The step-length adjustment mechanism will stop increasing the step-length if $k \leq 0$ and and the next increase in step-length results in an increase in the objective function.

3. If **simulate** returns `NaN`, the step-length will be decreased until **simulate** returns a valid result.

4. Because of the coordinate transformation, the inner products in the termination tests are inner-products in $L_2[a, b]$. Thus the tests are independent of the discretization level.

Bugs

1. Control bounds can be violated if using splines of order $\rho > 2$ unless the coordinate transformation is disabled by setting the variable `TRANSFORM` to zero in the code.

riots

Purpose

This is the main optimization program in RIOTS. It can only be used if the user has obtained one of two nonlinear programming algorithms: CFSQP or NPSOL[†]. Both of these algorithms are based on sequential quadratic programming (SQP) methods. CFSQP is a feasible point SQP method and NPSOL is an active-set method based on an augmented Lagrangian merit function. With CFSQP, **riots** can solve **OCP** in its most general version. With NPSOL, **riots** only allows a single objective function. On the other hand, because NPSOL is a highly refined, commercial package, it is much faster and more robust than CFSQP. Multiple objective functions can be handled indirectly using the transcription describe in Section 2.3.

The user is urged to check the validity of the user-supplied derivatives with the utility program **check_deriv** before attempting to use **riots**.

Calling Syntax

```
[u,x,f,g,lambda2] = riots([x0,{fixed,{x0min,x0max}}],u0,t,Umin,Umax,
                          params,[miter,{var,{fd,{feasbl}}}],ialg,
                          {[eps,epsneq,objrep,bigbnd]},{scaling},
                          disp,{lambda1});
```

Description of Inputs

The first six inputs are described in Table O3. The remainder are described here.

`miter` The maximum number of iterations allowed.

`var` Specifies a penalty on the piecewise derivative variation[‡] of the control to be added to the objective function. Can only be used with first and second order splines. Adding a penalty on the piecewise derivative variation of the control is useful if rapid oscillations are observed in the numerical solution. This problem often occurs for singular problems [3,116] in which trajectory constraints are active along singular

[†] CFSQP can be obtained for free by sending a request to Prof. André Tits (andre@eng.umd.edu). NPSOL can be purchased from Stanford Business Software, Inc., 2680 Bayshore Parkway, Suite 304, Mountain View, CA 94043, (415) 962-8719.

[‡] The piecewise derivative variation is smoothed to make it differentiable by squaring the terms in the summation. The smoothing can also be accomplished using an l_1 approximation by changing the define'd variable `L1` in riots.c. However, the l_1 approximation is not twice continuously differentiable and this can inhibit superlinear convergence.

arcs. The penalty should be ten to ten thousand times smaller than the value of the objective function at a solution. See Chapter 4.5 for a discussion of singular control problems and the piecewise derivative variation of the control.

`fd` If a non-zero value is specified, the gradients for all functions will be computed by finite-difference approximations. In this case **Dh**, **Dg**, and **Dl** will not be called. Default: 0.

`feasbl` (CFSQP only) If a non-zero value is specified, CFSQP will always check for constraint violations during its line searches before evaluating objective functions. Default: 0.

`ialg` Specifies the integration algorithm used by **simulate**.

`eps` Overall optimization tolerance. For NPSOL, `eps` is squared before calling NPSOL. See the SQP user's manual for more details. Default: 10^{-6}.

`epsneq` Nonlinear constraint tolerance. Default: 10^{-4}.

`objrep` For CFSQP, for problems without equality constraints, optimization will terminate if the relative change in objective function values is less than `objrep`. For NPSOL, `objrep` indicates function precision. For both, a value of 0 causes this features to be ignored. Default: 0.

`bigbnd` A number large than the largest magnitude expected for the decision variables. Default: 10^{6}.

`scaling` Allowable values are 00, 01, 10, 11, 12, 21, 22. Default: 00. See description below.

`disp` Specify zero for minimal displayed output. Default: 1.

`lambda1` Only applies to NPSOL. Controls warm starts. Default: 0. See description below.

Description of Outputs

The first two outputs are described in Table O4.

`f` The objective value at the obtained solution.

`g` Vector of constraint violations in the following order (*N.B.* linear constraints are treated as nonlinear constraint for systems with nonlinear dynamics):

Table O5

CFSQP	NPSOL
nonlinear endpoint inequality	linear endpoint inequality
nonlinear trajectory inequality	linear trajectory inequality
linear endpoint inequality	linear endpoint equality
linear trajectory inequality	nonlinear endpoint inequality
nonlinear endpoint equality	nonlinear trajectory inequality
linear endpoint equality	nonlinear endpoint equality

`lambda2` Vector of Lagrange multipliers. This output has two columns if NPSOL is used. The first column contains the Lagrange multipliers. The first $m(N + \rho - 1)$ components are the multipliers associated with the simple bounds on u. These are followed by the multipliers associated with the bounds on any free initial conditions. Next are the multipliers associated with the general constraint, given in the same order as the constraint violations in the output `g`. Last, for CFSQP, are the multipliers associated with the objective functions. If NPSOL is being used, the second column of `lambda2` contains information about the constraints which is used by **riots** if a warm start using `lambda1` is initiated (as described below).

Scaling

There are several heuristic scaling options available in **riots** for use with badly scaled problems. There are two scaling methods for objective functions and two scaling methods for constraints. These are selected by setting `scaling` to one of the two-digit number given in the following table:

Table O6

`scaling`	Objective Scaling Method	Constraint Scaling Method
00	no scaling	no scaling
01	no function scaling	constraint scaling 1
10	function scaling 1	no constraint scaling
11	function scaling 1	constraint scaling 1
12	function scaling 1	constraint scaling 2
21	function scaling 2	constraint scaling 1
22	function scaling 2	constraint scaling 2

In the following, *FACTOR* = 10 if CFSQP is linked with **riots** and *FACTOR* = 20 if NPSOL is linked with **riots**. Also, $\eta_0 = (u_0, \xi_0)$.

Objective Scaling 1: For each $\nu \in \mathbf{q}_o$, the ν-th objective function is scaled by

$$\gamma_o^\nu = \frac{1}{1 + |f^\nu(\eta_0)|} \, FACTOR \, .$$

Objective Scaling 2: For each $\nu \in \mathbf{q}_o$, let

$$S = (1 + \|\eta_0\|_\infty) / (100 \|\nabla f^\nu(\eta_0)\|_\infty)$$

$$\delta\eta = [\eta_0 - S \nabla f^\nu(\eta_0)]_\# \, ,$$

$$\gamma = \frac{1}{2} \left| \frac{\langle \delta\eta, \delta\eta \rangle_I}{f^\nu(\eta_0 + \delta\eta_0) - f^\nu(\eta_0) - \langle \nabla f^\nu(\eta_0), \delta\eta_0 \rangle_I} \right| \, ,$$

where $[\cdot]_\#$ is the projection operator that projects its argument into the region feasible with respect to the simple bounds on u and ξ, and I is the set of indices of η_0 corresponding to components which are in the interior of this feasible region (γ is the distance along the projected steepest descent direction, $\delta\eta$, to the minimum of a quadratic fit to $f(\cdot)$). If $\gamma \geq 10^{-4}$, scale the ν-th objective function by $\gamma_o^\nu = FACTOR \, \gamma$. Otherwise, compute $\gamma = \|\nabla f^\nu(\eta_0)\|$. If $\gamma \geq 10^{-3}$, set $\gamma_o^\nu = FACTOR \, \gamma$. Otherwise, use function scaling 1.

Constraint Scaling 1: For each $\nu \in \mathbf{q}_{ei}$, the endpoint inequality constraints are scaled by

$$\gamma_{ei}^\nu = \frac{1}{\max \{ \, 1, |g_{ei}^\nu(\eta_0)| \, \}} \, FACTOR \, ,$$

for each $\nu \in \mathbf{q}_{ee}$, the endpoint equality constraints are scaled by

$$\gamma_{ee}^\nu = \frac{1}{\max \{ \, 1, |g_{ee}^\nu(\eta_0)| \, \}} \, FACTOR \, ,$$

and, for each $\nu \in \mathbf{q}_{ti}$, the trajectory inequality constraints are scaled by

$$\gamma_{ti}^\nu = \frac{1}{\max \{ \, 1, \max_{k \, \in \, \{ 1,\ldots,N+1 \}} |l_{ti}^\nu(t_k, x_k, u_k)| \, \}} \, FACTOR \, .$$

Constraint Scaling 2: The trajectory constraint scalings are computed in the same way as for constraint scaling method 1. For each $\nu \in \mathbf{q}_{ei}$, the endpoint inequality constraints are scaled by $\gamma_{ei}^\nu = \gamma$ and, for each $\nu \in \mathbf{q}_{ee}$, the endpoint equality constraints are scaled by $\gamma_{ee}^\nu = \gamma$ where γ is determined as follows. If $|g(\eta_0))| \geq 10^{-3}$, let

$$\gamma = \frac{1}{|g(\eta_0)|} \, FACTOR \, ,$$

otherwise, if $\|\nabla g(\eta_0)\| \geq 10^{-3}$, let

$$\gamma = \frac{1}{\|\nabla g(\eta_0)\|} FACTOR ,$$

otherwise do not scale.

Scaling will not always reduce the amount of work required to solve a specific problem. In fact, it can be detrimental. In the following table, we show the number of iterations required to solve some problems (described in Appendix B) with and without function scaling. All of these problems were solved using second order splines on a uniform mesh with a discretization level of $N = 50$. For both NPSOL and CFSQP, the problems were solved using `scaling` set to 0, 10, and 20. No numbers are given for CFSQP in the last two rows since it was not able to solve the Goddard problem. It should be noted that none of these problems is seriously ill-conditioned.

Table O7

		NPSOL			CFSQP		
Problem	ialg	0	10	20	0	10	20
LQR	2	5	7	7	5	13	11
Rayleigh w/o endpoint constraint	2	18	17	14	25	24	19
Rayleigh with endpoint constraint	2	24	29	19	19	26	17
Goddard w/o trajectory constraint	4	69	29	45			
Goddard with trajectory constraint	4	22	17	19			

For the last row, **riots** was called with `var` $= 10^{-4}$. Constraint scaling did not have any affect on the number of iterations for these problems. Discussion of scaling issues can be found in [41,129,130].

Warm Starts

The input `lambda1` controls the warm-starting feature available with **riots** if it is linked with NPSOL. There are two types of warm starts.

The first type of warm start is activated by setting `lambda1=1`. If this warm start is used, the Lagrange multiplier estimates and Hessian estimate from the previous run will automatically be used as the starting estimates for the current run. This is useful if **riots** terminates because the maximum number of iterations has been reached and you wish to continue optimizing from where **riots** left off. This type of warm start can only be used if the previous call to **riots** specified `lambda1=-1` or `lambda1=1`. Setting `lambda1=-1` does not cause a warm-start, it just prepares for a warm start by the next call to **riots**.

The second type of warm start allows warm starting from the previous solution from **riots** but interpolated onto a new mesh and is only implemented for first and second order splines. It is activated by providing estimates of the Lagrange multipliers in the first column of input `lambda1` and the status of the constraints in the second column of `lambda1`. Typically, `lambda1` is produced by the program distribute which appropriately interpolates the `lambda2` output from the previous run of **riots** onto the new mesh. When `lambda1` is supplied in this way, **riots** estimates $H(\eta)$, the Hessian of the Lagrangian at the current solution point, by applying finite-differences to the gradients of all objective and constraint functions weighted by their Lagrange multipliers (and scalings if a scaling option has been specified).

The estimate $H(\eta)$ of the Hessian of the Lagrangian is computed by the program **comp_hess**. This computation requires $N + \rho + n_{\text{free x0}}$ system simulations (where $n_{\text{free x0}}$ is the number of free initial conditions) and twice as many gradient computations as there are objective functions and constraints with non-zero Lagrange multipliers. Also, if a non-zero value for `var` is specified, the second derivative of the penalty term on the piecewise derivative variation of the control is added to the Hessian estimate. When $\rho \leq 2$, the computation takes advantage of the symmetry of the Hessian by stopping the simulations and gradient computations once the calculations start filling the Hessian above its diagonal. After H is computed, it is converted into transformed coordinates using the formula $\tilde{H} = (\mathbf{M}_\alpha^{-1/2})^T H \mathbf{M}_\alpha^{-1/2}$, unless the transformation mechanism has been disabled.

Because NPSOL expects the Cholesky factorization of a positive definite Hessian estimate, the following additional steps are taken. First, a Cholesky factorization is attempted on $\tilde{H}$. If this fails (because $\tilde{H}$ is not positive definite) the computation continues with the following procedure. A singular value decomposition is performed to obtain the factorization $\tilde{H} = USV^T$, where S is the diagonal matrix of singular values of $\tilde{H}$. Next, each diagonal element, σ_i, of S is set to $\sigma_i = \max\{\sigma_i, \epsilon_{\text{mach}}^{1/3}\}$. Then, we set $\tilde{H} = USU^T$, which, because $\tilde{H} = \tilde{H}^T$, makes all negative eigenvalues of $\tilde{H}$ positive while preserving the eigenstructure of $\tilde{H}$. Finally, the Cholesky factorization of $\tilde{H}$ is computed.

Notes

1. Since NPSOL is not a feasible point algorithm, it is likely that intermediate iterates will violate some nonlinear constraints. If **riots** is linked with NPSOL and, during a linesearch, NPSOL tries to evaluate a function which produces a floating point error, it will try backtracking to a

smaller step-length[†]. Using this mechanism, it is possible to force NPSOL to keep iterates within a prescribed region by forcing a division by zero[‡] when iterates are outside that region. Typically, this should be done in conjunction with a constraint such that if the constraint violation is too great, a division by zero will occur. In this way, it is possible to specify the allowable amount of constraint violation. Some margin for constraint violations should be allowed so that superlinear convergence is not inhibited.

If **riots** is linked with CFSQP, the iterates will always be feasible with respect to the constraints if `feasbl` is set to a non-zero value.

2. Because of the coordinate transformation, the inner products in the termination tests correspond to inner-products in $L_2[a, b]$. Thus the tests are independent of the discretization level.

3. When linked with NPSOL, **riots** will produce a file called npsol.opt in the current working directory.

4. On return from a call to **riots**, the variable `opt_program` will be defined in the Matlab workspace. It will contain the string `'NPSOL'` or `'CFSQP'` according to which SQP method is linked with **riots**.

Bugs

1. Control bounds can be violated if using splines of order $\rho > 2$ unless the coordinate transformation is disabled. This is done by defining the pre-compiler symbol `TRANSFORM` to zero in the code.

2. **riots** uses the Matlab MEX function **mexCallMATLAB** to make calls to **simulate**. There is a bug in this function that interferes with the operation of `ctrl-C`. This problem can be circumvented by compiling **simulate** directly into **riots** (see instructions on compiling **riots**).

3. The full warm-start feature, which requires the computation of the Hessian using finite-differencing of the gradients, is not allowed if the input `fd` is set to a non-zero value.

[†] The backtracking feature of NPSOL requires a patch for the linesearch subroutine in NPSOL (see instructions for compiling).

[‡] Adding the statement `1.0/0.0;` to the user-supplied object code will not be allowed by most compilers and the statements `zero=0.0; 1.0/zero;` will probably not cause a floating point error if compiler optimization is turned on (and it could result in a bus error for the exception handling routine). Instead use `zero=0.0; zero = 1.0/zero;`.

7. UTILITY ROUTINES

There are several utility programs, some are used by the optimization programs and some are callable by the user. Those utility programs of interest to the user are described in this section. These are:

control_error Computes an estimate of the norm of the error of the computed solution. If ${\eta_N}^*$ is the computed solution and η^* is a local minimizer for problem **OCP**, the solution error is $\|{\eta_N}^* - \eta^*\|_{H_2}$.

distribute Redistributes the integration mesh according to one of several mesh refinement strategies including one which simply doubles the mesh. The control spline defined on the previous mesh will be interpolated onto the mesh. The order of the spline is allowed to change.

est_errors Returns an estimate of the global integration error for the fixed step-size Runge-Kutta methods and uses the variable step-size integration algorithm to obtain accurate measures of the objective functions, constraint violations and trajectories. It also returns the function space norm the free portion of the gradient of the augmented Lagrangian which is needed by **control_error**.

sp_plot Plots spline functions.

transform Computes a matrix which allows the L_2 inner product of two splines to be computed by taking the inner product of their coefficients.

control_error

Purpose

This function uses values computed by **est_errors** for solutions of **OCP** on different integration meshes to estimate $\|\eta_N - \eta^*\|_{H_2}$ for the current solution $\eta_N = (u_N, \xi_N)$ using results from Chapter 4.4.

Calling Syntax

```
[error,norm_zd]=control_error(x01,u1,t1,ze1,x02,u2,t2,ze2,{Tf})
```

Description

This program compares the two solutions $\eta_{N_1} = (\mathtt{u1}, \mathtt{x01})$ and $\eta_{N_2} = (\mathtt{u2}, \mathtt{x02})$, corresponding to the mesh sequences `t1` and `t2` to produce an estimate of $\|\eta_{N_2} - \eta^*\|_{H_2}$ where $\eta^* = (u^*, \xi^*)$ is a solution for **OCP**. For free final time problems, `Tf` should be set to the duration scale factor (see transcription for free final time problems in §2). Only the first columns of `x01` and `x02` are used. The inputs `ze1` and `ze2` are the norms of the free gradients of the augmented Lagrangians evaluated at η_{N_1} and η_{N_2}, respectively, which can be obtained from calls to **est_errors**.

The output `error` is the estimate of $\|\eta_{N_2} - \eta^*\|_{H_2}$ where

$$\|\eta_{N_2} - \eta^*\|_{H_2}^2 \doteq \|\mathtt{x02} - \xi^*\|_2^2 + \int_a^{a+(b-a)\mathtt{Tf}} \|u_2(t) - u^*(t)\|_2^2 dt \ ,$$

with $u_2(\cdot)$ the spline determined by the coefficients `u2`. The output `norm_zd` is $\|\eta_{N_2} - \eta_{N_1}\|_{H_2}$ where

$$\|\eta_{N_2} - \eta_{N_1}\|_{H_2}^2 \doteq \|\mathtt{x02} - \mathtt{x01}\|_2^2 + \int_a^{a+(b-a)\mathtt{Tf}} \|u_2(t) - u_1(t)\|_2^2 dt \ ,$$

with $u_1(\cdot)$ and $u_2(\cdot)$ the splines determined by the coefficients `u1` and `u2`, respectively.

Example

Let u_1 be the coefficients of the spline solution for the mesh `t1` and let u_2 be the coefficients of the spline solution for the mesh `t2`. Let λ_1 and λ_2 be the Lagrange multipliers (if the problem has state constraints) and let I_1 and I_2 be the index set of inactive control bounds returned by one of the optimization programs (if the problem has control bounds). The Lagrange multipliers and the inactive control bound index sets are also returned by the optimization routines. Then we can compute the errors, $e_1 = \|\eta_{N_1} - \eta^*\|_{H_2}$ and $e_2 = \|\eta_{N_2} - \eta^*\|_{H_2}$ as follows:

```
>> [int_error1,norm_gLa1] = est_errors(x0,u1,t1,1,ialg1,lambda1,I1);
>> [int_error2,norm_gLa2] = est_errors(x0,u2,t1,1,ialg2,lambda2,I2);
>> error1 = control_error(x0,u2,t2,norm_gLa2,x0,u1,t1,norm_gLa1,1);
>> error2 = control_error(x0,u1,t1,norm_gLa1,x0,u2,t2,norm_gLa2,1);
```

See Also: **est_errors**.

distribute

Purpose

This function executes various strategies for redistributing and refining the current integration mesh. It also interpolates the current control and Lagrange multipliers corresponding to trajectory constraints onto this new mesh.

Calling Syntax

```
[new_t,new_u,new_lambda,sum_lte]=distribute(t,u,x,ialg,lambda,
                                            n_free_x0,strategy,
                                            {FAC},{new_K},{norm})
```

Description of Inputs

`t` Row vector containing the sequence of breakpoints for the current mesh.

`u` The coefficients of the spline defined on the current mesh.

`x` Current state trajectory solution.

`ialg` Integration algorithm to be used during next simulation or optimization.

`lambda` Current Lagrange multiplier estimates from **riots**. Specify `lambda=[]` if you do not need new multipliers for a warm start of **riots**.

`n_free_x0` Number of free initial conditions. This value only affects the extension of Lagrange multipliers needed for a warm start of **riots**.

`strategy` Selects the redistribution strategy according to the following table:

`strategy`	Type of Redistribution
1	Movable knots, absolute local truncation error.
2	Fixed knots absolute local truncation error.
3	Double the mesh by halving each interval.
4	Just change spline order to `new_K`.
11	Movable knots, relative local truncation error.
12	Fixed knots, relative local truncation error.

For more information on these strategies, see Chapter 4.3.2. The quasi-uniformity constant in equations (4.3.13) and (4.3.24) is set to $\bar{\delta} = 50$. In *Step 2* of Strategy 2 (and 12), $\sigma = 1/4$.

`FAC` For use with strategies 1,2,11 and 12. If specified, the number of intervals in the new mesh is chosen to achieve an integration accuracy approximately equal to the current integration accuracy divided by `FAC`. If `FAC=[]` or `FAC=0`, the number of intervals in the new mesh will be the same as the previous mesh for strategies 1 and 11. For strategies 2 and 12, the relative errors $\bar{e}_k$ will be used without being pre-weighted by `FAC`.

`new_K` Specifies the order of the output spline with coefficients `new_u`. By default, `new_K` is the same as the order of the input spline with coefficients `u`.

`norm` Specifies the norm used to measure the integration error on each interval. If `norm=0`, then

$$e_k = \|\mathrm{lte}_k\|_2 \ , \ \ k = 1, \ldots, N \ .$$

If `norm=1`, then

$$e_k = \|\mathrm{lte}_k\|_\infty \ , \ \ k = 1, \ldots, N \ .$$

The quantity lte_k is an estimate of the local truncation error produced by the k-th integration (see description of **simulate**, form 7). Default: 0.

Description of Outputs

`new_t` Contains the sequence of breakpoints for the new mesh.

`new_u` Contains the coefficients of the spline of order `new_K` (if specified) interpolated from `u` onto the new mesh.

`new_lambda` Two column matrix of Lagrange multiplier estimates and associate constraint status indicators. Those multipliers (and indicators) corresponding to control bounds and trajectory constraints are extended to the new mesh. This is for use with the warm start facility of **riots** and only works with NPSOL-linked **riots**.

`sum_lte` An $(n+1)$-column vector of the accumulated local truncation errors produced by the integration:

$$\texttt{sum\_lte(i)} = \sum_{k=1}^{N} e_k^i \ , \ \ i = 1, \ldots, n+1 \ ,$$

where e_k^i is as computed above. The $(n+1)$-th component represents the accumulation of local truncation errors for the integrand of the first objective function.

Notes

1. The algorithm used in strategies 1 and 2 does not take into account the presence, if any, of trajectory constraints. Strategies 2 and 12 include a mechanism that tends to add mesh points at times, or near times, where trajectory constraints are active. The input `lambda` must be supplied for this mechanism to be used.

est_errors

Purpose

This function performs a high accuracy integration with LSODA to produce estimates of various quantities. One of these quantities is used by **control_error** to produce an estimate of $\|\eta_N - \eta^*\|_{H_2}$.

Calling Syntax

```
[int_error,norm_gLa,J,G,x,Ii] = est_errors([x0,{fixed}],u,t,Tf,
                                        ialg,lambda,{I_i})
```

Description of Inputs

`x0`	Initial conditions of the *current solution*. When one or more initial conditions are free variables, set `x0=x(:,1)` where `x` is the trajectory solution returned by one of the optimization programs.
`fixed`	An n-vector that indicates which components of `x0` are free variables. If `fixed(i)=0` then `x0(i)` is a free variable. Default: all ones.
`u`	Current control solution.
`t`	Sequence of breakpoints for the current integration mesh on the (nominal) time interval $[a, b]$.
`Tf`	The duration scale factor. For fixed final time problems, set `Tf=1`.
`ialg`	Integration algorithm used to produce the current solution.
`lambda`	Vector of Lagrange multiplier estimates (one or two columns depending on which optimization program produced `lambda`).
`I_i`	Index set of controls and free initial conditions that are not at their bounds (returned by one of the optimization program).

Description of Outputs

`int_error`	`int_error(i)`, $i = 1, \ldots, n+1$, is an estimate of the global integration error, $\lvert x^i_{N,N+1} - x^i(b)\rvert$, of the current solution computed by summing the local

truncation errors produced by the integration method specified by `ialg`. The local truncation errors are obtained by a call to **simulate** (form 7). If the discrete solver or the variable stepsize integration routine is being used, `int_error` is set to a vectors of zeros. If this is the only output requested, the rest of the calculations are skipped.

`norm_gLa` This is an estimate of the H_2 norm of the free gradient of the augmented Lagrangian $L_{c,\lambda}$ evaluated at the current solution $\eta = (u, \xi)$. The H_2 norm of the free gradient of the augmented Lagrangian is the norm restricted to the subspace of controls and initial conditions that are not constrained by their bounds. Let `grad_Lu` be the gradient of the augmented Lagrangian with respect to controls, `grad_Lx0` be the gradient of the augmented Lagrangian with respect to initial conditions and $\mathbf{M}_\alpha$ be the spline transformation matrix computed by **transform**. If `Ii` is the index set estimating the free portion of $\eta =$ `[u(:);xi(free_x0)]` (see below), then the free norm if computed as follows:

$$\|\nabla_{\text{free}} L_{c,\lambda}(\eta)\|_{H_2} = \texttt{gLM(Ii)'*gL(Ii)}\ ,$$

where

$$\texttt{gLM} = [\texttt{grad\_Lu(:)}\mathbf{M}_\alpha^{-1}\,;\texttt{grad\_Lx0(free\_x0)}]$$

and

$$\texttt{gL} = [\texttt{grad\_Lu(:)}\,;\texttt{grad\_Lx0(free\_x0)}]\ .$$

In forming the augmented Lagrangian, $\lambda =$ `lambda(:,1)` and $c_i = |\lambda_i|$. The quantity $\|\nabla_{\text{free}} L_{c,\lambda}(\eta)\|_{H_2}$ is used by **control_error** to estimate the error $\|\eta_N - \eta^*\|_{H_2}$.

`J` An estimate of the objective function at the current solution. This estimate is produced using LSODA.

`G` An estimate of the sum of constraint violations. This estimate is produced using LSODA.

`x` The solution trajectory as produced using LSODA.

`Ii` Set of indices that specify those time points in the mesh `t` that are contained in the estimate I of subintervals in $[a, b]$ on which the control solution is not constrained by a control bound followed by the indices of any free initial conditions that are not constrained by a bound. This index set is used by **control_error**. For the purpose of demonstration, consider a single input systems ($m = 1$) with no free initial conditions. Let

$$\hat{I} \doteq \bigcup_{k \in \texttt{I\_i}} [t_{k-1}, t_{k+1}] \,,$$

where $t_0 \doteq t_1$ and $t_{N+2} \doteq t_{N+1}$. $\hat{I}$ is an estimate of the time intervals on which the control bounds are inactive. From $\hat{I}$, the index set `Ii` is set to

$$\texttt{Ii} \doteq \{ \, k \mid t_k \in \hat{I} \, \} \ .$$

When there are multiple inputs, this procedure is repeated for each input. When there are free initial conditions, the indices of the unconstrained components of `x0(free_x0)` are added to the end of `Ii`.

Notes

1. If the user does not supply the derivative functions **Dh** and **Dl** then it will be necessary to change the statement `IALG=5` to `IALG=6` in the file est_errors.m.

See Also: **control_error**.

sp_plot

Purpose

This program allows the user to easily plot controls which are represented as splines.

Calling Syntax

```
val = sp_plot(t,u,{tau})
```

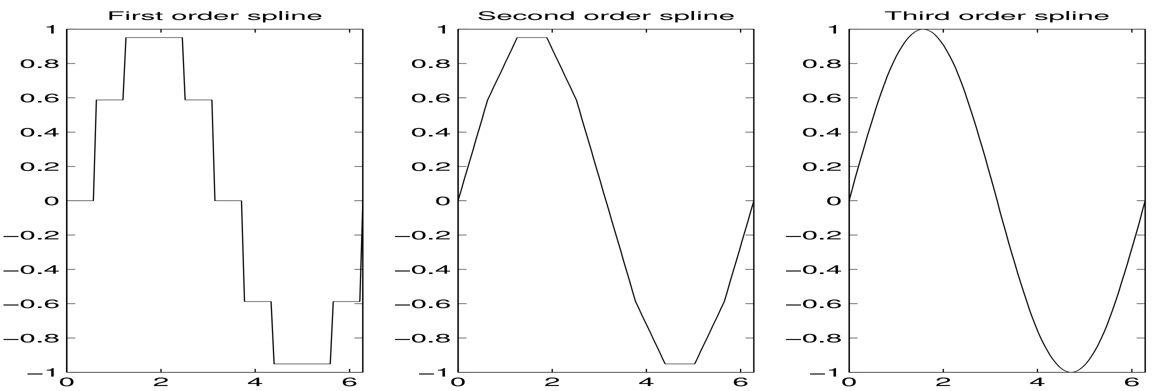
Description

Produces a plot of the spline with coefficients `u` defined on the knot sequence constructed from the integration mesh `t`. The order, ρ, of the spline is presumed equal to `length(u)` $- N + 1$. If `tau` is specified, u is not plotted, just evaluated at the times `tau`. Otherwise, `u` is plotted at 100 points with the same relative spacing as the breakpoints in `t`. Second order splines can also be plotted using the Matlab command **plot** instead of **sp_plot**.

If the input `tau` is not given, then the output is `val=[t;uval]` where `t` are the data points and `uval` are the data values; `uval` has the same number of rows as the input `u`. If the input `tau` is given, then the output is just `val=uval`.

Example. This example plots a first, second and third order spline approximation to one period of a sinusoid using ten data points. The splines are produced using the commands in the Matlab Spline Toolbox.

```
>> t=[0:2*pi/10:2*pi];
>> sp1 = spapi(t,t(1:10),sin(t(1:10)));
>> [dummy,u1] = spbrk(sp1);
>> knots2 = augknt(t,2); knots3 = augknt(t,3);
>> sp2 = spapi(knots2,t,sin(t));
>> [dummy,u2] = spbrk(sp2);
>> tau = aveknt(knots3,3);
>> sp3 = spapi(knots3,tau,sin(tau));
>> [dummy,u3] = spbrk(sp3);
>> sp_plot(t,u1); sp_plot(t,u2); sp_plot(t,u3);
```



transform

Purpose

This function produces the transformation matrix $\mathbf{M}_\alpha$. It is called by **riots** and **pdmin** to generate the spline coordinate transformation for the controls.

Calling Syntax

```
Malpha = transform(t,order)
```

Description

Given two splines u_1 and u_2 of order $\rho =$ `order` with coefficient α_1 and α_2 defined on the knot sequence with breakpoints given by `t`, $\langle u_1, u_2 \rangle_{L_2} = trace(\alpha_1 \mathbf{M}_\alpha \alpha_1^T)$. This function works with non-uniform meshes and with repeated interior knot points.

The output, `Malpha` is given in sparse matrix format. The transform matrix for $\rho = 1, 2, 3$, or 4 has been pre-computed for uniformly spaced mesh points. Also, if the inputs to the preceding call to **transform**, if there was a preceding call, were the same as the values of the current inputs, then the previously computed transform matrix is returned.

Example

This example generates two second order splines and computes their L_2 inner-product by integrating their product with the trapezoidal rule on a very fine mesh and by using $\mathbf{M}_\alpha$.

```
>> t = [0:.1:1];
>> knots = augknt(t,2);
>> coef1 = rand(1,11); coef2 = rand(1,11);
>> sp1 = spmak(knots,coef1);
>> sp2 = spmak(knots,coef2);
>> tau = [0:.0001:1];
>> u1 = fnval(sp1,tau);
>> u2 = fnval(sp2,tau);
>> inner_prod1 = trapz(tau,u1.*u2)

inner_prod1 = 0.2800

>> Malpha = transform(t,2);
>> inner_prod2 = coef1*Malpha*coef2'

inner_prod2 = 0.2800

>> inner_prod1-inner_prod2

ans = 1.9307e-09
```

8. INSTALLING, COMPILING AND LINKING RIOTS

This section describes how the components of RIOTS are compiled and linked together. Some of the specific details pertain to operation on a Sun SparcStation running SunOS 4.1 and may have to be modified for other systems. Some of the compiling and linking procedures discussed below use the shell script **cmex** that comes with Matlab. Please refer to the Matlab *External Interface Guide* and '$MATLAB/bin/README.mex' for details about **cmex**.

The following files are supplied with RIOTS:

Integration Routines	MEX programs and utilities	M-files
adams.c	.mexrc.sh	a_lagrng_fnc.m
discrete.c	cmex.link	armijo.m
euler.c	exist_in_workspace.c	aug_lagrng.m
lsoda_dummy.c	LBFGS.c	check_deriv.m
RK3.c	m_sys_link.c	check_grad.m
RK4.c	NEW_linesearch.f	comp_hess.m
trapezoid.c	riots.c	conj_grad.m
	simulate.c	control_error.m
	system.h	distribute.m
	utility.c	est_errors.m
	utility.h	eval_fnc.m
		extend.m
		fill_indices.m
		gestimate.m
		multiplier_update.m
		outer.m
		pdmin.m
		ppual.m
		project.m
		sort_lambda.m
		sp_plot.m
		transform.m

There are also a few other miscellaneous files: Contents.m, cmex.link, RIOTS.m, RIOTS_demo.m, RIOTSdem1.m, RIOTSdem2.m, RIOTSdem3.m, RIOTSdem4.m, RIOTS-dem5.m and RIOTSdem6.m. Additionally, there are examples of user code for several optimal control problems and a C-code template file in the subdirectory 'RIOTS/systems'. There is also an executable shell script, 'RIOTS/RIOTS_install.SunOS4' that performs most of the steps required to compile RIOTS. Read the comments at the top of that file for instructions. Also see the file 'RIOTS/README'.

SOME RULES FOR COMPILING.

There are several different programs that must be compiled before using RIOTS. Most of these programs are written in C, but some are written in Fortran. There are a few points that must be observed in order to ensure that all of the programs are compiled in a consistent manner.

- Use the compiler option '-cg87' because, on Sun platforms, Matlab comes linked with the old-style math library.
- Make sure that the alignment of double words is the same for all programs. That is, either all programs should be compiled with double words aligned on 8 byte boundaries, or none should. For the Sun supplied compilers, double alignment on 8 byte boundaries can be specified with the option '-dalign'. If the option '-fast' is specified, double alignment on 8 byte boundaries is automatically turned on.
- Make sure when compiling any Fortran code that the meaning of the Fortran real corresponds to the C double (8 bytes), and similarly for integers (4 bytes). This is the default behavior for the Sun Fortran compiler.
- Some problems were noticed when riots.c was compiled with optimization level -O2 and **riots** was used with the variable step-size integration algorithm. Optimization level -O3 did not produce any problems.

UNPACKING RIOTS.

The RIOTS package is distributed in a compressed UNIX tar file called RIOTS.tar.gz. The following procedure uncompresses this file and extracts RIOTS.

Step 1. Create a directory for RIOTS called 'RIOTS' by typing the following command at the UNIX prompt '%',

```
% mkdir RIOTS
```

Step 2. Uncompress RIOTS.tar.gz using the freely available GNU program gunzip†.

```
% gunzip RIOTS.tar.gz
```

Step 3. Extract the RIOTS files.

```
% tar xvf RIOTS.tar
```

After RIOTS has been extracted, the main programs will be located in the directory 'RIOTS' and the integration routines will be in the subdirectory 'RIOTS/drivers'. The user should also create a subdirectory, such as 'RIOTS/systems', for storing system functions and any system data saved from Matlab. Matlab should be started from this systems directory. Access to the functions in

† You can obtain gunzip via anonymous ftp to prep.ai.mit.edu. It is located in the file /pub/gnu/gzip-1.2.4.tar.

RIOTS can then be enabled by typing at the Matlab prompt

```
>> path(path,'..')
```

Note that it is not necessary for the 'systems' directory to be a sub-directory of the 'RIOTS' directory. If it is not, the *path* command should specify the full pathname for the 'RIOTS' directory.

Important: Included with RIOTS is a file called '.mexrc.sh' which must exist in the directory 'RIOTS' in order for **cmex** to work properly with the SunOS 4.1 operating system. This file should also be copied into the user's 'systems' or home directory. For other systems, '.mexrc.sh' should either be deleted or modified according to the instructions in '$MATLAB/bin/README.mex'. The shell variable 'MATLAB' should be set to the Matlab root directory. For example,

```
% setenv MATLAB /usr/local/matlab
```

If you are running a Bourne shell, you will need to use the export command in place of setenv.

Compiling simulate.

Before compiling and linking the programs the constitute **simulate**, the user must construct the library 'drivers.a' which contains the numerical integration routines. Most of the integration routines come supplied with RIOTS. The exceptions are the variable step-size integration routine LSODA and the linear algebra package, LINPACK, to which LSODA makes calls. If LINPACK is not already present on your machine[†], instructions for obtaining it can be received by sending email to 'netlib@ornl.gov' with no subject and the following message:

```
send index from linpack
```

You must compile LINPACK in double precision and convert it into a library called 'liblinpackd.a' using the UNIX commands 'ar' and 'ranlib'. You can also obtain LSODA by sending email to 'netlib@ornl.gov' with the following message:

```
      send lsoda.f from odepack
```

LSODA consists of fourteen files. Once the LSODA programs are received, they should be compiled and collected into a library called 'lsoda.a'. This can be accomplished with the following commands typed at the UNIX prompt:

```
% f77 -c -fast -O3 -cg87 *.f
% ar rcv lsoda.a *.o
% ranlib lsoda.a
```

[†] One place to look for LINPACK is in /usr/local/lib/liblinpackd.a.

```
% rm *.o
```

The resulting file 'lsoda.a' should be placed in the subdirectory 'RIOTS/drivers'.

Alternatively, you can use the dummy program lsoda_dummy.c, supplied with RIOTS, to avoid having to obtain and compile LSODA and LINPACK (see below). However, you will not be able to use the variable step-size option of **simulate**. Also, **est_errors**, which requires LSODA, cannot be used.

Compiling the integration routines. There are six integration programs included with RIOTS. They are: discrete.c, euler.c, trapezoid.c, RK3.c, RK4.c and adams.c.

These programs must be compiled and collected into a library called drivers.a as follows:

Step 1: Compile each integration routine. For example, to compile euler.c, use the following command at the UNIX prompt,

```
% cc -c -fast -O3 -cg87 -I.. -I$MATLAB/extern/include euler.c
```

where $MATLAB is the root directory for Matlab.

Step 2: Create the drivers.a library with the following UNIX commands,

```
% ar rcv drivers.a *.o
% ranlib drivers.a
```

If you do not have LSODA and LINPACK, you must also compile the program lsoda_dummy.c, *after creating drivers.a*, in the same way the other integration programs were compiled in *Step 1*. Then type

```
>> mv lsoda_dummy.o lsoda.a
```

The following command, typed at the UNIX prompt will compile program simulate.c in the 'RIOTS' directory.

```
% cc -c -I$MATLAB/extern/include -DMATLAB_MEX_FILE -fast simulate.c
```

Additionally, the program utility.c must be compiled:

```
% cc -c -I$MATLAB/extern/include -fast -O3 utility.c
```

Compiling the User-Supplied System Code

To link **simulate**, the user must supply object code that defines the optimal control problem. In the example above that object code was called 'my_system.o'. The following UNIX command will create 'my_system.o' from a file called 'my_system.c'.

```
% cc -c -fast -O3 my_system.c
```

It is a good idea to use a high level of optimization when compiling the user code since the user functions will be called many times during the course of each simulation. If the user-supplied system code will be supplied as M-files, you must compile the file ‘m_sys_link.c’ instead. This will create the object code file ‘m_sys_link.o’ which should then be linked with **simulate**. This object code makes the appropriate Matlab call-backs to the system m-files.

Linking in the User’s Optimal Control Problem. The next step is to create the executable MEX program **simulate** by linking in the user-supplied system code. This executable should be created in the user’s ‘systems’ directory. This step must be repeated each time a new optimal control problem is to be solved. In this example, the system code is called ‘my_system.o’. First define the environment variable `RIOTS_DIR` with the name of the ‘RIOTS’ directory and then link.

`% setenv RIOTS_DIR` *full_path_name_of_RIOTS*

`% cmex.link my_system.o`

You needn’t set the `RIOTS_DIR` variable if you are going to perform the linking in the ‘RIOTS’ directory.

Compiling riots.

Before compiling **riots**, one of the Sequential Quadratic Programming (SQP) codes NPSOL (version 4.0) or CFSQP (version 2.1) must be procured†. The efficacy of **riots** depends directly on the underlying optimization program; **riots** is much more effective with NPSOL than with CFSQP. Once obtained, the SQP program should be compiled according to the instructions included with its user’s manuals. If CFSQP is being used, the result should be two object files, ‘cfsqp.o’ and ‘qld.o’. For NPSOL, the result will be a library called ‘optlib.a’. In keeping with standard UNIX conventions, the name of this file should be changed to ‘libopt.a’ using the UNIX command ‘mv’.

When using NPSOL, the following changes should be made to the Fortran program ‘npsol-subs.f’ before compiling NPSOL:

1. Change the line

```
      if (ncdiff .eq. 0) then
```

to

```
      if (ncdiff .lt. 0) then
```

† CFSQP can be obtained for free by sending a request to Prof. André Tits (andre@eng.umd.edu). NPSOL can be purchased from Stanford Business Software, Inc., 2680 Bayshore Parkway, Suite 304, Mountain View, CA 94043, (415) 962-8719.

and change the line

```
if (nfdiff .lt. 0) then
```

to

```
if (nfdiff .eq. 0) then
```

2. Add the line

```
open( ioptns, file = 'npsol.opt', status = 'OLD' )
```

before the line

```
call opfile( ioptns, nout, inform, npkey )
```

3. Replace the subroutine `npsrch` with the code in the file 'NEW_linesearch.f'.

Next, **riots** can be compiled and linked using one of the following commands.
For NPSOL, use:

```
% cmex -O3 CASE=f -DNPSOL -L$NPSOL riots.c LIBS='-lopt $LIBS'
```

where $NPSOL is the directory where 'libopt.a' is located.
For CFSQP, use:

```
% cmex -O3 -I$CFSQP riots.c LIBS='$CFSQP/cfsqp.o $CFSQP/qld.o $LIBS'
```

where $CFSQP is the CFSQP directory.

Direct linking. With **riots** compiled and linked with the proceeding commands, it will make calls to **simulate** using the Matlab procedure **mexCallMATLAB**. There are two drawbacks to this. First, calling **simulate** via this Matlab procedure is somewhat slow. Secondly, there is a bug in **mexCallMATLAB** that prevents `ctrl-C` from causing an interrupt. To avoid these problems, **riots** can be compiled with **simulate** linked in directly. The following steps are required for direct linking:

Step 1. Make a direct version of 'simulate.o' using

```
% cc -c -I$MATLABextern/include -fast -DDIRECT \
-o simulate_direct.o simulate.c
```

Step 2. Compile and link **riots**.

`% setenv RIOTS_DIR` *full_path_name_of_RIOTS*

For NPSOL use:

```
% cmex CASE=f -O3 -DDIRECT -DNPSOL -DSTATIC $RIOTS_DIR/riots.c \
LIBS="my_system.o $RIOTS_DIR/simulate_direct.o $RIOTS_DIR/utility.o \
$RIOTS_DIR/drivers/drivers.a $RIOTS_DIR/drivers/lsoda.a \
/usr/tools/lib/liblinpackd.a $NPSOL/libopt.a"'$LIBS'
```

For CFSQP use:

```
% cmex CASE=f -DDIRECT -O3 -I$CFSQP $RIOTS_DIR/riots.c \
LIBS="$RIOTS_DIR/simulate_direct.o $ my_system.o CFSQP/cfsqp.o \
$CFSQP/qld.o $RIOTS_DIR/utility.o $RIOTS_DIR/drivers/drivers.a \
drivers/lsoda.a /usr/tools/lib/liblinpackd.a"'$LIBS'
```

Step 3. Remove the file ‘simulate_direct.o’.

```
% rm simulate_direct.o
```

The disadvantage to linking **simulate** directly into **riots** is that the size of the executable **riots** will be much larger. Also, in order for calls to **simulate** that are made outside of **riots** to work properly, **simulate** must still be compiled and linked with the user’s system code according to the instruction above for compiling **simulate**. Even if **simulate** is not also called directly by the user, there are many other programs in RIOTS that make calls to **simulate**. These programs will not work if **simulate** is not compiled and linked separately from **riots**.

Compiling the Other MEX Programs.

There two other MEX programs which must be compiled by typing at the UNIX prompt:

```
% cmex -O3 LBFGS.c
% cmex exist_in_workspace
```

Chapter 6

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

The work in this thesis presents a theoretical foundation for the solution of optimal control problems using Runge-Kutta integration methods. This theoretical foundation underpins the implementation of our software toolbox, RIOTS, which has been used to solve several challenging optimal control problems. However, the current version of RIOTS has shortcomings in some important areas. What follows is a discussion of areas in which RIOTS could be improved and suggestions for how to do so.

Automatic Differentiation of user-supplied functions. To solve an optimal control problems using RIOTS, the user must supply code for each of the functions and their derivatives used to that describe that problem. A common source of errors is the computation of function derivatives which can be quite complicated for nonlinear systems. Currently, RIOTS has two programs to check derivative calculations and to help locate any errors in these calculations. But, it would be much more useful to have derivative functions provided automatically using automatic differentiation [131,132]. Not only would this prevent errors in the derivative calculations, but it would spare the user the job of programming the derivatives.

Extension to Large-Scale Problems. The size of the mathematical programming problem created by discretizing an optimal control problem depends primarily on the discretization level N. The conjugate gradient and L-BFGS implementations of the projected descent algorithm presented in Chapter 3 are well-suited for handling very high discretization levels. These methods are used in the program **pdmin**. However, **pdmin** can only handle state constraints indirectly through penalty terms. The main program in, **riots**, is based on sequential quadratic programming and can handle state equality and inequality constraints. However, **riots** is not well-suited for high discretization levels because at each iteration the SQP algorithm computes a Hessian update using the BFGS formula and solves a dense quadratic program (QP) at each iteration to obtain a search direction. We list here several possible ways to overcome this problem:

- Instead of computing the Hessian estimate using the BFGS method, a Hessian estimate can be provided using the limited memory L-BFGS [86,94] method. Still, some means for efficiently solving the QP based on this estimate would be needed.

- The QP can be solved efficiently by taking into account the structure of the problem that gives rise to the QP. Specifically, the QP that is solved at each iteration is actually a linear/quadratic (LQ) optimal control problem. This structure can be used to develop solution methods for the QP that are much more efficient than standard QP algorithms. One approach is based on the method in [93] for recursively computing the Newton's direction for unconstrained problems in order N time and the differential dynamic programming method in [133-137]. The extension of this method to problems with control constraints is presented in [138,139]. Finally, an algorithm for the recursive computation of Newton's direction for problems with state constraints is developed in [140]. The main drawback to all of these methods is that they have only been developed for Euler's method with piecewise constant controls. It is likely that these methods can be extended to any fixed step-size Runge-Kutta method and to piecewise polynomial representation for the control. It is probably much more difficult to extend these methods to spline representations of the control (except linear splines which can be treated as piecewise linear functions with linear equality constraints at the breakpoints). But the main advantage of splines over piecewise polynomials is the smaller number of parameters. This advantage is not so important if the work to solve the QP is only order N.

  Another approach is to consider the two point boundary value problem (BVP) that arises necessary conditions for optimality of a solution to the LQ problem. This BVP is linear with constraints. Numerical methods based on multiple shooting for solving such BVP's are quite advanced. The solution of this BVP is the required search direction. Examples of this approach can be found in [5,6,24,114].

- Another major area of research involves discretizing the optimal control using collocation (fixed step-size, implicit Runge-Kutta) methods. In this procedure, finite-dimensional approximations of both the controls and states are parameterized and their parameters are used as the decision variables in the resulting mathematical programming problem. In this method, the differential equations describing the system dynamics are accounted for by requiring them to be satisfied at collocation points. There are two major disadvantages to this approach: *(i)* the number of decision variables in the mathematical program is dramatically increased and *(ii)* the collocation conditions introduce a huge system of nonlinear equality constraints into the problem.

There are, however, two major advantages that come with this approach. First, because the decision variables include the system states, simple bounds on the state trajectories or endpoints become simple bounds on the decision variables. These bounds can be handled very efficiently without having to compute gradients and, furthermore, feasibility with respect to these bounds can be easily maintained even with infeasible point optimization methods. Second, the Hessian of the Lagrangian is block diagonal because the second derivatives of the objective and constraint functions with respect to the control and state parameters are all block diagonal. Additionally, the Jacobian of the constraints is a banded matrix. Thus, the QP is sparse and can be solved using sparse linear algebra. A complete software package for discretizing optimal control problems using collocation and solving the discretized problems using a sparse SQP method is described in [38,105]. One obstacle with this approach that has not been fully resolved is how to obtain a sparse Hessian estimate. The method in [38] obtains the Hessian using sparse finite-differences. This is reasonably efficient. Alternatively, but less attractively, the user can be required to supply second derivatives so that the Hessian can be computed exactly. But for both of these choices, the fact that the Hessian is not, in general, positive definite causes serious difficulties in the solution of the QP. A new SQP algorithm which used the true Hessian without requiring any modification to the QP has been developed [141]. Another possibility is to use trust region methods. One avenue that has not yet been explored is to use the LANCELOT package [142] which is a trust region method for minimizing an augmented Lagrangian. Finally, another approach which we began to work on with some initial success is to attempt to produce a block diagonal, positive definite estimate of the Hessian using a modified BFGS update. Generally, the work on sparse Hessian updates has been disappointing. But the block diagonal structure of the Hessian is a specific case whose updates can, we believe, be obtain using the partitioned quasi-Newton update method [143,144] modified to handle the possibility of non-positive-definite updates for individual blocks. This modification can be either to skip such updates altogether or to use the Powell modification [145]. If the Hessian is positive definite, or even positive semi-definite, the QP can be solved by a sparse QP algorithm such as BQPD which is based on the algorithm in [146] and is available from that author. For surveys on the solution of large-scale optimization algorithms, the reader is referred to the following articles [147-149]. Finally, to obtain sparse Hessian and Jacobian matrices, the controls (and state trajectories) need to be represented as piecewise discontinuous functions (*e.g.* piecewise polynomial) rather than splines. This has the side-benefit of producing a transform matrix, $\mathbf{M}_N$, that is block diagonal and trivial to compute.

Trajectory constraints. Our current method of computing functions gradients with respect to the control is based on adjoint equations. There is one adjoint equation for each function. This is quite inefficient when there are trajectory constraints because for each trajectory constraint there is, in effect, one constraint function per mesh point. Thus, for an integration mesh with $N+1$ breakpoints, roughly N adjoint equations have to be solved to compute the gradients at each point of a trajectory constraint.

There are two approaches for greatly increasing the gradient computations for trajectory constraints. These two approaches can be used in conjunction with each other. First, it is really only necessary to compute gradients at points, t_k, where the trajectory constraints are active or near-active. The other mesh points should be ignored. Algorithms for selecting the active or almost active constraint are present in [99,150] along with convergence proofs. The second approach uses the state-transition (sensitivity) matrix, rather than adjoint variables, to compute gradients. The state-transition matrix is the solution of a matrix differential (or difference) equation. The solution of this equation requires the same amount of computation as solving n adjoint equations, where n is the number of state variables. But this equation is solved only once for a given control, u, regardless of how many gradients are required. Thus, if there are more than n gradients that need to be computed[†] it is more efficient to use the state-transition matrix rather than adjoint variables.

Stabilization of Iterates. One of the main limitations of the current implementation of RIOTS is that it is not well-equipped to deal with problems whose dynamics are highly unstable. For such problems, the iterates produced by the optimization routines in RIOTS can easily move into regions where the system dynamics "blow-up" if the initial control guess is not close to a solution. For instance, a very difficult optimal control problem is the Apollo re-entry problem [4]. This problem involves finding the optimum re-entry trajectory for the Apollo space capsule as it enters the Earth's atmosphere. Because of the physics of this problem, slight deviations of the capsules trajectory can cause the capsule to skip off the Earth's atmosphere or to burn up in the atmosphere. Either way, once an iterate is a control that drives the system into such a region of the state-space, there is no way for the optimization routine to recover. Moreover, in this situation, there is no way to avoid these regions of the state-space using control constraints.

This problem could be avoided using constraints on the system trajectories. However, this is a very expensive approach for our method (not for collocation-based methods), especially at

[†] Here we are not taking into account the fact that the the adjoint equation for a trajectory constraint at time t_k only has to be solved from time t_k (instead of t_N) to time t_0.

high discretization levels. Also, for optimization methods that are not feasible point algorithms, this approach still might not work. An intermediate solution is possible because it is really only necessary to check the trajectory constraints at a few points, called nodes, in the integration mesh. This can be accomplished as follows. Let t_k be one such node. Then define the decision variable $\tilde{x}_{k,0}$ which will be taken as the initial condition for integrating the differential equations starting at time t_k. This $\tilde{x}_{k,0}$ is allowed to be different than the value $\bar{x}_k$ of the state integrated up to time t_k. However, to ensure that these values do, in fact, coincide at a solution, a constraint of the form $g_k(u) \doteq \tilde{x}_{k,0} - \bar{x}_k = 0$ must be added at each node. Note that, for nonlinear systems, $g_k(u)$ is a nonlinear constraint. The addition of these node variables allows bounds on that states to be applied at each node point. This procedure is closely related to the multiple shooting method for solving boundary value problems and is an intermediate approach between using a pure control variable parameterization and a control/state parameterization (as in collocation methods). See [151] for a discussion of node placement for multiple shooting methods.

Diagonalization Strategies. In order to create a truly efficient algorithm for providing solutions of a specified accuracy to optimal control problems, the discretization strategy, an ‘‘outer loop’’, must be developed for increasing the discretization level in a systematic way. Such a strategy must be able to predict the integration and control solution errors, the amount of work required per iteration of an optimization algorithm in solving the discretized problem and the number of iterations required to obtain a solution of the discretized problem to a given accuracy. Using this information, the outer loop must specify at each outer iteration the discretization level, the order of the integration method, the order of the control representation, and the number of inner iterations to perform so that the overall amount of work required to solve the optimal control problem is minimized. Such a diagonalization strategy was developed in [57] for solving semi-infinite optimization problems. There are three major obstacles obstructing the use of that algorithm for optimal control problems. First, the diagonalization scheme proposed in [57] is based on linearly convergent optimization algorithms. However, we would like to use superlinearly convergent algorithms such as SQP methods for the inner loop. Second, the errors estimates needed by the diagonalization strategy, such the control solution error, are much harder to predict for optimal control problems. Currently, only first order bounds on the difference between the solution of the discretized problems and the true solution are known for general optimal control problems. This may be too conservative, especially when mesh redistribution schemes are to be used. Finally, an underlying assumption for the effective use of a diagonalization strategy is that using the solution from one outer iteration as the starting point for the next outer iteration, so-called ‘‘warm starts’’, will result in a significant reduction in the number of inner iterations needed. This may not always be true. For example, the program **riots** is based on an SQP algorithm that generates Hessian information using a BFGS update. Super-linear convergence can only begin once enough Hessian information is gathered so that the search directions are close to Newton's direction. Since restarting the SQP algorithm at a different discretization level causes the Hessian information to be lost, many iterations will be used just to obtain the requisite Hessian information, even when the starting point is close to the solution. Currently, our implementation of **riots** has the option of computing an initial Hessian approximation by finite-differences before starting the inner iterations. However, since the Hessian is dense, this takes a great deal of computational effort†. Another possibility is to somehow lift the Hessian information obtained from the previous outer iteration into the new, higher-dimensional space. We have tried several approaches for doing this and have not met with any success.

Mesh Refinement. The mesh refinement strategies presented in Chapter 4 are not suited for all situations. In particular, they contain no mechanism for directly placing mesh points at or near locations where control and/or trajectory constraints switch from active to inactive or *vice versa*. But, these are the locations where the solutions are likely to be least accurate. One simple, static adjustment strategy for placing mesh points near constraint switches is to simply place extra mesh points wherever such constraint activity transitions occur. Extra mesh points can also be placed throughout regions where the trajectory constraints are active. A version of this approach has been added to Strategy 2 (and 12) of the utility program **distribute** in the RIOTS package. However, changing the mesh also causes the solution to change. Thus, placing a mesh point where a constraint becomes active for one grid is probably not the correct point for a new grid. Also, accurate placement of mesh points may not be sufficient if the control solution loses smoothness at a constraint transition. For instance, if splines are used for the control representation, then it is impossible to achieve better than first order accuracy in the overall solution unless repeated knots are used at the point of discontinuity in the solution. A dynamic adjustment approach for locating mesh points near discontinuities is through the use of super-nodes [34] in the discretization. These super-nodes are, essentially, movable locations of extra spline knots that are positioned during the optimization to allow discontinuities in the spline or its derivatives.

It must also be remembered that the location of control discontinuities depends not only on the errors due to the control representation, but also the integration error. To circumvent this complication, it might be beneficial to employ a the two-phase approach [4,24,41] in which a

† In the discussion of extensions to large-scale problems, we discussed to alternatives for solving the QP to obtain a search direction. The first involves a recursive procedure for finding Newton's direction. The second involves using collocation as discretization because it produces a sparse Hessian. In both cases, the required second derivative information comes in small block matrices. Thus, for these approaches, finite-differencing, either on the first iteration or all iterations, is a feasible approach.

rough solution is first obtained using a direct optimization method. In these references, the rough solution provides structural information and a good initial guess for a more accurate solution obtained by solving the two point boundary value problem arising from the necessary conditions for optimality. Alternatively, super-nodes could be used in conjunction with a variable step-size integration method to refine the solution in a second phase direct approach similar to the first phase. The use of the variable step-size integration routine would remove the effect that moving the super-nodes has on the integration accuracy. Thus, the location the optimization algorithm chooses for the super-nodes in order to minimizes the objective function will coincide with the discontinuities in the true solution. The number of super-nodes needed can be ascertained by inspection of the rough solution obtained in the first phase.

Also, trajectory constraints are currently evaluated at discrete mesh points. In the phase II operation, trajectory constraint satisfaction can be more accurately guaranteed by constructing interpolating Hermite polynomials for the state in question over each mesh interval and requiring these polynomials to satisfy these constraints. Such a procedure is adopted in [104] and [24].

Other Issues and Extensions. Some other useful features for RIOTS would include:

- A graphical user interface. This would allow much easier access to the optimization programs and selection of options. Also, important information about the progress of the optimization such as error messages and warnings, condition estimates, step-sizes, constraint violations and optimality conditions could be displayed in a much more accessible manner.

- Dynamic linking. Currently, the user of RIOTS must re-link **simulate** for each new optimal control problem. It would be very convenient to be able to dynamically link in the object code for the optimal control problem directly from Matlab (without having to re-link **simulate**). There are dynamic linkers available but they do not work with Matlab's MEX facility.

- For problems with dynamics that are difficult to integrate, the main source of error in the solution to the approximating problems is due to the integration error. In this case, it would be useful to use an integration mesh that is finer than the control mesh. Thus, several integration steps would be taken between control breakpoints. By doing this, the error from the integration is reduced without increasing the size (the number of decision variables) of the approximating problem.

- The variable transformation needed to allow the use of a standard inner product on the coefficient space for the approximating problems adds extra computation to each function and gradient evaluation. Also, if the transformation is not diagonal, simple bound constraints on the controls are converted into general linear constraints. Both of these deficits can be removed for optimization methods that use Hessian information to obtain search directions. If the Hessian is computed analytically, then the transformation is not needed at all. If the Hessian is estimated using a quasi-Newton update, it may be sufficient to use the transformation matrix $\mathbf{M}_N$ or $\mathbf{M}_\alpha$ as the initial Hessian estimate (rather than the identity matrix) and dispense with the variable transformation. We have not performed this experiment; it may not work because the the updates will be constructed from gradients computed in non-transformed coordinates[†].

- It may be useful to allow the user to specify bounds on the control derivatives. This would be a simple matter for piecewise linear control representations. Also, currently the only way to specify general constraints on the controls is using mixed state-control trajectory constraints. This is quite inefficient since adjoint variables are computed but not needed for pure control constraints.

- Currently there is no mechanism in RIOTS for handling systems with time-delays or, more generally, integro-differential equations [153]. This would be a non-trivial extension.

- Add support for other nonlinear programming routines in **riots**.

- There have been very few attempts to make quantitative comparisons between different algorithms for solving optimal control problems. The few reports comparing algorithms [154,155], involve a small number of example problems, are inconclusive and are out of date. Therefore, it would be of great use to have an extensive comparison of some of the current implementations of algorithms for solving optimal control problems.

† With appropriate choice of H_0, quasi-Newton methods are invariant with respect to objective function scalings[95,152], but not coordinate transformations (which is variable scaling).

APPENDIX A

In this Appendix we collect a few results used in the analysis of Sections 4 and 5. We will continue to use the notation of Section 4: $\Delta = 1/N$, $t_k = k\Delta$, and $\tau_{k,i} = t_k + c_i\Delta$.

Lemma A.1. For representation **R1**, suppose that Assumptions 3.1*(a)*, 4.1' and 4.3 hold. For representation **R2**, suppose that Assumptions 3.1*(a)*, 4.1', and 4.6 hold. For any bounded subset $S \subset \mathbf{B}$, there exists a $\kappa < \infty$ such that for any $\eta = (\xi, u) \in S \cap \mathbf{H}_N$, $\|\delta_k\| \le \kappa\Delta^2$ for all $k \in \mathcal{N}$, where

$$\delta_k \doteq x^\eta(t_k) - x^\eta(t_{k+1}) + \Delta \sum_{i=1}^{s} b_i h(x^\eta(t_k), u[\tau_{k,i}]) \,, \quad k \in \mathcal{N}, \tag{A.1}$$

with $x^\eta(\cdot)$ the solution of the differential equation (2.3.1) and $u[\tau_{k,i}]$ defined by (2.4.6e) for representation **R1** or (2.4.11c) for representation **R2**.

Proof: Let $\tilde{b}_j$ and d_j be as defined in (2.4.10) and, for $j \in \mathbf{r}$, let $i_j \in I$ where I is given by (2.4.4a). Then, writing $x(\cdot) = x^\eta(\cdot)$, since the solution of (2.3.1) satisfies $x(t_{k+1}) = x(t_k) + \int_{t_k}^{t_{k+1}} h(x(t), u(t))\, dt$, we see that

$$\delta_k = \Delta \sum_{i=1}^{s} b_i h(x(t_k), u[\tau_{k,i}]) - \int_{t_k}^{t_{k+1}} h(x(t), u(t))\, dt$$

$$= \sum_{j=1}^{r} \int_{t_k+d_{j-1}}^{t_k+d_j} h(x(t_k), u[\tau_{k,i_j}])\, dt - \sum_{j=1}^{r} \int_{t_k+d_{j-1}}^{t_k+d_j} h(x(t), u(t))\, dt \,, \tag{A.2a}$$

because $d_j - d_{j-1} = \Delta\tilde{b}_j$, $u[\tau_{k,i_j}] = u[\tau_{k,i}]$ for all $i \in I_j$, $d_0 = 0$ and, by Assumption 4.1', $d_r = \Delta \sum_{j=1}^{r} \tilde{b}_j = \Delta \sum_{j=1}^{s} b_j = \Delta$. Since $d_j - d_{j-1} > 0$ by Assumption 4.1', we have that

$$\|\delta_k\| \le \sum_{j=1}^{r} \int_{t_k+d_{j-1}}^{t_k+d_j} \|h(x(t_k), u[\tau_{k,i_j}]) - h(x(t), u(t))\|\, dt$$

$$\le \sum_{j=1}^{r} \int_{t_k+d_{j-1}}^{t_k+d_j} \kappa_1[\|x(t_k) - x(t)\| + \|u[\tau_{k,i_j}] - u(t)\|]\, dt \,, \tag{A.2b}$$

where $\kappa_1 < \infty$ is as in Assumption 3.1*(a)*. Now, for $t \in [t_k, t_{k+1}]$, there exists $\kappa_2 < \infty$ such that

$$\|x(t_k) - x(t)\| \le \int_{t_k}^{t} \|h(x(t), u(t))\|\, dt \le \int_{t_k}^{t_{k+1}} \kappa_2[\|x(t)\| + 1]\, dt \tag{A.3}$$

by Assumption 3.1*(a)* and the fact that S is bounded. Also because S is bounded, if follows from Theorem 3.2*(ii)* that there exists $L < \infty$ such that $\|x(t)\| \le \kappa_3[\|\xi\| + 1] \le L$. Thus, for $t \in [t_k, t_{k+1}]$, $\|x(t_k) - x(t)\| \le \int_{t_k}^{t_{k+1}} \kappa_2[L+1]\, dt = \Delta\kappa_2(L+1)$. Next, for representation **R1**, for any $k \in \mathcal{N}$, $j \in \mathbf{r}$ and $t \in [t_k + d_{j-1}, t_k + d_j)$, $\|u[\tau_{k,i_j}] - u(t)\| \le \kappa_U \Delta$, where κ_U is used in (2.4.15a), since, by construction, $\bar{u} \in \mathbf{U}_N^1$ is a Lipschitz continuous polynomial on $[t_k, t_{k+1})$ with Lipschitz constant κ_U independent of N, $\tau_{k,i_j} \in [t_k, t_k + \Delta]$ by Assumption 4.3, and $0 \le d_j \le \Delta$ for $j = 0, \ldots, r$ by Assumption 4.1' which implies that $[t_k + d_{j-1}, t_k + d_j) \subset [t_k, t_k + \Delta)$. The same holds for representation **R2** since $u \in L_N^2$ is constant on $t \in [t_k + d_{j-1}, t_k + d_j)$ and $\tau_{k,i_j} \in [t_k + d_{j-1}, t_k + d_j]$ by Assumption 4.6. Therefore,

$$\|\delta_k\| \le \sum_{j=1}^{r} \int_{t_k+d_{j-1}}^{t_k+d_j} \kappa_1(\kappa_2(L + 1\,) + \kappa_U)\Delta\, dt = \kappa\Delta \sum_{j=1}^{r} \int_{t_k+d_{j-1}}^{t_k+d_j} dt = \kappa\Delta^2\,, \tag{A.4}$$

where $\kappa = \kappa_1(\kappa_2(L + 1\,) + \kappa_U)$. This completes our proof. □

Remark A.2. The result in Lemma A.1 can be shown to hold even if the constraints on $\|u_K T_j\|$ in the definition (2.4.15a) of $\bar{\mathbf{U}}_N^1$ were removed if $h(x, u) = \tilde{h}(x) + Bu$ and the RK method is order r. Starting from equation (A.2a), we have

$$\delta_k = \sum_{j=1}^{r} \int_{t_k+d_{j-1}}^{t_k+d_j} \tilde{h}(x(t_k)) - \tilde{h}(x(t))\, dt + \Delta \sum_{j=1}^{r} \tilde{b}_j B u[\tau_{k,j}] - \int_{t_k}^{t_{k+1}} Bu(t)\, dt\,. \tag{A.5a}$$

The first term is $O(\Delta^2)$ by the argument already presented. For the remaining part, we see that

$$B\left(\Delta \sum_{j=1}^{r} \tilde{b}_j u[\tau_{k,j}] - \int_0^{\Delta} u(t + t_k)\, dt \right) = 0\,, \tag{A.5b}$$

since a ρ-th order Runge-Kutta method, $\rho \ge r$, integrates the equation $\dot{x} = u(t + t_k)$ exactly for any r-th order polynomial u.

□

The next lemma concerns the functions $K_{k,i} = K_i(\bar{x}_k, \omega_k)$ of the RK method defined by (2.4.3a,b). The proof of this result is easily obtained from the proof for Lemma 222A in [8, p. 131].

Lemma A.3 Suppose Assumptions 3.1*(a)* holds. Let $S \subset \mathbf{B}$ be bounded. Then there exists $L < \infty$ and $N^* < \infty$ such that for all $N \ge N^*$, $\eta \in S \cap H_N$, $k \in \mathcal{N}$ and $i \in \mathbf{s}$,

$$\|K_{k,i} - h(\bar{x}_k, u[\tau_{k,i}])\| \le L\Delta\,. \tag{A.6}$$

□

Next, we present a proof of Lemma 4.10.

Proof of Lemma 2.4.10.

(i) Convergence. Let $\eta = (\xi, u) \in S \cap \mathbf{H}_N$ and, for $k \in \mathcal{N}$, let $e_k \doteq \bar{x}_k^\eta - x^\eta(t_k)$. Then, $\|e_0\| = 0 \leq \kappa\Delta$ and by adding and subtracting terms,

$$e_{k+1} = \bar{x}_k^\eta + \Delta \sum_{i=1}^{s} b_i K_{k,i} - x^\eta(t_{k+1})$$

$$= e_k + \left(x^\eta(t_k) - x^\eta(t_{k+1}) + \Delta \sum_{i=1}^{s} b_i h(x^\eta(t_k), u[\tau_{k,i}]) \right) + \Delta \sum_{i=1}^{s} b_i \left(K_{k,i} - h(x^\eta(t_k), u[\tau_{k,i}]) \right). \quad (A.7)$$

The norm of the second term in this expression is bounded by $\kappa_1 \Delta^2$ by Lemma A.1 where $\kappa_1 < \infty$. Using Lemma A.3, Assumption 3.1*(a)*, and the fact that $|b_i| \leq 1$ by Assumption 4.1', we conclude for the third term that, there exists $\kappa_2 < \infty$ such that

$$\Delta \left\| \sum_{i=1}^{s} b_i \left(K_{k,i} - h(x^\eta(t_k), u[\tau_{k,i}]) \right) \right\|$$

$$\leq \Delta \sum_{i=1}^{s} \|K_{k,i} - h(\bar{x}_k^\eta, u[\tau_{k,i}])\| + \Delta \sum_{i=1}^{s} \|h(\bar{x}_k^\eta, u[\tau_{k,i}]) - h(x^\eta(t_k), u[\tau_{k,i}])\|$$

$$\leq \Delta^2 L s + \Delta \kappa_2 s \|e_k\| . \quad (A.8)$$

Thus, for all $k \in \mathcal{N}$,

$$\|e_{k+1}\| \leq (1 + \kappa_2 \Delta s) \|e_k\| + \kappa_3 \Delta^2 . \quad (A.9)$$

where $\kappa_3 = \kappa_1 + Ls$. Solving (A.9), we see that for all $k \in \mathcal{N}$, $\|e_k\| \leq (1 + \kappa_2 \Delta s)^N \|e_0\| + \kappa_3{}' \Delta \leq \kappa\Delta$. This proves (2.4.18a).

(ii) Rate of Convergence. We prove (2.4.18c) in two steps. First suppose that $H_N = H_N^1 = \mathbb{R}^n \times L_N^1$ and let $\eta_1 = (\xi, u_1) \in S \cap H_N^1$ be given. The expansion based on higher-order derivatives (see [8]) needed to prove (2.4.18c) requires smoothness of $h(x, u)$ between time steps. The stated assumptions on the piecewise smoothness of $u_1(\cdot)$ provide this smoothness. Alternatively, the result can also be shown to hold without this assumption on u_1 if the differential equations describing the system dynamics are linear and time-invariant with respect to u since the RK method provides exact quadrature integration for $u \in L_N$ in this case. In either case, using the same type of reasoning as in the proof of Lemma A.1, we conclude that there exists $\kappa < \infty$, independent of η, such that (2.4.18c) holds for representation **R1**. Next, to prove (2.4.18c) for representation **R2**, let $H_N = H_N^2 = \mathbb{R}^n \times L_N^2$. Let $\eta_2 = (\xi, u_2) \in S \cap H_N^2$ be given and let $\eta_1 = (\xi, u_1) \in H_N^1$ with $u_1 = (V_{\mathbf{A},N}^1)^{-1}(V_{\mathbf{A},N}^2(u_2))$ so that $V_{\mathbf{A},N}^1(u_1) = V_{\mathbf{A},N}^2(u_2)$. Then for any $t \in [0, 1]$,

$$\|x^{\eta_1}(t) - x^{\eta_2}(t)\| = \| \int_0^t h(x^{\eta_1}(s), u_1(s)) - h(x^{\eta_2}(s), u_2(s))ds \|$$

$$\leq \| \int_0^t \tilde{h}(x^{\eta_1}(s)) - \tilde{h}(x^{\eta_2}(s)) + B(u_1(s) - u_2(s))ds \|$$

$$\leq \kappa_1 \int_0^t \|x^{\eta_1}(s) - x^{\eta_2}(s)\| ds + \| \int_0^t B(u_1(s) - u_2(s))ds \| , \qquad \text{(A.10a)}$$

by Assumption 3.1*(a)*. Using the Bellman-Gronwall lemma, we conclude that for any $t \in [0, 1]$,

$$\|x^{\eta_1}(t) - x^{\eta_2}(t)\| \leq \kappa_1 e^{\kappa_1} \|B\| \| \int_0^t (u_1(s) - u_2(s))ds \| . \qquad \text{(A.10b)}$$

Now, let $\dot{z}^1(t) \doteq u_1(t)$, $t \in [0, 1]$, $z^1(0) = \xi$ and $\dot{z}^2(t) \doteq u_2(t)$, $t \in [0, 1]$, $z^2(0) = \xi$. Let $\bar{z}_k^1$ and $\bar{z}_k^2$, $k \in \mathcal{N}$ be the computed solution of $z^1(t)$ and $z^2(t)$, respectively, using the RK method under consideration. We note that $\bar{z}_k^1 = \bar{z}_k^2$ for all $k \in \mathcal{N}$ since $V_{\mathbf{A},N}^1(u_1) = V_{\mathbf{A},N}^2(u_2)$. Then, since u_1 is an r-th order polynomial, any ρ-th order RK method, $\rho \geq r$, integrates $\dot{z}^1(t)$ exactly. Hence, $\bar{z}_k^1 = z^1(t_k)$, for all $k \in \mathcal{N}$. Also, from (2.4.3a,b),

$$\bar{z}_{k+1}^2 = \bar{z}_k^2 + \sum_{i=1}^{s} b_i u_2[\tau_{k,i}] = \bar{z}_k^2 + \sum_{j=1}^{r} \int_{t_k+d_{j-1}}^{t_k+d_j} u_2(s)ds = z^2(t_{k+1}) , \qquad \text{(A.10c)}$$

since $\tau_{k,j} \in [t_k + d_{j-1}, t_k + d_j)$ (by Assumption 4.6) with $u_2(\cdot)$ constant on these intervals, and $d_r = \Delta$ by Assumption 4.1'. Since $\bar{z}_k^1 = \bar{z}_k^2$, we must have

$$z^1(t_k) - z^2(t_k) = \bar{z}_k^1 - \bar{z}_k^2 = 0 , \quad \forall\, k \in \mathcal{N}. \qquad \text{(A.10d)}$$

Hence, we conclude that

$$\| \int_0^{t_k} (u_1(s) - u_2(s))ds \| = \|z^1(t_k) - z^2(t_k)\| = 0 . \qquad \text{(A.10e)}$$

Therefore,

$$\|x^{\eta_2}(t_k) - \bar{x}_k^{\eta_2}\| \leq \|x^{\eta_2}(t_k) - x^{\eta_1}(t_k)\| + \|x^{\eta_1}(t_k) - \bar{x}_k^{\eta_1}\| + \|\bar{x}_k^{\eta_1} - \bar{x}_k^{\eta_2}\| \leq \kappa'/N^{\rho} , \quad \forall\, k \in \mathcal{N}, \qquad \text{(A.10f)}$$

where we have used (A.10b) and (A.10e) for the first term, the fact that $\|x^{\eta_1}(t_k) - \bar{x}_k^{\eta_1}\| \leq \kappa_2/N^{\rho}$ since (2.4.18c) holds for $\eta_1 \in S \cap H_N^1$ by the first part of this discussion for the second term, and the fact that $\bar{x}_k^{\eta_1} = \bar{x}_k^{\eta_2}$ since $u_1[\tau_{k,i}] = u_2[\tau_{k,i}]$ for the third term. Thus (2.4.18c) holds for representation **R2** under the stated conditions. □

Lemma A.4. Suppose that Assumptions 3.1, 4.1' and 4.3 hold for representation **R1** and that Assumptions 3.1, 4.1', and 4.6 hold for representation **R2**. For any $S \subset \mathbf{B}$ bounded, there exists $\kappa < \infty$ and $N^* < \infty$ such that for any $\eta \in S \cap \mathbf{H}_N$ and $N \geq N^*$,

$$\| \bar{p}_k^\nu - p^\nu(t_k) \| \leq \frac{\kappa}{N} \ , \ \ k \in \{ 0, \ldots, N \} \ , \ \ \nu \in \mathbf{q} \ , \tag{A.14}$$

where $p^\nu(\cdot)$ is the solution to the adjoint differential equation (2.3.6c) and $\{ \bar{p}_k^\nu \}_{k=0}^N$ is the solution to the corresponding adjoint difference equation (2.5.5d).

Proof. Proceeding as in the proof of Lemma 4.10*(i)*, if we define $e_{k+1} \doteq \bar{p}_{k+1}^\nu - p^\nu(t_{k+1})$ we can show that

$$\| e_k \| \leq L_1 \| e_{k+1} \| + L_2 \Delta^2 \ , \quad k \in \mathcal{N}, \tag{A.15}$$

where $L_1, L_2 < \infty$, using *(i)* the fact that

$$\bar{p}_k^\nu = F_x(\bar{x}_k, \bar{u}_k)^T \bar{p}_{k+1} = \bar{p}_{k+1}^\nu + \Delta \sum_{i=1}^{s} b_i h_x(\bar{x}_k, u[\tau_{k,i}])^T \bar{p}_{k+1} + O(\Delta^2) \ , \tag{A.16}$$

(ii) Lemma A.1 with $h(x(t_k), u[\tau_{k,i}])$ replaced by $-h_x(x(t_k), u[\tau_{k,i}])^T p^\nu(t_{k+1})$ and *(iii)* the result of Lemma 4.10*(i)* that $\| x(t_k) - \bar{x}_k \| \leq \kappa \Delta$ for all $k \in \mathcal{N}$. Now, by Assumption 3.1*(b)* and Lemma 4.10*(i)*, there exists $\kappa_1 < \infty$ such that

$$\| e_N \| = \| \bar{p}_N^\nu - p^\nu(1) \| \leq \| \zeta_x(\xi, \bar{x}_N)^T - \zeta_x(\xi, x(1))^T \| \leq \kappa_1 \| \bar{x}_N - x(1) \| \leq \kappa_2 \Delta \ , \tag{A.17}$$

where $\kappa_2 = \kappa \kappa_1$. Thus, solving (A.15) we conclude that for all $k \in \mathcal{N}$,

$$\| e_k \| \leq (L_1)^N \| e_N \| + L_2{}' \Delta \ , \tag{A.18}$$

which, with (A.17), proves (A.14). □

APPENDIX B

This appendix describes several optimal control problem examples that are used in Chapter 3 and the RIOTS user's manual.

Problem: LQR [42].

$$\min_u J(u) \doteq \int_0^1 0.625x^2 + 0.5xu + 0.5u^2 \, dt$$

subject to

$$\dot{x} = \tfrac{1}{2}x + u \quad ; \quad x(0) = 1 \ .$$

This problem has an analytic solution given by

$$u^*(t) = -(\tanh(1-t) + 0.5)\cosh(1-t)/\cosh(1) \ , \quad t \in [0,1] \ ,$$

with optimal cost $J^* = e^2 \sinh(2)/(1+e^2)^2 \approx 0.380797$.

Problem: Rayleigh [26,156].

$$\min_u J(u) \doteq \int_0^{2.5} x_1^2 + u^2 \, dt$$

subject to

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) & x_1(0) &= -5 \\ \dot{x}_2(t) &= -x_1(t) + [1.4 - 0.14x_2^2(t)]x_2(t) + 4u(t) & x_2(0) &= -5 \end{aligned}$$

A constrained version of this problem is formed by including the state constraint

$$x_1(2.5) = 0 \ .$$

Problem: Bang [3, p. 112].

$$\min_{u,T} J(u,T) \doteq T$$

subject to

$$\dot{x}_1 = x_2 \quad ; \quad x_1(0) = 0 \ , \quad x_1(T) = 300$$

$$\dot{x}_2 = u \quad ; \quad x_2(0) = 0 \ , \quad x_2(T) = 0 \ ,$$

and

$$-2 \le u(t) \le 1 \ , \ \forall\, t \in [0,T] \ .$$

This problem has an analytic solution which is given by $T^* = 30$ and

	$0 \le t < 20$	$20 \le t \le 30$
$u^*(t)$	1	-2
$x_1^*(t)$	$t^2/2$	$-t^2 + 60t - 600$
$x_2^*(t)$	t	$60 - 2t$

Problem: Obstacle [78].

$$\min_{u} J(u) \doteq 5x_1(2.9)^2 + x_2(2.9)^2$$

subject to

$$\dot{x}_1 = x_2 \qquad x_1(0) = 1$$

$$\dot{x}_2 = u - 0.1(1 + 2x_1^2)x_2 \qquad x_2(0) = 1$$

$$-1 \le u(t) \le 1 \ , \ \forall\, t \in [0,1]$$

$$1 - 9(x_1(t) - 1)^2 - \left(\frac{x_2(t) - 0.4}{0.3}\right)^2 \le 0 \ , \ \forall\, t \in [0,1]$$

$$-0.8 - x_2(t) \le 0 \ , \ \forall\, t \in [0,1] \ .$$

Problem: Goddard Rocket, Maximum Ascent [157].

$$\max_{u,T} J(u,T) \doteq h(T)$$

subject to:

$$\dot{v} = \frac{1}{m}(u - D(h,v)) - \frac{1}{h^2}\ , \ \ D(h,v) = \tfrac{1}{2}\, C_D A \rho_0 v^2 e^{\beta(1-h)} \qquad v(0) = 0$$

$$\dot{h} = v \qquad h(0) = 1$$

$$\dot{m} = -\frac{1}{c}\, u \qquad m(0) = 1 \ \ ; \ \ m(T) = 0.6$$

$$0 \le u(t) \le 3.5\ , \ \ \forall\, t \in [0,T]\ .$$

where $\beta = 500$, $C_D = 0.05$ and $A\rho_0 = 12{,}400$. The variables used above have the following meanings:

v	vertical velocity
h	radial altitude above earth ($h = 1$ is earth's surface)
m	mass of vehicle
u	thrust
c	specific impulse (impulse per unit mass of fuel burned, $c = 0.5$)
ρ	air density ($\rho = \rho_0 e^{\beta(1-h)}$)
q	dynamic pressure ($q = \frac{1}{2}\rho v^2$)
D	drag

The endpoint constraint $m(T) = 0.6$ means that there is no more fuel left in the rocket. Another version of this problem includes the trajectory constraint

$$Aq(t) \le 10\ , \ \ \forall\, t \in [0,T]\ .$$

This is a upper bound on the dynamic pressure experienced by the rocket during ascent.

Problem: Switch [3, pp. 120-123,37].

$$\min_u J(u) \doteq \int_0^1 \tfrac{1}{2}\, u^2 \, dt$$

subject to

$$\dot{x} = v \quad ; \quad x(0) = 0\ , \ \ x(1) = 0$$

$$\dot{v} = u \quad ; \quad v(0) = 1\ , \ \ v(1) = -1$$

$$x(t) - L \le 0\ , \ \ \forall\, t \in [0, 1]\ ,$$

with $L = 1/9$. This problem has an analytic solution. For any L such that $0 < L \le 1/6$, the solution is $J^* = \frac{4}{9L}$ with

	$0 \le t < 3L$	$3L \le t < 1 - 3L$	$1 - 3L \le t \le 1$
$u^*(t)$	$-\frac{2}{3L}\,(1 - \frac{t}{3L})$	0	$-\frac{2}{3L}\,(1 - \frac{1-t}{3L})$
$v^*(t)$	$(1 - \frac{t}{3L})^2$	0	$(1 - \frac{1-t}{3L})^2$
$x^*(t)$	$L(1 - (1 - \frac{t}{3L})^3)$	L	$L(1 - (1 - \frac{1-t}{3L})^3)$

REFERENCES

1. L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes,* John Wiley & Sons, New York (1962). (translated from Russian by K.N. Trirogoff)
2. E. Polak, *Computational Methods in Optimization,* Academic Press, New York (1971).
3. A. E. Bryson and Y. Ho, *Applied Optimal Control,* Hemisphere Publishing Corp. (1975). (revised printing)
4. O. Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operations Research* **37** pp. 357-373 (1992).
5. H. J. Pesch, "Real-time computation of feedback controls for constrained optimal control problems. Part I: Neighbouring extremals," *Optimal Control Applications and Methods* **10** pp. 129-145 (1989).
6. D. J. Mook and J. Lew, "Multiple shooting algorithms for jump-discontinuous problems in optimal control and estimation," *IEEE Trans. Autom. Ctnrl* **36**(8) pp. 979-983 (1991).
7. E. Polak, T. H. Yang, and D. Q. Mayne, "A method of centers based on barrier functions for solving optimal control problems with continuum state and control constraints," *SIAM J. Control and Optim.* **31**(1) pp. 159-179 (1993).
8. O. Pironneau and E. Polak, "A dual method for optimal control problems with initial and final boundary constraints," *SIAM J. Control* **11**(3) pp. 534-549 (1973).
9. D. Q. Mayne and E. Polak, "Feasible directions algorithms for optimization problems with equality and inequality constraints," *Math. Prog.* **11** pp. 67-80 (1976).
10. D. Q. Mayne and E. Polak, "A feasible directions algorithm for optimal control problems with terminal inequality constraints," *IEEE Trans. Autom. Control* **22**(5) pp. 741-751. (1977).
11. D. Q. Mayne and E. Polak, "An exact penalty function algorithm for control problems with control and terminal equality constraints," *J. Optim. Theory and Appl.* **32**(2) pp. 211-246 (1980). Part 1
12. D. Q. Mayne and E. Polak, "An exact penalty function algorithm for control problems with control and terminal equality constraints," *J. Optim. Theory and Appl.* **32**(3) pp. 345-364 (1980). Part 2

13. D. Q. Mayne and E. Polak, "An exact penalty function algorithm for control problems with state and control constraints," *IEEE Trans. Autom. Cntrl* **32**(5) pp. 380-387 (1987).

14. K. Shimizu and S. Ito, "Constrained optimization in Hilbert space and a generalized dual quasi-Newton algorithm for state-constrained optimal control problems," *IEEE Trans. Autom. Cntrl.* **39**(5) pp. 982-986 (1994).

15. P. R. Turner and E. Huntley, "Self-scaling variable metric methods in Hilbert space with applications to control problems," *Optimal Control Applications and Methods* **1** pp. 155-166 (1980).

16. H.R. Sirisena and K.S. Tan, "Computation of constrained optimal controls using parameterization techniques," *IEEE Trans. Autom. Cntrl.* **19**(4) pp. 431-433 (1974).

17. D. Shih and F. Kung, "Optimal control of deterministic systems via shifted Legendre polynomials," *IEEE Trans. Autom. Cntrl* **31**(5) pp. 451-454 (1986).

18. J. Vlassenbroeck and R. V. Dooren, "A Chebyshev technique for solving nonlinear optimal control problems," *IEEE Trans. Autom. Cntrl.* **33**(4) pp. 333-340 (1988).

19. E. R. Edge and W. F. Powers, "Function-space quasi-Newton algorithms for optimal control problems with bounded controls and singular arcs," *J. Optim. Theory and Appl.* **20**(4) pp. 455-479 (1976).

20. E. Polak and D. Q. Mayne, "First order, strong variations algorithms for optimal control problems with terminal inequality constriants," *J. Optim. Theory and Appl.* **16**(3/4) pp. 303-325 (1975).

21. C. T. Kelley and E. W. Sachs, "Quasi-Newton methods and unconstrained optimal control problems," *SIAM J. Control and Optim.* **25**(6) pp. 1503-1516 (1987).

22. J. K. Willoughby and B. L. Pierson, "A constraint-space conjugate gradient method for function minimization and optimal control problems," *Int. J. Control* **14** pp. 1121-1135 (1971).

23. A. Miele and T. Wang, "Primal-dual properties of sequential gradient-restoration algorithms for optimal control problems 2: General problem," *J. Math. Anal. and Appl.* **119** pp. 21-54 (1986).

24. K. C. P. Machielsen, "Numerical Solution of Optimal Control Problems with State Constraints by Sequential Quadratic Programming in Function Space," in *CWI-Tract*, Centrum voor Wiskunde en informatica, Amsterdam, the Netherlands (1988).

25. L. S. Jennings, M. E. Fisher, K. L. Teo, and C. J. Goh, "MISER3: Solving optimal control problems---an update," *Advances in Engineering software* **14**(13) pp. 190-196 (1991).

26. N. B. Nedeljković, “New algorithms for unconstrained nonlinear optimal control problems,” *IEEE Trans. Autom. Cntrl.* **26**(4) pp. 868-884 (1981).

27. H. R. Sirisena and F. S. Chou, “Convergence of the control parmaeterization Ritz method for nonlinear optimal control problems,” *J. Optim. Theory and Appl.* **29**(3) pp. 369-382 (1979).

28. W. W. Hager, “The Ritz-Trefftz method for state and control constrained optimal control problem,” *SIAM J. Numer. Anal* **12**(6) pp. 854-867 (1975).

29. F.H. Mathis and G.W. Reddien, “Ritz-Trefftz approximations in optimal control,” *SIAM J. Control and Optim.* **17**(2) pp. 307-310 (1979).

30. F. H. Mathis, “An L_∞ error estimate for the Ritz-Trefftz approximation in optimal control,” *Mathematics and Computers in Simulation* **XXIII** pp. 188-190 (1981).

31. W. W. Hager, “Dual approximations in optimal control,” *SIAM J. Control and Optim.* **22**(3) pp. 423-465 (1984).

32. J. G. Renfro, A. M. Morshedi, and O. A. Asbjornsen, “Simultaneous optimization and solution of systems described by differential equations,” *Comput. chem. Engng.* **11**(5) pp. 503-517 (1987).

33. J. E. Cuthrell and L. T. Biegler, “On the optimization of differential-algebraic process systems,” *AIChE Journal* **3**(1/2) pp. 1257-1270 (1987).

34. J. E. Cuthrell and L. T. Biegler, “Simultaneous optimization and solution methods for batch reactor control profiles,” *Computers Chem. Engng* **13** pp. 49-62 (1989).

35. C. P. Neuman and A. Sen, “A suboptimal control algorithm for constrained problems using cubic splines,” *Automatica* **9** pp. 601-613 (1973).

36. G. W. Reddien, “Collocation at Gauss points as a discretization in optimal control,” *SIAM J. Control and Optim.* **17**(2) pp. 298-306 (1979).

37. O. Stryk, “Numerical solution of optimal control problems by direct collocation,” *International Series of Numerical Methematics* **111** pp. 129-143 (1993).

38. J. T. Betts and P. D. Frank, “A sparse nonlinear optimization algorithm,” *J. Optim. Theory and Appl.* **82**(3) pp. 519-541 (1994).

39. G. Pillo, L. Grippo, and F. Lampariello, “A class of structured quasi-Newton algorithms for optimal control problems,” pp. 101-107 in *Proc. IFAC Appl. of Nonlinear Prog. to Optim. and Control*, , Palo Alto (1983).

40. C. R. Hargraves and S. W. Paris, “Direct trajectory optimization using nonlinear

programming and collocation," *J. Guidance* **10** pp. 338-342 (1987).

41. O. Stryk, "Numerische Losung optimaler Steuerungsprobleme: Diskretisierung, Parameteroptimierung und erechnung der adjungierten Variablen," Diploma-Math., Munchen University of Technology, VDI Verlag, Germany (1995).

42. W.W. Hager, "Rates of convergence for discrete approximations to unconstrained control problems," *SIAM J. Numer. Anal.* **13**(4) pp. 449-472 (1976).

43. E. Polak, "On the use of consistent approximations in the solution of semi-infinite optimization and optimal control problems," *Math. Prog.* **62** pp. 385-415 (1993).

44. J. W. Daniel, *The Approximate Minimization of Functionals,* Prentice-Hall, New Jersey (1971).

45. H. Attouch, *Variational Convergence for Functions and Operators,* Pitman, London (1984).

46. S. Doleck, G. Salinetti, and R. J.B. Wets, "Convergence of functions: equisemicontinuity," *Transactions of the American Mathematical Society*, (276) p. 429 (1983).

47. J. P. Aubin and H. Frankowska, *Set-Valued Analysis,* Birkhauser, Boston (1990).

48. B. M. Budak, E. M. Berkovich, and E. N. Solov'eva, "Difference approximations in optimal control problems," *SIAM J. Control* **7**(1) pp. 18-31 (1969).

49. J. Cullum, "Discrete approximations to continuous optimal control problems," *SIAM J. Control* **7**(1) pp. 32-49 (1969).

50. J. Cullum, "An Explicit procedure for discretizing continuous, optimal control problems," *Journal of Optim. Theory and Appl.* **8**(1) pp. 15-35 (1971).

51. J. W. Daniel, "The Ritz-Galerkin method for abstract optimal control problems," *SIAM J. Control* **11**(1) pp. 53-63 (1973).

52. W. E. Bosarge, O. G. Johnson, R. S. McKnight, and W. P. Timlake, "The Ritz-Galerkin procedure for nonlinear control problems," *SIAM J. Numer. Anal* **10**(1) pp. 94-111 (1973).

53. B. Sh. Mordukhovich, "On difference approximations of optimal control systems," *J. Appl. Math. Mech* **42** pp. 452-461 (1978).

54. B. Sh. Mordukhovich, *Methods of approximation in optimal control problems*, (in Russian) 1988.

55. E. Polak and L. He, "Rate-preserving discretization strategies for semi-infinite programming and optimal control," *SIAM J. Control and Optim.* **30**(3) pp. 548-572 (1992).

56. V. Veliov, "Second-order discrete approximations to linear differential inclusions," *SIAM J. Numer. Anal.* **29**(2) pp. 439-451 (1992).

57. L. He and E. Polak, “An optimal diagonalization strategy for the solution of a class of optimal design problems,” *IEEE Trans. on Autom. Contr.* **35** pp. 258-267 (1990).

58. T. E. Baker and E. Polak, “On the optimal control of systems described by evolution equations,” *SIAM J. Control and Optim.* **32** pp. 224-260 (1994).

59. F. H. Clarke, *Optimization and Nonsmooth Analysis,* Wiley-Interscience, New York (1983).

60. Haim Brezis, *Analyse Fonctionnelle,* Masson, Paris (1983). (in French)

61. J. C. Butcher, *The Numerical Analysis of Ordinary Differential Equations,* John Wiley and Sons, England (1987).

62. J. D. Lambert, *Numerical Methods for Ordinary Differential Systems,* John Wiley and Sons, England (1991).

63. Carl de Boor, *A Practical Guide to Splines,* Springer-Verlag, New York (1978).

64. E. Polak and L. He, “Unified steerable phase I-phase II method of feasible directions for semi-infinite optimization,” *J. Optim. Theory and Appl.* **69**(1) pp. 83-107 (1991).

65. C. Lawrence, J. L. Zhou, and A. L. Tits, “User’s guide for CFSQP version 2.1: A C Code for solving (large scale) constrained nonlinear (minimax) optimization problems,” TR-94-16rl, Institute for Systems Researce, Univ. of Maryland (1994).

66. U. Ascher and G. Bader, “Stability of collocation at Gaussian points,” *SIAM J. Numer. Anal.* **23**(2) pp. 412-422 (1986).

67. R. Schere and H Turke, “Reflected and transposed Runge-Kutta methods,” *BIT* **23** pp. 262-266 (1983).

68. J. M. Lane an R. F. Riesenfeld, “A theoretical development for the computer generation of piecewise polynomial surfaces,” *IEEE Trans. Patern Anal. and Machine Inteligence* **2** pp. 35-46 (1980).

69. W. Boehm, “Inserting new knots into B-spline curves,” *CAD* **12** pp. 199-216 (1980).

70. R. Qu and J. A. Gregory, “A subdivision algorithm for non-uniform B-splines,” pp. 432-436 in *Approximation Theory, Spline Functions and Applications*, ed. S. P. Singh (ed.), Kluwer Academic Publishers, Boston (1992).

71. W. Rudin, *Real and Complex Analysis,* McGraw-Hill, New York (1987).

72. G. H. Golub and C. F. Loan, *Matrix Computations,* Johns Hopkins University Press (1989). (second edition)

73. A. A. Goldstein, “Convex programming in Hilbert space,” *Bull. Amer. Math. Soc.* **70** pp. 709-710 (1964).

74. E. S. Levitin and B. T. Polyak, "Constrained minimization problems," *USSR Comput. Math. Math. Phys.* **6** pp. 1-50 (1966). (English transl. in *Zh. Vychisl. Mat. i Mat. Fiz.*, vv. 6, pp. 787-823, 1965)

75. D. P. Bertsekas, "On the Goldstein-Levitin-Poljak gradient projection method," *IEEE Trans. Autom. Cntrl.* **21**(2) pp. 174-184 (1976).

76. D. P. Bertsekas, "Projected Newton methods for optimization problems with simple constraints," *SIAM J. Control and Optim.* **20**(2) pp. 221-246 (1982).

77. J. C. Dunn, "A projected Newton method for minimization problems with nonlinear inequality constraints," *Numer. Math.* **53** pp. 377-409 (1988).

78. V. H. Quintana and E. J. Davison, "Clipping-off gradient algorithms to compute optimal controls with constrained magnitude," *Int. J. Control* **20**(2) pp. 243-255 (1974).

79. A. R. Conn, N. Gould, and P. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM J. Numer. Anal.* **25** pp. 433-460 (1988).

80. A. R. Conn, N. Gould, and P. L. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM J. Numer. Anal.* **28**(2) pp. 545-572 (1991).

81. R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," Technical Report NAM-08, EECS Dept., Northwestern Univ. (1994).

82. J. J. More and G. Toraldo, "Algorithms for bound constrained quadratic programming problems," *Numer. Math.* **55** pp. 377-400 (1989).

83. P. H. Calamai and J. J. More, "Projected gradient methods for linearly constrained problems," *Math. Prog.* **39** pp. 93-116 (1987).

84. C. T. Kelley and E. W. Sachs, "Solution of optimal control problems by a pointwise projected Newton method," *SIAM J. Contr. and Optim.* **43** pp. 1731-1757 (1995).

85. E. Polak, R. W. Sargent, and D. J. Sebastian, "On the convergence of sequential minimization algorithms," *J. Optim. Theory and Appl.* **14** pp. 439-442 (1974).

86. J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Mathematics of Computation* **35**(151) pp. 773-782 (1980).

87. M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Math. Prog.* **12** pp. 241-254 (1977).

88. D. G. Luenberger, *Linear and Nonlinear Programming,* Addison-Wesley Pub. Co., Reading, Massachusetts (1984). (second edition)

89. D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods,* Academic Press, New York (1982).

90. D. G. Luenberger, “Convergence rate of a penalty-function scheme,” *J. Optim. Theory and Appl.* **7**(1) pp. 39-51 (1971).

91. J. C. Gilbert and J. Nocedal, “Global convergence properties of conjugate gradient methods for optimization,” *SIAM J. Optimization* **2**(1) pp. 21-42 (1992).

92. D. P. Bertsekas, “Partial conjugate graident methods for a class of optimal control problems,” *IEEE Trans. Autom. Cntrl.* **19**(3) pp. 209-217 (1974).

93. J. C. Dunn and D. P. Bertsekas, “Efficient dynamic programming implementations of Newton’s method for unconstrained optimal control problems,” *J. Optim. Theory and Appl.* **63**(1) pp. 23-38 (1989).

94. D. C. Liu and J. Nocedal, “On the limited memory BFGS method for large scale optimization,” *Math. Prog.* **45** pp. 503-528 (1989).

95. D. F. Shanno and K. H. Phua, “Matrix conditioning and nonlinear optimization,” *Math. Prog.* **14** pp. 149-160 (1978).

96. X. Zou, I. M. Navon, M. Berger, K. H. Phua, T. Schlick, and F. X. Dimet, “Numerical experience with limited-memory quasi-Newton and truncated newton methods,” *SIAM J. Optim.* **3**(3) pp. 582-608 (1993).

97. W. W. Hager, “Lipschitz continuity for constrained processes,” *SIAM J. Control and Optim.* **17**(3) pp. 321-338 (1979).

98. W. W. Hager and G. Strang, “Free boundaries and finite elements in one dimension,” *Math. Comp.* **29**(132) pp. 1020-1031 (1975).

99. J. E. Higgins and E. Polak, “An ε-active barrier-function method for solving minimax problems,” *Appl. Math. Optim.* **23** pp. 275-297 (1991).

100. M. E. Hosea and L. F. Shampine, “Estimating the error of the classic Runge-Kutta formula,” *Applied Math. and Comp.* **66** pp. 217-226 (1994).

101. R. Eng, “Error estimates for Runge-Kutta type solutions to systems of ordinary differential equations,” *Computer Journal* **12**(2) pp. 166-170 (1969).

102. F. Ceschnino and J. Kuntzmann, *Numerical Solution of Initial Value Problems,* Prentice-Hall, Englewood Cliffs, NJ (1966). (English transl.)

103. L. F. Shampine and H. A. Watts, “Comparing error estimators for Runge-Kutta methods,” *Mathematics of Computation* **25**(115) pp. 445-455 (1971).

104. C. Jansch and M. Paus, "Aircraft trajectory optimization with direct collocation using movable gridpoints," pp. 262-267 in *Proc. American Control Conference*, , San Diago (1990).

105. J. T. Betts and W. P. Huffman, "Path-constrained trajectory optimization using sparse sequential quadratic programming," *J. Guidance, Control, and Dynamics* **16**(1) pp. 59-68 (1993).

106. F. B. Lee and L. Markus, *Foundations of Optimal Control Theory,* John Wiley, New York (1967).

107. V. M. Alekseev, V. M. Tikhomirov, and V. M. Fomin, *Optimal Control,* Consultants Bureau, New York (1987). (translated from Russian by V.M. Volosov)

108. J. C. Dunn, "Second-order optimality conditions in sets of L^∞ functions with range in a polyhedron.," *SIAM J. Control and Optim.* **33**(5) pp. 1603-1635 (1995).

109. J. C. Dunn, "L^2 sufficient conditions for end-constrained optimal control problems with inputs in a polyhedron.," *pre-print*, (1996).

110. J. C. Dunn, "On L^2 sufficient conditions and the gradient projection method for optimal control problems.," *SIAM J. Control and Optim.*, (July, 1996).

111. J. C. Dunn and T. Tian, "Variants of the Kuhn-Tucker sufficient conditions in cones of nonnegative functions," *SIAM J. Control and Optim.* **30**(6) pp. 1361-1384 (1992).

112. H. Maurer, "First and second order sufficient optimality conditions in mathematical programming and optimal control," pp. 163-177 in *Mathematical Programming Study*, North-Holland Publishing Co. (1981).

113. A. L. Dontchev, W. W. Hager, A. B. Poore, and B. Yang, "Optimality, stability, and convergence in nonlinear control," *Applied Math. and Optim.* **31**(3) pp. 297-326 (1995).

114. H. J. Pesch, "Real-time computation of feedback controls for constrained optimal control problems. Part II: A Correction Method Based on Multiple Shooting," *Optimal Control Applications and Methods* **10** pp. 147-171 (1989).

115. R. Bulirsch, E. Nerz, H. J. Pesch, and O. Stryk, "Combining direct and indirect methods in optimal control: range maximization of a hang glider," pp. 273-288 in *International Series of Numerical Mathematics*, (1993).

116. D. J. Bell and D. H. Jacobson, *Singular Optimal Control Problems,* Academic Press, London (1975).

117. J. P. McDanell and W. F. Powers, "Necessary conditions for joining optimal singular and nonsingular subarcs," *SIAM J. Control* **9** pp. 161-173 (1971).

118. B. Goh, “Compact forms of the generalized Legendre-Clebsch conditions and the computation of singular control trajectories,” pp. 3410-3413 in *Proc. ACC*, , Seattle, WA (June 1995).

119. H. Seywald, “Trajectory optimization based on differential inclusion,” *J. Guidance, Control and Dynamics* **17**(3) pp. 480-487 (1994).

120. Y. Chen and J. Huang, “A numerical algorithm for singular optimal control synthesis using continuation methods,” *Optimal Control Appl. and Methods* **15** pp. 223-236 (1994).

121. S. A. Dadebo and K. B. McAuley, “On the computation of optimal singular controls,” pp. 150-155 in *Proceedings of the 4th IEEE Conference of Control Applications*, (Sept. 1995).

122. K. L. Teo and L. S. Jennings, “Optimal control with a cost on changing control,” *J. Optim. Theory and Appl.* **68** pp. 335-357 (1991).

123. R. T. Rockafellar, “Monotone operators and the proximal point algorithm,” *SIAM J. Control and Optim.* **14**(5) pp. 877-898 (1976).

124. R. T. Rockafellar, “Augmented Lagrangians and applications of the proximal point algorithm in convex programming,” *Mathematics of Operations Research* **1**(2) pp. 97-116 (1976).

125. P. E. Gill, W. M. Murray, M. A. Saunders, and M. H. Wright, “User’s guide for NPSOL (Version 4.0): A Fortran package for nonlinear programming,” technical report SOL 86-2, Systems Opitimzation Laboratory, Stanford University (1986).

126. J. Cullum, “Finite-dimensional approximations of state-constrained continuous optimal control problems,” *SIAM J. Control* **10**(4) pp. 649-670 (1972).

127. K. Radhakrishnan and A. C. Hindmarsh, “Description and use of LSODE, the Livermore Solver for Ordinary Differential Equations,” NASA Reference Publ. 1327 (1993).

128. L. R. Petzold, “Automatic selection of methods for solving stiff and nonstiff systems of differential equations,” *SIAM J. Sci. Stat. Comput.* **4** pp. 136-148 (1983).

129. P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization,* Academic Press, London (1981).

130. L. T. Biegler and J. E. Cuthrell, “Improved infeasible path optimization for sequential modular simulators--II: the optimization algorithm,” *Computers & Chemical Engineering* **9**(3) pp. 257-267 (1985).

131. A. Griewank, D. Juedes, and J. Utke, *ADOL-C: A package for the automatic differentiation of algorithms written in C/C++,* Argonne National Laboratory, ftp://info.mcs.anl.gov/pub/ADOLC (December 1993).

132. A. Griewank, “On automatic differentiation,” Preprint MCS-P10-1088, Argonne National Laboratory, ftp://info.mcs.anl.gov/pub/tech_reports/reports (October 1988).

133. D. M. Murray and S. J. Yakowitz, “Differential dynamic programming and Newton’s method for discrete optimal control problems,” *Journal of Optim. Theory and Appl.* **43**(3) pp. 395-414 (1984).

134. J. F. A. Pantoja, “Differential dynamic programming and Newton’s method,” *Int. J. Control* **47**(5) pp. 1539-1553 (1988).

135. S. K. Mitter, “Successive approximation methods for the solution of optimal control problems,” *Automatica* **3** pp. 135-149 (1966).

136. D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming,* American Elsevier Pub. Co., New York (1970).

137. A. Rakshit and S. Sen, “Sequential rank-one/rank-two updates for quasi-Newton differential dynamic programing,” *Optimal Control Appl. and Methods* **11** pp. 95-101 (1990).

138. J. F. A. Pantoja and D. Q. Mayne, “Sequential quadratic programming algorithm for discrete optimal control problems with control inequality constraints,” *Int. J. Control* **53**(4) pp. 823-836 (1991).

139. T. F. Coleman and A. Liao, “An efficient trust region method for unconstrained discrete-time optimal control problems,” *Computational Optimization and Applications* **4** pp. 47-66 (1995).

140. Henrik Jonson, “Newton Method for Solving Non-linear Optimal Control Problems with Genereal constraints,” Ph.D. Dissertation, Linkoping Studies in Science and Technology (1983).

141. R. W. H. Sargent, “A new SQP algorithm for large-scale nonlinear programming,” C95 36, Centre for Process Systems Engineering, Imperial College, London (1995).

142. A. R. Conn, N. Gould, and P. L. Toint, “LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization,” in *Springer Series in Computational Mathematics*, Springer-Verlag, Berlin (1992).

143. A. Griewank and P. L. Toint, “Local convergence analysis for partitioned quasi-Newton updates,” *Numer. Math.* **39** pp. 429-448 (1982).

144. P. L. Toint, “Global convergence of the partitioned BFGS algorithm for convex partially separable optimization,” *Math. Prog.* **36** pp. 290-306 (1986).

145. M. J. D. Powell, “A fast algorithm for nonlinearly constrained optimization calculations,” pp. 144-157 in *Lecture Notes in Mathematics 630, Numerical Analysis*, ed. G.A. Watson

(ed.), Springer-Verlag (1978).

146. R. Fletcher, “Resolving degeneracy in quadratic programming,” *Annals of Operations Research* **47** pp. 307-334 (1993).

147. W. Murray, “Algorithms for large nonlinear problems,” pp. 172-185 in *Mathematical Programming--State of the Art*, (1994).

148. J. Nocedal, “Recent advances in large-scale nonlinear optimization,” pp. 208-219 in *Mathematical Programming--State of the Art*, (1994).

149. A. R. Conn, N. Gould, and P. L. Toint, “Large-scale nonlinear constrained optimization: a current survey,” pp. P 287-332 in *Algorithms for Continuous Optimization*, ed. E. Spedicator (ed.), Kluwer Academic Publishers, Boston (1994).

150. J. L. Zhou and A. L. Tits, “An SQP algorithm for finely discretized continuous minimax problems and other minimax problems with many objective functions,” *to appear in SIAM J. Optimization*, ().

151. U. Ascher, R. Mattheij, and R. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations,* Prentice Hall, Englewood Cliffs, NJ (1988).

152. S. S. Oren, “Perspectives on self-scaling variable metric algorithms,” *J. Optim. Theory and Appl.* **37**(2) pp. 137-147 (1982).

153. F.H. Mathis and G.W. Reddien, “Difference approximations to control problems with functional arguments,” *SIAM J. Control and Optim.* **16**(3) pp. 436-449 (1978).

154. D. I. Jones and J. W. Finch, “Comparison of optimization algortihms,” *Int. J. Control* **40** pp. 747-761 (1984).

155. S. Strand and J. G. Balchen, “A Comparison of Constrained Optimal Control Algorithms,” pp. 439-447 in *IFAC 11th Triennial World Congress*, , Estonia, USSR (1990).

156. D. Talwar and R. Sivan, “An Efficient Numerical Algorithm for the Solution of a Class of Optimal Control Problems,” *IEEE Trans. Autom. Cntrl.* **34**(12) pp. 1308-1311 (1989).

157. H. Seywald and E. M. Cliff, “Goddard Problem in Presence of a Dynamic Pressure Limit,” *J. Guidance, Control and Dynamics* **16**(4) pp. 776-781 (1993).