Global Dynamic Optimization

by

Adam Benjamin Singer

Submitted to the Department of Chemical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Chemical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author .....
Department of Chemical Engineering
June 10, 2004

Certified by .....
Paul I. Barton
Associate Professor of Chemical Engineering
Thesis Supervisor

Accepted by .....
Daniel Blankschtein
Professor of Chemical Engineering
Chairman, Committee for Graduate Students

Global Dynamic Optimization

by

Adam Benjamin Singer

Submitted to the Department of Chemical Engineering
on June 10, 2004, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Chemical Engineering

Abstract

My thesis focuses on global optimization of nonconvex integral objective functions subject to parameter dependent ordinary differential equations. In particular, efficient, deterministic algorithms are developed for solving problems with both linear and nonlinear dynamics embedded. The techniques utilized for each problem classification are unified by an underlying composition principle transferring the nonconvexity of the embedded dynamics into the integral objective function. This composition, in conjunction with control parameterization, effectively transforms the problem into a finite dimensional optimization problem where the objective function is given implicitly via the solution of a dynamic system. A standard branch-and-bound algorithm is employed to converge to the global solution by systematically eliminating portions of the feasible space by solving an upper bounding problem and convex lower bounding problem at each node. The novel contributions of this work lie in the derivation and solution of these convex lower bounding relaxations.

Separate algorithms exist for deriving convex relaxations for problems with linear dynamic systems embedded and problems with nonlinear dynamic systems embedded. However, the two techniques are unified by the method for relaxing the integral in the objective function. I show that integrating a pointwise in time convex relaxation of the original integrand yields a convex underestimator for the integral. Separate composition techniques, however, are required to derive relaxations for the integrand depending upon the nature of the embedded dynamics; each case is addressed separately.

For problems with embedded linear dynamic systems, the nonconvex integrand is relaxed pointwise in time on a set composed of the Cartesian product between the parameter bounds and the state bounds. Furthermore, I show that the solution of the differential equations is affine in the parameters. Because the feasible set is convex pointwise in time, the standard result that a convex function composed with an affine function remains convex yields the desired result that the integrand is convex under composition. Additionally, methods are developed using interval arithmetic to derive the exact state bounds for the solution of a linear dynamic system. Given a nonzero tolerance, the method is rigorously shown to converge to the global solution in a

finite time. An implementation is developed, and via a collection of case studies, the technique is shown to be very efficient in computing the global solutions.

For problems with embedded nonlinear dynamic systems, the analysis requires a more sophisticated composition technique attributed to McCormick. McCormick's composition technique provides a method for computing a convex underestimator for the integrand given an arbitrary nonlinear dynamic system provided that convex underestimators and concave overestimators can be given for the states. Because the states are known only implicitly via the solution of the nonlinear differential equations, deriving these convex underestimators and concave overestimators is a highly nontrivial task. Based on standard optimization results, outer approximation, the affine solution to linear dynamic systems, and differential inequalities, I present a novel method for constructing convex underestimators and concave overestimators for arbitrary nonlinear dynamic systems. Additionally, a method is derived to compute state bounds for nonquasimonotone ordinary differential equations. Given a nonzero tolerance, the relaxation method is proven to yield finite convergence to the global solution within a branch-and-bound framework. A detailed implementation for solving problems with nonlinear dynamic systems embedded is described. The implementation includes a compiler that automatically applies the theory and generates a Fortran residual file defining the upper and lower bounding problems. This residual file is utilized in conjunction with a discontinuity locking numerical differential equation solver, a local optimizer, and a custom branch-and-bound code to solve globally dynamic optimization problems with embedded nonlinear ordinary differential equations. To enable the comparison of the efficiency of the algorithm, several literature case studies are examined. A detailed analysis of a chemical engineering case study is performed to illustrate the utility of the algorithm for solving realistic problems.

Thesis Supervisor: Paul I. Barton
Title: Associate Professor of Chemical Engineering

Acknowledgments

I have always posited that the primary advantage of studying at MIT is neither the faculty nor the resources. No, I believe the primary advantage of studying at MIT is the high quality of one's coworkers. Not only did I learn a great deal from any number of countless conversations in the office, but it has also been my pleasure to work with the very talented students, postdocs, and research associates who have surrounded me for the past four and a half years. I am grateful that I have been afforded this small space to thank some people individually. I'd like to start by thanking Dr. John Tolsma for software support and, in particular, for support with the tools required for the implementation of the linear theory. I know I have teased John mercilessly about his software, but I hope he realizes that I have a great deal of respect for the work he has performed. While discussing software, I'd like to thank Jerry Clabaugh who was always available to answer general computer questions, especially those questions concerning the C programming language. I'd like to thank Dr. Edward Gatzke for some of the initial work he performed and code he shared concerning his branch-and-bound implementation. I'd like to thank Dr. David Collins with whom I bounced around many ideas on many occasions. Sometimes I feel David and I spent more time together fixing lab computers than doing research. I'd like to thank Dr. Binita Bhattacharjee for her assistance when we TAed 10.551 together. I think performing that duty together likely made the experience less painful for both of us. I'd like to thank Cha Kun Lee for daily discussions ranging anywhere from work related things to the NBA playoffs, and just about everything in between. I'd like to thank Alexander Mitsos for finally discovering the correct avenue to pursue to convince the department to buy us modern office chairs to replace the office chairs inherited from the Pilgrims. I'd like to thank Dr. Benoît Chachuat for our many conversations concerning both the nonlinear theory and it's implementation. I'd like to thank James Taylor for experimental data and assistance with the problems in Chapter 9. Finally, I'd like to thank all the other members of our research group I didn't mention by name and the research group as a whole simply for putting up with me for the last four and half

years.

The above list of people represent the individuals I wish to thank mostly for their professional assistance. However, completing a thesis involves much more than merely academics. First and foremost, I would like to thank my parents, Marc and Ellen, who have always been present to guide and advise me both personally and professionally. Without them, this thesis would not have existed, for on many occasions, their foresight into the benefits of this degree and their constant encouragement alone kept me from quitting out of annoyance. Next, I'd like to thank my friends in the student office, Suzanne Easterly, Jennifer Shedd, Annie Fowler, and Mary Keith. I think it was beyond their job description to listen to my incessant complaining. Last, but not least, I'd like to thank Sharron McKinney, who taught me a great deal about both life and love. At a time in my life full of questions, she helped me find answers within myself. I am more appreciative of the lessons I learned from her than of any I have ever learned in the hallowed halls of this institute.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The objective of this thesis is to develop efficient, deterministic algorithms for globally solving nonconvex optimization problems with an integral objective function subject to ordinary differential equations (ODEs). While the work was originally targeted at infinite dimensional variational problems, the main emphasis of this thesis concerns solving finite dimensional optimization problems containing parameter dependent ODEs. Because the problems addressed in this thesis contain finite dimensional degrees of freedom, a standard branch-and-bound algorithm was employed for converging sequences of rigorous upper and lower bounds to the global solution of the problem. In fact, the crux of the thesis itself is the efficient computation of convex relaxations for dynamic problems. During the course of my research, two seemingly distinct algorithms were developed: one for problems containing linear dynamic systems and one for problems containing nonlinear dynamic systems. However, both algorithms are unified by their approach to relaxing integral objective functions via pointwise in time relaxations of an integrand and their utilization of affine relaxations for the solutions of parameter dependent ODEs. Only after constructing both algorithms separately did I realize that the linear approach can be viewed as a true subset of the nonlinear approach.

In addition to developing efficient algorithms, I was also interested in developing efficient implementations of these algorithms, and I believe that overall, more research time was devoted to implementation than theory. Three generations of branch-and-

bound have been implemented and a discontinuity locking implementation for a stiff integration routine was written. Two complete implementations of the relaxation theory have been developed (one linear and one nonlinear), while a third generation based on algorithmic differentiation techniques is forthcoming. Despite four years of work on the subject, I believe I have merely scratched the surface; hopefully, I have completed enough work to allow and to inspire future generations of researchers to continue studying global dynamic optimization. Already, this work has been extended to encompass hybrid systems and mixed-integer dynamic optimization. New work is beginning in the optimization of differential-algebraic equations and partial differential equations. The remainder of this chapter motivates the research topic, reviews prior art, and outlines the structure of the thesis.

1.1 Motivation and Literature Review

Strong economic incentives exist for conducting chemical and biological process operations in the most efficient manner possible, particularly in the commodities sectors. With globalization of the world economy, these incentives will become even greater in the future. Moreover, an increasingly demand driven and competitive global economy is motivating many chemical and biological products to be manufactured in campaign continuous processes. These are flexible, continuous processes that produce a variety of different but related products or grades of product. Each individual product is made during a short, demand driven campaign, and then a changeover occurs to a new product or grade; this cycle continues repeatedly. One example includes polymer processes where many different molecular weight grades are produced in a single plant. Another example is in fertilizer processes, where many different N/K/P mixes are demanded by the market on a fluctuating basis. Campaign continuous processes may spend up to 50% of their total operating time in transitions between steady-states; thus, the performance of the process during these transients becomes as crucial as the steady-state performance. Therefore, consideration of operability and transient issues simultaneously with steady-state issues must be thoroughly addressed in order

to achieve the most economic overall process design.

Another particularly important aspect of the study of process operations is the relevance to environmental concerns. Most of the environmental impact of any continuous process is typically created during major transients, including off specification product, high energy consumption, byproducts from suboptimal reactor operating conditions, etc. This derives from the fact that steady-state design has become such a refined art that processes producing large quantities of waste at steady-state have become economically not viable. Process safety is another area in which process transients play a crucial role. A much wider variety of hazardous situations can develop in a plant that experiences frequent transients and constant changeovers; thus, much more careful and thorough attention must be paid to mitigating these hazards.

A majority of chemical and biological products are manufactured in processes that are operated in an inherently transient manner rather than at some nominal steady-state. Examples include batch processes, semi-continuous processes, and periodic process that operate at a cyclic steady-state. The design of inherently transient systems is, in essence, an exercise in optimal process operation. The batch mode of operation is the preferred method of manufacturing in the synthetic pharmaceutical, biotechnology, specialty polymers, electronic materials, and specialty agrochemicals industries. Moreover, much speculation presently exists that product design will emerge as the new paradigm of chemical engineering, and the related new chemical products will all be manufactured via batch processes. All of these considerations motivate research on design procedures for optimizing the performance of inherently transient manufacturing systems.

The mathematical framework in this thesis is designed to tackle problems in process operations for which a detailed, typically nonlinear, differential equation model is employed to describe the transient behavior of the physical system. Often, one is interested in determining input profiles or parameters for a dynamic model for the purpose of optimizing the operation of a system over some period of time according to a specified performance metric. Such problems are referred to as dynamic optimization problems or open-loop optimal control problems. Examples include deter-

mination of optimal operating profiles for batch unit operations [75], determination of optimal plant-wide recipes for batch processes [23], fitting of chemical reaction kinetics parameters to data [17], determination of optimal changeover policies [33], determination of catalyst blend profiles for PFRs [57], optimal drug scheduling for cancer chemotherapy [62], process safety analysis [1], optimization of chemical vapor deposition processes [74], etc.

One of the primary distinctions between optimizing a dynamic system and optimizing a steady-state system is that the degrees of freedom in a dynamic system lie in an infinite dimensional space while the degrees of freedom in a standard optimization problem lie in a finite dimensional space. For this reason, the methods employed for solving standard optimization problems cannot be immediately applied to dynamic systems; rather extensions of standard techniques must be employed. The techniques utilized for solving dynamic problems fall under two broad frameworks: variational methods and discretization methods. The sequel addresses these two methods separately expounding upon the virtues and deficiencies of each. I note, however, that the main thrust of this thesis lies in solving dynamic optimization problems via partial discretization.

The first technique, the variational approach, encompasses the classical methods of the calculus of variations and many of the modern methods of optimal control. These methods approach the problem in the original infinite dimensional space and attempt to determine stationary functions via the solution of the Euler-Lagrange equations. The variational approach for solving dynamic optimization problems is extremely attractive because by addressing the optimization problem in the infinite dimensional space, the problem can be solved in its original form without any mathematical transformations. Hence, the solution is guaranteed to be a rigorous solution to the original problem. Despite this benefit, the variational approach has several major drawbacks. Inherently, the Euler-Lagrange equations are difficult to solve numerically because they amount to solving two point boundary value problems. Complicating this issue is the addition of Lagrangian inequality constraints on the state and control variables, an omnipresent artifact of practical optimal control problems. Many efforts

have been made to address the solution of variational problems subject to bounded states beginning with the work of Valentine [92]. Among many other authors, determining necessary conditions for variational problems with inequality constrained state variables has been addressed by Dreyfus [27], Berkovitz [16], Chang [22], Speyer and Bryson [82], and Jacobson and Lele [50, 51]. Despite significant advances in identifying necessary and sufficient conditions for bounded problems, control constraints cause almost as many numerical difficulties as unbounded problems. Additionally, as noted, most of the work concerning variational problems with inequality constrained state variables has been restricted to determining necessary conditions for yielding stationary functions. However, any global optimization scheme requires that optimality conditions be both necessary and sufficient because the set of solutions satisfying only necessary conditions is actually a superset of the set containing all minimizing functions. In fact, not only does satisfying a necessary condition not yield the global minimum, the satisfaction of this condition does not guarantee even that a local minimum has been found. Unfortunately, the necessary and sufficient conditions are known only to match identically for the special cases of unconstrained and control constrained convex problems.

In addition to the variational approach for solving dynamic optimization problems, another approach exists based on discretization. While discretization has the disadvantage that it is only an approximation of the infinite dimensional problem, it possesses the tremendous advantage that it transforms the original infinite dimensional problem into a problem lying at least partially in a finite space; therefore, the problem can often be solved by standard nonlinear programming methods. Discretization can be subdivided into two broad classifications known as simultaneous and sequential. The simultaneous method is a complete discretization of both state and control variables, often achieved via collocation [90, 69]. While completely transforming a dynamic system into a system of algebraic equations eliminates the problem of optimizing in an infinite dimensional space, simultaneous discretization has the unfortunate side effect of generating a multitude of additional variables yielding large, unwieldy nonlinear programs (NLPs) that are often impractical to solve numerically.

Sequential discretization is usually achieved via control parameterization [18, 85], in which the control variable profiles are approximated by a series of basis functions in terms of a finite set of real parameters. These parameters then become the decision variables in a dynamic embedded NLP. Function evaluations are provided to this NLP via numerical solution of a fully determined initial value problem (IVP), which is given by fixing the control profiles. This method has the advantages of yielding a relatively small NLP and exploiting the robustness and efficiency of modern IVP and sensitivity solvers [60, 32].

For a number of years, researchers have known that dynamic optimization problems encountered in chemical engineering applications exhibit multiple local minima almost pathologically [12, 59, 57]. This property, which can be attributed to nonconvexity of the functions participating in most chemical engineering models, implies that standard local optimization methods will often yield suboptimal solutions to problems. Suboptimality can have direct economic, safety, and environmental impacts if a suboptimal operating policy is implemented on a real process. The classical approach to nonconvex variational problems is to reformulate them so that they are convex and then apply the relevant necessary and sufficient Euler-Lagrange conditions [89]. However, only a very few small problems are accessible to such analysis. This deficiency has motivated researchers to develop global optimization algorithms for nonconvex dynamic optimization problems. Of particular interest are deterministic global optimization algorithms, or those methods that rigorously prove finite convergence of the algorithm to global optimality. The usage of these algorithms therefore guarantees the optimal operating policy has been found for the considered formulation.

A great deal of research has been devoted to the application of stochastic optimization algorithms to overcome convergence only to local minima. In chemical engineering, this body of research has been dominated by Banga and coworkers (e.g., [12, 20]) and Luus and coworkers (e.g., [59, 57]). While these methods often perform better than those mentioned above, they are typically very computationally expensive, and they typically have difficulty with highly constrained problems because they tend not to converge to solutions at which multiple constraints are active. Most im-

portantly, however, even the best stochastic search method cannot guarantee locating the global solution in a finite number of iterations.

Galperin and Zheng [39] have developed a method for globally optimizing optimal control problems via a measure theory approach. Their contribution to dynamic optimization is based upon their previous work [38] in which they propose a global algorithm for solving nonlinear observation and identification problems. This method requires determining the Lebesgue measure of level sets of the objective function. While this technique theoretically guarantees determination of a global minimum, implementation for large problems is impractical due to the sampling required for generating level sets.

As previously discussed, one of the most important industrial applications of dynamic simulation and optimization is to model changeover operations in chemical processes. Often, the chemicals produced during changeovers are unsuitable for market. Therefore, a very important dynamic optimization problem consists of determining the control procedure that minimizes the time required to switch from one operating condition to another. Canon *et al.* [19] have examined this problem and have outlined a solution technique for problems obeying a special structure. For final time minimization of a linear time-invariant system with piecewise constant controls subject to endpoint constraints, they have shown that via a time discretization approach, the problem can be reformulated as a sequence of linear programs. Because linear programs are inherently convex, their method achieves a global solution for the reformulated problem.

Many modern, general methods for deterministic global optimization in Euclidean spaces rely on the notion of a convex relaxation for a nonconvex function [65]. As the name implies, a convex relaxation is a convex function that underestimates a nonconvex function on the set of interest. The highest possible convex underestimator is termed the convex envelope of the nonconvex function [31]. Because convex underestimating problems can be solved to guaranteed global optimality with local solution techniques, they are employed to generate rigorous lower bounds on nonconvex problems. For this reason, convex underestimators are ideally suited as the relaxations

required by a deterministic branch-and-bound algorithm for solving nonconvex NLPs [65, 31, 49]. Floudas and co-workers have demonstrated that a convex underestimator can be constructed for any twice continuously differentiable nonconvex function via a shift of the diagonal elements of the Hessian matrix of the nonconvex function [61]. The shift parameter, α, is utilized in conjunction with a branch-and-bound algorithm for the global solution of NLPs; this NLP solution technique has been termed αBB. In later work, Floudas *et al.* demonstrated a method by which rigorous values for α may be computed via interval analysis [5, 4], provided the functional form of the optimization problem is known explicitly as an elementary function and can be manipulated symbolically.

Building upon the αBB framework, Esposito and Floudas [30, 29] describe what is probably the first practical algorithm for deterministic global optimization of dynamic systems. Subject to mild assumptions, they have shown that an NLP with a dynamic embedded system (potentially derived from the control parameterization of a dynamic optimization problem) is twice continuously differentiable; hence, similarly to αBB, a shift parameter for the Hessian can be found to yield a convex underestimator for any nonconvex terms. This shift parameter, denoted β, is used in conjunction with a branch-and-bound algorithm to solve a dynamic embedded NLP; this technique has been termed βBB. As previously discussed, the rigorous calculation of a Hessian shift parameter requires the explicit functional form of the function to be relaxed. However, in a dynamic embedded problem, the functional dependence of the objective function and constraints on the control parameters is rarely known explicitly. Rather, the functional dependence is implied computationally by the numerical integration of the embedded dynamic system. Hence, except in analytically tractable problems, the interval analysis methods for generating rigorous β parameters are not applicable. To circumvent this problem, several heuristic methods have been proposed to compute β. Unfortunately, all of the proposed methods rely either on arbitrary selection of β or on sampling to determine estimates of the true Hessian matrix. Not only is this method of calculating β not rigorous, in fact, these methods cannot even provide guarantees that the derived underestimator is convex over the entire region of interest. Furthermore,

in order to generate a reasonable value for β, the Hessian matrix requires a relatively large number of sampled points. Because the size of a uniform sample space grows exponentially with the number of decision variables, practical implementation of βBB becomes intractable rather quickly.

Papamichail and Adjiman [71] have presented the first truly rigorous deterministic global optimization algorithm for problems with ODEs embedded. As with βBB, a branch-and-bound algorithm is considered. The uniqueness of the approach is in its rigorous construction of a convex relaxation. Rather than attempt to convexify every term in the objective and constraint functionals with state variables participating, their method substitutes a new real valued decision variable for each state at a fixed time. This reformulation relegates any nonconvexity arising from the state variables to equality constraints with both the new variables and the state variables appearing explicitly. This new equality constraint is then relaxed by deriving a convex lower bounding inequality and a concave upper bounding inequality. Any other nonconvexities in the resulting transformed lower bounding problem are also relaxed via standard convexification techniques. However, because a state variable's functional dependence on the optimization parameters is only known indirectly via the solution of a system of differential equations, two techniques based on differential inequalities are presented for rigorously bounding these solutions. The first technique utilizes differential inequalities to provide constant (relative to the embedded parameters) bounds on the states. Analogous to the βBB approach, the second technique utilizes an α shift parameter and an overpowering quadratic term to convexify or concavify a state while retaining a functional dependence on the embedded parameter. The α parameter is determined rigorously by a combination of interval arithmetic techniques and the simultaneous solution of second order sensitivity equations. This method is computationally expensive, particularly if the number of optimization parameters and constraints grows large, because the computation of second order sensitivities grows quadratically in the number of optimization parameters. Moreover, the method is applicable only to objective functionals and constraints involving the state variables at fixed time points.

In recent years, there has been growing interest in problems in process design and operations that can be formulated as dynamic optimization problems coupled with integer decisions; these problems are known as mixed-integer dynamic optimization (MIDO) problems [7, 8]. Binary or integer decision variables are often introduced in an optimization formulation to represent the inclusion or exclusion of elements in design, discontinuities in the model, and temporal sequencing decisions. If the underlying model is dynamic, such a formulation yields a MIDO problem. Examples thus far reported in the literature include the synthesis of integrated batch processes [7, 8], the interaction of design and control and operations [66, 67, 80], kinetic model reduction [73, 9], and the design of batch distillation columns [81]. Similarly, problems in safety verification have been formulated as a mixed-integer optimization problem with a linear, discrete time model [26]; use of a more realistic nonlinear, continuous time model would directly yield a MIDO formulation. One should note that the use of a deterministic global optimization scheme is absolutely vital here, for formal safety verification requires a rigorous proof that the formulated safety property is satisfied by the model.

In addition to the recent interest in MIDO problems, there has also been growing interest in the optimization of dynamic systems containing discontinuities (events); these problems are termed hybrid discrete/continuous systems. As an example, these formulations can be used to design major process transients such as start-up and shut-down procedures [36, 14]. The most interesting class of problems is that in which the sequence of discontinuities along the optimal trajectory can vary with parameters in the decision space, and the optimal sequence of events is sought; such optimization problems have been shown to be nonsmooth [36, 37]. This suggests a MIDO formulation for hybrid discrete/continuous dynamic optimization in which binary variables are introduced to represent the different sequences of events that can occur [14]. Recently, it has been proposed to address the optimal design of campaign continuous processes via a combination of MIDO and hybrid dynamic optimization formulations [13].

Standard algorithms for general mixed-integer nonlinear optimization in Euclidean

spaces assume that the participating functions are convex [42, 28, 34]. It is well known that if such an algorithm is applied to problems in which the participating functions are nonconvex, the algorithm converges to arbitrary suboptimal solutions [55, 78, 11]. Convergence to such arbitrary suboptimal solutions is far worse than convergence to a local minimum because the algorithms add constraints during the solution process that arbitrarily exclude portions of the search space. These observations extend identically to MIDO problems [7, 8]; however, this has not stopped several authors from applying these unsuitable algorithms to MIDO problems indiscriminately [66, 67, 80, 81], although one wonders about the utility of the answers found. Deterministic global optimization is really the only valid method to solve both mixed-integer optimization and MIDO problems. In recent years, a number of general deterministic global optimization algorithms have emerged for the solution of mixed-integer nonlinear programs and integer nonlinear programs in which the participating functions are nonconvex [78, 3, 53, 54, 52]. In particular, the algorithms developed in [52] are easily extensible to MIDO problems provided that the gradients of the objective function and constraints can be computed and provided that the global dynamic optimization subproblem can be solved for fixed integer realization.

1.2 Structural Outline of the Thesis

As previously stated, the objective of this thesis is to develop efficient algorithms for solving nonconvex dynamic optimization problems to guaranteed global optimality. The thesis begins by stating the general mathematical formulation of the problem and introducing the basic notation used throughout the thesis. As part of the mathematical formulation, a solution to the problem is shown to exist, and the general outline of the solution strategy is introduced. The next chapter introduces the features common to solving problems with embedded linear dynamics and problems with embedded nonlinear dynamics. In particular, a general convexity result for an integral objective function and the affine nature of the solution to a system of linear differential equations are introduced. The subsequent chapters address problems with

linear dynamics and problems with nonlinear dynamics separately. For each problem type, one chapter is devoted to the problem specific theory and another chapter is devoted to the implementation of the theory. For linear problems, the theory includes the composition result of affine functions with convex functions and the exact state bounds for linear dynamic systems. In the chapter concerning the implementation of the linear theory, special considerations are given to exploiting the unique properties of numerically solving linear problems, and several small example problems are considered. Building upon the concepts developed in the linear theory, the theory for solving problems with embedded nonlinear dynamics is presented. In particular, the nonlinear theory focuses on bounding nonquasimonotone systems, constructing linear convex and concave relaxations for the solution of a system of nonlinear ODEs, and McCormick's composition theorem as a technique for relaxing the integrand. The chapter concerning the implementation of the nonlinear theory primarily emphasizes the automatic generation of convex relaxations for dynamic problems. Additionally, consideration is given to some of the unique requirements of integrating the automatically generated relaxations. Following the implementation chapter, several small example problems are considered to demonstrate the utility of the implementation. The next chapter in the thesis addresses the application of the theory and implementation to a practical chemical engineering case study. Next, the preliminary results obtained concerning variational problems are briefly examined. The thesis is concluded with a discussion of the outstanding issues related to the efficient solution of global dynamic optimization problems and potential future directions for research.

Chapter 2

Problem Statement and Solution Strategy

In this chapter, the general mathematical form of the problem addressed in this thesis is stated, and the existence of a minimum is proven. Additionally, the general solution strategy for solving the central problem of the thesis is introduced. Throughout the thesis, bold, lowercase letters are utilized for vectors, and bold uppercase letters are utilized for matrices. The symbol p is used to represent parameters, x is used to represent states, and t is used to represent the independent variable often referred to as time. P represents a nonempty, compact, convex subset of the Euclidean space $\mathbb{R}^{n_p}$ that contains the parameter, and X is used to represent some subset of the Euclidean space $\mathbb{R}^{n_x}$ such that $\mathbf{x}(t,\mathbf{p}) \in X \ \forall \ (t,\mathbf{p}) \in (t_0,t_f] \times P$. The symbol $\mathcal{X}$ is restricted for a time dependent superset (possibly with equality) of X that represents the state bounds, usually as computed via the solution of a bounding system of differential equations. Additionally, lowercase letters are used to represent integrands, and their corresponding capital letters represent the integrals associated with those integrands. An underlined variable in any equation means that the variable is fixed while all other variables in the equation are allowed to vary.

2.1 Problem Statement and Existence of a Minimum

The objective of this thesis is to compute a global minimum of Problem 2.4 defined below. To ensure a clear understanding of the difference between a local minimum and a global minimum, the following definition is introduced:

Definition 2.1. Let f be a function $f : P \to \mathbb{R}$. $f(\mathbf{p}^*)$ is a local minimum value of f at $\mathbf{p}^* \in P$ if for $\varepsilon > 0$, there exists a neighborhood $N_\varepsilon(\mathbf{p}^*)$ such that

$$f(\mathbf{p}^*) \leq f(\mathbf{p}) \tag{2.1}$$

for all $\mathbf{p} \in N_\varepsilon(\mathbf{p}^*) \cap P$. $f(\mathbf{p}^*)$ is a global minimum value of f at $\mathbf{p}^* \in P$ if Equation 2.1 holds for all $\mathbf{p} \in P$.

I now define a special classification of discontinuities that are permissible in the general problem formulation.

Definition 2.2. Let $Z \subset \mathbb{R}^d$ and $f : [t_0, t_f] \times Z \to \mathbb{R}$ is an integrable function. f is said to have a finite number of stationary simple discontinuities in t if the following conditions hold:

1. For each $\underline{t}$ fixed in $[t_0, t_f]$, $f(\underline{t}, \mathbf{z})$ is continuous on Z.

2. For each $\underline{\mathbf{z}}$ fixed in Z, $f(t, \underline{\mathbf{z}})$ possesses at most a finite number of simple discontinuities. Additionally, $f(t, \underline{\mathbf{z}})$ must be defined at any point of discontinuity.

Remark 2.3. I note that the above definition immediately implies the following: For all $\underline{\mathbf{z}}$ fixed in Z, discontinuities may only occur on the fixed partition $P : t_0 < t_1 < \ldots < t_n < t_{n+1} = t_f$ (i.e., the partition may not vary with $\mathbf{z}$). This type of discontinuity typically arises from the control parameterization of optimal control problems with fixed intervals of support for the basis functions.

The general problem is now stated as follows:

Problem 2.4.

$$\min_{\mathbf{p}} J(\mathbf{p}) = \phi(\mathbf{x}(t_f, \mathbf{p}), \mathbf{p}) + \int_{t_0}^{t_f} \ell(t, \mathbf{x}(t, \mathbf{p}), \mathbf{p})\ dt$$

subject to

$$g_i(\mathbf{x}(t_f, \mathbf{p}), \mathbf{p}) + \int_{t_0}^{t_f} h_i(t, \mathbf{x}(t, \mathbf{p}), \mathbf{p})\ dt \leq \mathbf{0}, \quad i = 1, ..., nc$$

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{p})$$
$$\mathbf{x}(t_0, \mathbf{p}) = \mathbf{x}_0(\mathbf{p})$$

where $\mathbf{p} \in P$; ϕ is a continuous mapping $\phi : X \times P \to \mathbb{R}$; ℓ is a Lebesgue integrable mapping $\ell : (t_0, t_f] \times X \times P \to \mathbb{R}$; f_i is a Lipschitz continuous mapping $f_i : (t_0, t_f] \times X \times P \to \mathbb{R}$, $i = 1, \ldots, n_x$; $\mathbf{x}_0$ is a continuous mapping $\mathbf{x} : P \to \mathbb{R}^{n_x}$, $i = 1, \ldots, n_x$, $\mathbf{g}$ is a continuous mapping $g_i : X \times P \to \mathbb{R}$; and $\mathbf{h}$ is a Lebesgue integrable mapping $h_i : (t_0, t_f] \times X \times P \to \mathbb{R}$. nc is the number of point and/or isoperimetric constraints. Furthermore, the functions $\ell, \mathbf{h}$ are only permitted a finite number of stationary simple discontinuities in t. Additionally, $\mathbf{f}$ may be relaxed to contain a finite number of stationary simple discontinuities provided Lipschitz continuity is preserved between the points of discontinuity.

Remark 2.5. This problem formulation represents any problem possessing parameter dependent ODEs. This formulation arises in solving, for example, parameter estimation problems, control parameterizations of optimal control problems, and continuous relaxations of mixed-integer dynamic optimization problems. Trivially, the problem formulation may be extended to any finite number of point and integral terms on a fixed partition. Each term may be relaxed individually, and the sum of these relaxations is a relaxation for the sum of the nonconvex terms. Additionally, the problem formulation may be extended to handle varying time problems by fixing the final time and scaling the differential equations by an optimization variable. Similarly, optimal control problems with time varying intervals may be reformulated by scaling in each control interval.

Before proving the existence of a minimum to Problem 2.4, several additional comments should be made. First, the integrands in Problem 2.4 are permitted to be Lebesgue integral. Typically, only Riemann integrable functions are studied in this thesis; however, this restriction is not necessary for the development of the theory and is thus not asserted. Second, the problem formulation only explicitly considers inequality constraints. Of course, equality constraints are immediately handled within this formulation, for any equality can be written as a set of two inequalities. Finally, in addressing Problem 2.4, I speak of a solution to the differential equations defined in Problem 2.4 in the sense of Carathéodory [24]. That is, a solution is considered admissible if the differential equations are satisfied almost everywhere.

The existence of a minimum of Problem 2.4 is now proven in two steps. In the first step, the objective function for the problem is shown to be continuous, and in the second step, Weierstrass's Theorem is asserted to prove the existence of a minimum.

Proposition 2.6. *Consider the integral $L(\mathbf{p})$ in the objective function defined in Problem 2.4. Then, $L(\mathbf{p})$ is continuous on P.*

Proof. By hypothesis, the discontinuities allowed in ℓ and $\mathbf{f}$ are restricted to a finite number of stationary simple discontinuities in t. Let there be n such discontinuities each found at points $t = t_i$ and write $L(\mathbf{p})$ as follows:

$$L(\mathbf{p}) = \int_{t_0}^{t_1} \ell(t, \mathbf{x}(t,\mathbf{p}), \mathbf{p})\, dt + \cdots + \int_{t_{i-1}}^{t_i} \ell(t, \mathbf{x}(t,\mathbf{p}), \mathbf{p})\, dt + \cdots + \int_{t_n}^{t_{n+1}=t_f} \ell(t, \mathbf{x}(t,\mathbf{p}), \mathbf{p})\, dt. \quad (2.2)$$

Choose any integral $\int_{t_{i-1}}^{t_i} \ell(t, \mathbf{x}(t,\mathbf{p}), \mathbf{p})\, dt$ from the right hand side of Equation 2.2. By construction, ℓ is at most discontinuous at its endpoints (in time). Extend ℓ to be continuous over its respective domain and consider the equivalent integral with the extended function. The equivalence is established because the integrands are equal almost everywhere. Because $\mathbf{x}$ is continuous on the compact set $[t_{i-1}, t_i] \times P$, the set X (assuming construction as a mapping of the solution of the differential equations on P) is also compact. Hence, ℓ is continuous on the compact set $[t_{i-1}, t_i] \times X \times P$, and by

trivial extension of Theorem 26.1 [64], the integral is continuous over P. The integral was chosen arbitrarily, implying any integral from Equation 2.2 is continuous. Hence, $L(\mathbf{p})$ is comprised of the sum of continuous functions and is therefore continuous.

□

Corollary 2.7. *Assume that the feasible space of Problem 2.4 is nonempty. Then, there exists a minimum to Problem 2.4.*

Proof. Because ϕ is continuous by hypothesis, by the above proposition, $J = \phi + L$ as defined in Problem 2.4 is a continuous mapping from a nonempty compact set into $\mathbb{R}$. Therefore, by Theorem 4.16 [77], the existence of a minimum to Problem 2.4 is guaranteed.

□

2.2 Solution Strategy

Having demonstrated the existence of a minimum of Problem 2.4, the remainder of this thesis is devoted to deriving efficient algorithms and implementations for computing this minimum. As defined, the degrees of freedom for Problem 2.4 lie completely in a Euclidean space; the dynamic variables appearing in the problem can be treated simply as intermediate variables appearing in compositions. At this level of abstraction, the optimization problem can be solved in the original Euclidean space via standard techniques provided that both function evaluations and gradients are available from the dynamic system. Indeed, this is the general solution strategy applied to solving Problem 2.4, and the standard Euclidean optimization technique employed is the branch-and-bound algorithm.

Branch-and-bound algorithms for global optimization of nonlinear programming problems first appeared in the literature in the late 1960s with the work of Falk and Soland [31] and mid 1970s with the work of McCormick [65]. The objective of a branch-and-bound algorithm is to generate a sequence of rigorous upper bounds for the optimization problem and a sequence of rigorous lower bounds for the optimization

problem such that the two sequences eventually converge to a global minimum for the problem. By the term "rigorous bounds," I mean that the upper bound is theoretically guaranteed to exceed the global minimum of the problem at hand and the lower bound is theoretically guaranteed to be less than the global minimum. In general, any feasible point for the problem may be considered to yield an upper bound, for by its definition, the global minimum value of a problem is less than or equal to the objective function value of every other feasible point. Typically, however, local solution of the upper bounding problem is relatively inexpensive. Therefore, a local solution of the upper bounding problem is taken as the upper bound. Computing a rigorous lower bound is a much more difficult task and is, in fact, the primary focus of this thesis. In solving Problem 2.4, the lower bound is always taken as the minimum value of a convex relaxation. An underestimating relaxation of a function is any function that is pointwise less than the original function, while an overestimating relaxation of a function is any function that is pointwise greater than the original function. Often, the term relaxation is used without a modifier. Whether the function is underestimating or overestimating is then implied by context. As a general rule, convex relaxations are always underestimators, and concave relaxations are always overestimators. Convex functions, which I explore in much greater detail in subsequent chapters, have the special property that a local solution is guaranteed to be a global solution to the convex problem. Therefore, by minimizing a convex relaxation, a rigorous lower bound is obtained by construction. After obtaining both upper and lower bounds, the two bounds are compared. If the lower bound is within some finite, positive tolerance of the upper bound, then the algorithm terminates, and the upper bound is the estimate for the global minimum value (within the tolerance).

Unfortunately, termination at the root node rarely occurs, and branching is required to generate a sequence of converging upper and lower bounds. Branching is an exhaustive search procedure in which the parameter space is partitioned (usually a split in one coordinate direction of a hyperrectangle); the bounding procedure is then repeated in each new child node of the branch-and-bound tree. By construction, the convex relaxations become tighter as the search space is partitioned and approach the

objective function pointwise as the search space approaches degeneracy. That is, as the parameter space is branched, the solution to the convex relaxation in each child node increases relative to the solution of the convex relaxation in the parent node. After an infinite amount of branching, the search space reduces to a degenerate point, and the value of the objective function and the convex relaxation are equal at this degenerate point. During successive bounding steps, if the solution to a node's convex relaxation is greater than the current best upper bound (the incumbent solution) then this node is removed from the search space, for clearly this node cannot contain a global minimum of the problem. This process of deleting regions of the search space is referred to as fathoming. In this manner, successively branching and bounding the space leads to a sequence of upper bounds and a sequence of lower bounds that converge to each other (and hence the global minimum value) after infinite branching and bounding steps. Obviously, a practical algorithm must converge in a finite amount of time. Fortunately, infinite convergence of a branch-and-bound algorithm immediately implies that the algorithm converges to the global minimum within an ε tolerance in finite time. I return to the topic of branch-and-bound convergence after the construction of the convex relaxations.

Chapter 3

Relaxation of an Integral and the Affine Solution of a Linear System

In this chapter, I examine two topics that are precursors to developing convex relaxations for Problem 2.4. Although seemingly disjoint topics, relaxing an integral and the affine solution of a linear system are presented together because they are common prerequisites to the relaxation approaches for both problems with embedded linear dynamic systems and problems with embedded nonlinear dynamic systems.

3.1 Convex Relaxations for an Integral

Until now, the term convexity has been used without rigorous definition; this intentional oversight is now corrected. The definitions for convex set and convex function are introduced below. Both concepts have geometric interpretations; thus, pictures are included for each topic to assist in visualizing the definitions.

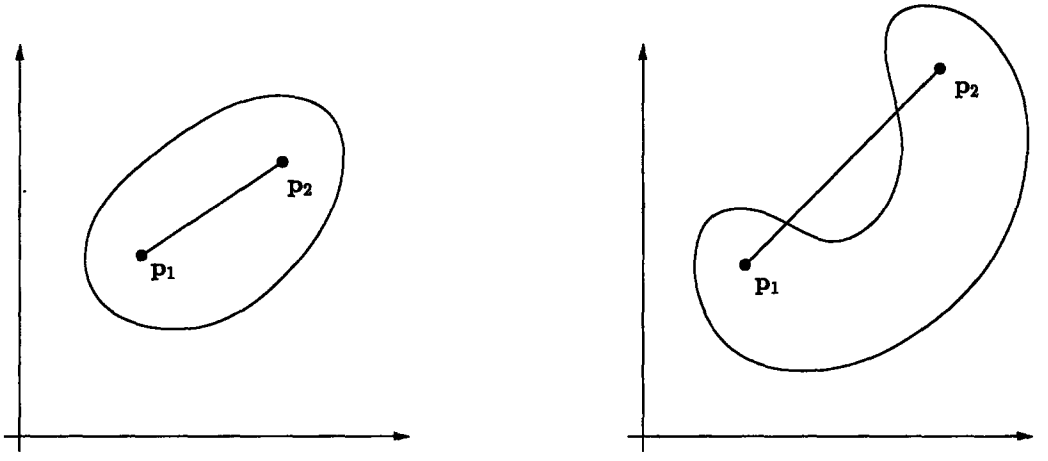
Definition 3.1 (Convex Set). Let P be a subset of $\mathbb{R}^{n_p}$. P is convex if

$$\lambda \mathbf{p}_1 + (1-\lambda)\mathbf{p}_2 \ \in P, \qquad \forall\, \mathbf{p}_1, \mathbf{p}_2 \in P,\ \forall\, \lambda \in [0,1].$$

The above definition for a convex set is typically referred to as the geometric definition of convexity, for the definition implies that if a line segment joining any

two points of a set also lies wholly within the set, then the set is convex. If any part of this line segment lies outside the set, then the set is nonconvex. Figure 3-1 below illustrates this concept.

Figure 3-1: Geometric notion of a convex set



Convex Set Nonconvex Set

Having defined a convex set, a convex function is now defined. The geometric definition of convexity (for functions) is employed thereby separating the definition of a convex function from a function's differentiability.

Definition 3.2 (Convex Function). Let $f : P \rightarrow \mathbb{R}$, where P is a nonempty, convex subset of $\mathbb{R}^{n_p}$. The function f is said to be convex on P if

$$f(\lambda \mathbf{p}_0 + (1-\lambda)\mathbf{p}_1) \leq \lambda f(\mathbf{p}_0) + (1-\lambda) f(\mathbf{p}_1)$$

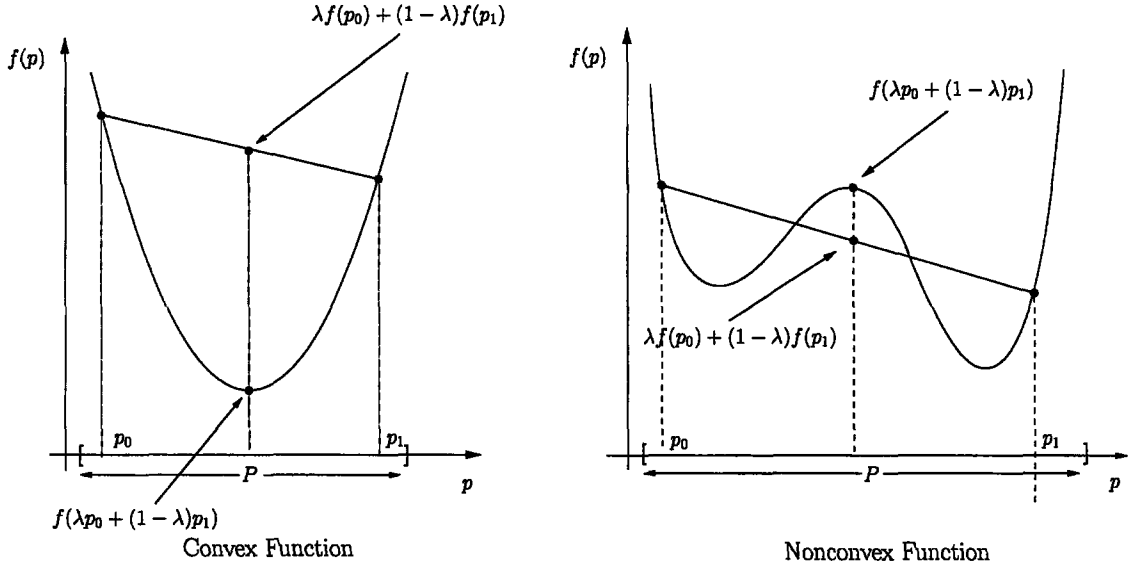
for each $\mathbf{p}_0$, $\mathbf{p}_1 \in P$ and for each $\lambda \in (0,1)$.

Remark 3.3. A function $f(\mathbf{p})$ is concave if $-f(\mathbf{p})$ is convex.

The geometric interpretation of Definition 3.2 is that if any line segment joining two points on the function lies on or above the function for all points in between, then the function is convex. If any portion of this line segment lies below the function, then the function is nonconvex. Figure 3-2 below illustrates the geometric definition

of a convex function. Convex functions are of tantamount importance to developing global optimization routines because any minimum of a convex function is global. Computationally, this implies that a rigorous lower bound for a function may be obtained simply by minimizing locally a convex relaxation of the function. This result concerning the minimization of convex functions is a classical result of optimization theory and is not proven here.

Figure 3-2: Geometric notion of a convex function



The primary objective of this section is to provide a technique for constructing a convex relaxation of an integral. By the monotonicity property of the integral, if integrand $u(t, \mathbf{p})$ underestimates integrand $\ell(t, \mathbf{p})$, then the function $U(\mathbf{p})$ underestimates the function $L(\mathbf{p})$. Furthermore, Definition 3.2 implies that a partially convex integrand implies a convex integral. The term partially convex refers to any function which is convex for each fixed $\underline{t} \in (t_0, t_f]$. Combining integral monotonicity and the definition of convexity, I am then able to prove that if a partially convex function $u(t, \mathbf{p})$ underestimates $\ell(t, \mathbf{p})$, then $U(\mathbf{p})$ is a convex relaxation for $L(\mathbf{p})$. The analysis begins by proving the monotonicity of a parameter dependent integral.

Lemma 3.4 (Integral Monotonicity). *Let* $\mathbf{p} \in P$, $t \in (t_0, t_f]$, *and* $\ell_1, \ell_2 : (t_0, t_f] \times$

$P \to \mathbb{R}$ *such that* ℓ_1, ℓ_2 *are integrable. If*

$$\ell_1(\underline{t}, \mathbf{p}) \leq \ell_2(\underline{t}, \mathbf{p}) \qquad \forall\ (\underline{t}, \mathbf{p}) \in (t_0, t_f] \times P$$

then

$$L_1(\mathbf{p}) = \int_{t_0}^{t_f} \ell_1(t, \mathbf{p})\ dt \leq \int_{t_0}^{t_f} \ell_2(t, \mathbf{p})\ dt = L_2(\mathbf{p}) \qquad \forall\ \mathbf{p} \in P.$$

Proof. Theorem 6.12b in [77] holds $\forall\ \mathbf{p} \in P$.

□

I now prove that a partially convex integrand implies a partially convex integral.

Theorem 3.5 (Integral Convexity). *Let* $\mathbf{p} \in P$, $t \in (t_0, t_f]$, *and* $\ell : (t_0, t_f] \times P \to \mathbb{R}$ *be Lebesgue integrable. If* $\ell(t, \mathbf{p})$ *is convex on* P *for each fixed* $\underline{t} \in (t_0, t_f]$, *then*

$$L(\mathbf{p}) = \int_{t_0}^{t_f} \ell(t, \mathbf{p})\ dt$$

is convex on P.

Proof. By hypothesis, $\ell(t, \mathbf{p})$ is convex on $P\ \forall\ \underline{t} \in (t_0, t_f]$. Therefore, given $\lambda \in (0, 1)$, I have

$$\ell(\underline{t}, \lambda\mathbf{p} + (1-\lambda)\mathbf{p}_0) \leq \lambda\ell(\underline{t}, \mathbf{p}) + (1-\lambda)\ell(\underline{t}, \mathbf{p}_0) \qquad \forall\ (\underline{t}, \mathbf{p}), (\underline{t}, \mathbf{p}_0) \in (t_0, t_f] \times P.$$

By Lemma 3.4, the above equation implies

$$\int_{t_0}^{t_f} \ell(t, \lambda\mathbf{p} + (1-\lambda)\mathbf{p}_0)\ dt \leq \int_{t_0}^{t_f} \lambda\ell(t, \mathbf{p})\ dt + \int_{t_0}^{t_f} (1-\lambda)\ell(t, \mathbf{p}_0)\ dt \qquad \forall\ \mathbf{p}, \mathbf{p}_0 \in P,$$

which by inspection equals

$$L(\lambda\mathbf{p} + (1-\lambda)\mathbf{p}_0) \leq \lambda L(\mathbf{p}) + (1-\lambda)L(\mathbf{p}_0) \qquad \forall\ \mathbf{p}, \mathbf{p}_0 \in P.$$

By Definition 3.2, $L(\mathbf{p})$ is convex on P.

□

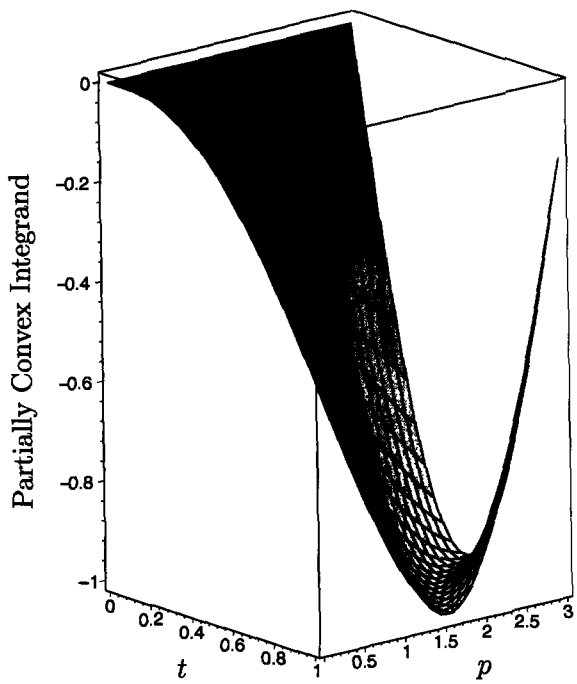
An interesting implication of requiring only partial convexity of the integrand in Theorem 3.5 is that even if an integrand is nonconvex on $(t_0, t_f] \times P$, its integral is still convex. The following example illustrates this principle.

Example 3.6. Consider $p \in [0, 3]$ and the following integral:

$$J(p) = \int_0^1 -\sin(pt)\ dt.$$

The integrand is nonconvex on $[0, 1] \times [0, 3]$; however, the integrand is partially convex on [0,3] for each fixed $\underline{t} \in [0, 1]$. By Theorem 3.5, $J(p)$ is convex on [0,3]. Figure 3-3 below depicts the partially convex integrand, while Figure 3-4 illustrates the convex integral.

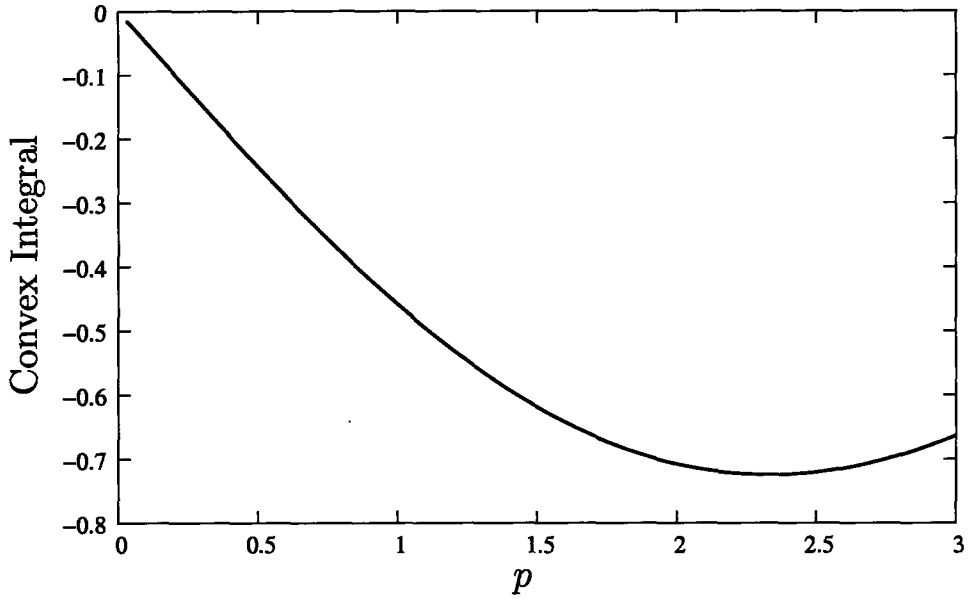
Figure 3-3: Partially convex integrand from Example 3.6



Corollary 3.7 (Integral Relaxation). *Let* $\mathbf{p} \in P$, $t \in (t_0, t_f]$, *and* $u, \ell : (t_0, t_f] \times P \to \mathbb{R}$ *be Lebesgue integrable. If* $u(t, \mathbf{p})$ *is convex on* $P\ \forall\ \underline{t} \in (t_0, t_f]$ *and*

$$u(\underline{t}, \mathbf{p}) \leq \ell(\underline{t}, \mathbf{p}) \qquad \forall\ (\underline{t}, \mathbf{p}) \in (t_0, t_f] \times P,$$

Figure 3-4: Convex integral from Example 3.6



then $U(\mathbf{p})$ *is convex on* P *and* $U(\mathbf{p}) \leq L(\mathbf{p})$, *where*

$$U(\mathbf{p}) = \int_{t_0}^{t_f} u(t, \mathbf{p})\, dt.$$

Proof. The proof is immediately evident from Lemma 3.4 and Theorem 3.5. □

Remark 3.8. An analogous result applies for concave overestimators.

Corollary 3.7 enables one to calculate a convex underestimator for an integral by integrating a relaxation of the integrand, provided this relaxation is convex on P for each fixed $\underline{t} \in (t_0, t_f]$. Because a continuous set contains an uncountable number of points, computing an underestimator at each fixed $\underline{t} \in (t_0, t_f]$ is clearly impossible numerically. Obviously, in order to solve problems practically, numerical integration methods are required; fixed time quantities are simply computed at each integration time step.

As presented, Theorem 3.5, and consequently Corollary 3.7, apply to integrands for which partial convexity is known on the parameter space P. If the functional dependence on $\mathbf{p}$ for an integrand were explicit, computing a convex relaxation for

this integrand would follow directly from standard techniques. However, Problem 2.4 contains an integrand that may have implicit dependence on parameters via composition with the state variables. Therefore, in order to apply Corollary 3.7 to Problem 2.4, a method for determining convexity of composite functions must first be introduced. This topic is addressed separately for problems with embedded linear dynamic systems in Chapter 4 and for problems with embedded nonlinear dynamic systems in Chapter 6.

3.2 The Affine Solution of Linear Systems

The second fundamental theoretical building block of this thesis is the affine solution of a system of linear time varying ordinary differential equations. The general form of a linear dynamic system is given by the following:

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{p} + \mathbf{r}(t) \tag{3.1a}$$

$$\mathbf{E}_0\mathbf{x}(t_0) + \mathbf{E}_f\mathbf{x}(t_f) = \mathbf{B}_{IC}\mathbf{p} + \mathbf{r}_{IC} \tag{3.1b}$$

where $\mathbf{A}(t)$ is continuous on $(t_0, t_f]$, and $\mathbf{B}(t)$ and $\mathbf{r}(t)$ are piecewise continuous on $(t_0, t_f]$. While this problem formulation allows for a general boundary condition, a more familiar initial value formulation is achieved if $\mathbf{E_0} = \mathbf{I}$ and $\mathbf{E_f} = \mathbf{0}$, where $\mathbf{I}$ is the identity matrix and $\mathbf{0}$ is the zero matrix. Unless otherwise noted, I use the term linear dynamic system to mean either the boundary condition formulation or the initial condition formulation; the distinction is obvious from the context.

According to Theorem 6.1 in [10], a solution to the embedded linear boundary value problem (for continuous $\mathbf{B}$) exists and is unique provided the matrix $\mathbf{Q} = \mathbf{E_0} + \mathbf{E_f}\mathbf{\Phi}(t_f, t_0)$ is nonsingular, where $\mathbf{\Phi}(t_f, t_0)$ is the transition matrix, which is defined by the solution of the following differential equation:

$$\frac{d\mathbf{\Phi}(t, t_0)}{dt} = \mathbf{A}(t)\mathbf{\Phi}(t, t_0)$$

with $\Phi(t_0, t_0) = \mathbf{I}$. Because the differential equations can always be reformulated as a finite collection of differentiable linear systems, the aforementioned statement implies that the solution in the sense of Carathéodory also exists provided $\mathbf{x}$ is defined at the corners. I am interested only in those problems for which a solution to the embedded dynamic system exists and is unique; therefore, $\mathbf{Q}$ is assumed to be nonsingular throughout. Having established existence and uniqueness of $\mathbf{x}(t, \mathbf{p})$, I now address the functional form of the solution. From linear systems theory (see [94] for an introduction), the following provides a general solution to Equation 3.1:

$$\mathbf{x}(t, \mathbf{p}) = \Phi(t, t_0)\mathbf{x}(t_0, \mathbf{p}) + \int_{t_0}^{t} \Phi(t, \tau)\mathbf{B}(\tau)\mathbf{p}\ d\tau + \int_{t_0}^{t} \Phi(t, \tau)\mathbf{r}(\tau)\ d\tau. \tag{3.2}$$

Solving for the boundary condition and substituting the initial condition into Equation 3.2 results in

$$\mathbf{x}(t, \mathbf{p}) = \left\{ \Phi(t, t_0)(\mathbf{E_0} + \mathbf{E_f}\Phi(t_f, t_0))^{-1} \left[\mathbf{C} - \mathbf{E_f} \int_{t_0}^{t_f} \Phi(t_f, \tau)\mathbf{B}(\tau)\ d\tau \right] + \int_{t_0}^{t} \Phi(t, \tau)\mathbf{B}(\tau)\ d\tau \right\} \mathbf{p} + \Phi(t, t_0)(\mathbf{E_0} + \mathbf{E_f}\Phi(t_f, t_0))^{-1} \left[\mathbf{d} - \mathbf{E_f} \int_{t_0}^{t_f} \Phi(t_f, \tau)\mathbf{r}(\tau)\ d\tau \right] + \int_{t_0}^{t} \Phi(t, \tau)\mathbf{r}(\tau)\ d\tau.$$

Although the above equation defining the solution to Equation 3.1 is unwieldy at best, the equation possesses the following structural form:

$$\mathbf{x}(t, \mathbf{p}) = \mathbf{M}(t)\mathbf{p} + \mathbf{n}(t), \tag{3.3}$$

where $\mathbf{M}$ and $\mathbf{n}$ reduce to real valued matrices for fixed time. The importance of the above result is that for each fixed $\underline{t} \in (t_0, t_f]$, Equation 3.3 is affine in the parameter $\mathbf{p}$. Trivially from Definition 3.2, an affine equation is both convex and concave. Equation 3.3 represents one of the few general statements that can be made concerning convexity or concavity for the solution of a differential equation based upon the structure of the differential equation itself. For optimization problems with linear dynamic

systems embedded, the affine nature of the solution of a linear dynamic system leads directly to a composition result for convexifying an integrand. Additionally, Equation 3.3 permits the relatively efficient numerical computation of exact state bounds for a system of parameter dependent linear differential equations. For optimization problems with nonlinear dynamic systems embedded, Equation 3.3 leads directly to a technique where the solution of a system of nonlinear ODEs is relaxed by the solution of a system of bounding linear ODEs. These relaxations are then directly utilized in a composition technique to convexify an integrand. The utility of Equation 3.3 is described in much greater detail in the subsequent chapters pertaining to the solution of Problem 2.4 subject to both linear and nonlinear embedded dynamic systems.

Chapter 4

Relaxation Theory for Problems with Linear Dynamics Embedded

In this chapter, I consider the solution of Problem 2.4 subject to a parameter dependent linear differential equation. The restricted linear problem is stated below.

Problem 4.1.

$$\min_{\mathbf{p}} J(\mathbf{p}) = \phi(\mathbf{x}(t_f,\mathbf{p}),\dot{\mathbf{x}}(t_f,\mathbf{p}),\mathbf{p}) + \int_{t_0}^{t_f} \ell(t,\mathbf{x}(t,\mathbf{p}),\dot{\mathbf{x}}(t,\mathbf{p}),\mathbf{p})\ dt$$

subject to

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{p} + \mathbf{r}(t)$$
$$\mathbf{E}_0\mathbf{x}(t_0) + \mathbf{E}_f\mathbf{x}(t_f) = \mathbf{B}_{IC}\mathbf{p} + \mathbf{r}_{IC}$$

where $\mathbf{p} \in P$; ϕ is a continuous mapping $\phi : X \times \dot{X} \times P \to \mathbb{R}$; ℓ is a Lebesgue integrable mapping $\ell : (t_0, t_f] \times X \times \dot{X} \times P \to \mathbb{R}$; $\mathbf{A}(t)$ is continuous, and $\mathbf{B}(t)$ and $\mathbf{r}(t)$ are piecewise continuous. The function ℓ is only permitted a finite number of stationary discontinuities in t. The set $\dot{X}$ is defined as an enclosure of the right hand sides of the differential equations.

Remark 4.2. For simplicity, isoperimetric and point constraints have not been included explicitly in the formulation. Trivially, given a relaxation strategy for the objective function, these constraints are immediately handled by the same technique.

Furthermore, the reader should note that Problem 4.1 is not a direct restriction of Problem 2.4, for Problem 4.1 permits the derivative of the state variables directly in the objective function. The function ϕ may be dependent on the state variables at any finite number of fixed time points in addition to the final time.

In order to solve Problem 4.1, three distinct steps must be performed. First, the objective function must be relaxed. The point term in the objective function is relaxed via standard techniques, and the integral term is relaxed via Corollary 3.7. As noted in the previous chapter, in order to apply Corollary 3.7, a composition technique must be applied to the integrand. Thus, relaxing the objective function for Problem 4.1 effectively reduces to providing a composition technique for relaxing an integrand defined implicitly by an embedded linear dynamic system. Second, bounds must be computed for the solution of the linear dynamic system. As will be seen shortly, the necessity for state bounds is a direct consequence of the composition technique required for relaxing the integrand. Finally, I show that the derived relaxation technique leads to a convergent branch-and-bound algorithm for solving Problem 4.1.

4.1 Affine Composition with a Convex Function

From Corollary 3.7, in order to relax an integral, a partially convex relaxation is required for its integrand. From Problem 4.1, the integrand in the objective function is defined as a composition of the state variables and the derivatives of the state variables. By treating the domain of the integrand as a Euclidean subset $(t_0, t_f] \times X \times \dot{X} \times P$, a relaxation for the integrand can be generated via standard techniques. However, as stated, Corollary 3.7 requires partial convexity of the integrand on the set P. From Equation 3.3, the solution to the embedded linear dynamic system is known to be affine. The following proposition, which states that the composition $u[\mathbf{x}(\mathbf{p})]$ is convex on P if $u(\cdot)$ is convex on X and $\mathbf{x}(\mathbf{p})$ is affine, is therefore introduced.

Proposition 4.3. *Let X be a convex subset of $\mathbb{R}^{n_x}$, P be a convex subset of $\mathbb{R}^{n_p}$, $u : X \to \mathbb{R}$ be a function convex on X, and $\mathbf{x} : P \to \mathbb{R}^{n_x}$ be an affine function*

of the form $\mathbf{x}(\mathbf{p}) = \mathbf{M}\mathbf{p} + \mathbf{n}$, *where range*$(\mathbf{x}) \subset X$. *Then, the composite function* $u \circ \mathbf{x} : \; P \rightarrow \mathbb{R}$ *is a convex function on* P.

Proof. Given $\lambda \in (0,1)$, by Definition 3.2, I have

$$u(\lambda \mathbf{x} + (1-\lambda)\mathbf{x}_0) \leq \lambda u(\mathbf{x}) + (1-\lambda)u(\mathbf{x}_0) \quad \forall\, \mathbf{x}, \mathbf{x}_0 \in X.$$

However, by hypothesis, range$(\mathbf{x}) \subset X$. Therefore, I have

$$u[\lambda \mathbf{x}(\mathbf{p}) + (1-\lambda)\mathbf{x}(\mathbf{p}_0)] \leq \lambda u[\mathbf{x}(\mathbf{p})] + (1-\lambda)u[(\mathbf{x}(\mathbf{p}_0)] \quad \forall\, \mathbf{p}, \mathbf{p}_0 \in P,$$

where the change of sets is evident from the definition of the mapping $\mathbf{x}$. I now perform the following algebraic operations:

$$\begin{aligned} u[\lambda \mathbf{x}(\mathbf{p}) + (1-\lambda)\mathbf{x}(\mathbf{p}_0)] &= u[\lambda(\mathbf{M}\mathbf{p} + \mathbf{n}) + (1-\lambda)(\mathbf{M}\mathbf{p}_0 + \mathbf{n})] \\ &= u[\lambda \mathbf{M}\mathbf{p} + \lambda \mathbf{n} + \mathbf{M}\mathbf{p}_0 - \lambda \mathbf{M}\mathbf{p}_0 - \lambda \mathbf{n} + \mathbf{n}] \\ &= u[\mathbf{M}(\lambda \mathbf{p} + (1-\lambda)\mathbf{p}_0) + \mathbf{n}] \\ &= u[\mathbf{x}(\lambda \mathbf{p} + (1-\lambda)\mathbf{p}_0)] \quad \forall\, \mathbf{p}, \mathbf{p}_0 \in P. \end{aligned}$$

Therefore,

$$u[\mathbf{x}(\lambda \mathbf{p} + (1-\lambda)\mathbf{p}_0)] \leq \lambda u[\mathbf{x}(\mathbf{p})] + (1-\lambda)u[\mathbf{x}(\mathbf{p}_0)] \quad \forall\, \mathbf{p}, \mathbf{p}_0 \in P,$$

which by definition shows that $u[\mathbf{x}(\mathbf{p})]$ is convex on P.

□

Remark 4.4. The proposition applies to $\dot{\mathbf{x}}$ as well, for $\dot{\mathbf{x}}$ may be written as

$$\dot{\mathbf{x}} = (\mathbf{A}\mathbf{M} + \mathbf{B})\mathbf{p} + \mathbf{A}\mathbf{n} + \mathbf{r}.$$

In order to utilize Proposition 4.3 to derive a convex relaxation for Problem 4.1, a partially convex underestimator for $\ell(t, \mathbf{x}(t,\mathbf{p}), \dot{\mathbf{x}}(t,\mathbf{p}), p)$ must be derived on the set $\mathcal{X}(\underline{t}) \times \dot{X}(\underline{t}) \times P$ for each fixed $\underline{t} \in (t_0, t_f]$. Given bounds sets $\mathcal{X}(\underline{t})$ and $\dot{\mathcal{X}}(\underline{t})$, deriving

a partially convex relaxation for ℓ follows from standard techniques. However, no method has yet been described for computing the sets $\mathcal{X}(\underline{t})$ and $\dot{\mathcal{X}}(\underline{t})$. This omission is now rectified.

4.2 Computing State Bounds for Linear Dynamic Systems

For linear dynamic systems, interval analysis may be employed to compute the exact bounds for both $\mathbf{x}$ and $\dot{\mathbf{x}}$. The following two propositions formalize the technique. The notation $[\mathbf{z}]$ is shorthand notation used to represent an interval extension of the variable $\mathbf{z}$ where the interval itself is given by $[\mathbf{z}^L, \mathbf{z}^U]$. The reader unfamiliar with interval analysis is referred to the book by Moore [68] as an introduction to interval arithmetic.

Proposition 4.5. *Consider the following set of differential equations and boundary conditions:*

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{p} + \mathbf{r}(t)$$
$$\mathbf{E_0}\mathbf{x}(t_0) + \mathbf{E_f}\mathbf{x}(t_f) = \mathbf{B}_{IC}\mathbf{p} + \mathbf{r}_{IC}.$$

If $\mathbf{p} \in P = [\mathbf{p}^L, \mathbf{p}^U]$, *then the exact state bounds for the solution of the differential equations are given pointwise in time as the natural interval extension (on P) of Equation 3.3:*

$$[\mathbf{x}](\underline{t}, [\mathbf{p}]) = \mathbf{M}(\underline{t})[\mathbf{p}] + \mathbf{n}(\underline{t}). \tag{4.1}$$

Proof. The functional form of the solution of the differential equation is given by Equation 3.3. For any fixed $\underline{t} \in (t_0, t_f]$,

$$\mathbf{x}(\underline{t}, \mathbf{p}) = \mathbf{M}(\underline{t})\mathbf{p} + \mathbf{n}(\underline{t}), \tag{4.2}$$

where $\mathbf{M}(\underline{t})$ and $\mathbf{n}(\underline{t})$ are constants for each fixed $\underline{t}$. The interval extension of Equation 4.2 is taken to yield

$$[\mathbf{x}](\underline{t}, [\mathbf{p}]) = \mathbf{M}(\underline{t})[\mathbf{p}] + \mathbf{n}(\underline{t}). \tag{4.3}$$

I note that since Equation 4.2 is a rational function (as defined by Moore [68, pp. 19-21]), Equation 4.3 is simply the natural interval extension of Equation 4.2 (i.e., Equation 4.3 is identically Equation 4.2 with the variables replaced by intervals and the algebraic operations replaced by interval arithmetic operations). Thus, because Equation 4.3 above is a rational interval function and a natural interval extension of $\mathbf{x}(\underline{t}, \mathbf{p})$, by Corollary 3.1 in [68],

$$\text{range}\left(\mathbf{x}(\underline{t}, [\mathbf{p}])\right) \subseteq [\mathbf{x}](\underline{t}, [\mathbf{p}]). \tag{4.4}$$

Furthermore, because $[\mathbf{x}](\underline{t}, [\mathbf{p}])$ is a natural interval extension of a rational function in which each variable occurs only once and to the first power, Equation 4.4 holds with equality. Because t was fixed arbitrarily for any $t \in (t_0, t_f]$, Equation 4.4 holds pointwise with equality for all $t \in (t_0, t_f]$.

□

Proposition 4.6. *For the differential equations defined in Proposition 4.5 above, the bounds for* $\dot{\mathbf{x}}(t, \mathbf{p})$ *are given pointwise in time by the following interval equation:*

$$[\dot{\mathbf{x}}](\underline{t}, [\mathbf{p}]) = (\mathbf{A}(\underline{t})\mathbf{M}(\underline{t}) + \mathbf{B}(\underline{t}))[\mathbf{p}] + \mathbf{A}(\underline{t})\mathbf{n}(\underline{t}) + \mathbf{r}(\underline{t}). \tag{4.5}$$

Proof. The functional form of the solution to the differential equation is given by Equation 3.3. For any fixed $\underline{t} \in (t_0, t_f]$, I substitute Equation 3.3 into the differential equations to yield

$$\dot{\mathbf{x}} = \mathbf{A}(\underline{t})(\mathbf{M}(\underline{t})\mathbf{p} + \mathbf{n}(\underline{t})) + \mathbf{B}(\underline{t})\mathbf{p} + \mathbf{r}(\underline{t}).$$

The above equation **must** be factored to yield

$$\dot{\mathbf{x}}(\underline{t}, \mathbf{p}) = (\mathbf{A}(\underline{t})\mathbf{M}(\underline{t}) + \mathbf{B}(\underline{t}))\mathbf{p} + \mathbf{A}(\underline{t})\mathbf{n}(\underline{t}) + \mathbf{r}(\underline{t}).$$

The interval extension of the above equation is taken to yield

$$[\dot{\mathbf{x}}](\underline{t},[\mathbf{p}]) = (\mathbf{A}(\underline{t})\mathbf{M}(\underline{t}) + \mathbf{B}(\underline{t}))[\mathbf{p}] + \mathbf{A}(\underline{t})\mathbf{n}(\underline{t}) + \mathbf{r}(\underline{t}). \tag{4.6}$$

By an identical argument as in the proof of Proposition 4.5, I now have that Equation 4.6 is a natural interval extension of a rational function in which each variable occurs only once and to the first power. Thus,

$$\text{range}\,(\dot{\mathbf{x}}(\underline{t},[\mathbf{p}])) = [\dot{\mathbf{x}}](\underline{t},[\mathbf{p}]).$$

Because t was fixed arbitrarily for any $\underline{t} \in (t_0, t_f]$, the above equation holds pointwise for all $t \in (t_0, t_f]$.

□

Several important comments need to be made concerning the state bounds. First, as illustrated by the two propositions, the bounds depend only upon the differential equations and the bounds on $\mathbf{p}$. Thus, the bounds are completely independent of the convex relaxation technique chosen for the optimization problem. Second, because the bounds compute the exact range of the interval equations, the bounds generate the tightest possible envelope on the solution of the differential equations that does not exclude any solution trajectory for any $\mathbf{p} \in P$. Finally, the state bounds are at least piecewise continuous with only a finite number of stationary simple discontinuities in t. This fact preserves the piecewise continuity of the objective function for the convex underestimating problem and hence guarantees by Corollary 2.7 that the convex relaxation attains its minimum value over P. The continuity of the state bounds is formalized by the following corollary:

Corollary 4.7. *The bounds $\mathbf{x}^L(t)$ and $\mathbf{x}^U(t)$ as determined from Proposition 4.5 are continuous on $(t_0, t_f]$.*

Proof. The proof follows immediately from Equation 3.2, Proposition 4.5, and interval arithmetic.

□

Remark 4.8. By a similar argument, the bounds on $\dot{\mathbf{x}}(t, \mathbf{p})$ as determined from Proposition 4.6 are piecewise continuous with stationary simple discontinuities only in t.

Because Propositions 4.5 and 4.6 only defined pointwise in time sets, the following notation is introduced to define time varying bounding sets.

Definition 4.9.

$$\mathcal{X}(\underline{t}) = \{\mathbf{x}(\underline{t}, \mathbf{p}) \mid \mathbf{p} \in P,\ \underline{t} \text{ fixed } \in [t_0, t_f]\}$$

$$\dot{\mathcal{X}}(\underline{t}) = \{\dot{\mathbf{x}}(\underline{t}, \mathbf{p}) \mid \mathbf{p} \in P,\ \underline{t} \text{ fixed } \in [t_0, t_f]\}$$

$$\mathcal{X} = \bigcup_{\underline{t} \in [t_0, t_f]} \mathcal{X}(\underline{t})$$

$$\dot{\mathcal{X}} = \bigcup_{\underline{t} \in [t_0, t_f]} \dot{\mathcal{X}}(\underline{t}).$$

Remark 4.10. It is clear that $\mathcal{X} \subseteq X$ and $\dot{\mathcal{X}} \subseteq \dot{X}$. Additionally, from Theorems 3.1, 3.4, 3.5 in [76]; Equation 3.3; and the above definitions, it is evident that the set $\mathcal{X}(\underline{t}) \times \dot{\mathcal{X}}(\underline{t}) \times P$ is convex for all $\underline{t} \in [t_0, t_f]$.

An example is now presented to illustrate the nuances of analytically calculating the exact state bounds for a linear dynamic system. A numerical technique for computing the exact state bounds is given in the following chapter.

Example 4.11. Consider the following system of linear differential equations:

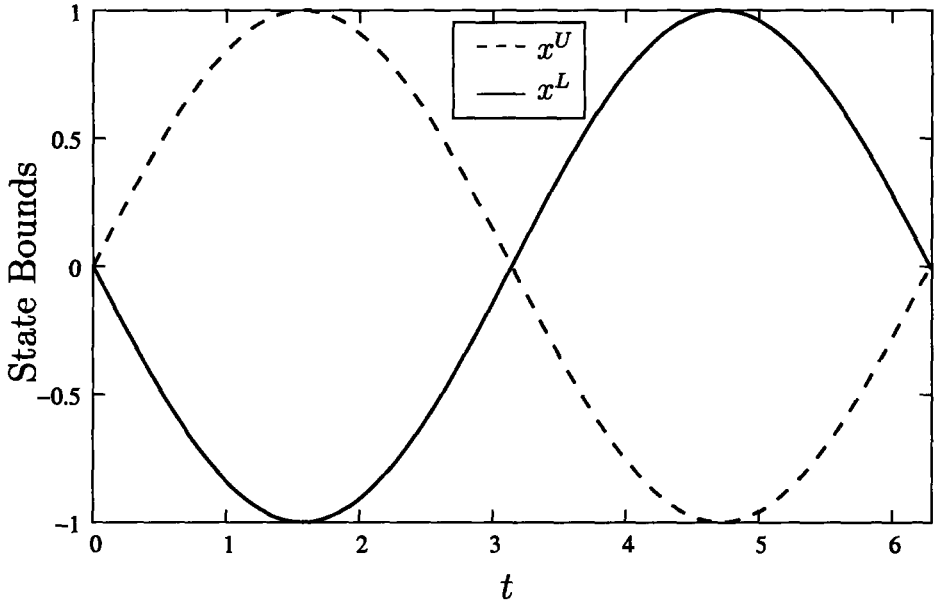
$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -x_1$$
$$x_1(0) = 0,\ x_2(0) = p$$
$$p \in [-1, 1].$$

The analytical solution to the above differential equations is

$$x_1(t, p) = p \sin(t) \quad \text{and} \quad x_2(t, p) = p \cos(t). \tag{4.7}$$

Because the solutions for x_1 and x_2 are analogous, the reminder of this example is restricted to a discussion of the bounds for x_1. A naïve approach to calculating the tightest bounds would be to set $x_1^L = x_1(t, p^L)$ and $x_1^U = x_1(t, p^U)$. This yields the incorrect results in Figure 4-1. The correct approach to finding the tightest bounds

Figure 4-1: Incorrect bounds for Example 4.11



for x_1 is to construct the natural interval extensions of x_1. For any fixed $\underline{t} \in (t_0, t_f]$, the natural interval extension of Equation 4.7 is

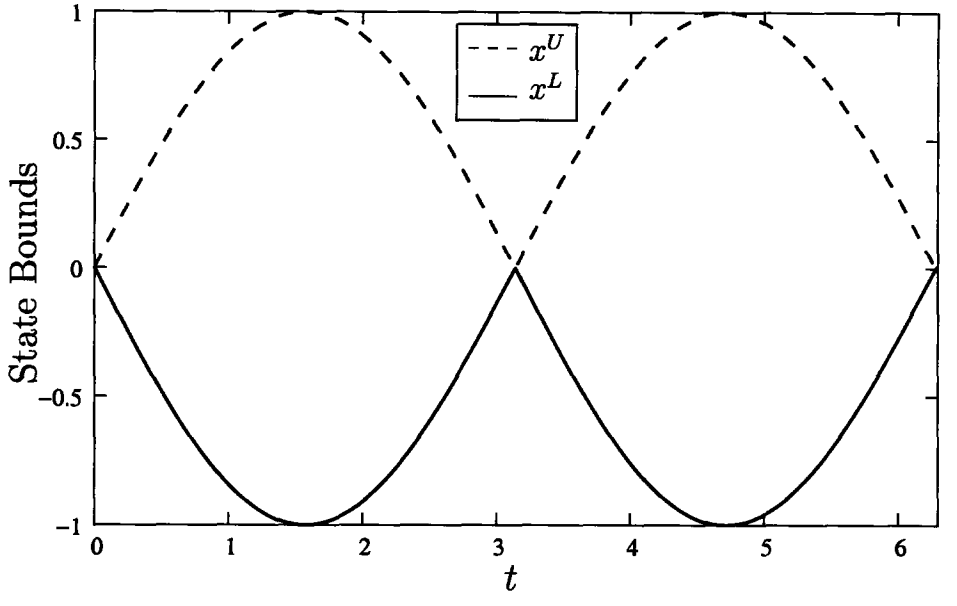
$$[x_1] = [-1, 1]\sin(\underline{t}).$$

From interval arithmetic, the tightest bounds are now found pointwise in time by the following formulae:

$$x_1^L(\underline{t}) = \min\{-1\sin(\underline{t}), 1\sin(\underline{t})\} \quad \text{and} \quad x_1^U(\underline{t}) = \max\{-1\sin(\underline{t}), 1\sin(\underline{t})\} \ \forall\, \underline{t} \in (t_0, t_f].$$

A plot of the correct state bounds for x_1 is shown in Figure 4-2.

Figure 4-2: Correct bounds for Example 4.11



4.3 Linear Dynamic Relaxations and Branch-and-Bound Convergence

In this section, I combine the results of the previous sections into one theorem for relaxing Problem 4.1 and show that the derived relaxations lead to a convergent branch-and-bound algorithm. A trivial example is then examined to demonstrate the application of the relaxation theorem for an analytically tractable problem. A numerical implementation of the linear theory is presented in the following chapter. The following theorem provides a method for relaxing Problem 4.1:

Theorem 4.12. *Consider*

$$L(\mathbf{p}) = \int_{t_0}^{t_f} \ell(t, \mathbf{x}(t,\mathbf{p}), \dot{\mathbf{x}}(t,\mathbf{p}), \mathbf{p})\ dt$$

subject to

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{B}(t)\mathbf{p} + \mathbf{r}(t)$$
$$\mathbf{E}_0\mathbf{x}(t_0) + \mathbf{E}_f\mathbf{x}(t_f) = \mathbf{B}_{IC}\mathbf{p} + \mathbf{r}_{IC}.$$

Let $X = \mathcal{X}$ *and* $\dot{X} = \dot{\mathcal{X}}$. *If*

$$u(t, \mathbf{x}(t,\mathbf{p}), \dot{\mathbf{x}}(t,\mathbf{p}), \mathbf{p}) \leq \ell(t, \mathbf{x}(t,\mathbf{p}), \dot{\mathbf{x}}(t,\mathbf{p}), \mathbf{p}) \qquad \forall\ (\underline{t}, \mathbf{p}) \in (t_0, t_f] \times P$$

and if $u(t, \mathbf{x}(t,\mathbf{p}), \dot{\mathbf{x}}(t,\mathbf{p}), \mathbf{p})$ *is convex on* $\mathcal{X}(\underline{t}) \times \dot{\mathcal{X}}(\underline{t}) \times P$ *for each fixed* $\underline{t} \in (t_0, t_f]$, *then*

$$U(\mathbf{p}) = \int_{t_0}^{t_f} u(t, \mathbf{x}(t,\mathbf{p}), \dot{\mathbf{x}}(t,\mathbf{p}), \mathbf{p})\ dt$$

is convex on P *and* $U(\mathbf{p}) \leq L(\mathbf{p})$. *That is,* $U(\mathbf{p})$ *is a convex relaxation for* $L(\mathbf{p})$ *on* P.

Proof. That $U(\mathbf{p}) \leq L(\mathbf{p})$ is trivial from integral monotonicity (Lemma 3.4). The convexity of $U(\mathbf{p})$ is established by applying Proposition 4.3 in Corollary 3.7.

□

In order to justify the use of convex relaxations constructed via Theorem 4.12, I must show that these relaxations lead to a convergent branch-and-bound algorithm. A branch-and-bound algorithm is said to be at least infinitely convergent if the selection operation is bound improving and the bounding operation is consistent (Theorem IV.3 [48]). Additionally, convergence requires the ability to delete relaxed partitions for which the intersection of this partition and the feasible space is empty. By definition, a selection operation is bound improving if at least after a finite number of steps, at least one partition element where the actual lower bound is attained is selected for further partitioning (Definition IV.6 [48]). By Definition IV.4 [48], a bounding operation is consistent if, at every step, any unfathomed partition element can be further refined, and if any infinite sequence of nested partitions has the property that the lower bound in any partition converges to the upper bound of this same partition. In other words, not only must every unfathomed partition be refineable, but as any infinite sequence of nested partitions approaches its limit set, its lower bound must converge to the upper bound. In global NLP branch-and-bound algorithms, fathoming of a partition occurs only when its lower bound is greater than the current best upper bound (or within an ε tolerance). Therefore, partitions containing the global minimum are

fathomed only at termination. By construction, the branching strategy employed is exhaustive. In an exhaustive search, such as bisecting the variable satisfying

$$\arg\min_i \left| p_i^U - p_i^L \right|,$$

by the finite dimensionality of the problem, every unfathomed partition is selected for further partitioning after a finite number of steps. Thus, the retention of partitions on which the global minimum is attained and the utilization of an exhaustive search together imply a bound improving selection operation. Furthermore, because P is a subset of a Euclidean space, by the Archimedean property of the real numbers, any unfathomed partition can always be refined (by bisection on a coordinate axis, for example); therefore, the first criterion of a consistent bounding operation is trivially satisfied. However, it is not immediately obvious whether or not the integral underestimators obey the second property for consistent bounding. Proving convergence of a branch-and-bound algorithm for solving Problem 4.1 reduces to proving that any infinite sequence of nested partitions has the property that the lower bound in any partition converges to the upper bound of this same partition, where in this case, the lower bounds are defined by Theorem 4.12. Equivalently, convergence follows if the lower bound in any partition converges pointwise to the objective function in this same partition as the Euclidean norm of the partition approaches zero (the interval approaches degeneracy). The proof of this statement requires the assumption that the relaxation for the integrand of $U(\mathbf{p})$ itself possesses a consistent bounding operation with monotonic pointwise convergence. This assumption is justified because the convex underestimators utilized in standard optimization problems possess the property that as the interval decreases, the convex underestimators become higher with monotonic pointwise convergence (see [65] or [61] for examples). That the relaxations derived via Theorem 4.12 possess a consistent bound operation is now proven.

Theorem 4.13. *Consider the function $L(\mathbf{p})$ and convex underestimator $U(\mathbf{p})$ as defined by Theorem 4.12. If the interval in any partition approaches degeneracy, then $U(\mathbf{p})$ in this partition converges pointwise to $L(\mathbf{p})$ in this same partition.*

Proof. Choose any partition and any fixed $\underline{t} \in (t_0, t_f]$. Let $[\mathbf{p}^L, \mathbf{p}^U]$ be the bounds of the partition and let $[\mathbf{x}^L(\underline{t}), \mathbf{x}^U(\underline{t})]$ and $[\dot{\mathbf{x}}^L(\underline{t}), \dot{\mathbf{x}}^U(\underline{t})]$ denote the state bounds as defined by Proposition 4.5 and Proposition 4.6 respectively. From Equations 4.1 and 4.5, as the interval $[\mathbf{p}^L, \mathbf{p}^U]$ approaches the degenerate value of $\mathbf{p}^*$, the intervals $[\mathbf{x}^L(\underline{t}), \mathbf{x}^U(\underline{t})]$ and $[\dot{\mathbf{x}}^L(\underline{t}), \dot{\mathbf{x}}^U(\underline{t})]$ respectively approach implied degenerate values denoted $\mathbf{x}^*(\underline{t})$ and $\dot{\mathbf{x}}^*(\underline{t})$. From Theorem 4.12, the integrand underestimator u is convex on the subset $\mathcal{X}(\underline{t}) \times \dot{\mathcal{X}}(\underline{t}) \times P \ \forall \ \underline{t} \in (t_0, t_f]$ and hence is generated in the partition by the intervals $[\mathbf{p}^L, \mathbf{p}^U]$, $[\mathbf{x}^L(\underline{t}), \mathbf{x}^U(\underline{t})]$, and $[\dot{\mathbf{x}}^L(\underline{t}), \dot{\mathbf{x}}^U(\underline{t})]$. Suppose that at each step i, the interval $[\mathbf{p}^L, \mathbf{p}^U]_i$ is bisected (or reduced in some other manner) such that as $i \to \infty$, $[\mathbf{p}^L, \mathbf{p}^U]_i \to \mathbf{p}^*$, which in turn implies $[\mathbf{x}^L(\underline{t}), \mathbf{x}^U(\underline{t})]_i \to \mathbf{x}^*(\underline{t})$ and $[\dot{\mathbf{x}}^L(\underline{t}), \dot{\mathbf{x}}^U(\underline{t})]_i \to \dot{\mathbf{x}}^*(\underline{t})$ (the reduction is possible because exhaustive branching is assumed). By construction, I have the following sequence:

$$u_i \uparrow \ell \quad \text{as} \quad i \to \infty \quad \text{for } \underline{t} \in [t_0, t_f],$$

where the convergence arises because the bounds on u are all approaching degeneracy and the underestimator u is assumed to posses a consistent bounding operation with monotonic convergence to ℓ. Because $\underline{t}$ was fixed arbitrarily, the convergence is true for all $t \in (t_0, t_f]$. By the monotone convergence theorem (Theorem 1, Section 2.3 in [2]),

$$L(\mathbf{p}^*) = \int_{t_0}^{t_f} \ell \, dt = \lim_{i \to \infty} \int_{t_0}^{t_f} u_i \, dt = U(\mathbf{p}^*).$$

Because the partition was arbitrarily chosen, the convergence is applicable to any partition.

□

Remark 4.14. Strictly speaking, the monotone convergence theorem only applies to positive sequences. Of course, if u is not positive over all of $(t_0, t_f]$, then u can be divided into u_+ and u_-, and the integral can be written as the difference of integrals possessing only positive integrands. The monotone convergence theorem is then applied piecewise. Additionally, the monotone convergence theorem is applicable to measurable functions and the Lebesgue integral. Of course, this poses no difficulty

because Riemann integrability implies Lebesgue integrability.

Until now, I have avoided discussing the requirement that any relaxed partitions for which the intersection of this partition and the feasible space is empty must be deleted. In general, deletion by certainty is a reasonably difficult task for general nonconvex problems. For most of the problems considered in this thesis, deletion by certainty is trivial because the problems are constrained only by bounds on the parameters. The relaxed partitions in the branch-and-bound algorithm are always subsets of P; therefore, the intersection of any relaxed partition and the feasible space is always nonempty. For problems containing point or isoperimetric constraints, because partitions are always relaxed by convex relaxations, Theorem 4.13, combined with exhaustive partitioning, implies that deletion by infeasibility is certain in the limit. This is obvious, for in the limit, the relaxed partition converges to the feasible space.

Having now proven the convergence of a branch-and-bound algorithm for finding a global minimum of Problem 4.1, this chapter is concluded with a small example illustrating the basics of solving Problem 4.1. A very simple example problem with an analytical solution was chosen to emphasize the convexity theory and state bounding techniques. Moreover, this low dimensional example problem permits graphical visualization of the solution and hence immediate verification of the results. A numerical implementation is discussed in the following chapter, and this implementation is applied to problems more complicated than that considered in Example 4.15 below.

Example 4.15. Consider the following nonconvex integral:

$$\min_{p\in[-4,4]} L(p) = \int_0^1 -x^2 \, dt$$

subject to

$$\dot{x} = -2x + p$$
$$x(0) = 1.$$

In order to construct a convex underestimator, the integrand is considered as a function on a Euclidean space. Since the integrand is a univariate concave function, its

convex envelope is the secant connecting the two end points. Utilizing Theorem 4.12 yields the following convex underestimator for $L(p)$:

$$U(p) = \int_0^1 (-(x^U)^2 + (x^L)^2)(x - x^L)/(x^U - x^L) - (x^L)^2 \, dt. \tag{4.8}$$

In order to complete the convex underestimator, the state bounds are needed. The following equation is the solution of the embedded differential equation:

$$x(t,p) = [1/2 - \exp(-2t)/2]p + \exp(-2t).$$

The natural interval extension of the above equation is constructed generating the following two implied state bounds equations:

$$\begin{aligned} x^L(t) &= [1/2 - \exp(-2t)/2]p^L + \exp(-2t) \\ x^U(t) &= [1/2 - \exp(-2t)/2]p^U + \exp(-2t). \end{aligned}$$

Upon substitution into Equation 4.8, the following convex underestimator for the objective function $L(p)$ is derived for the root node:

$$U(p) = \int_0^1 (\exp(-4t) - \exp(-2t))p + 8\exp(-2t) - 5\exp(-4t) - 4 \, dt.$$

Figure 4-3 below depicts the objective function with its convex underestimator. In this case, the underestimator is the convex envelope. From Figure 4-3, one observes that two minima occur, a local minimum at $p = -4$ and the global minimum at $p = +4$. Starting a local optimizer at $p < p_{max}$ finds the local minimum, and branching is required in order to find the global minimum. The subset P is bisected, and the process of convex relaxation is applied to each partition yielding the results found in Figure 4-4. From Figure 4-4, one clearly sees that the left region is fathomed and the global minimum $P = -2.516$ is found at $p = +4$.

Figure 4-3: Objective function and relaxation for Example 4.15 at the root node

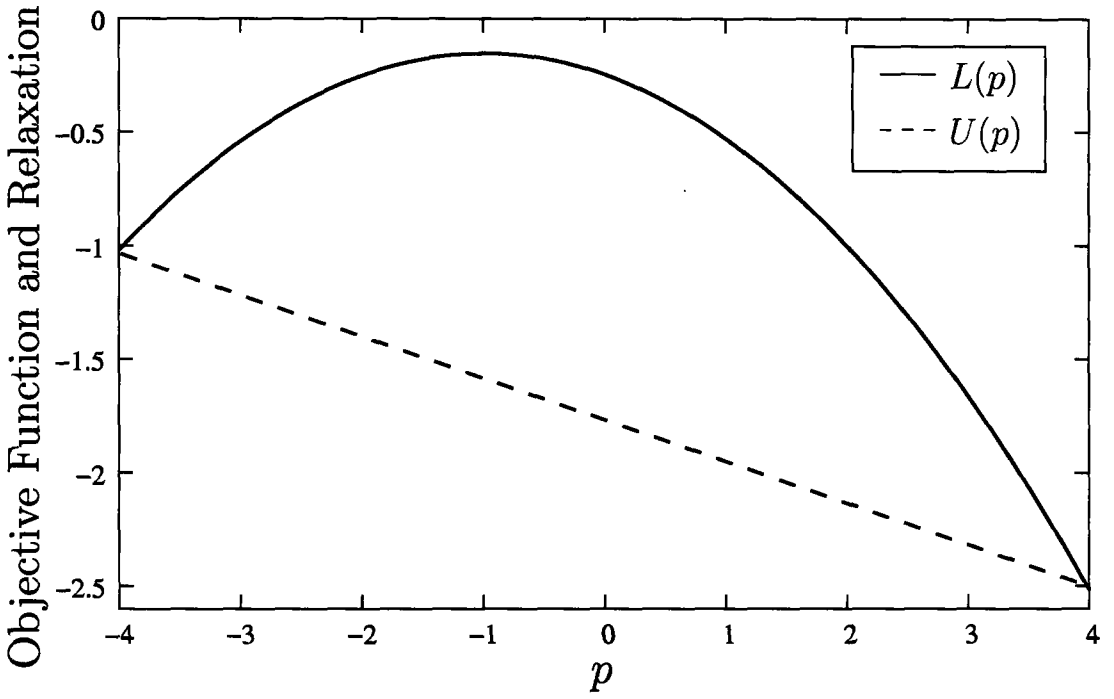
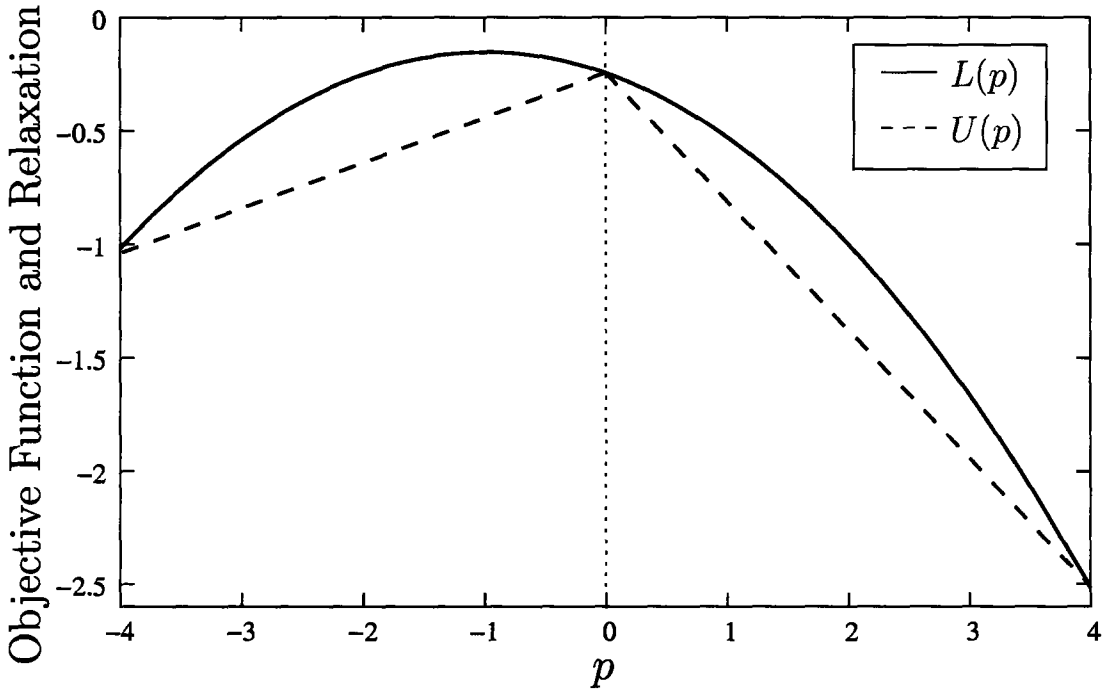


Figure 4-4: Objective function and relaxation for Example 4.15 after the first bisection



Chapter 5

Implementation for Problems with Linear Dynamics Embedded

In this chapter, I examine the numerical aspects of globally solving Problem 4.1 introduced in the previous chapter. In particular, this chapter discusses numerical solution of the upper bounding problem, the lower bounding problem, and computation of the state bounds as defined by Proposition 4.5. To demonstrate the feasibility of the implementation, several numerical case studies are examined.

5.1 The Three Subproblems: Upper Bound, Lower Bound, and State Bounds

As previously stated, the optimization algorithm utilized to calculate the global minimum of Problem 4.1 is based upon branch-and-bound concepts for determining the global minimum of standard optimization problems. Essentially, three subproblems exist: deriving and solving locally an upper bounding problem on each partition of P, deriving and solving locally a convex relaxation on each partition of P, and computing state bounds for the linear dynamics on each partition of P. For Problem 4.1, because the branch-and-bound procedure is performed on P, a subset of a Euclidean space, standard techniques are utilized for branching the set. For most practical

optimization problems (and those considered here), the set P is an n_p-dimensional hyperrectangle. The heuristic for selecting on which variable in any partition is selected for branching is the coordinate with the largest difference between its current upper and lower bound ($\arg\max_i |p_i^U - p_i^L|$); the branching itself is performed by bisecting the chosen coordinate. This branching scheme is shown to be exhaustive in [48]. A best first search (lowest lower bound on the objective function) criterion is utilized for node selection; this is obviously bound improving. A large number of heuristics have been developed to accelerate the performance of branch-and-bound algorithms (e.g., see [78, 79]). These techniques could be used to improve the computational statistics reported in this chapter. However, I have deliberately kept my branch-and-bound implementation simple in order to facilitate reproduction of and comparison to my results. I now examine the three subproblems in detail. To simplify the notation, unless specifically required, subscripts on variables will not be used to denote partitions; inclusion into a partition is implied by context.

5.1.1 Computing an Upper Bound

In any branch-and-bound algorithm, several techniques exist for determining the upper bound on the minimum value in any particular partition. Because I only seek a value guaranteed to be greater than or equal to the global minimum value in a partition, by definition, the objective function value at any feasible point satisfies the upper bounding criterion. However, I have chosen to satisfy the upper bounding criterion by solving for a local minimum; the optimization is performed by NPSOL [43]. For each fixed $\mathbf{p}$ chosen by the optimizer, a numerical value for the objective function and its gradient must be provided. Although the optimization itself is performed on a Euclidean space, the computations of the objective function and its gradient are nontrivial tasks due to the embedding of the dynamic system. From Problem 4.1, the objective function is rewritten as the following system of equations:

$$J(\mathbf{p}) = \phi(\mathbf{x}(t_f, \mathbf{p}), \dot{\mathbf{x}}(t_f, \mathbf{p}), \mathbf{p}) + z_{ubp}(t_f, \mathbf{p}) \tag{5.1a}$$

$$\dot{z}_{ubp} = \ell(t, \mathbf{x}(t,\mathbf{p}), \dot{\mathbf{x}}(t,\mathbf{p}), \mathbf{p}) \tag{5.1b}$$

$$z_{ubp}(0) = 0. \tag{5.1c}$$

Clearly, upon integration of $\dot{z}_{ubp}$, the original objective function is recovered. Therefore, the above system of equations is coupled with the original embedded linear dynamic system and numerically integrated in order to calculate the objective function. The gradient of $J(\mathbf{p})$ is found by differentiating Equation 5.1a and application of the chain rule, where $\partial z_{ubp}/\partial \mathbf{p}$ is merely the sensitivity of z_{ubp} to the parameters $\mathbf{p}$ at the final time. The numerical integration code chosen for this task was DSL48S [32], a numeric component of DAEPACK [88, 87]. DSL48S was chosen because of its ability to compute efficiently dynamic sensitivities in parallel to numerical integration. Since DSL48S was designed for large, sparse, stiff systems, it incurs significant overhead for the small examples considered here. However, using DSL48S allows the implementation to easily scale to large, sparse dynamic systems.

5.1.2 Computing State Bounds

In order to compute a convex relaxation for Problem 4.1, state bounds must first be computed because Theorem 4.12 requires convexification of the integrand on the set $\mathcal{X}(\underline{t}) \times \dot{\mathcal{X}}(\underline{t}) \times P$ for each fixed $\underline{t} \in (t_0, t_f]$. Because the state bounds can be computed independently of the lower bound, one could compute them once at each node and store the values for successive lower bounding function calls. For simplicity, however, the state bounds are computed in parallel with the convex relaxation. As described in the previous chapter, Equation 4.1 is utilized to compute $\mathcal{X}$ and Equation 4.5 is utilized to compute $\dot{\mathcal{X}}$. Both equations are computed pointwise in time via natural interval arithmetic from a combination of the original linear differential system and the affine solution quantities $\mathbf{M}(\underline{t})$ and $\mathbf{n}(\underline{t})$ as defined by Equation 3.3. Therefore, computing the state bounds effectively reduces to computing $\mathbf{M}(\underline{t})$ and $\mathbf{n}(\underline{t})$. The reader is reminded that pointwise in time computations are performed at each integration step.

From Equations 3.2 and 3.3, the following equations are extracted defining $\mathbf{M}(t)$

and $\mathbf{n}(t)$:

$$\mathbf{M}(t) = \mathbf{\Phi}(t,t_0)(\mathbf{E_0}+\mathbf{E_f}\mathbf{\Phi}(t_f,t_0))^{-1}\left[\mathbf{C}-\mathbf{E_f}\int_{t_0}^{t_f}\mathbf{\Phi}(t_f,\tau)\mathbf{B}(\tau)\,d\tau\right]+\int_{t_0}^{t}\mathbf{\Phi}(t,\tau)\mathbf{B}(\tau)\,d\tau \tag{5.2}$$

$$\mathbf{n}(t) = \mathbf{\Phi}(t,t_0)(\mathbf{E_0}+\mathbf{E_f}\mathbf{\Phi}(t_f,t_0))^{-1}\left[\mathbf{d}-\mathbf{E_f}\int_{t_0}^{t_f}\mathbf{\Phi}(t_f,\tau)\mathbf{r}(\tau)\,d\tau\right]+\int_{t_0}^{t}\mathbf{\Phi}(t,\tau)\mathbf{r}(\tau)\,d\tau. \tag{5.3}$$

Differential equations for $\mathbf{M}(t)$ and $\mathbf{n}(t)$ can be derived by several different methods. As previously discussed, $\mathbf{M}(t)$ and $\mathbf{n}(t)$ are both defined from Equation 3.3, which is the affine form of the solution of the embedded linear dynamic system. By inspection, $\mathbf{M}(t) = \partial\mathbf{x}/\partial\mathbf{p}$, the sensitivity of the state equations with respect to the optimization parameters. From the embedded linear dynamic system, the system of equations is differentiated with respect to $\mathbf{p}$ to yield

$$\frac{d}{dt}\frac{\partial\mathbf{x}}{\partial\mathbf{p}} = \mathbf{A}(t)\frac{\partial\mathbf{x}}{\partial\mathbf{p}} + \mathbf{B}(t).$$

The substitution $\mathbf{M}(t) = \partial\mathbf{x}/\partial\mathbf{p}$ and the interchange of differentiation operators yields the desired result. The boundary condition is found by differentiating the boundary condition for the embedded linear dynamic system:

$$\mathbf{E}_0\mathbf{M}(t_0) + \mathbf{E}_f\mathbf{M}(t_f) = \mathbf{B}_{IC}.$$

$\mathbf{n}(t)$ is found from the following problem:

$$\dot{\mathbf{n}} = \mathbf{A}(t)\mathbf{n} + \mathbf{r}(t)$$
$$\mathbf{E}_0\mathbf{n}(t_0) + \mathbf{E}_f\mathbf{n}(t_f) = \mathbf{r}_{IC}.$$

That the solution to the above differential equation matches the formula given for $\mathbf{n}(t)$ in Equation 5.3 is easily verifiable by integration.

Having derived differential equations defining $\mathbf{M}$ and $\mathbf{n}$, I now address the practical computation of the state bounds. As previously stated, theoretically, Equations 4.1 and 4.5 are calculated at every fixed time in the interval $(t_0, t_f]$. However, because

this interval is a connected subset of $\mathbb{R}$, there are an uncountable number of fixed points $\underline{t}$. Therefore, for the purposes of numerically integrating the lower bounding system, fixed $\underline{t}$ calculations are performed at each integration step. Thus, at each integration step, in order to implement interval arithmetic, Equations 4.1 and 4.5 are rewritten as follows for $i = 1, \ldots, n_x$:

$$x_i^L(\underline{t}) = n_i(\underline{t}) + \sum_{j=1}^{n_p} \min\{M_{ij}(\underline{t})p_j^L, M_{ij}(\underline{t})p_j^U\} \tag{5.4a}$$

$$x_i^U(\underline{t}) = n_i(\underline{t}) + \sum_{j=1}^{n_p} \max\{M_{ij}(\underline{t})p_j^L, M_{ij}(\underline{t})p_j^U\} \tag{5.4b}$$

$$\dot{x}_i^L(\underline{t}) = r_i(\underline{t}) + \sum_{j=1}^{n_x} A_{ij}n_j + \sum_{j=1}^{n_p} \min\{(M_{ij}^*(\underline{t})p_j^L, M_{ij}^*(\underline{t})p_j^U\} \tag{5.4c}$$

$$\dot{x}_i^U(\underline{t}) = r_i(\underline{t}) + \sum_{j=1}^{n_x} A_{ij}n_j + \sum_{j=1}^{n_p} \max\{(M_{ij}^*(\underline{t})p_j^L, M_{ij}^*(\underline{t})p_j^U\} \tag{5.4d}$$

where $\mathbf{M}^* \equiv \mathbf{AM} + \mathbf{B}$. By definition, the lower bound, p_i^L, is always less than or equal to the upper bound, p_i^U. Therefore, the minimum and maximum operators in the above equations only select different functions when $M_{ij}(t)$ or $M_{ij}^*(t)$ have zero crossings during an integration step. While this phenomena does not occur with every differential system, its presence creates difficulties, for it generates state dependent discontinuities in the integration of the convex lower bounding problem. For this reason, DSL48E [87], the differential-algebraic numerical integration component of DAEPACK capable of handling events, is utilized for computing the state bounds.

5.1.3 Lower Bounding Problem

Now that a method has been elucidated for computing the state bounds, I address computing a lower bound for Problem 4.1. Similar to the upper bounding problem, the lower bounding problem is a local optimization of an integral objective function subject to an embedded linear dynamic system. For the global optimization algorithm, the lower bounding problem is always chosen to be a convex relaxation of the original problem. Theorem 4.12 prescribes the method used to derive a convex under-

estimator for Problem 4.1, a method which is based upon convexifying the integrand of the objective function pointwise in time. Because the lower bounding problem is convex by construction, by Theorem 3.4.2 in [15], a minimum found via a local optimization routine is a global minimum on the respective partition. Therefore, NPSOL is again chosen as the optimization routine. As with the upper bounding problem, the integral is replaced by a differential equation:

$$\begin{aligned} U(\mathbf{p}) &= \phi_{lbp}(\mathbf{x}(t_f,\mathbf{p}),\dot{\mathbf{x}}(t_f,\mathbf{p}),\mathbf{p}) + z_{lbp}(t_f,\mathbf{p}) \\ \dot{z}_{lbp} &= u(t,\mathbf{p},\mathbf{x}(t,\mathbf{p}),\mathbf{x}^U(t),\mathbf{x}^L(t),\dot{\mathbf{x}}(t,\mathbf{p}),\dot{\mathbf{x}}^U(t),\dot{\mathbf{x}}^L(t)) \\ z_{lbp}(0) &= 0. \end{aligned}$$

As before, the gradient for the objective function of the lower bounding problem is found by differentiating $U(\mathbf{p})$, and the appropriate sensitivity equations are employed in order to yield this result. Clearly from differentiation, $\mathbf{M}$ is the sensitivity of $\mathbf{x}$ with respect to $\mathbf{p}$. Because $\mathbf{M}$ is explicitly calculated in parallel with computing the state bounds, an additional integration for calculating the sensitivity is unnecessary. The fact that the corrector matrix employed in the numerical integration of the embedded linear dynamic system is the same as that for computing the columns of $\mathbf{M}$ and $\mathbf{n}$ can be exploited via a staggered corrector scheme [32] to improve greatly the efficiency of the lower bounding problem. This, however, is not currently implemented.

5.1.4 Intersection of State Bounds and Intersection with State Bounds

Two of the more interesting numerical phenomena that arise in the algorithm for globally solving Problem 4.1 are the intersection of the state bounds and the intersection of a state solution trajectory with the state bounds. In a typical Euclidean optimization problem, the phenomenon of decision variable bounds intersection does not arise, for intersecting bounds indicate the search set for that variable has been reduced to degeneracy, immediately implying the solution for that variable at the particular node because no other feasible points would exist (this is the infinite con-

vergence solution with an exhaustive branching strategy). The other phenomenon that causes difficulty is the intersection of a state solution trajectory with a state bound. Of course, in Euclidean optimization, the analogous situation occurs quite frequently and poses no difficulty. In fact, for the extreme case of linear programming, the optimal point always lies at an active constraint. In dynamic optimization, the intersection of state bounds and the intersection of a state solution trajectory with a state bound occur quite frequently, and their occurrences potentially create havoc in the numerical integration of the lower bounding problem. The intersection of state bounds is addressed first.

At first glance, the intersection of state bounds would seem to imply the degeneracy of the set. However, because the state bounds are functions of time, their intersection may occur at a finite number of points in time even when the measure of the set is greater than zero. This situation arises when the solution of the embedded linear system contains a point in time whose value is independent of the embedded parameters. More commonly though, the situation arises when the initial conditions to the linear system are parameter independent causing the upper and lower bounds of the state variables to intersect at the initial time. Often, the equations defining convex relaxations contain the difference of the state bounds in the denominator (for example, the convex relaxation of a univariate concave function). Provided the intersection of the state bounds occurs only at a finite number of points in time, the collection of these occurrences form a set of measure zero, and integration theory permits the integration of a function containing these removable point discontinuities. Numerically, of course, these discontinuities generate residual calculations evaluating to "not a number," an unacceptable result. The solution of this dilemma is postponed until after the discussion of the intersection of state bounds with state solution trajectories, for the numerical solution to both problems is identical.

The other common numerical dilemma in working with the state bounds occurs when a candidate solution to the optimization problem causes a state variable to intersect with the state bounds. By the construction of the state bounds in Equations 5.4a and 5.4b, clearly, a state solution intersects its bound when the candidate

solution for $\mathbf{p}$ equals a vertex of the hyperrectangle defining its bound. Typically, the intersection is over a finite time interval rather than at individual time points. While this construction seems theoretically harmless, this situation induces chattering [91] due to numerical noise smaller than the integration tolerance. This phenomenon manifests itself in identifying events in functions such as the mid function (the mid function is paramount in computing McCormick relaxations and arises frequently). I present below a brief sketch of event location as it pertains only to the problem at hand. Much more will be said concerning event detection and location in Chapter 7.

I assume that the logical expressions that may cause events during the numerical integration may be decomposed into one or more real-valued relational expressions involving $<$, $\leq$, $>$, or $\geq$. For example, the logical expression

$$(x > 5) \text{ AND } (y < 10 \text{ OR } y > 20)$$

may be decomposed into three relational expressions, or logical atoms: $x > 5$, $y < 10$, and $y > 20$. Rearranging these logical atoms into the form $g \geq 0$ defines discontinuity functions (e.g., $x - 5$, $10 - y$, and $y - 20$). These discontinuity functions have the property that logical expressions may change in value when one or more of the discontinuity functions cross zero. The event location algorithms described in this chapter rely on finding zero crossings of the real-valued discontinuity functions to locate the events [21].

Now, suppose I wish to determine the middle value of three numbers $a(\underline{t}), b(\underline{t}), c(\underline{t})$ at every time step within an integration interval, and further suppose that the switching of the mid function from one argument to the next at time $\underline{t}^*$ causes an integration event in a state equation. Determining the middle value of three numbers obviously requires the direct comparison of the numbers. Let $d = a(\underline{t}) - b(\underline{t})$ be the discontinuity function derived from the comparison $a(\underline{t}) \geq b(\underline{t})$. An integration event occurs when d crosses 0, which clearly corresponds to equality in the direct comparison. If a and b intersect only at a finite number of points in time, the event location algorithm is well-behaved. However, if a and b are equal over a finite interval, chattering ensues.

This follows because $a = b$ only within a given integration tolerance. Therefore, even though $a = b$ analytically, the discontinuity function may cross zero at every integration step. This situation causes the event location algorithm to fail, and the numerical integration reaches an impasse.

The simplest solution to the above two problems is merely to construct the state bounds in a manner that eliminates the inherent difficulties. As described in the previous chapter, Equations 5.4a and 5.4b yield the tightest possible bounds on the solution to the embedded linear system. However, proving infinite convergence of the branch-and-bound algorithm does not require the tightest bounds. Rather, in order to guarantee infinite convergence of the algorithm, the only property required of the state bounds is that any infinite sequence of nested partitions has the property that in the limit partition, the lower bound approaches the objective function. To rectify the inherent numerical difficulties in using the tightest possible state bounds, I write the following convergent, yet nonintersecting bounds as a modification of the implied state bounds:

$$x_i^{L^*} = x_i^L - \varepsilon \cdot \|\mathbf{p}^U - \mathbf{p}^L\| \qquad i = 1, \ldots, n_x \tag{5.5a}$$

$$x_i^{U^*} = x_i^U + \varepsilon \cdot \|\mathbf{p}^U - \mathbf{p}^L\| \qquad i = 1, \ldots, n_x \tag{5.5b}$$

for some $\varepsilon > 0$ and suitable norm. Clearly, $\mathbf{x}^{L^*}$ and $\mathbf{x}^{U^*}$ rigorously bound $\mathbf{x}$. Furthermore, these bounds retain the necessary properties to ensure infinite convergence of the branch-and-bound algorithm (the necessary modification to the proof of Theorem 4.13 is trivial). An important note is that additional variables must be created in the implementation of the modified state bounds. This allows the state event location to be performed with the original bounds defined in Equations 5.4a and 5.4b, while actual numerical calculations are performed using the modified state bounds defined by Equation 5.5a and 5.5b. Obviously, for all $\mathbf{p}^U \neq \mathbf{p}^L$, the following holds with strict inequality:

$$x_i^{L^*} < x_i < x_i^{U^*} \qquad i = 1, \ldots, n_x.$$

Therefore, the two aforementioned problems are eliminated, for the modified bounds

can never intersect, and x_i can never intersect the modified bounds. The numerical routine is well-behaved provided that the solution is obtained before the norm on the partition approaches the integration tolerance. For notational convenience, the modified state bounds are written without the asterisk for the remainder of this chapter.

5.2 Dynamic Extensions to Standard Convex Relaxation Techniques

In order to relax Problem 4.1, convex relaxations for the integrand of the objective function must be derived via standard techniques. Relaxing the integrand is not always a trivial task, for the integrand itself is likely to be defined as a composition of both the states and parameters on the Euclidean set $\mathcal{X}(\underline{t}) \times \dot{\mathcal{X}}(\underline{t}) \times P$. In this thesis, McCormick's underestimators [65] and αBB underestimators [5] are the primary techniques utilized for convex relaxation (of course, special structure is exploited whenever possible). This section details the dynamic extension of these two convex relaxation techniques for solving Problem 4.1.

5.2.1 Use of McCormick's Underestimators

McCormick presents a method for obtaining a convex underestimator for any nonconvex, factorable program on a Euclidean space. I restate a condensed version of McCormick's result in Theorem 5.1 below. A brief discussion follows extending this result to dynamic systems. The interested reader is referred to McCormick's original paper [65] for details on this technique as it pertains to Euclidean optimization.

Theorem 5.1. *Let $S \subset \mathbb{R}^{n_x}$ be a nonempty convex set. Consider the function $V[v(\mathbf{x})]$ where $v : S \rightarrow \mathbb{R}$ is continuous, and let $S \subset \{\mathbf{x} \mid v(\mathbf{x}) \in [a, b]\}$. Suppose that a convex function $c(\mathbf{x})$ and a concave function $C(\mathbf{x})$ satisfying*

$$c(\mathbf{x}) \leq v(\mathbf{x}) \leq C(\mathbf{x}), \qquad \forall\, \mathbf{x} \in S$$

are available. Denote a convex relaxation of $V(\cdot)$ *on* $[a,b]$ *by* $e_V(\cdot)$*, denote a concave relaxation of* $V(\cdot)$ *on* $[a,b]$ *by* $E_V(\cdot)$*, let* z_{min} *be a point at which* $V(\cdot)$ *attains its infimum on* $[a,b]$*, and let* z_{max} *be a point at which* $V(\cdot)$ *attains its supremum on* $[a,b]$*. If the above conditions are satisfied, then*

$$u(\mathbf{x}) = e_V[\text{mid}\ \{c(\mathbf{x}), C(\mathbf{x}), z_{min}\}]$$

is a convex underestimator for $V[v(\mathbf{x})]$ *on* S*, and*

$$o(\mathbf{x}) = E_V[\text{mid}\ \{c(\mathbf{x}), C(\mathbf{x}), z_{max}\}]$$

is a concave overestimator for $V[v(\mathbf{x})]$ *on* S*, where the* mid *function selects the middle value of three scalars.*

Proof. I prove the theorem only for $u(\mathbf{x})$; the proof for $o(\mathbf{x})$ is analogous. First, the convex relaxation $e_V(\cdot)$ is broken up into the sum of three terms: the convex nonincreasing part, the convex nondecreasing part, and the minimum value. The convex nonincreasing part is defined as

$$e_V^D(v) = e_V\left(\min\{v, z_{min}\}\right) - A$$

and the convex nondecreasing part by

$$e_V^I(v) = e_V\left(\max\{v, z_{min}\}\right) - A,$$

where $A = e_V(z_{min})$. Obviously, then,

$$e_V(v) = e_V^D(v) + e_V^I(v) + A.$$

Consider the chain of inequalities

$$V[v(\mathbf{x})] \ \geq \ e_V[v(\mathbf{x})] = e_V^D[v(\mathbf{x})] + e_V^I[v(\mathbf{x})] + A$$

$$
\begin{aligned}
&\geq e_V^D[\min\{b, C(\mathbf{x})\}] + e_V^I[\max\{a, c(\mathbf{x})\}] + A \\
&= e_V[\min\{b, C(\mathbf{x}), z_{min}\}] + e_V[\max\{a, c(\mathbf{x}), z_{min}\}] - A \\
&= e_V[\min\{C(\mathbf{x}), z_{min}\}] + e_V[\max\{c(\mathbf{x}), z_{min}\}] - A \\
&= e_V[\mathrm{mid}\{c(\mathbf{x}), C(\mathbf{x}), z_{min}\}]
\end{aligned}
$$

The convexity of the underestimating function follows because the minimum of two concave functions is concave, the maximum of two convex functions is convex, a convex nonincreasing function of a concave function is convex, and a convex nondecreasing function of a convex function is convex. □

Remark 5.2. In his seminal paper [65], McCormick generalized the results of Theorem 5.1 to terms of the general form $T[t(\mathbf{x})] + U[u(\mathbf{x})] \cdot V[v(\mathbf{x})]$. Furthermore, the paper explains how Theorem 5.1 may be applied recursively to generate a convex underestimator for any factorable program.

McCormick notes the following two important facts concerning his convex underestimator $u(\mathbf{x})$; these are precisely the results necessary for constructing an infinitely convergent branch-and-bound optimization algorithm:

1. The underestimator $u(\mathbf{x})$ is "tight" in the sense that as the size of $[a, b]$ decreases, the value of the underestimator increases. The size of an interval is understood to represent some suitable norm such as the Euclidean norm.

2. If the interval $[a, b]$ becomes degenerate (i.e., $a = b$), then $u(\mathbf{x}) = V[v(\mathbf{x})]$.

As presented, Theorem 5.1 applies to Euclidean optimization problems where the inner function of the composition is bounded in range. Utilizing the exact state bounds as defined by Proposition 4.5, time varying enclosures of the solution of the differential equations for all $\mathbf{p}$ in a hyperrectangle P may be obtained. I label this interval $\mathcal{X}(t) = [\mathbf{x}^L(t), \mathbf{x}^U(t)]$ and also make extensive use of its pointwise in time analog $\mathcal{X}(\underline{t}) = [\mathbf{x}^L(\underline{t}), \mathbf{x}^U(\underline{t})]\ \forall\ \underline{t} \in (t_0, t_f]$. Because the interval $\mathcal{X}(t)$ (and hence $\mathcal{X}(\underline{t})$) rigorously bounds the range of $\mathbf{x}(t, \mathbf{p})$ and because $\mathcal{X}(t)$ shrinks to degeneracy as P

shrinks to degeneracy, I have a range interval $[a, b]$ satisfying Theorem 5.1 by computing interval extensions for $v(\mathbf{x})$ on $\mathcal{X}(\underline{t})$. In the case where recursion is required to generate an underestimator for the integrand, recursive interval extensions provide the additional range bounding sets; interval monotonicity ensures that the recursively derived range enclosures also approach degeneracy as the parameter space approaches degeneracy. To complete the justification for using Theorem 5.1 for dynamic problems, I must show that the composition is performed on a Euclidean space rather than on the function space $\mathcal{X}$. Clearly, for each fixed $\underline{t} \in (t_0, t_f]$, $\mathcal{X}(\underline{t}) \subset \mathbb{R}^{n_x}$ and is therefore on a Euclidean space. Additionally, any fixed time set created by interval extensions on $\mathcal{X}(\underline{t})$ is also a subset of a Euclidean space. This justifies utilizing Theorem 5.1 in conjunction with Theorem 4.12 to derive convex relaxations for Problem 4.1.

5.2.2 Use of αBB Underestimators

The second main convexification technique utilized in this study is αBB, a relaxation technique based on the augmentation of a nonconvex twice continuously differentiable function by a quadratic term of sufficient magnitude to yield a resultant function with a positive semidefinite Hessian on the set of interest. The construction of the quadratic term ensures that the relaxation underestimates the original function. Typically, αBB convex underestimators are much "weaker" (have lower objective function values at their minima) than the underestimators generated by McCormick's methods. Therefore, solving problems with αBB often requires more branch-and-bound nodes and hence more computation. However, αBB has several advantages over other convex relaxation techniques. First, the method is applicable to all twice continuously differentiable functions provided the user can provide an interval Hessian, a process automated by DAEPACK's interval code generation extensions. Second, the method always provides twice continuously differentiable convex relaxations, a substantial benefit when using standard optimization packages such as NPSOL. The implementation of αBB for Problem 4.1 is described below. The interested reader is referred to the articles by Adjiman *et al.* [4, 5] for details concerning the theory and imple-

mentation of αBB for Euclidean optimization problems and the multitude of rigorous polynomial complexity techniques for computing the α parameter.

Given a function $\ell(\mathbf{x}) \in C^2(\mathbb{R}^{n_x})$, Adjiman *et al.* [5] provide the following formula for relaxing $\ell(\mathbf{x})$ on the restricted domain $[\mathbf{x}^L, \mathbf{x}^U]$:

$$u(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n_x} \alpha_i (x_i^L - x_i)(x_i^U - x_i),$$

where α_i's are positive scalars. Furthermore, the relaxation $u(\mathbf{x})$ is convex if

$$H_u(\mathbf{x}) = H_\ell(\mathbf{x}) + 2\Delta \tag{5.6}$$

is positive semidefinite on $[\mathbf{x}^L, \mathbf{x}^U]$, where $H_\ell(\mathbf{x})$ is the Hessian of $\ell(\mathbf{x})$ and Δ is a diagonal matrix whose diagonal elements are α_i. The extension of αBB to Problem 4.1 is trivial. As previously stated, according to Theorem 4.12, deriving a convex underestimator for an integral entails deriving a convex underestimator for its integrand over the Euclidean space $\mathcal{X}(\underline{t}) \times \dot{\mathcal{X}}(\underline{t}) \times P$. Clearly, for an integrand $\ell[\mathbf{x}(\underline{t})] \in C^2(\mathcal{X}(\underline{t}))\ \forall\ \underline{t} \in [t_0, t_f]$, the following is a convex underestimator for $\ell(\mathbf{x})$ on $\mathcal{X}(\underline{t})$ at each fixed $\underline{t} \in [t_0, t_f]$:

$$u[\mathbf{x}(\underline{t})] = \ell[\mathbf{x}(\underline{t})] + \sum_{i=1}^{n_x} \alpha_i(\underline{t})[x_i^L(\underline{t}) - x_i(\underline{t})][x_i^U(\underline{t}) - x_i(\underline{t})]$$

provided suitable $\alpha_i(\underline{t})$ can be found such that $H_u(\mathbf{x}(\underline{t}))$ is positive semidefinite on $\mathcal{X}(\underline{t})$ for each fixed time point. In order for αBB to yield the tightest possible convex underestimator, the algorithm calculates individual α_i at each integration step utilizing the scaled Gerschgorin technique as described by Theorem 3.13 in [5]. Natural interval extensions are utilized in calculating the interval extension of the Hessian of the integrand, and a value of $\mathbf{d} = \mathbf{1}$ is used to avoid scaling difficulties associated with state bound intersections. The extension to integrands with functionality depending upon the parameter or the derivative of the states is obvious.

5.2.3 Convex Relaxation of Bilinear Terms

Bilinear terms (terms of the form x_1x_2) appear so often in optimization problems that the convex relaxation of these terms merits a brief, yet separate discussion. In his seminal exposition on the computability of global solutions to factorable nonconvex programs [65], McCormick states that the convex envelope of a bilinear term is given by the following equation:

$$u(\mathbf{x}) = \max\{x_1^L x_2 + x_2^L x_1 - x_1^L x_2^L,\ x_1^U x_2 + x_2^U x_1 - x_1^U x_2^U\}. \tag{5.7}$$

While Equation 5.7 provides the tightest possible convex relaxation, the function is not continuously differentiable. Because this inherent nonsmoothness upsets standard local optimization algorithms, most authors introduce extra variables and inequality constraints to generate an equivalent continuously differentiable optimization problem. However, the introduction of additional variables to replace state variables is not permitted theoretically by our algorithm for solving Problem 4.1, for this addition creates optimization degrees of freedom in an infinite dimensional space. The following example illustrates this point.

Example 5.3. Consider the objective function

$$\min_{\mathbf{p}\in P} L(\mathbf{p}) = \int_{t_0}^{t_f} x_1x_2\ dt$$

subject to constraints of the same form as those of Problem 4.1. Rather than employ the nonsmooth convex envelope of bilinear terms, a smooth reformulation is often considered where a new variable w is introduced and new inequalities are added. This yields the following convex relaxation:

$$U(\mathbf{p}, w) = \int_{t_0}^{t_f} w\ dt$$

subject to the original constraints and the new constraints

$$w \geq x_1^L(t) \cdot x_2 + x_2^L(t) \cdot x_1 - x_1^L(t) \cdot x_2^L(t)$$
$$w \geq x_1^U(t) \cdot x_2 + x_2^U(t) \cdot x_1 - x_1^U(t) \cdot x_2^U(t)$$
$$w \in C[t_0, t_f],$$

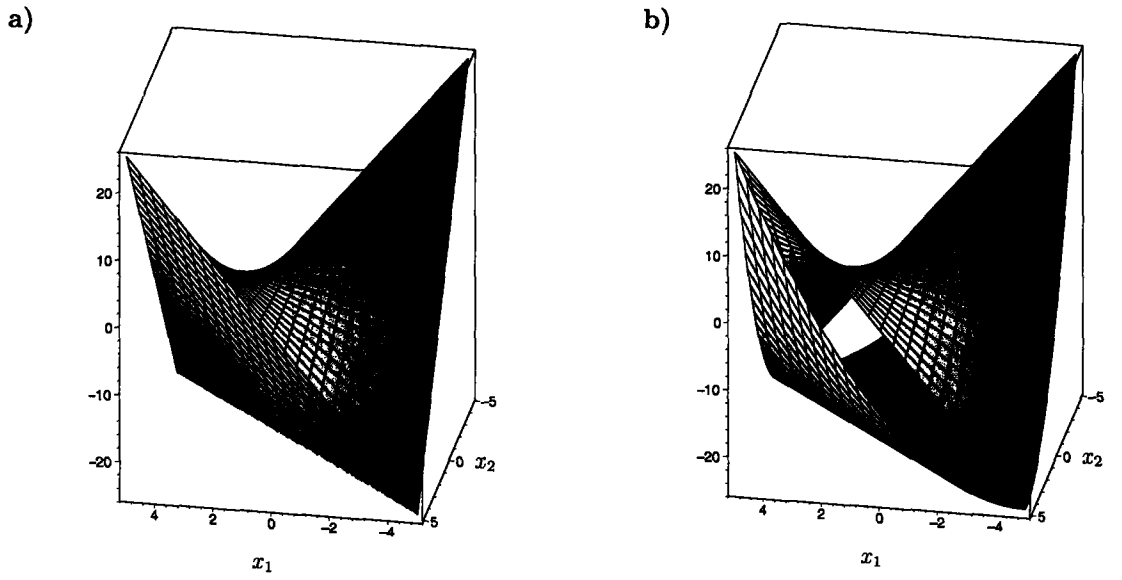
where $C[t_0, t_f]$ is the space of continuous functions on the interval $[t_0, t_f]$. Without introducing the new variable w, the embedded ODEs and boundary conditions allow the elimination of $\mathbf{x}$ from the objective function enabling us to consider only an optimization problem on P. Unfortunately, the newly added Lagrangian inequalities on w do not permit the elimination of w from the convex relaxation. Hence, the optimization problem must be considered on the space $P \times C[t_0, t_f]$ subject to the Lagrangian inequality constraints. Problems of this nature must be addressed by a variational approach (see Chapter 10).

An alternative to exploiting the convex envelope of the bilinear term is to construct an underestimator via αBB. Letting $\ell(\mathbf{x}) = x_1 x_2$, one can easily deduce that the Hessian of ℓ is constant; hence, the interval extension of the Hessian of ℓ is degenerate. Therefore, for dynamic systems, the values for α_i are independent of both time and the state bounds (and thus implicitly parameter bounds). The α_i values are calculated to be $1/2$ yielding the following dynamic convex underestimator:

$$u_{bl}[\mathbf{x}(\underline{t})] = x_1(\underline{t})x_2(\underline{t}) + [(x_1^L(\underline{t}) - x_1(\underline{t}))(x_1^U(\underline{t}) - x_1(\underline{t})) + (x_2^L(\underline{t}) - x_2(\underline{t}))(x_2^U(\underline{t}) - x_2(\underline{t}))]/2 \qquad \forall\, \underline{t} \in [t_0, t_f]. \quad (5.8)$$

Equation 5.8 is the preferred convex underestimator for bilinear terms appearing in Problem 4.1 and is also used for any bilinear terms occurring in McCormick's factorable programming decomposition. A comparison of the strength of the convex envelope and the αBB result on the Euclidean space $[-5, 5] \times [-5, 5]$ is presented in Figure 5-1 below.

Figure 5-1: Comparison of bilinear relaxation techniques: a) McCormick convex envelope b) αBB



5.3 Case Studies

The remainder of this chapter is dedicated to exploring the algorithm for globally solving Problem 4.1 with numerical examples ranging from dynamic extensions to literature problems to contrived examples designed to emphasize particular concepts. All of the example problems in this section have been intentionally designed with a low dimensionality for several reasons. First and foremost of these reasons is that I wish solely to emphasize the application and feasibility of the developed algorithm and not the existing techniques concerning Euclidean optimization, convex relaxation, and numerical integration. In fact, despite the inevitable increase in computational time associated with higher dimensional problems, the astute reader will recognize that the scaling of the algorithm to higher dimensional problems is immediate and has already been extensively addressed in the literature. That is, to increase the dimensionality of the problem increases either the size of the parameter space or the size of the function space. Essentially, increasing the size of the function space amounts to requiring the numerical integration of a large-scale DAE at each function call from the local optimizer. Since the exploitation of sparsity enables polynomial

algorithms for the numerical integration of systems containing hundreds of thousands of equations, solving a problem with even several hundred state variables is a trivial extension for any robust numerical integrator. Furthermore, because the optimization itself is on a Euclidean space, increasing the dimensionality of the parameter space can only illustrate the inherent weaknesses of utilizing pure branch-and-bound as a global optimization technique. Many authors, in particular Sahinidis *et al.* [78, 79], have already addressed such issues. The uniqueness of this algorithm lies in the juxtaposition of numerical integration and global optimization with convex relaxations of dynamic problems, not in the application of the individual techniques themselves.

Two event location approaches were employed for solving these case studies. Both approaches rely on identifying the zero crossings of the discontinuity functions as defined above. The first approach, called bisection, locates events by comparing the sign of the discontinuity function between the current and previous time steps. If the sign changes, then the zero crossing of the relevant discontinuity function is identified using a bisection algorithm [70]. As described in [72], this algorithm can fail in some cases. The second event location algorithm, called rigorous, employs the method developed by Park and Barton [72]. In this approach, new algebraic equations of the form $z_i = g_i$ are created and augmented to the original DAE, where g_i is the discontinuity function and z_i is the new discontinuity variable. This algorithm places the discontinuity functions under integration error control, ensuring accurate detection of events. During integration, potential zero crossings of the discontinuity functions are monitored using an algorithm based on interval arithmetic. This algorithm not only guarantees the zero crossing will be identified, but also that the first, and thus correct, crossing will be found. Once the event is identified, it is then "polished" to avoid the phenomenon of discontinuity sticking [72]. The rigorous approach guarantees that the event location will be performed robustly and accurately. In most circumstances, the rigorous approach will be only slightly more expensive than the bisection approach. However, the problems considered here do not obey this maxim. First, appending the relatively large number of discontinuity functions to the original DAE significantly alters the performance of the numerical integration, resulting

in more integration steps and Jacobian evaluations. Second, since many of the discontinuity function trajectories lie very close to zero (at least initially if the initial condition is not a function of the parameters), the cost of rigorous event detection based on interval methods is costly. Fortunately, for the problems examined here, the event location algorithm based on bisection performs reliably.

In order to solve dynamic optimization problems, three tolerances must be specified: the integration tolerance, the local optimization tolerance, and the branch-and-bound global tolerance. Because the numerical integration is a subproblem of the local optimization, the local optimization tolerance should be no tighter than the integration tolerance. For the same reason, the global optimization tolerance should be no tighter than the local optimization tolerance. For these examples, the absolute and relative integration tolerances were set to 10^{-8}, the local optimization tolerance was set to 10^{-5}, the absolute tolerance for the global optimization was set to 10^{-3}, and the relative tolerance for the global optimization was set to 10^{-3}. All code was compiled with gcc 3.2, and all simulations were performed on an AMD Athlon XP2000+ processor operating at 1667 MHz running SuSE Linux kernel 2.4.

5.3.1 Small Numerical Example

I now revisit the following problem first posed in the previous chapter.

Problem 5.4.

$$\min_{p\in[-4,4]} L(p) = \int_0^1 -x^2 \, dt$$

subject to

$$\dot{x} = -2x + p$$
$$x(0) = 1.$$

As was demonstrated in the previous chapter, in the space of the state variable, a convex underestimator for L is given by

$$U(p) = \int_0^1 (-(x^U)^2 + (x^L)^2)(x - x^L)/(x^U - x^L) - (x^L)^2 \, dt.$$

This problem was solved numerically via the developed code. The results, found in Table 5.1, are consistent with those obtained analytically. From Table 5.1, one sees

Table 5.1: Numerical results for Problem 5.4

Relaxation	Nodes	Problem	Int. calls	p_{min}	Obj. fcn.	Time[a] (s)	Time[b] (s)
convex envelope	3	lower	11	4.000	-2.516	0.07	0.23
		upper	10				
αBB	3	lower	11	4.000	-2.516	0.09	0.25
		upper	10				

[a]CPU time using bisection for event detection.
[b]CPU time using fully rigorous event detection.

that solving the problem via an αBB relaxation and solving the problem via the convex envelope yield virtually identical statistics. This immediately follows because a trivial analysis shows that αBB yields the convex envelope of this problem. The miniscule increase difference in CPU time derives from the Gerschgorin calculation to obtain α at every integration step.

5.3.2 Dynamic Extension to McCormick's Example Problem

The convex relaxation of the integrand for Problem 5.5 was first considered by McCormick [65] as an application of his convex relaxation technique. However, McCormick only considered optimization on Euclidean spaces; finding a convex underestimator for $\ell(\mathbf{x}) = (\exp(x_1) + x_2)^2$ subject to a dynamic system greatly increases the complexity. In this example, the chosen dynamic system illustrates the applicability of the algorithm to embedded linear systems with a forcing function. The remainder of this subsection details the analysis necessary for practical implementation of Problem 5.5 and the numerical results from the implementation. Additionally, Problem 5.5 was also solved utilizing an αBB underestimator. Figure 5-2 below shows the objective function with each underestimator.

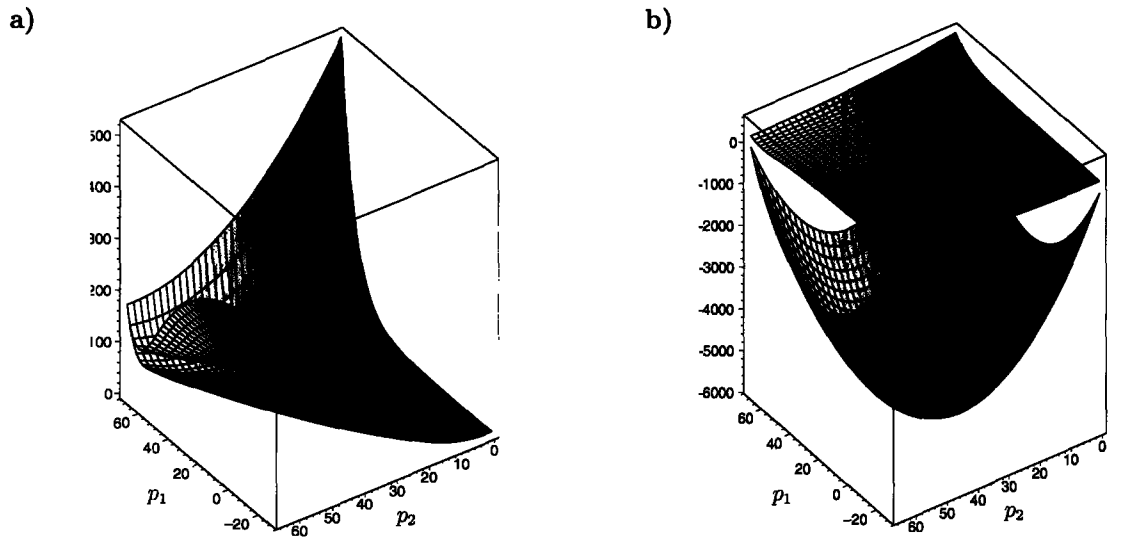
Problem 5.5.

$$\min_{\mathbf{p}} \; L(\mathbf{p}) = \int_0^{0.5} (\exp(x_1) + x_2)^2 \; dt$$

subject to

$$\dot{x}_1 = x_1 + p_1/10 + \sin(t)$$
$$\dot{x}_2 = x_1 - 2x_2 - 2p_2$$
$$x_1(0) = 0,\ x_2(0) = 0$$
$$\mathbf{p} \in [-30, 70] \times [0, 66].$$

Figure 5-2: Dynamic extension to McCormick's example problem at the root node: **a)** Objective function and McCormick's underestimator. **b)** Objective function and αBB underestimator



The first step in deriving a convex underestimator for the integrand of Problem 5.5 is identifying the composition that defines ℓ. For this problem, the composition is trivially written as

$$\ell = z^2$$
$$z = \exp(x_1) + x_2.$$

Following this composition, both a convex underestimating function and a concave overestimating function must be computed for z on $\mathcal{X}(\underline{t})$. For this problem, the convex underestimator on $\mathcal{X}(\underline{t})$ is trivially the function

$$c[\mathbf{x}(\underline{t})] = \exp[x_1(\underline{t})] + x_2(\underline{t}), \quad \forall\ \underline{t} \in [0, 0.5].$$

That $\mathbf{x}$ depends on $\mathbf{p}$ is explicitly dropped from the notation to emphasize that

$c(\mathbf{x})$ is a convex relaxation of z on the Euclidean space $\mathcal{X}(\underline{t})$ and not on the space P. In order to derive a concave overestimator for z, I examine the two functions $z_1 = \exp(x_1)$ and $z_2 = x_2$ separately. Trivially, because z_2 is an affine function of x_2, the concave overestimator for z_2 on $\mathcal{X}(\underline{t})$ is $C_2(x_2) = x_2(\underline{t})$. In order to derive a concave overestimator for z_1, I employ the following formula from McCormick [65]:

$$C_1(x) = \frac{\exp(x^U) - \exp(x^L)}{x^U - x^L}x + \frac{x^U \exp(x^L) - x^L \exp(x^U)]}{x^U - x^L}, \quad x \in [x^L, x^U].$$

One of the complications of the dynamic system in Problem 5.5 is that the bounds on the state variables are time varying. Utilizing the implied state bounds defined by Equations 5.4 and 5.5, the concave overestimator for z_1 on $\mathcal{X}(\underline{t})$ is given $\forall\ \underline{t} \in [0, 0.5]$ by

$$C_1[x_1(\underline{t}))] = \frac{\exp[x_1^U(\underline{t})] - \exp[x_1^L(\underline{t})]}{x_1^U(\underline{t}) - x_1^L(\underline{t})}x_1(\underline{t}) + \frac{x_1^U(\underline{t}) \exp[x_1^L(\underline{t})] - x_1^L(\underline{t}) \exp[x_1^U(\underline{t})]}{x_1^U(\underline{t}) - x_1^L(\underline{t})}.$$

Now, because $C_1[x_1(\underline{t})] + C_2[x_2(\underline{t})] \geq z_1 + z_2$ and because the sum of concave functions remains concave, I have that

$$C[\mathbf{x}(\underline{t})] = C_1[x_1(\underline{t})] + C_2[x_2(\underline{t})] = \frac{\exp[x_1^U(\underline{t})] - \exp[x_1^L(\underline{t})]}{x_1^U(\underline{t}) - x_1^L(\underline{t})}x_1(\underline{t}) + \frac{x_1^U(\underline{t}) \exp[x_1^L(\underline{t})] - x_1^L(\underline{t}) \exp[x_1^U(\underline{t})]}{x_1^U(\underline{t}) - x_1^L(\underline{t})} + x_2(\underline{t})$$

is a valid concave overestimator for z on $\mathcal{X}(\underline{t})\ \forall\ \underline{t} \in [0, 0.5]$. The next step in deriving a convex relaxation for Problem 5.5 is to calculate z_{min}, the point at which $\ell = z^2$ attains its minimum on the range of $z = \exp[x_1(\underline{t}, \mathbf{p})] + x_2(\underline{t}, \mathbf{p})$. Because the exponential function is inclusion monotonic, I can construct a superset of the range of z by taking the natural interval extension of z using the implied state bounds (Theorem 3.1 [68]). Denoting this superset by $Z = [z^L(\underline{t}), z^U(\underline{t})]$, I can write the following function for $z_{min}(\underline{t})\ \forall\ \underline{t} \in [0, 0.5]$:

$$z_{min}(\underline{t}) = \begin{cases} z^L(\underline{t}) & \text{if } z^L(\underline{t}) > 0 \\ z^U(\underline{t}) & \text{if } z^U(\underline{t}) < 0 \\ 0 & \text{otherwise.} \end{cases}$$

Noting $\forall\ \underline{t} \in [0, 0.5]$ that z^2 is the convex envelope of $\ell\ \forall\ z \in [z^L(\underline{t}), z^U(\underline{t})]$, according to McCormick's convex relaxation technique [65], I have the following convex underestimator for the integrand

$$u(\underline{t}, \mathbf{p}) = \left[\text{mid}\left\{c(\mathbf{x}(\underline{t}, \mathbf{p})), C(\mathbf{x}(\underline{t}, \mathbf{p})), z_{min}(\underline{t})\right\}\right]^2, \quad \forall\ \underline{t} \in [0, 0.5].$$

By Theorem 4.12,

$$U(\mathbf{p}) = \int_0^{0.5} u(t, \mathbf{p})\ dt$$

is a convex underestimator for the objective function of Problem 5.5.

Using both McCormick's underestimator and αBB, Problem 5.5 was solved globally; the results of these experiments are found in Table 5.2. As expected, each relaxation technique yields the same solution within the specified tolerances. However, the αBB underestimator requires more than fifty times the number of nodes to solve the problem than McCormick's underestimator. This behavior is expected, for αBB provides an underestimator that is much weaker than McCormick's underestimator for this problem (see Figure 5-2).

Table 5.2: Numerical results for Problem 5.5

Relaxation	Nodes	Problem	Int. calls	$\mathbf{p}_{min}$	Obj. fcn.	Time[a]	Time[b]
McCormick	9	lower	63	(14.398, 4.427)	0.0588	0.88	2.95
		upper	28				
αBB	503	lower	3292	(14.398, 4.427)	0.0588	46.1	124.8
		upper	1682				

[a]CPU s using bisection for event detection.
[b]CPU s using fully rigorous event detection.

5.3.3 Dynamic Extension to the Himmelblau Function

The next example is a dynamic extension to a classical optimization problem noted for its demonstration of multiple local minima. The objective function with its convex underestimators is plotted in Figure 5-3 below.

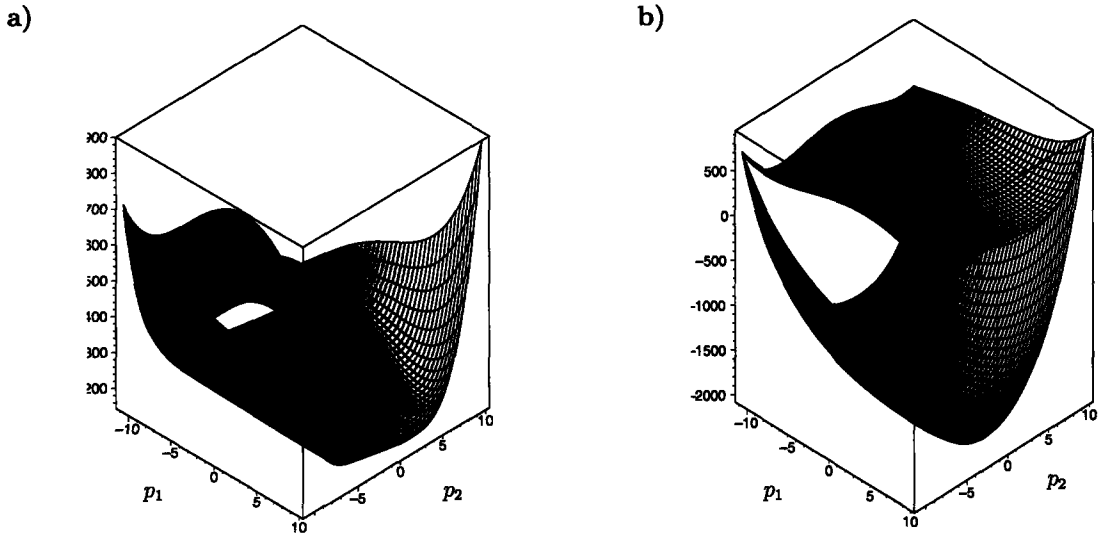
Problem 5.6.

$$\min_{\mathbf{p}} \; L(\mathbf{p}) = \int_0^3 \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2 \; dt$$

subject to

$$\dot{x}_1 = 0.1x_1 + 0.2x_2 + 0.1p_1$$
$$\dot{x}_2 = 0.15x_1 - 0.12x_2 + 0.2p_2$$
$$x_1(0) = 0, \; x_2(0) = 0$$
$$\mathbf{p} \in [-11, 10] \times [-11, 10].$$

Figure 5-3: Dynamic extension to the Himmelblau function at the root node: **a)** Objective function and McCormick underestimator. **b)** Objective function and αBB underestimator.



I again employ the methods thus far detailed to apply McCormick's Euclidean convex relaxation techniques to construct a convex underestimator for the objective function of Problem 5.6. I begin by identifying the composition that defines the integrand:

$$\ell = z_1^2 + z_2^2$$
$$z_1 = x_1^2 + x_2 - 11$$
$$z_2 = x_1 + x_2^2 - 7.$$

Because the sum of convex functions remains convex, the construction proceeds by

simultaneously constructing convex underestimators for both z_1 and z_2 on $\mathcal{X}(\underline{t})$. The next step in the relaxation process is to find convex underestimators and concave overestimators for z_1 and z_2. Fortuitously, both z_1 and z_2 are themselves convex; therefore, my work reduces merely to finding concave overestimators. For each z_i, $(i = 1, 2)$, the function is divided into a quadratic term and an affine term. The affine term is itself concave, and the quadratic term is overestimated by the secant connecting the endpoints. The sum of the affine term and the secant thus provides a concave overestimator for each z_i. The concave and convex underestimators for z_1 and z_2 on $\mathcal{X}(\underline{t})$ are given by

$$\begin{aligned}
c_{z_1}[\mathbf{x}(\underline{t})] &= x_1^2(\underline{t}) + x_2(\underline{t}) - 11 \\
c_{z_2}[\mathbf{x}(\underline{t})] &= x_1(\underline{t}) + x_2(\underline{t})^2 - 7 \\
C_{z_1}[\mathbf{x}(\underline{t})] &= [x_1^L(\underline{t}) + x_1^U(\underline{t})]x_1(\underline{t}) - x_1^L(\underline{t}) \cdot x_1^U(\underline{t}) + x_2(\underline{t}) - 11 \\
C_{z_2}[\mathbf{x}(\underline{t})] &= [x_2^L(\underline{t}) + x_2^U(\underline{t})]x_2(\underline{t}) - x_2^L(\underline{t}) \cdot x_2^U(\underline{t}) + x_1(\underline{t}) - 7,
\end{aligned}$$

where the applicability of each relaxation is restricted to $\underline{t} \in [0, 3]$. Utilizing the implied state bounds, I now construct supersets of the ranges of z_1 and z_2 via the natural interval extensions to yield the two range sets $Z_1 = [z_1^L(\underline{t}), z_1^U(\underline{t})]$ and $Z_2 = [z_2^L(\underline{t}), z_2^U(\underline{t})]$. As in Problem 5.5, I write functions describing $z_{i,min}$ for all $\underline{t} \in [0, 3]$:

$$z_{1,min}(\underline{t}) = \begin{cases} z_1^L(\underline{t}) & \text{if } z_1^L(\underline{t}) > 0 \\ z_1^U(\underline{t}) & \text{if } z_1^U(\underline{t}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$z_{2,min}(\underline{t}) = \begin{cases} z_2^L(\underline{t}) & \text{if } z_2^L(\underline{t}) > 0 \\ z_2^U(\underline{t}) & \text{if } z_2^U(\underline{t}) < 0 \\ 0 & \text{otherwise.} \end{cases}$$

Because z_i^2 is the convex envelope of $z_i \; \forall \; z_i \in [z_i^L(\underline{t}), z_i^U(\underline{t})]$ and $\forall \; \underline{t} \in [0, 3]$, using McCormick's convex relaxation technique [65], I have the following convex underes-

timator for the integrand:

$$u(\underline{t}, \mathbf{p}) = \left[\text{mid}\ \{c_{z_1}(\mathbf{x}(\underline{t}, \mathbf{p})), C_{z_1}(\mathbf{x}(\underline{t}, \mathbf{p})), z_{1,min}(\underline{t})\}\right]^2 + \left[\text{mid}\ \{c_{z_2}(\mathbf{x}(\underline{t}, \mathbf{p})), C_{z_2}(\mathbf{x}(\underline{t}, \mathbf{p})), z_{2,min}(\underline{t})\}\right]^2.$$

Finally, by Theorem 4.12,

$$U(\mathbf{p}) = \int_0^3 u(t, \mathbf{p})\ dt$$

is a convex underestimator for the objective function of Problem 5.6.

McCormick's relaxation technique and αBB were both used to solve Problem 5.6 globally; the results of these calculations are found in Table 5.3. Again, the tighter of the two relaxation techniques, McCormick's method, solves Problem 5.6 with fewer nodes. However, the difference in the required number of nodes between αBB and McCormick's method is not as extreme for this problem as for Problem 5.5. This result is expected because the αBB underestimator is somewhat tighter for this problem relative to the tightness of the αBB underestimator for Problem 5.5. Additionally, for this problem, αBB has a lower cost per node than McCormick's method. This phenomena arises because the inherent smoothness of the αBB objective function causes less events and hence less integration steps for each function call.

Table 5.3: Numerical results for Problem 5.6

Relaxation	Nodes	Problem	Int. calls	$\mathbf{p}_{min}$	Obj. fcn.	Time[a]	Time[b]
McCormick	29	lower	195	(-11.0, 8.85)	220.7	4.5	11.1
		upper	113				
αBB	53	lower	373	(-11.0, 8.85)	220.7	6.9	6.9
		upper	190				

[a]CPU s using bisection for event detection.
[b]CPU s using fully rigorous event detection.

5.3.4 Dynamic Extension to the Six-hump Camelback Problem

The following problem is a dynamic extension to the six-hump camelback function. In particular, this problem was chosen to exhibit the handling of an objective function with a bilinear term. Additionally, the embedded linear dynamic system was chosen to illustrate bounds with multiple events. Pictures of the objective function and underestimators at the root node are found in Figure 5-4. The implied state bounds with a candidate optimal state trajectory are found in Figures 5-5 and 5-6.

Problem 5.7.

$$\min_{\mathbf{p}} \; L(\mathbf{p}) = \int_0^2 4x_1^2 - 2.1x_1^4 + x_1^6/3 + x_1x_2 - 4x_2^2 + 4x_2^4 \; dt$$

subject to

$$\begin{aligned} \dot{x}_1 &= x_1 + 10.9x_2 + p_2 \\ \dot{x}_2 &= -10x_1 - 5x_2 - 3p_2 \\ x_1(0) &= p_1/4, \; x_2(0) = 0 \\ \mathbf{p} &\in [0,5] \times [-7, 5.5]. \end{aligned}$$

The integrand for the objective function of the dynamic camelback problem consists of three types of terms: univariate convex ($4x_1^2$, $x_1^6/3$, $4x_2^4$), univariate concave ($-2.1x_1^4$, $-4x_2^2$), and bilinear (x_1x_2). Because the sum of convex functions is also convex, a convex relaxation for the integrand is obtained by individually convexifying the nonconvex terms. The convex envelope for $-2.1x_1^4$ on $\mathcal{X}(\underline{t})$ is

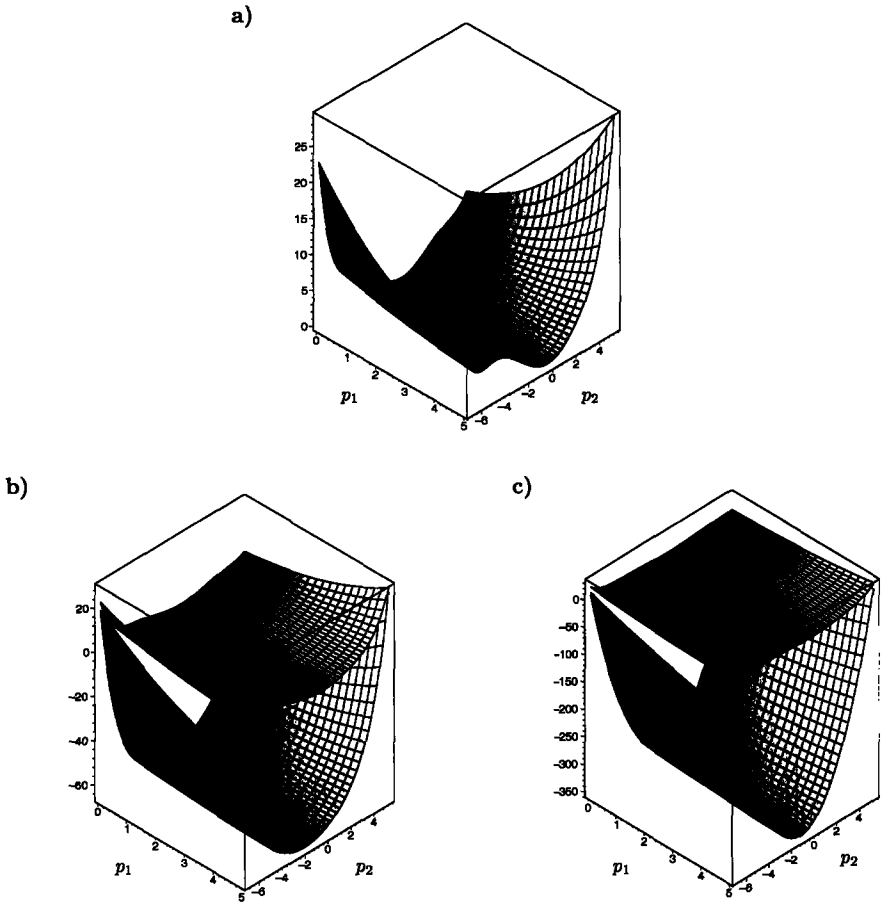
$$u_1(\underline{t}) = -2.1[x_1^L(\underline{t})]^4 - 2.1\left[[x_1^U(\underline{t})]^2 + [x_1^L(\underline{t})]^2\right][x_1^U(\underline{t}) + x_1^L(\underline{t})][x_1(\underline{t}) - x_1^L(\underline{t})],$$

and the convex envelope for $-4x_2^2$ on $\mathcal{X}(\underline{t})$ is

$$u_2(\underline{t}) = 4x_2^U(\underline{t})x_2^L(\underline{t}) - 4[x_2^U(\underline{t}) + x_2^L(\underline{t})]x_2(\underline{t}).$$

The convex underestimator x_1x_2 on $\mathcal{X}(\underline{t})$ is given by Equation 5.8. Therefore, by

Figure 5-4: Dynamic extension to the six-hump camelback function at the root node: **a)** Objective function. **b)** Objective function and derived underestimator. **c)** Objective function and αBB underestimator



Theorem 4.12,

$$U(\mathbf{p}) = \int_0^2 4x_1^2 + u_1 + x_1^6/3 + u_{bl} + u_2 + 4x_2^4 \, dt \tag{5.9}$$

is a convex underestimator for the objective function of Problem 5.7.

The solution to Problem 5.7 for both the underestimator in Equation 5.9 and the αBB underestimator is found in Table 5.4 below. Unlike the previous problems, αBB takes fewer nodes to solve this problem than the underestimator derived from special structure. At first, this result seems contradictory to the maxim that tighter underestimators require fewer branch-and-bound nodes. In fact, the reality is that

Figure 5-5: State bounds for Problem 5.7 with $x_1(t)$ for $p_1 = 2.5$ and $p_2 = -0.75$

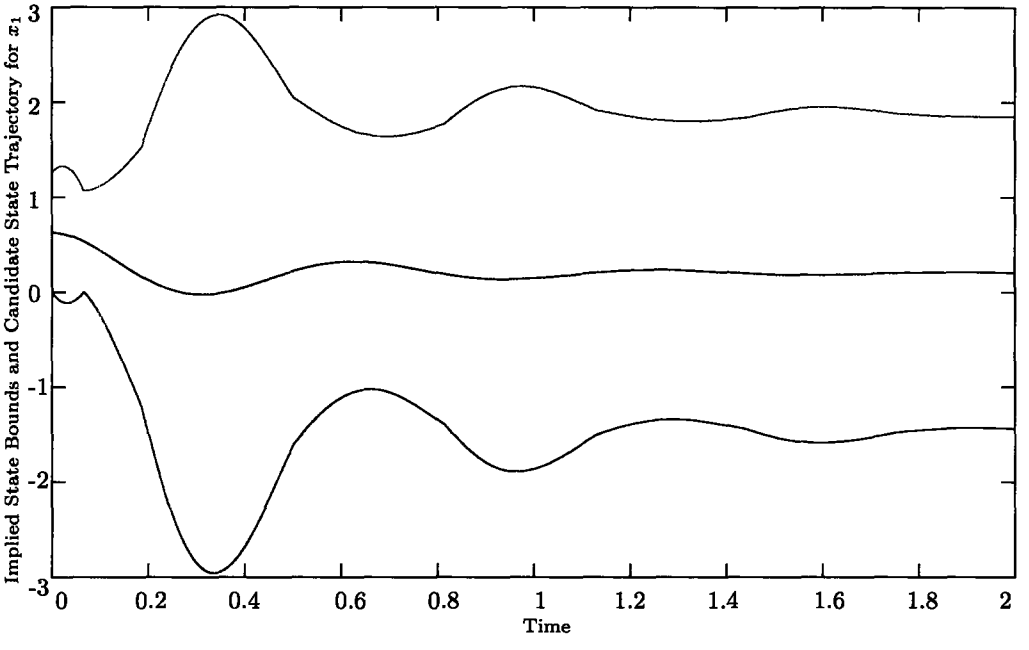


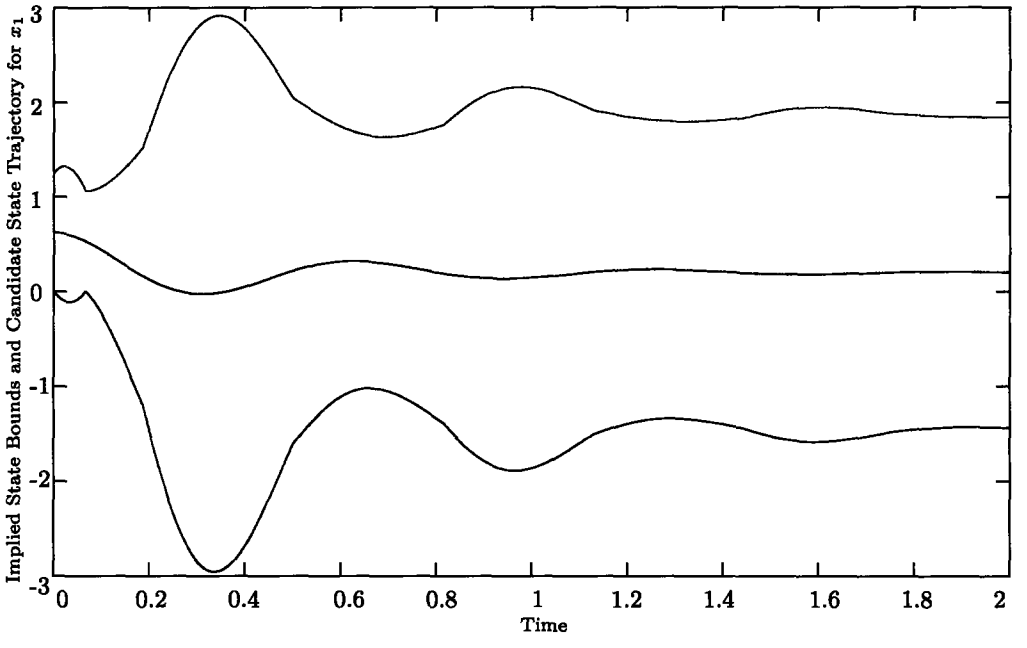
Figure 5-4 is somewhat misleading. While the underestimator derived from special structure is tighter than the αBB underestimator at the root node, as the parameter bounds shrink, the αBB underestimator actually becomes tighter than the special structure underestimator. However, one sees from Table 5.4 that despite requiring fewer nodes, using αBB requires more time than using the special structure. This phenomenon arises because the linear system in this problem exhibits many events. This increase in events in turn requires more integration steps, and the expense of calculating α values at each integration step via the scaled Gerschgorin technique becomes prohibitive.

Table 5.4: Numerical results for Problem 5.7

Relaxation	Nodes	Problem	Int. calls	$\mathbf{p}_{min}$	Obj. fcn.	Time[a]	Time[b]
special structure	131	lower	819	0.000	(0.00, 0.00)	44.3	71.4
		upper	457				
αBB	111	lower	756	0.000	(0.00, 0.000)	70.9	95.7
		upper	392				

[a]CPU (s) using bisection for event detection.
[b]CPU (s) using fully rigorous event detection.

Figure 5-6: State bounds for Problem 5.7 with $x_2(t)$ for $p_1 = 2.5$ and $p_2 = -0.75$



5.3.5 Optimization of a Generalized Twice-Differentiable Function

The following problem illustrates two general principles. First, using αBB as the relaxation technique, objective functions with arbitrarily complex, twice-differentiable integrands can be optimized with the algorithm. Second, the optimization algorithm presented here is applicable to linear time varying embedded dynamic systems as well as linear time invariant embedded dynamic systems. A picture of the objective function is found in Figure 5-7.

Problem 5.8.

$$\min_{\mathbf{p}} \; L(\mathbf{p}) = \int_0^1 \cos x_1 \sin x_2 + x_1/(x_2^2+1) \; dt$$

such that

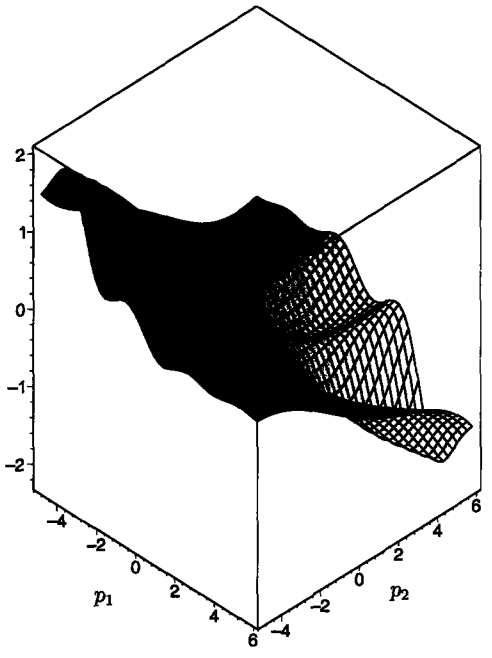
$$\dot{x}_1 = t x_1 + p_1$$

$$\dot{x}_2 = x_1 - x_2 + p_1 - p_2$$

$$x_1(0) = 0, \; x_2(0) = 0$$

$$\mathbf{p} \in [-5, 6] \times [-5, 6].$$

Figure 5-7: Objective function for Problem 5.8



While it may be theoretically possible to derive a McCormick underestimator for Problem 5.8, this would be a difficult challenge. For this reason, the problem is particularly suited for solution with an αBB underestimator, for this relaxation technique is implemented automatically for any twice-continuously differentiable integrand. The solution to Problem 5.8 is found in Table 5.5. The answer is graphically verifiable from Figure 5-7.

Table 5.5: Numerical results for Problem 5.8

Relaxation	Nodes	Problem	Int. calls	$\mathbf{p}_{min}$	Obj. fcn.	Time[a]	Time[b]
αBB	301	lower	1699	4.251	(6.00, -2.24)	41.3	41.3
		upper	800				

[a]CPU (s) using bisection for event detection.
[b]CPU (s) using fully rigorous event detection.

5.3.6 Dynamic Himmelblau Function Revisited

The objective of this section is to illustrate the use of the algorithm to solve an optimal control problem via control parameterization [18]. Because the emphasis of

this problem is not on the relaxation of the integrand, I have chosen to select an objective function for which this analysis has already been performed.

Problem 5.9.

$$\min_u \; L(u) = \int_0^1 \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2 \; dt$$

such that

$$\begin{aligned} \dot{x}_1 &= 0.1x_1 + x_2 + 8u \\ \dot{x}_2 &= x_1 - 0.1x_2 - 8u \\ x_1(0) &= 0, \; x_2(0) = 0 \\ -1 &\leq u \leq 1. \end{aligned}$$

To solve Problem 5.9, two different control parameterizations were selected: a piecewise linear control parameterization enforcing continuity between each of the ten equally spaced time partitions and a piecewise constant control parameterization on ten equally spaced time partitions. Letting P_i represent each time partition, for the piecewise linear case, u is parameterized by the following equation:

$$u(t) = \sum_{i=1}^{10} 1_{P_i}(t) \left[(10t - i + 1)p_{i+1} + (i - 10t)p_i\right]$$

such that

$$p_i \in [-1, 1], \quad i = 1, \ldots, 11$$

where the characteristic function is defined by

$$1_{P_i}(t) = \begin{cases} 1 & \text{if } t \in P_i \\ 0 & \text{if } t \notin P_i. \end{cases}$$

Defining $u(t)$ for the piecewise constant case is obvious. In the parameter space, multistart local optimization has shown the objective function to exhibit at least two local minima.

Problem 5.9 was solved using McCormick's underestimator as defined in the analysis of Problem 5.6. In order to facilitate convergence of this problem, optimality

based range reduction tests 1 and 2 from Sahinidis and Ryoo [78] were applied at each node. Range reduction at a given node was repeated if the reduction of the domain for any variable met or exceeded 25%. The numerical results are found in Table 5.6 below.

Table 5.6: Numerical results for Problem 5.9

Parameterization	Nodes	Problem	Int. calls	Obj. fcn.	Time[a]
piecewise constant	3311	lower	46320	43.8	19124
		upper	21015		
piecewise linear	10045	lower	122424	43.8	39516
		upper	68887		

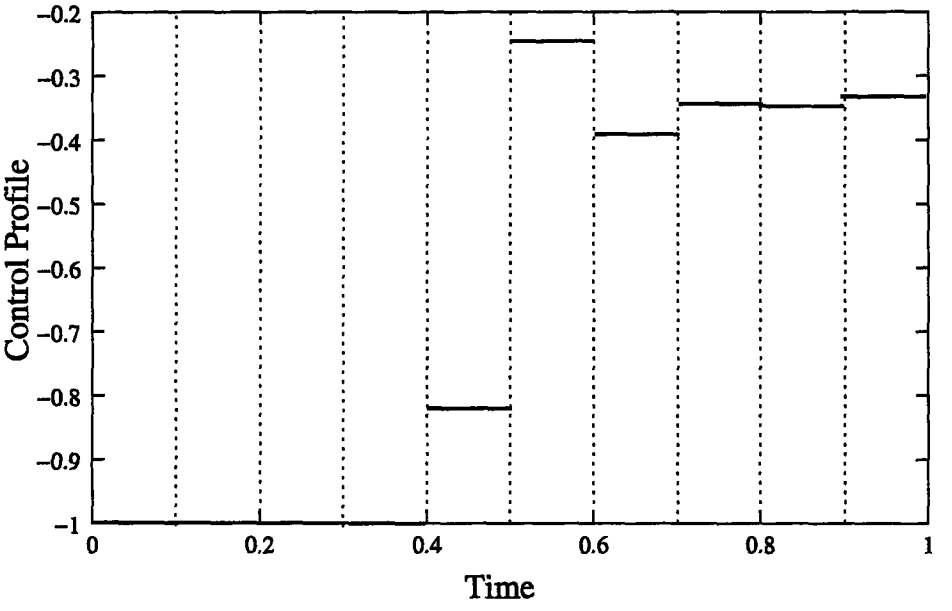
[a]CPU (s) using bisection for event detection.

The control profiles for the piecewise constant and piecewise linear case are illustrated in Figures 5-8 and 5-9 respectively. The problem was solved to a relative tolerance of 0.01. The large number of nodes required to solve this problem derives from the relative insensitivity of of the objective function to the control function in the vicinity of the minimum, a common feature of optimal control problems. The relatively high cost per node is due to each function call requiring ten separate integrations, one per time partition.

5.3.7 Scaling of the Algorithm

As previously mentioned, the scaling of the algorithm is a function of both the number of parameters and the number of states. As any optimization solver employing branch-and-bound, the algorithm scales exponentially with the number of parameters. The number of states simply affects the time required to compute an individual function call; however, high quality integration codes yield polynomial scaling for increasing numbers of states. The following problem is designed to illustrate the scaling of the algorithm with the simultaneous increase of both the number of parameters and the number of states. For this problem, post-processing optimality based reduce heuristics [78] were employed to accelerate convergence of the branch-and-bound algorithm; the problems were solved to 0.001 relative tolerance.

Figure 5-8: Piecewise constant control profile for Problem 5.9



Problem 5.10.

$$\min_{\mathbf{p}\in[-7,6]^{n_p}} L(\mathbf{p}) = \int_0^1 \left(x_{n_p-1}^2 + x_{n_p} - 11\right)^2 + \left(x_{n_p-1} + x_{n_p}^2 - 7\right)^2 \, dt$$

such that

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{p}$$
$$\mathbf{x}(0) = \mathbf{0}$$

where $\mathbf{A}$ is an $n_p \times n_p$ matrix given by

$$\mathbf{A} = \begin{bmatrix} -1 & 0 & 0 & \cdots \\ 1 & -1 & 0 & \cdots \\ 0 & 1 & -1 & \cdots \\ \vdots & & & \ddots \end{bmatrix}$$

and $\mathbf{B}$ is an $n_p \times n_p$ matrix given by $\mathbf{B} = \mathbf{I}$.

The results from solving Problem 5.10 are found in Table 5.7 below.

Figure 5-9: Piecewise linear control profile for Problem 5.9

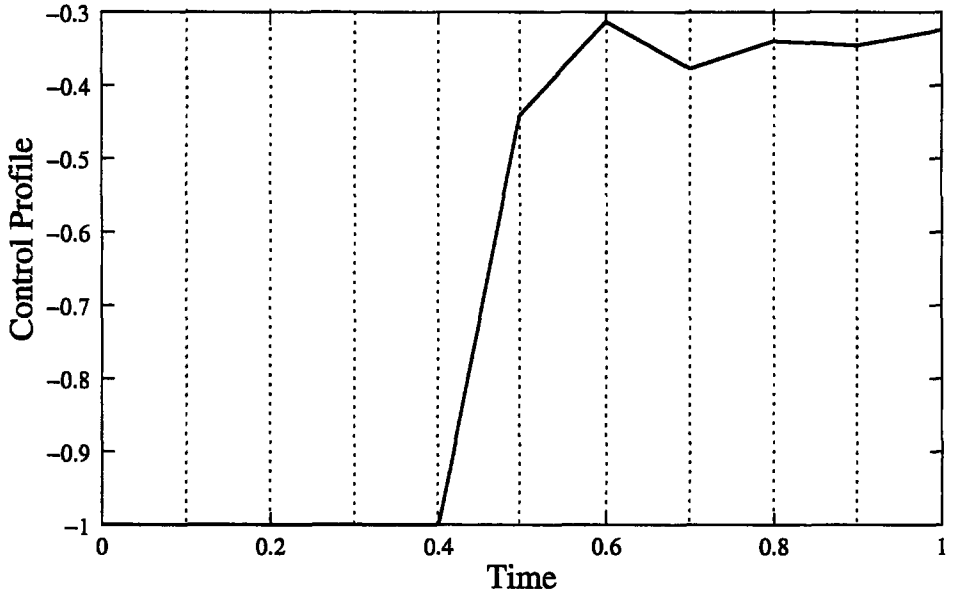


Table 5.7: Numerical results for Problem 5.10

n_p	Nodes	Obj. fcn.	Time[a]
2	27	58.3	3.4
3	47	56.7	23.4
4	53	55.3	58.4
5	51	54.8	125.3
6	63	54.7	278.6
7	517	54.7	2329.9
7[b]	161	54.7	1177.4

[a]CPU (s) using bisection for event detection.
[b]Root node probing employed to accelerate convergence

Chapter 6

Relaxation Theory for Problems with Nonlinear Dynamics Embedded

This chapter begins a sequence of chapters concerning the solution of Problem 2.4 with parameter dependent nonlinear dynamic systems embedded. As with the study of Problem 4.1, three steps exist for solving Problems with nonlinear dynamics: relaxing the integrand with a composition technique, generating state bounds, and proving convergence of the branch-and-bound algorithm. As will be demonstrated shortly, the theory for solving the nonlinear problem is substantially more complicated. The mathematical problem formulation is stated below.

Problem 6.1.

$$\min_{\mathbf{p}} J(\mathbf{p}) = \phi(\mathbf{x}(t_f, \mathbf{p}), \mathbf{p}) + \int_{t_0}^{t_f} \ell(t, \mathbf{x}(t, \mathbf{p}), \mathbf{p})\, dt$$

subject to

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(t, \mathbf{x}, \mathbf{p}) \\ \mathbf{x}(t_0, \mathbf{p}) &= \mathbf{x}_0(\mathbf{p}) \end{aligned} \tag{6.1}$$

where $\mathbf{p} \in P$; ϕ is a continuous mapping $\phi : X \times P \to \mathbb{R}$; ℓ is a Lebesgue integrable mapping $\ell : (t_0, t_f] \times X \times P \to \mathbb{R}$; f_i is a Lipschitz continuous mapping $f_i : (t_0, t_f] \times$

$X \times P \rightarrow \mathbb{R}$, $i = 1, \ldots, n_x$; x_{i0} is a continuous mapping $x_{i0} : P \rightarrow \mathbb{R}$, $i = 1, \ldots, n_x$.

6.1 State Bounds for Nonquasimonotone Differential Equations

Analogously to solving the linear problem, state bounds are requisite for constructing a relaxation for the integrand in Problem 6.1. However, unlike bounding the solution of a linear dynamic system, no generalities can be stated concerning the structure of the solution of a system of nonlinear ordinary differential equations. Thus, the bounding strategy cannot be based on an interval enclosure of the solution of the embedded differential equations. Instead, the majority of research concerning the bounding of differential equations is based upon the study of differential inequalities. While the roots of differential inequalities may be traced to the early part of the 20th century, the book by Walter [93] provides an excellent overview of the salient concepts. One important application of differential inequalities is computing time varying lower and upper bounds on the solution of systems of ODEs. Essentially, the analysis demonstrates that solving differential equations whose right hand sides underestimate the right hand sides of the original system at all points including the initial condition provides rigorous lower bounds for the original state variables. An analogous result applies to constructing upper bounds. Harrison [45] extended these results from Walter to parameter embedded ODEs. Unfortunately, except under stringent conditions of quasimonotonicity (quasimonotonicity is defined later in this chapter), the bounds generated via differential inequalities are typically weak and often suffer from bounds explosion, a situation in which the solution of the bounding differential equations tends rapidly toward infinity rendering the resulting bounds practically useless. The term "the wrapping effect" has been coined to describe this bounds explosion. A vast literature exists on fighting the wrapping effect as it relates to validated solutions of differential equations. In this section, however, I focus on a technique that employs engineering understanding of problems to generate convergent

state bounds for solving Problem 6.1.

As previously mentioned, the classical theory of differential inequalities provides a method for computing time varying bounds on the solution of a system of differential equations. In particular, I am interested in computing a superset of the image of the solution of a system of ODEs on a parameter set. The traditional techniques such as Harrison's technique depend exclusively on information obtained from the mathematical statement of the differential equations and their corresponding initial conditions. However, differential equations of practical interest to scientists and engineers are derived from physical systems for which more information is known about the behavior of the system dynamics than the information embodied by the differential equations alone. For example, for a system undergoing decay, the state never exceeds its initial condition. In another example, differential equations modeling mass and heat transfer obey conservation principles. In this section, I demonstrate how these natural bounds and differential invariants may be applied in conjunction with differential inequalities to produce tight bounds on the solution of ODEs. I make one very important note before beginning this discussion which is that this technique is not applicable to bounds imposed upon the system such as path constraints or bounds on a control variable. The technique is only applicable given additional information that naturally governs the evolution of the solution of the ODEs.

I begin by presenting a variant of a fundamental Lemma from Walter [93]:

Lemma 6.2. *Suppose the vector functions $\boldsymbol{\varphi}(t)$ and $\boldsymbol{\psi}(t)$ are differentiable a.e. in $(t_0, t_f]$. For some index i and fixed $\underline{t} \in (t_0, t_f]$, if $\dot{\varphi}_i(\underline{t}) \leq \dot{\psi}_i(\underline{t})$ when $\boldsymbol{\varphi}(\underline{t}) \leq \boldsymbol{\psi}(\underline{t})$, $\varphi_i(\underline{t}) = \psi_i(\underline{t})$ then we have precisely one of the following two cases:*

(i) $\boldsymbol{\varphi} \leq \boldsymbol{\psi}$ *in* $(t_0, t_f]$

(ii) $\boldsymbol{\varphi}(t_{0+}) \leq \boldsymbol{\psi}(t_{0+})$ *does not hold, i.e., there exists an arbitrary small* $\bar{t} \in (t_0, t_f]$ *such that* $\varphi_i(\bar{t}) > \psi_i(\bar{t})$ *for at least one index* i.

Proof. The proof is divided into two cases. In the first case, I assume the comparison on the derivative is strict (*i.e.* $\dot{\varphi}_i(\underline{t}) < \dot{\psi}_i(\underline{t})$). Assume that neither (i) nor (ii) holds.

From (ii), this implies that $\varphi(t_{0+}) \leq \psi(t_{0+})$, and consequently there exists some $\bar{t} \in (t_0, t_f]$ such that $\varphi \leq \psi$ for $t_0 < t < \bar{t}$. Because I have also assumed the contrary of Hypothesis (i), it must follow that there exists some point $\underline{t} \in (t_0, t_f]$, $\underline{t} \geq \bar{t}$ such that $\varphi(\underline{t}) \leq \psi(\underline{t})$ and $\varphi_i(\underline{t}) = \psi_i(\underline{t})$ for at least one index i. Therefore, for this i, the following inequality holds:

$$\frac{\varphi_i(t) - \varphi_i(\underline{t})}{t - \underline{t}} \geq \frac{\psi_i(t) - \psi_i(\underline{t})}{t - \underline{t}} \qquad t_0 < t < \underline{t}$$

(the denominator is negative). By passing to the limit $t \rightarrow \underline{t}$, I have $\dot{\varphi}_i \geq \dot{\psi}_i$ at $\underline{t}$, which contradicts the assumption of the first case. I now address the second case for which I assume that equality holds in the derivative comparison (*i.e.* $\dot{\varphi}_i(\underline{t}) = \dot{\psi}_i(\underline{t})$). Before beginning the argument, I remind the reader that φ and ψ are uniquely determined by the solution of their respective differential equations, and I need not consider the possibility of these functions being multiply defined for a given $t \in (t_0, t_f]$. Now, consider the existence of some first point $t_0^* \in (t_0, t_f]$ to the right or left of $\underline{t}$. If $\varphi > \psi$ or $\varphi_i \neq \psi_i$ at t_0^* then I need not consider t_0^* further for the hypotheses of the lemma no longer hold. Therefore, I assume $\varphi \leq \psi$ and $\varphi_i = \psi_i$ at t_0^*. If $\dot{\varphi}_i > \dot{\psi}_i$, then the hypotheses of the lemma again do not hold and I need not consider t_0^* further. If $\dot{\varphi}_i < \dot{\psi}_i$, then the first case holds, and I have already proven the lemma for this case. If $\dot{\varphi}_i = \dot{\psi}_i$ at t_0^*, then another $t_1^* \in (t_0, t_f]$ must exist immediately to the right or left of t_0^*. The procedure is repeated until a $t^* \in (t_0, t_f]$ for which $\dot{\varphi}_i < \dot{\psi}_i$ is found. If no such point is obtainable in $(t_0, t_f]$ where the hypotheses of the lemma hold with $\dot{\varphi}_i = \dot{\psi}_i$, then (i) holds trivially with equality for index i. □

Remark 6.3. Clearly, a symmetric result for $\varphi \geq \psi$ holds.

I wish to extend Lemma 6.2 to a method for bounding the solution of Equation 6.1 given prior knowledge of a set $\overline{\mathbf{X}}(t, \mathbf{p})$ that is known independently to contain the solution of Equation 6.1. The derivation is performed in several steps. First, I present an extension of Walter's fundamental result concerning the construction of superfunctions and subfunctions given the set $\overline{\mathbf{X}}(t, \mathbf{p})$. From this fundamental theorem, two corollaries are constructed. The first corollary extends the result to

differential equations with embedded parameters and yields a parameter dependent solution for which the inequality holds pointwise in the parameter. A second corollary follows that illustrates a method for constructing a solution valid for all parameter values in a given set. This second corollary yields the desired result which bounds the image of the solution to a parameter embedded ODE on a subset of a Euclidean space.

Theorem 6.4. *Let* $\mathbf{x}(t)$ *be a solution of*

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{6.2}$$

in $(t_0, t_f]$ *and further suppose that* $\mathbf{x}(t) \in \overline{\mathcal{X}}(t)$, *where* $\overline{\mathcal{X}}(t)$ *is known independently from the solution of Equation 6.2. For* $i = 1, \ldots, n_x$, *if*

(i) $v_i(t_0) \leq x_i(t_0) \leq w_i(t_0)$

and if $\forall\ \mathbf{v}(t), \mathbf{w}(t) \in G(t)$

(ii) $\dot{v}_i = g_i(t, \mathbf{v}, \mathbf{w}) \leq \inf_{\substack{\mathbf{z} \in \overline{\mathcal{X}}(t) \cap G(t) \\ z_i = v_i(t)}} f_i(t, \mathbf{z})$

(iii) $\dot{w}_i = h_i(t, \mathbf{v}, \mathbf{w}) \geq \sup_{\substack{\mathbf{z} \in \overline{\mathcal{X}}(t) \cap G(t) \\ z_i = w_i(t)}} f_i(t, \mathbf{z})$

where $G(t) = \{\mathbf{z} \mid \mathbf{v}(t) \leq \mathbf{z} \leq \mathbf{w}(t)\}$, *then*

$$\mathbf{v}(t) \leq \mathbf{x}(t) \leq \mathbf{w}(t) \quad \forall\ t \in (t_0, t_f].$$

Proof. Either equality holds throughout and the theorem is trivially satisfied or Hypothesis (i), (ii), and (iii) guarantee that if there exists some $\underline{t} \in (t_0, t_f]$ such that $\mathbf{v}(\underline{t}) \leq \mathbf{x}(\underline{t}) \leq \mathbf{w}(\underline{t})$, $v_i(\underline{t}) = x_i(\underline{t}) = w_i(\underline{t})$ for some index i, I have

$$\begin{aligned}
\dot{v}_i &= g_i(\underline{t}, \mathbf{v}, \mathbf{w}) \leq \inf_{\substack{\mathbf{z} \in \overline{\mathcal{X}}(t) \cap G(t) \\ z_i = v_i(t)}} f_i(\underline{t}, \mathbf{z}) \leq f_i(\underline{t}, \mathbf{x}) = \dot{x}_i \\
\dot{w}_i &= h_i(\underline{t}, \mathbf{v}, \mathbf{w}) \geq \sup_{\substack{\mathbf{z} \in \overline{\mathcal{X}}(t) \cap G(t) \\ z_i = w_i(t)}} f_i(\underline{t}, \mathbf{z}) \geq f_i(\underline{t}, \mathbf{x}) = \dot{x}_i,
\end{aligned}$$

where the last inequality in each equation above holds because I have already established $\mathbf{x}(\underline{t}) \in G(\underline{t})$, and $\mathbf{x}(\underline{t}) \in \overline{\mathcal{X}}(\underline{t})$ by hypothesis. I have now established the hypothesis of Lemma 6.2 and therefore know that either 6.2.(i) or 6.2.(ii) holds. However, (i) is inconsistent with 6.2.(ii). □

The following corollary extends Theorem 6.4 to pointwise in $\mathbf{p}$ bounds for Equation 6.1. While not needed for bounding the solution of a parameter dependent differential equation, Corollary 6.5 is needed directly for proving the results on convex bounding in the subsequent section.

Corollary 6.5. *Let $\mathbf{x}(t, \mathbf{p})$ be a solution of Equation 6.1 and let $\mathbf{x}(t, \mathbf{p}) \in \overline{\mathcal{X}}(t, \mathbf{p})$ for each $\mathbf{p} \in P$, where $\overline{\mathcal{X}}(t, \mathbf{p})$ is known independently from the solution of Equation 6.1. If for $i = 1, \ldots, n_x$ and for each $\mathbf{p} \in P$*

(i) $v_i(t_0, \mathbf{p}) \leq x_i(t_0, \mathbf{p}) \leq w_i(t_0, \mathbf{p})$

and if $\forall\ \mathbf{v}(t, \mathbf{p}), \mathbf{w}(t, \mathbf{p}) \in G(t, \mathbf{p})$

(ii) $$\dot{v}_i = g_i(t, \mathbf{v}, \mathbf{w}, \mathbf{p}) \leq \inf_{\substack{\mathbf{z} \in \overline{\mathcal{X}}(t,\mathbf{p}) \cap G(t,\mathbf{p}) \\ z_i = v_i(t,\mathbf{p})}} f_i(t, \mathbf{z}, \mathbf{p})$$

(iii) $$\dot{w}_i = h_i(t, \mathbf{v}, \mathbf{w}, \mathbf{p}) \geq \sup_{\substack{\mathbf{z} \in \overline{\mathcal{X}}(t,\mathbf{p}) \cap G(t,\mathbf{p}) \\ z_i = w_i(t,\mathbf{p})}} f_i(t, \mathbf{z}, \mathbf{p}),$$

where $G(t, \mathbf{p}) = \{\mathbf{z} \mid \mathbf{v}(t, \mathbf{p}) \leq \mathbf{z} \leq \mathbf{w}(t, \mathbf{p})\}$, then for each fixed $\mathbf{p} \in P$,

$$\mathbf{v}(t, \mathbf{p}) \leq \mathbf{x}(t, \mathbf{p}) \leq \mathbf{w}(t, \mathbf{p}) \qquad \forall\ t \in (t_0, t_f].$$

Proof. Theorem 6.4 holds for any fixed $\mathbf{p} \in P$. □

The following corollary is the fundamental result for bounding the image of a parameter embedded differential equation on the parameter set P. The resulting bounding equations are not necessarily exact. That is, the derived bounding set may actually be a superset of the true image of the solution on P.

Corollary 6.6. *Let* $\mathbf{x}(t,\mathbf{p})$ *be a solution of Equation 6.1 and let* $\mathbf{x}(t,\mathbf{p}) \in \overline{\mathcal{X}}(t,\mathbf{p})$ *for each* $\mathbf{p} \in P$*, where* $\overline{\mathcal{X}}(t,\mathbf{p})$ *is known independently from the solution of Equation 6.1. Furthermore, let* $\overline{\mathcal{X}}(\underline{t})$ *be defined pointwise in time by*

$$\overline{\mathcal{X}}(\underline{t}) = [\inf_{\mathbf{q}\in P} \overline{\mathcal{X}}(\underline{t},\mathbf{q}),\ \sup_{\mathbf{q}\in P} \overline{\mathcal{X}}(\underline{t},\mathbf{q})].$$

If for $i = 1,\ldots,n_x$,

(i) $v_i(t_0) \leq \inf_{\mathbf{q}\in P} x_i(t_0,\mathbf{q})$

(ii) $w_i(t_0) \geq \sup_{\mathbf{q}\in P} x_i(t_0,\mathbf{q})$

and if $\forall\ \mathbf{v}(t),\mathbf{w}(t) \in G(t)$

(iii) $\dot{v}_i = g_i(t,\mathbf{v},\mathbf{w}) \leq \inf_{\substack{\mathbf{z}\in\overline{\mathcal{X}}(t)\cap G(t),\ \mathbf{q}\in P \\ z_i = v_i(t)}} f_i(t,\mathbf{z},\mathbf{q})$

(iv) $\dot{w}_i = h_i(t,\mathbf{v},\mathbf{w}) \geq \sup_{\substack{\mathbf{z}\in\overline{\mathcal{X}}(t)\cap G(t),\ \mathbf{q}\in P \\ z_i = w_i(t)}} f_i(t,\mathbf{z},\mathbf{q}),$

then

$$\mathbf{v}(t) \leq \mathbf{x}(t,\mathbf{p}) \leq \mathbf{w}(t) \qquad \forall\ (t,\mathbf{p}) \in (t_0,t_f] \times P.$$

Proof. The proof is a trivial extension of the proof of Theorem 6.4. □

Remark 6.7. Because the functions $\mathbf{v}$ and $\mathbf{w}$ from the above corollary bound $\mathbf{x}(t,\mathbf{p})$ for all values of $\mathbf{p} \in P$, when used in the above context, these state bounds are labeled $\mathbf{x}^L$ and $\mathbf{x}^U$, respectively. Pointwise in time, the interval $[\mathbf{x}^L(\underline{t}),\mathbf{x}^U(\underline{t})]$ is labeled as $\mathcal{X}(\underline{t})$ for each fixed $\underline{t} \in (t_0,t_f]$.

From inspection of Corollary 6.6, the most difficult aspect of applying the corollary appears to be bounding the solutions of the parametric optimization problems defining the right hand sides of the bounding differential equations. While computing the exact solution to the optimization problem would yield the tightest bounds possible from this method, actually solving the optimization problems at each integration step in a numerical integration would be a prohibitively expensive task. Instead, the solution of the optimization problem on the right hand side of the differential equation is

bounded by interval arithmetic [68] pointwise in time. To apply interval arithmetic, the set $\overline{X}(\underline{t}) \cap G(\underline{t})$ must be representable as an interval pointwise in time. For some fixed $\underline{t} \in (t_0, t_f]$, let $\overline{X}(\underline{t}) = [\overline{\mathbf{x}}^L(\underline{t}), \overline{\mathbf{x}}^U(\underline{t})]$, then, pointwise in time, the set $\overline{X}(\underline{t}) \cap G(\underline{t})$ is given by

$$[\max\{\overline{x}_1^L(\underline{t}), x_1^L(\underline{t})\}, \min\{\overline{x}_1^U(\underline{t}), x_1^U(\underline{t})] \times \cdots \times [\max\{\overline{x}_{n_x}^L(\underline{t}), x_{n_x}^L(\underline{t})\}, \min\{\overline{x}_{n_x}^U(\underline{t}), x_{n_x}^U(\underline{t})].$$

I note that given inclusion monotonic interval functions [68] for the right hand sides of the differential equations, the bounds approach the original function when the parameter space approaches degeneracy. Therefore, satisfying the right hand side of the bounding differential inequalities is a relatively straightforward task.

Another interesting aspect of formulating the bounding differential inequalities as constrained optimization problems is that these optimization problems may have no feasible point. Assume that such an infeasibility occurs in the bounding problem for the ith variable. Such infeasibilities usually arise from the equality constraint on the ith variable. This situation immediately implies that the ith bound lies outside the set $\overline{X}(t)$. In this case, any finite value for the right hand side of the differential equation is valid, for any finite instantaneous rate of change in the ith bounding variable ensures that the bound remains outside $\overline{X}(t)$. In practice, when employing interval techniques, the interval computation provides a finite value for the right hand side of the differential equation regardless of the feasibility of the optimization problem.

In some special cases, the optimization problems of Corollary 6.6 are trivially satisfied by the structure of the right hand sides of Equation 6.1. In particular, vast simplifications can be applied under certain monotonicity assumptions. The concepts of monotonicity and quasimonotonicity are now defined (see also Walter [93]).

Definition 6.8. An n-vector function $\mathbf{f}(t, \mathbf{z})$ is monotone increasing in $\mathbf{z}$ if for $i = 1, \ldots, n$, $f_i(t, \mathbf{z})$ is monotone increasing in each of the variables z_j with z_k $(k \neq j)$ fixed for all z_k allowable in the domain of definition of $\mathbf{f}$. A vector function $\mathbf{f}(t, \mathbf{z})$ is quasimonotone increasing in $\mathbf{z}$ if for $i = 1, \ldots, n$, $f_i(t, \mathbf{z})$ is monotone increasing

in each of the variables z_j $(j \neq i)$ with z_k $(k \neq j)$ fixed for all z_k allowable in the domain of definition of $\mathbf{f}$. Analogous definitions hold for monotone decreasing and quasimonotone decreasing.

If the function f_i in Corollary 6.5 is monotone in several variables, then the hypotheses 6.5.(ii) and 6.5.(iii) can be greatly sharpened because this additional information simplifies the optimization of f_i. Let $\overline{\mathbf{x}}^L(t,\mathbf{p})$ be a lower bounding trajectory of the set $\overline{\mathcal{X}}(t,\mathbf{p})$, and let $\overline{\mathbf{x}}^U(t,\mathbf{p})$ be analogously defined. Furthermore, assume that f_i is monotone increasing [monotone decreasing] in z_j $(j \neq i)$ (note that monotonicity in $\mathbf{p}$ is not required for fixed $\mathbf{p}$). Then from Corollary 6.5, Hypothesis (ii), I know that with respect to the z_j direction $(j \neq i)$, the minimum of f_i over $\overline{\mathcal{X}}(t,\mathbf{p}) \cap G(t,\mathbf{p})$ occurs at $z_j = \max\{v_j(t,\mathbf{p}), \overline{x}_j^L(t,\mathbf{p})\}$ $[z_j = \min\{w_j(t,\mathbf{p}), \overline{x}_j^U(t,\mathbf{p})\}]$. Similarly, for Corollary 6.5, Hypothesis (ii), I know that with respect to the z_j direction $(j \neq i)$, the maximum of f_i over $\overline{\mathcal{X}}(t,\mathbf{p}) \cap G(t,\mathbf{p})$ occurs at $z_j = \min\{w_j(t,\mathbf{p}), \overline{x}_j^U(t,\mathbf{p})\}$ $[z_j = \max\{v_j(t,\mathbf{p}), \overline{x}_j^L(t,\mathbf{p})\}]$. If the function $\mathbf{f}$ is quasimonotone increasing or quasimonotone decreasing in all variables $z_j, j \neq i$, the above results reduce to the following theorem.

Theorem 6.9. *Let $\overline{\mathcal{X}}(t,\mathbf{p})$ be defined as in Corollary 6.5 (where if $\overline{\mathcal{X}}(t,\mathbf{p})$ is not representable as an interval, we assume an interval superset) and let*

$$v_j^{max} = \max\{v_j, \overline{x}_j^L\} \quad \textit{and} \quad w_j^{min} = \min\{w_j, \overline{x}_j^U\}.$$

If the function $\mathbf{f}(t,\mathbf{z},\mathbf{p})$ is quasimonotone increasing in $\mathbf{z}$ and if for $i = 1,\ldots,n_x$

(i) $\mathbf{v}(t_0,\mathbf{p}) \leq \mathbf{x}(t_0,\mathbf{p}) \leq \mathbf{w}(t_0,\mathbf{p})$

(ii) $\dot{v}_i = g_i(t,\mathbf{v},\mathbf{p}) \leq f_i(t,v_i,v_{j\neq i}^{max},\mathbf{p})$ *in* $(t_0,t_f]$

(iii) $\dot{w}_i = h_i(t,\mathbf{w},\mathbf{p}) \geq f_i(t,w_i,w_{j\neq i}^{min},\mathbf{p})$ *in* $(t_0,t_f]$

then for each fixed $\mathbf{p} \in P$,

$$\mathbf{v}(t,\mathbf{p}) \leq \mathbf{x}(t,\mathbf{p}) \leq \mathbf{w}(t,\mathbf{p}) \quad \textit{in} \quad (t_0,t_f].$$

Proof. The monotonicity assumption on f_i implies that $\inf_{\substack{\mathbf{z}\in\overline{X}(t,\mathbf{p})\cap G(t,\mathbf{p}) \\ z_i=v_i(t,\mathbf{p})}} f_i(t,\mathbf{z},\mathbf{p})$ is attained at $\mathbf{v}=\left(v_i, v_{j\neq i}^{max}\right)$; g_i is less than this infimum by construction. An analogous result holds for h_i and the supremum of f_i. The result immediately follows from an application of Corollary 6.5. □

Remark 6.10. An analogous result holds for a quasimonotone decreasing function. Similar simplifications also hold for Corollary 6.6 under quasimonotonicity assumptions.

The theorem concerning bounding quasimonotone differential equations concludes the theoretical presentation of deriving state bounds for Problem 6.1. A fully worked example is now presented to illustrate the application of the state bounding results. In particular, construction of state bounds for a nonquasimonotone system via Corollary 6.6 is emphasized.

Example 6.11. Consider the following bimolecular, reversible chemical reaction:

$$A + B \rightleftharpoons C$$

with forward rate constant k_f and reverse rate constant k_r. Given a bounding set $[k_f^L, k_f^U] \times [k_r^L, k_r^U]$ for the rate constants, I wish to compute bounds for the concentrations of the three chemical species over a specified time interval given the following differential equations describing the evolution of the species concentrations:

$$\dot{x}_A = -k_f x_A x_B + k_r x_C$$
$$\dot{x}_B = -k_f x_A x_B + k_r x_C$$
$$\dot{x}_C = k_f x_A x_B - k_r x_C$$

$$x_A(0) = x_{A0},\ x_B(0) = x_{B0},\ x_C(0) = x_{C0}.$$

Without loss of generality, I will assume that $x_{B0} \geq x_{A0}$.

In order to compute state bounds for Example 6.11, a tradeoff exists between the ease of computing bounds and the quality of the computed bounds. For this example,

several different approaches to applying Corollary 6.6 are demonstrated. The bounds are first derived without employing *a priori* information concerning the behavior of the states using Harrison's Theorem [45]. This analysis yields the following differential equations:

$$\begin{aligned}
\dot{x}_A^L &= -\max\{\min\{k_f^L x_A^L,\, k_f^U x_A^L\} x_B^L,\, \min\{k_f^L x_A^L,\, k_f^U x_A^L\} x_B^U,\, \max\{k_f^L x_A^L, \\
&\quad k_f^U x_A^L\} x_B^L,\, \max\{k_f^L x_A^L,\, k_f^U x_A^L\} x_B^U\} \\
&\quad + \min\{k_r^L x_C^L,\, k_r^L x_C^U,\, k_r^U x_C^L,\, k_r^U x_C^U\} && \text{(6.3a)} \\
\dot{x}_B^L &= -\max\{\min\{k_f^L x_A^L,\, k_f^L x_A^U,\, k_f^U x_A^L,\, k_f^U x_A^U\} x_B^L,\, \max\{k_f^L x_A^L,\, k_f^L x_A^U, \\
&\quad k_f^U x_A^L,\, k_f^U x_A^U\} x_B^L\} + \min\{k_r^L x_C^L,\, k_r^L x_C^U,\, k_r^U x_C^L,\, k_r^U x_C^U\} && \text{(6.3b)} \\
\dot{x}_C^L &= \min\{\min\{k_f^L x_A^L,\, k_f^L x_A^U,\, k_f^U x_A^L,\, k_f^U x_A^U\} x_B^L,\, \min\{k_f^L x_A^L,\, k_f^L x_A^U,\, k_f^U x_A^L, \\
&\quad k_f^U x_A^U\} x_B^U,\, \max\{k_f^L x_A^L,\, k_f^L x_A^U,\, k_f^U x_A^L,\, k_f^U x_A^U\} x_B^L,\, \max\{k_f^L x_A^L,\, k_f^L x_A^U, \\
&\quad k_f^U x_A^L,\, k_f^U x_A^U\} x_B^U\} - \max\{k_r^L x_C^L,\, k_r^U x_C^L\} && \text{(6.3c)} \\
\dot{x}_A^U &= -\min\{\min\{k_f^L x_A^U,\, k_f^U x_A^U\} x_B^L,\, \min\{k_f^L x_A^U,\, k_f^U x_A^U\} x_B^U,\, \max\{k_f^L x_A^U, \\
&\quad k_f^U x_A^U\} x_B^L,\, \max\{k_f^L x_A^U,\, k_f^U x_A^U\} x_B^U\} \\
&\quad + \max\{k_r^L x_C^L,\, k_r^L x_C^U,\, k_r^U x_C^L,\, k_r^U x_C^U\} && \text{(6.3d)} \\
\dot{x}_B^U &= -\min\{\min\{k_f^L x_A^L,\, k_f^L x_A^U,\, k_f^U x_A^L,\, k_f^U x_A^U\} x_B^U,\, \max\{k_f^L x_A^L,\, k_f^L x_A^U, \\
&\quad k_f^U x_A^L,\, k_f^U x_A^U\} x_B^U\} + \max\{k_r^L x_C^L,\, k_r^L x_C^U,\, k_r^U x_C^L,\, k_r^U x_C^U\} && \text{(6.3e)} \\
\dot{x}_C^U &= \max\{\min\{k_f^L x_A^L,\, k_f^L x_A^U,\, k_f^U x_A^L,\, k_f^U x_A^U\} x_B^L,\, \min\{k_f^L x_A^L,\, k_f^L x_A^U,\, k_f^U x_A^L, \\
&\quad k_f^U x_A^U\} x_B^U,\, \max\{k_f^L x_A^L,\, k_f^L x_A^U,\, k_f^U x_A^L,\, k_f^U x_A^U\} x_B^L,\, \max\{k_f^L x_A^L,\, k_f^L x_A^U, \\
&\quad k_f^U x_A^L,\, k_f^U x_A^U\} x_B^U\} - \min\{k_r^L x_C^U,\, k_r^U x_C^U\} && \text{(6.3f)}
\end{aligned}$$

subject to the initial conditions

$$x_A^L(0) = x_A^U(0) = x_{A0},\ x_B^L(0) = x_B^U(0) = x_{B0},\ x_C^L(0) = x_C^U(0) = x_{C0}.$$

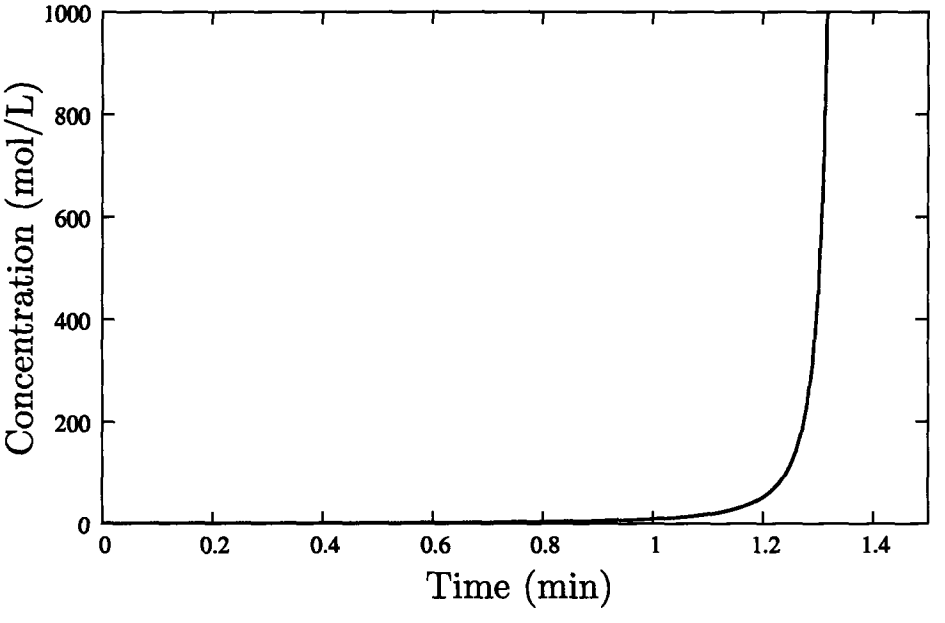
Despite their unwieldy appearance, because the right hand sides of the above differential equations are Lipschitz continuous, numerically integrating the differential

equations is not difficult. In order to compute a numerical solution, the following values are assigned to the constants:

$$k_f^L = 100,\ k_f^U = 500,\ k_r^L = 0.001,\ k_r^U = 0.01,\ x_{A0} = 1,\ x_{B0} = 1.5,\ x_{C0} = 0.5$$

where the units for concentration are mol/L, the units for the forward rate constant are L/(mol·min), and the units for the reverse rate constant are min^{-1}. The time interval of interest is set to 1.5 minutes. A plot of the upper bound for species B is shown in Figure 6-1 below. Clearly, from the figure, the bounding method is not acceptable. From physical considerations, I know that the upper bound should not exceed the sum of initial concentrations of species B and C in the system. However, because the system is not quasimonotone, the upper bound for species B explodes (in fact, the upper bound for the concentration of all species explodes). For Figure 6-1, the integration was terminated when the bound exceeded 10^3.

Figure 6-1: Upper bound for species B from Example 6.11 without utilizing *a priori* information



In order to correct the bounds explosion, Corollary 6.6 is now applied to the system in Example 6.11 utilizing the *a priori* information from considering species

mole balances on the system. Given the assumption that x_{B0} is greater than x_{A0}, from the kinetics of Example 6.11, the following inequalities hold:

$$\begin{aligned} 0 \leq x_A(t) &\leq x_{A0} + x_{C0} \\ x_{B0} - x_{A0} \leq x_B(t) &\leq x_{B0} + x_{C0} \\ 0 \leq x_C(t) &\leq x_{A0} + x_{C0}. \end{aligned}$$

Corollary 6.6 may now be implemented by several different methods; three methods are presented here. The first method is a trivial extension of applying interval arithmetic. Recalling that the optimization is now performed over the intersection of the physical and differential bounds, by setting $\overline{\mathbf{X}}(t) = [\overline{\mathbf{x}}^L(t), \overline{\mathbf{x}}^U(t)]$ and $G(t) = [\mathbf{x}^L(t), \mathbf{x}^U(t)]$, the right hand side can be represented as an optimization over the set

$$\begin{aligned} \overline{\mathbf{X}}(t) \cap G(t) = [\max\{x_A^L(t), \overline{x}_A^L(t)\}, \min\{x_A^U(t), \overline{x}_A^U(t)\}] \times \cdots \\ \times [\max\{x_C^L(t), \overline{x}_C^L(t)\}, \min\{x_C^U(t), \overline{x}_C^U(t)\}]. \quad (6.4) \end{aligned}$$

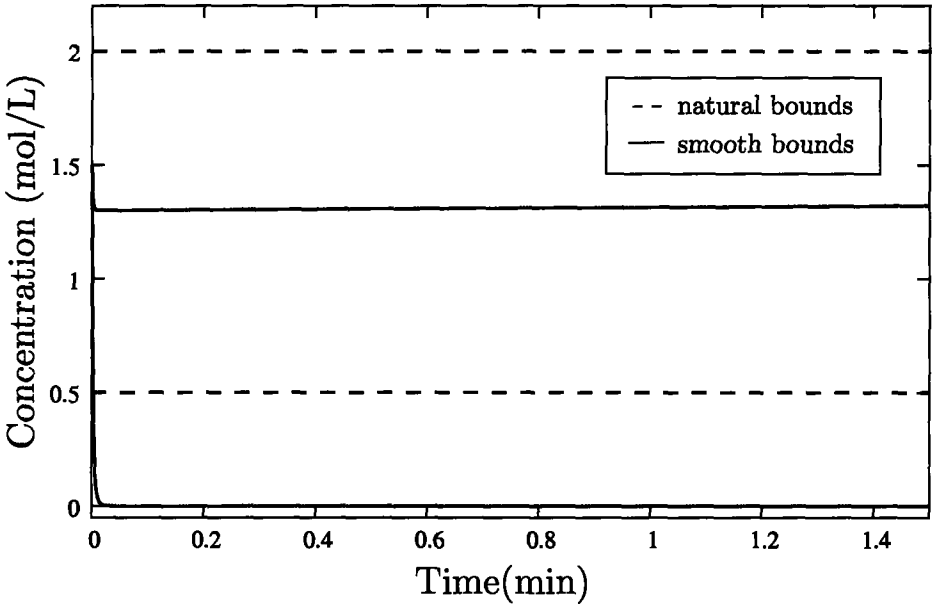
Therefore, much tighter bounds are achieved by simply assigning the following values

$$\begin{aligned} x_A^L &:= \max\{0, x_A^L\} \\ x_A^U &:= \min\{x_{A0} + x_{C0}, x_A^U\} \\ x_B^L &:= \max\{x_{B0} - x_{A0}, x_B^L\} \\ x_B^U &:= \min\{x_{B0} + x_{C0}, x_B^U\} \\ x_C^L &:= \max\{0, x_C^L\} \\ x_C^U &:= \min\{x_{A0} + x_{C0}, x_C^U\} \end{aligned}$$

into the right hand sides of Equation 6.3. The resulting bounds for species B together with the natural bounds are shown in Figure 6-2. There are several important features to note in these new bounds. First and foremost, the bounds no longer explode

because when the right hand sides of the differential equations exceed a limit (dictated by the interval arithmetic), the right hand side of the differential equation becomes a constant thus bounding the slope of the state. Second, the bounds are smooth. This phenomenon occurs because the right hand sides of the differential equations are continuous assuring the everywhere existence of the derivatives of the states. Finally, while the upper bound is an improvement over the natural bound, the lower bound is still not at least as good as the natural bounds on the system.

Figure 6-2: Bounds for species B from Example 6.11 utilizing *a priori* information



The second application of Corollary 6.6 alleviates the problem that the integrated state bounds exceed the *a priori* bounds at the expense of smoothness of the resulting bounds and ease of numerical integration. In the discussion following Corollary 6.6, the case where the optimization problem becomes infeasible was discussed. In addition to optimizing over the constraint set $\overline{\mathbb{X}}(t) \cap G(t)$, I must also remember that I am subject to the equality constraint $z_i = v_i(t)$ for the lower bounding problems and $z_i = w_i(t)$ for the upper bounding problems. However, because of the constraint set $\overline{\mathbb{X}}(t)$, the equality $z_i = v_i(t)$ is impossible to satisfy when $x_i^L(t) < \overline{x}_i^L(t)$, and the equality $z_i = w_i(t)$ is impossible to satisfy when $x_i^U(t) > \overline{x}_i^U(t)$. In this situation,

I am free to choose any finite number for the right hand side of the differential equation. The obvious choice for this number is zero, for this forces the integrated bound to equal the natural bound. For the upper bound, choosing a positive number yields a bound looser than the natural bound and choosing a negative number causes chattering in the numerical integrator. Therefore, for $i = 1, \ldots, n_x$, very tight bounds can be obtained via Corollary 6.6 by defining the right hand sides of the ODEs to equal the differential equations derived via interval analysis if $x_i^L, x_i^U \in \overline{X}(t)$ and letting $\dot{x}_i^L = \dot{x}_i^U = 0$ otherwise. For Example 6.11, the lower bound for species B from this analysis and the lower bound as previously computed via the interval arithmetic approach to Corollary 6.6 are shown for comparison in Figure 6-3 below.

Figure 6-3: Bounds for species B from Example 6.11 utilizing *a priori* information and bounds truncation

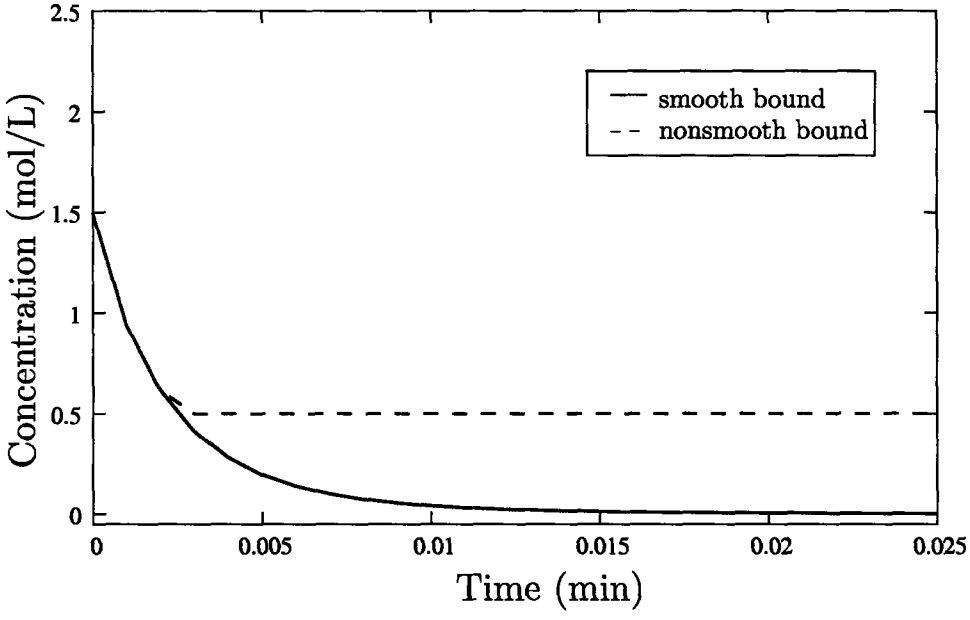
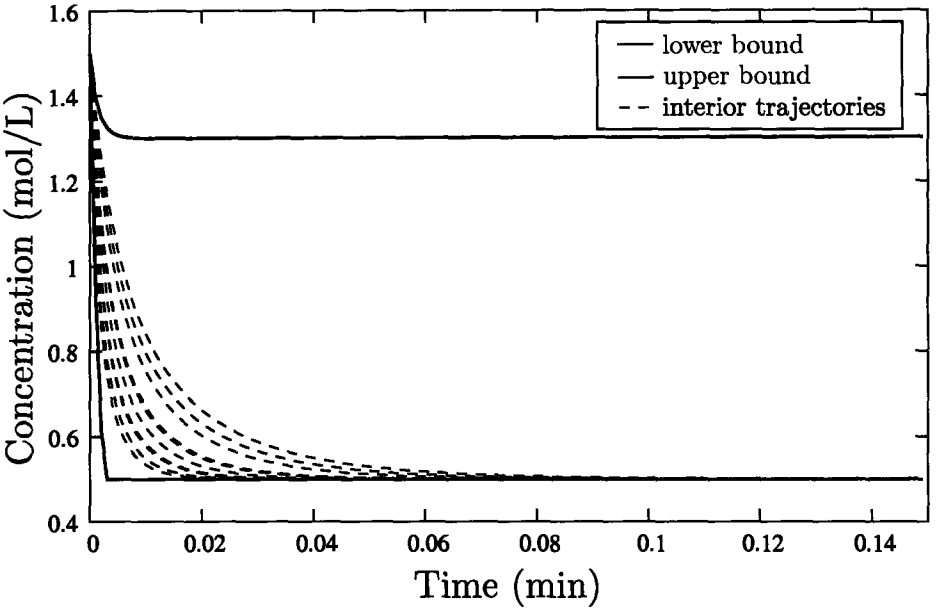


Figure 6-4 illustrates the upper and lower bounds as generated by the second method along with ten interior trajectories derived from random values for k_f and k_r.

In the third method of bounding the differential equations, I incorporate an invariant in the differential equation defining the system. From an extent of reaction

Figure 6-4: Bounds for species B from Example 6.11 utilizing *a priori* information and bounds truncation along with ten randomly generated interior trajectories



analysis, the following equality may be derived for this system:

$$x_A + x_B + 2x_C = x_{A0} + x_{B0} + 2x_{C0}. \tag{6.5}$$

Corollary 6.6 is now applied where the natural bounding set $\overline{\mathcal{X}}(t)$ is defined by the intersection of the plane in Equation 6.5 and $\overline{\mathcal{X}}(t)$ as defined by Equation 6.4. Effectively, for each bounding differential equation, Equation 6.5 is utilized to eliminate one variable from the optimization problem defining the right hand side of the bounding differential equation. The selection of which variables are eliminated in each optimization problem is not unique. Theoretically, the addition of this invariant creates more constrained optimization problems, and the infimum [supremum] of these new optimization problems are at least no worse than the infimum [supremum] of the optimization problems derived without the invariant. In practice, however, because the optimization problems are estimated using interval arithmetic, the quality of the state bound is very dependent on the quality of the interval arithmetic methods employed for bounding the right hand sides of the differential equations. Both elimination of

variables and distinct factorization strategies can lead to vastly different outcomes, and these different outcomes sometimes lead to worse state bounds than not including the invariant at all.

For this example, the invariant was used to eliminate x_C from the optimization problems defining $\dot{x}_A^L$, $\dot{x}_A^U$, $\dot{x}_B^L$, $\dot{x}_B^U$ and x_B from the equations defining $\dot{x}_C^L$, $\dot{x}_C^U$. The resulting differential equations are given by

$$
\begin{aligned}
\dot{x}_A^L &= -\max\{\min\{k_f^L x_A^L,\, k_f^U x_A^L\}\max\{x_B^L,\, 0.5\},\, \min\{k_f^L x_A^L,\, k_f^U x_A^L\}\min\{x_B^U,\, 2\},\\
&\quad \max\{k_f^L x_A^L,\, k_f^U x_A^L\}\max\{x_B^L,\, 0.5\},\, \max\{k_f^L x_A^L,\, k_f^U x_A^L\}\min\{x_B^U,\, 2\}\} +\\
&\quad \min\{k_r^L(1.75 - 0.5x_A^L - \max\{0.5\max\{x_B^L,\, 0.5\},\, 0.5\min\{x_B^U,\, 2\}\}),\\
&\quad k_r^L(1.75 - 0.5x_A^L - \min\{0.5\max\{x_B^L,\, 0.5\},\, 0.5\min\{x_B^U,\, 2\}\}),\\
&\quad k_r^U(1.75 - 0.5x_A^L - \max\{0.5\max\{x_B^L,\, 0.5\},\, 0.5\min\{x_B^U,\, 2\}\}),\\
&\quad k_r^U(1.75 - 0.5x_A^L - \min\{0.5\max\{x_B^L,\, 0.5\},\, 0.5\min\{x_B^U,\, 2\}\})\}\\
\dot{x}_B^L &= -\max\{\min\{k_f^L\max\{x_A^L,\, 0\},\, k_f^L\min\{x_A^U,\, 1.5\},\, k_f^U\max\{x_A^L,\, 0\},\\
&\quad k_f^U\min\{x_A^U,\, 1.5\}\}x_B^L,\, \max\{k_f^L\max\{x_A^L,\, 0\},\, k_f^L\min\{x_A^U,\, 1.5\},\, k_f^U\max\{x_A^L,\, 0\},\\
&\quad k_f^U\min\{x_A^U,\, 1.5\}\}x_B^L\} + \min\{k_r^L\{1.75 - \max\{0.5\max\{x_A^L,\, 0\},\, 0.5\min\{x_A^U,\, 1.5\}\}\\
&\quad -0.5x_B^L\},\, k_r^L(1.75 - \min\{0.5\max\{x_A^L,\, 0\},\, 0.5\min\{x_A^U,\, 1.5\}\} - 0.5x_B^L),\\
&\quad k_r^U(1.75 - \max\{0.5\max\{x_A^L,\, 0\},\, 0.5\min\{x_A^U,\, 1.5\}\} - 0.5x_B^L),\, k_r^U(1.75\\
&\quad - \min\{0.5\max\{x_A^L,\, 0\},\, 0.5\min\{x_A^U,\, 1.5\}\} - 0.5x_B^L)\}\\
\dot{x}_C^L &= \min\{(3.5 - \min\{x_A^U,\, 1.5\} - 2x_C^L)\min\{k_f^L\max\{x_A^L,\, 0\},\, k_f^L\min\{x_A^U,\, 1.5\},\\
&\quad k_f^U\max\{x_A^L,\, 0\},\, k_f^U\min\{x_A^U,\, 1.5\}\},\, (3.5 - \max\{x_A^L,\, 0\} - 2x_C^L)\cdot\\
&\quad \min\{k_f^L\max\{x_A^L,\, 0\},\, k_f^L\min\{x_A^U,\, 1.5\},\, k_f^U\max\{x_A^L,\, 0\},\, k_f^U\min\{x_A^U,\, 1.5\}\},\\
&\quad (3.5 - \min\{x_A^U,\, 1.5\} - 2x_C^L)\max\{k_f^L\max\{x_A^L,\, 0\},\, k_f^L\min\{x_A^U,\, 1.5\},\\
&\quad k_f^U\max\{x_A^L,\, 0\},\, k_f^U\min\{x_A^U,\, 1.5\}\},\, (3.5 - \max\{x_A^L,\, 0\} - 2x_C^L)\cdot\\
&\quad \max\{k_f^L\max\{x_A^L,\, 0\},\, k_f^L\min\{x_A^U,\, 1.5\},\, k_f^U\max\{x_A^L,\, 0\},\, k_f^U\min\{x_A^U,\, 1.5\}\}\} -\\
&\quad \max\{k_r^L x_C^L,\, k_r^U x_C^L\}\\
\dot{x}_A^U &= -\min\{\min\{k_f^L x_A^U,\, k_f^U x_A^U\}\max\{x_B^L,\, 0.5\},\, \min\{k_f^L x_A^U,\, k_f^U x_A^U\}\min\{x_B^U,\, 2\},
\end{aligned}
$$

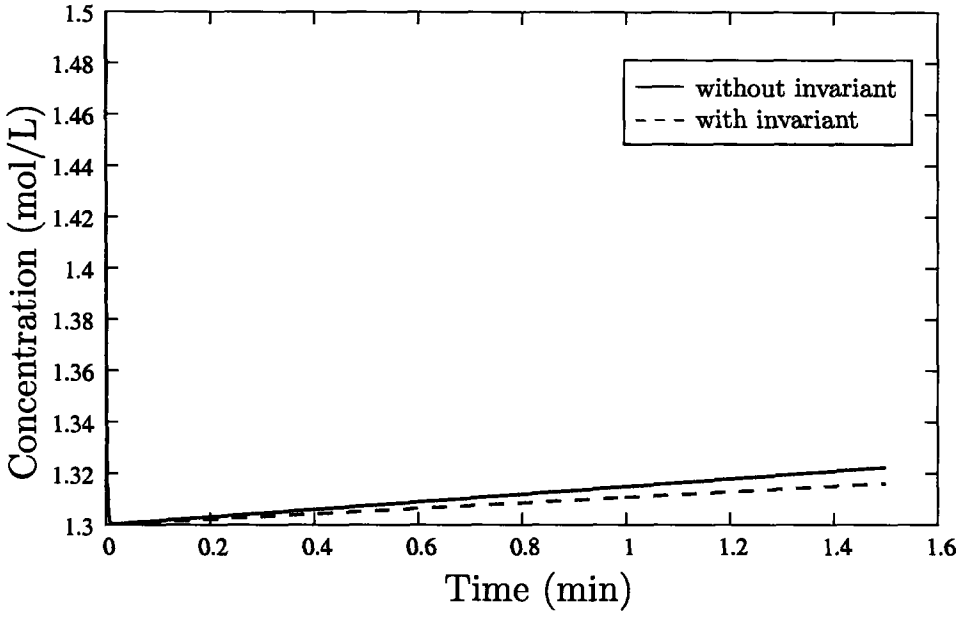
$$
\begin{aligned}
& \max\{k_f^L x_A^U,\, k_f^U x_A^U\} \max\{x_B^L,\, 0.5\},\, \max\{k_f^L x_A^U,\, k_f^U x_A^U\} \min\{x_B^U,\, 2\}\} + \\
& \max\{k_r^L(1.75 - 0.5x_A^U - \max\{0.5 \max\{x_B^L,\, 0.5\},\, 0.5 \min\{x_B^U,\, 2\}\}), \\
& k_r^L(1.75 - 0.5x_A^U - \min\{0.5 \max\{x_B^L,\, 0.5\},\, 0.5 \min\{x_B^U,\, 2\}\}), \\
& k_r^U(1.75 - 0.5x_A^U - \max\{0.5 \max\{x_B^L,\, 0.5\},\, 0.5 \min\{x_B^U,\, 2\}\}), \\
& k_r^U(1.75 - 0.5x_A^U - \min\{0.5 \max\{x_B^L,\, 0.5\},\, 0.5 \min\{x_B^U,\, 2\}\})\}
\end{aligned}
$$

$$
\begin{aligned}
\dot{x}_B^U = & -\min\{\min\{k_f^L \max\{x_A^L,\, 0\},\, k_f^L \min\{x_A^U,\, 1.5\},\, k_f^U \max\{x_A^L,\, 0\}, \\
& k_f^U \min\{x_A^U,\, 1.5\}\} x_B^U,\, \max\{k_f^L \max\{x_A^L,\, 0\},\, k_f^L \min\{x_A^U,\, 1.5\},\, k_f^U \max\{x_A^L,\, 0\}, \\
& k_f^U \min\{x_A^U,\, 1.5\}\} x_B^U\} + \max\{k_r^L(1.75 - \max\{0.5 \max\{x_A^L,\, 0\},\, 0.5 \min\{x_A^U,\, 1.5\}\} \\
& -0.5x_B^U),\, k_r^L(1.75 - \min\{0.5 \max\{x_A^L,\, 0\},\, 0.5 \min\{x_A^U,\, 1.5\}\} - 0.5x_B^U),\, k_r^U)1.75 \\
& - \max\{0.5 \max\{x_A^L,\, 0\},\, 0.5 \min\{x_A^U,\, 1.5\}\} - 0.5x_B^U), \\
& k_r^U(1.75 - \min\{0.5 \max\{x_A^L,\, 0\},\, 0.5 \min\{x_A^U,\, 1.5\}\} - 0.5x_B^U)\}
\end{aligned}
$$

$$
\begin{aligned}
\dot{x}_C^U = & \max\{(3.5 - \min\{x_A^U,\, 1.5\} - 2x_C^U) \min\{k_f^L \max\{x_A^L,\, 0\},\, k_f^L \min\{x_A^U,\, 1.5\}, \\
& k_f^U \max\{x_A^L,\, 0\},\, k_f^U \min\{x_A^U,\, 1.5\}\},\, (3.5 - \max\{x_A^L,\, 0\} - 2x_C^U) \cdot \\
& \min\{k_f^L \max\{x_A^L,\, 0\},\, k_f^L \min\{x_A^U,\, 1.5\},\, k_f^U \max\{x_A^L,\, 0\},\, k_f^U \min\{x_A^U,\, 1.5\}\}, \\
& \max\{k_f^L \max\{x_A^L,\, 0\},\, k_f^L \min\{x_A^U,\, 1.5\},\, k_f^U \max\{x_A^L,\, 0\},\, k_f^U(3.5 - \min\{x_A^U,\, 1.5\} \\
& -2x_C^U) \min\{x_A^U,\, 1.5\},\, (3.5 - \max\{x_A^L,\, 0\} - 2x_C^U) \max\{k_f^L \max\{x_A^L,\, 0\}, \\
& k_f^L \min\{x_A^U,\, 1.5\},\, k_f^U \max\{x_A^L,\, 0\},\, k_f^U \min\{x_A^U,\, 1.5\}\}\} - \min\{k_r^L x_C^U,\, k_r^U x_C^U\}.
\end{aligned}
$$

The marginal improvement of utilizing the reaction invariant is illustrated for the upper bound of species B in Figure 6-5 below. Even for this simple example problem, different factorizations of the objective function prior to applying interval arithmetic can lead to bounds worse than those computed without the invariant.

Figure 6-5: Bounds for species B from Example 6.11 utilizing the reaction invariant



6.2 McCormick Composition and Relaxing the Integral Objective Function

As described in Chapter 3, generating a convex relaxation for Problem 6.1 requires generating a convex relaxation for the integrand of the objective function. From Corollary 3.7, the convex relaxation for the integrand is required on the set $(t_0, t_f] \times P$ for each fixed $\underline{t} \in (t_0, t_f]$. In Chapter 4, the affine nature of the solution of a system of linear differential equations was utilized to prove the composition result that a partially convex integrand on $\mathcal{X}(\underline{t}) \times \dot{\mathcal{X}}(\underline{t}) \times P$ composed with a linear system was partially convex on P. This result immediately implied Corollary 4.12, a direct method for the construction of a convex relaxation for Problem 4.1. However, because no general statement can be made concerning the solution of the embedded nonlinear dynamic system of Problem 6.1, an alternative composition technique must be employed; this technique, attributed to McCormick [65], was first introduced as Theorem 5.1 in the previous chapter.

In Chapter 5, Theorem 5.1 was utilized to construct convex relaxations on $\mathcal{X}(\underline{t}) \times$

$\dot{\mathcal{X}}(\underline{t}) \times P$ for a composite integrand; the composition of the state variables with the parameters via the solution to the embedded linear dynamic system was addressed via Proposition 4.3. This strategy was employed because exploiting the affine nature of the solution of the linear system enabled an elegant method for handling the state composition. For nonlinear dynamic systems, the affine solution of linear systems cannot be directly applied to the solution of the differential equations. However, McCormick's composition technique is directly applicable to relaxing the solution of the embedded nonlinear differential equations provided that both a convex and concave relaxation for $\mathbf{x}(\underline{t}, \mathbf{p})$ are known. To emphasize its utilization to relax the solution of the nonlinear differential equations, McCormick's theorem is rewritten below using notation consistent with the dynamic system. The extension to dynamic systems is immediate following the same arguments as those following Theorem 5.1.

Theorem 6.12. *Let $P \subset \mathbb{R}^{n_p}$ be convex. Consider the function $V[x(\underline{t}, \mathbf{p})]$ where $x : P \to \mathbb{R}$ is continuous, and let $[a, b]$ be a set containing the image of $\mathbf{x}(\underline{t}, \mathbf{p})$ on P such that $[a, b]$ approaches degeneracy as P approaches degeneracy. Suppose that a convex function $c(\underline{t}, \mathbf{p})$ and a concave function $C(\underline{t}, \mathbf{p})$ satisfying*

$$c(\underline{t}, \mathbf{p}) \leq v(\underline{t}, \mathbf{p}) \leq C(\underline{t}, \mathbf{p}), \qquad \forall\ \mathbf{p} \in P$$

are available. Denote a convex relaxation of $V(\cdot)$ on $[a, b]$ by $e_V(\cdot)$, denote a concave relaxation of $V(\cdot)$ on $[a, b]$ by $E_V(\cdot)$, let z_{min} be a point at which $V(\cdot)$ attains its infimum on $[a, b]$, and let z_{max} be a point at which $V(\cdot)$ attains its supremum on $[a, b]$. If the above conditions are satisfied, then

$$u(\underline{t}, \mathbf{p}) = e_V[\text{mid}\ \{c(\underline{t}, \mathbf{p}), C(\underline{t}, \mathbf{p}), z_{min}(\underline{t})\}]$$

is a convex underestimator for $V[x(\underline{t}, \mathbf{p})]$ on P, and

$$o(\underline{t}, \mathbf{p}) = E_V[\text{mid}\ \{c(\underline{t}, \mathbf{p}), C(\underline{t}, \mathbf{p}), z_{max}(\underline{t})\}]$$

is a concave overestimator for $V[x(\mathbf{p})]$ on P, where the mid *function selects the middle*

value of three scalars.

Proof. See Theorem 5.1 □

Remark 6.13. For embedded linear differential systems, Theorem 6.12 reduces to Proposition 4.3. This follows because e_V and E_V reduce to the identify function, and since $c(\underline{t}, \mathbf{p}) = C(\underline{t}, \mathbf{p}) = x(\underline{t}, \mathbf{p}) = \mathbf{M}(\underline{t})\mathbf{p} + \mathbf{n}(\underline{t})$, the mid function reduces to $x(\underline{t}, \mathbf{p})$. Therefore, relaxing Problem 6.1 exactly reduces to solving Problem 4.1 when Equation 6.1 is restricted to a linear system (if the construction of state bounds is considered an independent task).

Clearly, the difficulty in applying Theorem 6.12 to Problem 6.1 is that obtaining $c_i(\underline{t}, \mathbf{p})$ and $C_i(\underline{t}, \mathbf{p})$ for any given state $x_i(\underline{t}, \mathbf{p})$ is not obvious given that no general information is available concerning the structure of the solution of Equation 6.1. Before demonstrating a method for obtaining $\mathbf{c}(\underline{t}, \mathbf{p})$ and $\mathbf{C}(\underline{t}, \mathbf{p})$ for general nonlinear differential equations, an example is presented that demonstrates utilizing Theorem 6.12 for directly relaxing an integral. An analytically tractable differential equation was chosen to emphasize the utilization of the theorem; numerical examples are considered in the following chapter.

Example 6.14. Consider the optimization problem

$$\min_{p \in [-3,3]} L(p) = \int_0^1 -[x(t,p)]^2 \, dt$$

such that

$$\dot{x} = px$$
$$x(0) = -1.$$

This example develops a convex relaxation for $L(p)$ at the root node. The analytical solution of the differential equation is

$$x(t,p) = -\exp(pt), \tag{6.6}$$

and this allows me to compute explicitly the objective function:

$$L(p) = \begin{cases} -1 & \text{if } p = 0 \\ \left[1 - \exp(2p)\right]/(2p) & \text{otherwise.} \end{cases}$$

Because the objective function is nonconvex, a global optimization technique is warranted; hence, a convex relaxation at the root node is required. From Corollary 3.7, I know that to generate a convex underestimator for the integral requires a convex underestimator for the integrand. First, the composition for the integrand is identified. In this case, I trivially have

$$\ell(z) = -z^2,$$

where $z = x(t,p)$ is given by the solution of the differential equation (in this case given explicitly by Equation 6.6). From Theorem 6.12, a convex underestimator for the integrand is given pointwise in time by

$$u(\underline{t},p) = e\left(\text{mid}\left\{c(\underline{t},p), C(\underline{t},p), z_{min}(\underline{t})\right\}\right) \qquad \forall\ \underline{t} \in [0,1].$$

Therefore, I must identify $e(\cdot)$, compute $\mathcal{X}(t)$ and $z_{min}(\underline{t})$, and derive functions $c(\underline{t},p)$ and $C(\underline{t},p)$. Because t is always positive, for any fixed $\underline{t} \in [0,1]$, $x(\underline{t},p)$ is monotonically decreasing. This fact enables me to trivially calculate $\mathcal{X}(t)$ by interval extensions to yield

$$\mathcal{X}(t) = [-\exp(p^U t), -\exp(p^L t)] = [-\exp(3t), -\exp(-3t)].$$

For each $\underline{t} \in [0,1]$, ℓ is concave on $Z = \mathcal{X}(\underline{t})$; therefore, $e(z)$ is simply the secant given by

$$e(\underline{t},z) = \left[x^U(\underline{t}) + x^L(\underline{t})\right]\left[x^L(\underline{t}) - z\right] - \left[x^L(\underline{t})\right]^2 \qquad \forall\ \underline{t} \in [0,1].$$

Because $\left[x^U(\underline{t}) + x^L(\underline{t})\right] < 0\ \forall\ \underline{t} \in [0,1]$, I know that $z_{min}(\underline{t}) = x^L(\underline{t})\ \forall\ \underline{t} \in [0,1]$. To complete the analysis, I must derive the convex underestimator $c(\underline{t},p)$ and the concave overestimator $C(\underline{t},p)$. The second partial derivative of Equation 6.6 with

respect to p is less than or equal to zero for all $(t,p) \in [0,1] \times P$; thus, the function is always concave. Trivially, I have

$$C(\underline{t},p) = -\exp(p\underline{t}) \qquad \forall\, \underline{t} \in [0,1]$$

and

$$c(\underline{t},p) = [\exp(p^L\underline{t}) - \exp(p^U\underline{t})](p-p^L)/(p^U-p^L) - \exp(p^L\underline{t}) \qquad \forall\, \underline{t} \in [0,1].$$

Combining, the pointwise in time underestimator for the integrand is given by

$$\mathcal{U}(\underline{t},p) = \left[x^U(\underline{t}) + x^L(\underline{t})\right]\left[x^L(\underline{t}) - \text{mid}\left\{c(\underline{t},p),\, C(\underline{t},p),\, z_{min}(\underline{t})\right\}\right] - \left[x^L(\underline{t})\right]^2 \qquad \forall\, \underline{t} \in [0,1].$$

By Corollary 3.7, a relaxation for the integral is given by

$$U(p) = \int_0^1 \mathcal{U}(t,p)\; dt.$$

The remainder of this section is devoted to deriving $\mathbf{c}$ and $\mathbf{C}$, the convex and concave relaxations for the solution of the embedded nonlinear differential system. I begin by defining a notation for the pointwise in time linearization of a function at a reference trajectory.

Definition 6.15. Let $\mathcal{L}_{\mathbf{f}}(t,\mathbf{x},\mathbf{p})\big|_{\mathbf{x}^*(t),\mathbf{p}^*}$ be a linearization of $\mathbf{f}(t,\mathbf{x},\mathbf{p})$ at $(\mathbf{x}^*(t),\mathbf{p}^*)$. The linearization is given for $i = 1,\ldots,n_x$ by the following:

$$\mathcal{L}_{f_i}(t,\mathbf{x},\mathbf{p})\big|_{\mathbf{x}^*(t),\mathbf{p}^*} = f_i(t,\mathbf{x}^*(t),\mathbf{p}^*) + \sum_{j=1}^{n_x} \left.\frac{\partial f_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} \left(x_j - x_j^*(t)\right) + \sum_{j=1}^{n_p} \left.\frac{\partial f_i}{\partial p_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} \left(p_j - p_j^*\right). \quad (6.7)$$

The following theorem is utilized for constructing convex underestimators and concave overestimators for the solution of a parameter dependent ODE.

Theorem 6.16. *Consider Differential Equation 6.1 and the set $\mathcal{X}(t)$ as defined in*

Remark 6.7. Let P be convex. For $i = 1, \ldots, n_x$ and each fixed $\underline{t} \in (t_0, t_f]$, let $u_i(\underline{t}, \mathbf{x}, \mathbf{p})$ be a convex underestimator for f_i on $\mathcal{X}(\underline{t}) \times P$ and let $o_i(\underline{t}, \mathbf{x}, \mathbf{p})$ be a concave overestimator for f_i on $\mathcal{X}(\underline{t}) \times P$. For $i = 1, \ldots, n_x$, consider the differential equations

$$\begin{aligned} \dot{c}_i &= g_{c,i}(t, \mathbf{c}, \mathbf{C}, \mathbf{p}) = \inf_{\substack{\mathbf{z} \in \mathcal{C}(t,\mathbf{p}) \\ z_i = c_i(t)}} \mathcal{L}_{u_i}(t, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(t), \mathbf{p}^*} \\ \dot{C}_i &= g_{C,i}(t, \mathbf{c}, \mathbf{C}, \mathbf{p}) = \sup_{\substack{\mathbf{z} \in \mathcal{C}(t,\mathbf{p}) \\ z_i = C_i(t)}} \mathcal{L}_{o_i}(t, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(t), \mathbf{p}^*} \end{aligned}$$

for some reference trajectory $(\mathbf{x}^*(\underline{t}), \mathbf{p}^*) \in \mathcal{X}(\underline{t}) \times P$ (differentiability of u_i and o_i is assumed along the reference trajectory for fixed $\underline{t} \in (t_0, t_f]$) with initial conditions

$$\mathbf{c}(t_0) \leq \mathbf{x}(t_0, \mathbf{p}) \leq \mathbf{C}(t_0) \qquad \forall\ \mathbf{p} \in P, \tag{6.8}$$

where $\mathcal{C}(t, \mathbf{p}) = \{\mathbf{z} \mid \mathbf{c}(t, \mathbf{p}) \leq \mathbf{z} \leq \mathbf{C}(t, \mathbf{p})\}$. Then for each fixed $\underline{t} \in (t_0, t_f]$, $\mathbf{c}(\underline{t}, \mathbf{p})$ is a convex underestimator for $\mathbf{x}(\underline{t}, \mathbf{p})$ on P and $\mathbf{C}(\underline{t}, \mathbf{p})$ is a concave overestimator for $\mathbf{x}(\underline{t}, \mathbf{p})$ on P.

Proof. Because for each fixed $\underline{t} \in (t_0, t_f]$, $u_i(\underline{t}, \mathbf{x}, \mathbf{p})$ is a convex underestimator for f_i on $\mathcal{X}(\underline{t}) \times P$, the linearization $\mathcal{L}_{u_i}(\underline{t}, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(\underline{t}), \mathbf{p}^*}$ validly supports f_i on $\mathcal{X}(\underline{t}) \times P$ from below. Analogously, $\mathcal{L}_{o_i}(\underline{t}, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(\underline{t}), \mathbf{p}^*}$ supports f_i on $\mathcal{X}(\underline{t}) \times P$ from above. For each fixed $\underline{t} \in (t_0, t_f]$ and all $(\mathbf{z}, \mathbf{p}) \in \mathcal{X}(\underline{t}) \times P$, the pointwise inequalities

$$\begin{aligned} \mathcal{L}_{u_i}(\underline{t}, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(\underline{t}), \mathbf{p}^*} &\leq u_i(\underline{t}, \mathbf{z}, \mathbf{p}) \leq f_i(\underline{t}, \mathbf{z}, \mathbf{p}) \\ \mathcal{L}_{o_i}(\underline{t}, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(\underline{t}), \mathbf{p}^*} &\geq o_i(\underline{t}, \mathbf{z}, \mathbf{p}) \geq f_i(\underline{t}, \mathbf{z}, \mathbf{p}) \end{aligned}$$

are therefore established for $i = 1, \ldots, n_x$, which immediately imply

$$\dot{c}_i = \inf_{\substack{\mathbf{z} \in \mathcal{C}(t,\mathbf{p}) \\ z_i = c_i(t)}} \mathcal{L}_{u_i}(t, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(t), \mathbf{p}^*} \leq \inf_{\substack{\mathbf{z} \in \mathcal{X}(t) \cap \mathcal{C}(t,\mathbf{p}) \\ z_i = c_i(t)}} f_i(t, \mathbf{z}, \mathbf{p}) \tag{6.9a}$$

$$\dot{C}_i = \sup_{\substack{\mathbf{z} \in \mathcal{C}(t,\mathbf{p}) \\ z_i = C_i(t)}} \mathcal{L}_{o_i}(t, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(t), \mathbf{p}^*} \geq \sup_{\substack{\mathbf{z} \in \mathcal{X}(t) \cap \mathcal{C}(t,\mathbf{p}) \\ z_i = C_i(t)}} f_i(t, \mathbf{z}, \mathbf{p}) \tag{6.9b}$$

for $i = 1, \ldots, n_x$. I note that the infimum [supremum] over $\mathcal{C}(t, \mathbf{p})$ is less [greater]

than the infimum [supremum] over $\mathcal{X}(t) \cap \mathcal{C}(t, \mathbf{p})$, for the infimum [supremum] over a larger set cannot increase the tightness of the bound. Given Initial Condition 6.8 and Equations 6.9a and 6.9b, the hypotheses of Corollary 6.5 are satisfied, and we have that for each $\mathbf{p} \in P$,

$$\mathbf{c}(\underline{t}, \mathbf{p}) \leq \mathbf{x}(\underline{t}, \mathbf{p}) \leq \mathbf{C}(\underline{t}, \mathbf{p})$$

in $(t_0, t_f]$. Because $\mathcal{L}_{u_i}(\underline{t}, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(t),\mathbf{p}^*}$ and $\mathcal{L}_{o_i}(\underline{t}, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(t),\mathbf{p}^*}$ are both affine in $\mathbf{z}$ for fixed $\underline{t} \in (t_0, t_f]$, the infimum and supremum are attained at the vertices of the set $\mathcal{C}(t, \mathbf{p})$ $(z_i = c_i(t))$ and are respectively given by

$$\begin{aligned}\dot{c}_i = \inf_{\substack{\mathbf{z}\in\mathcal{C}(t,\mathbf{p}) \\ z_i = c_i(t)}} \mathcal{L}_{u_i}(t, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(t),\mathbf{p}^*} &= u_i(t, \mathbf{x}^*(t), \mathbf{p}^*) + \left.\frac{\partial u_i}{\partial x_i}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} (c_i - x_i^*(t)) \\ &+ \sum_{j\neq i} \min\left\{ \left.\frac{\partial u_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} c_j,\ \left.\frac{\partial u_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} C_j \right\} - \left.\frac{\partial u_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} x_j^*(t) \\ &+ \sum_{j=1}^{n_p} \left.\frac{\partial u_i}{\partial p_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} (p_j - p_j^*) \end{aligned} \tag{6.10}$$

and

$$\begin{aligned}\dot{C}_i = \sup_{\substack{\mathbf{z}\in\mathcal{C}(t,\mathbf{p}) \\ z_i = C_i(t)}} \mathcal{L}_{o_i}(t, \mathbf{z}, \mathbf{p})\big|_{\mathbf{x}^*(t),\mathbf{p}^*} &= o_i(t, \mathbf{x}^*(t), \mathbf{p}^*) + \left.\frac{\partial o_i}{\partial x_i}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} (C_i - x_i^*(t)) \\ &+ \sum_{j\neq i} \max\left\{ \left.\frac{\partial o_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} c_j,\ \left.\frac{\partial o_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} C_j \right\} - \left.\frac{\partial o_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} x_j^*(t) \\ &+ \sum_{j=1}^{n_p} \left.\frac{\partial o_i}{\partial p_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} (p_j - p_j^*) \end{aligned} \tag{6.11}$$

for $i = 1, \ldots, n_x$. By construction, $\mathbf{c} \leq \mathbf{C}$. Therefore,

$$\min\left\{\left.\frac{\partial u_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} c_j,\ \left.\frac{\partial u_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} C_j\right\} = \begin{cases} \left.\dfrac{\partial u_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} c_j & \text{if } \partial u_i/\partial x_j\big|_{\mathbf{x}^*(t),\mathbf{p}^*} \geq 0 \\ \left.\dfrac{\partial u_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} C_j & \text{otherwise} \end{cases} \tag{6.12}$$

and

$$\max\left\{\left.\frac{\partial o_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} c_j,\ \left.\frac{\partial o_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} C_j\right\} = \begin{cases} \left.\dfrac{\partial o_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} c_j & \text{if } \partial o_i/\partial x_j\big|_{\mathbf{x}^*(t),\mathbf{p}^*} \leq 0 \\ \left.\dfrac{\partial o_i}{\partial x_j}\right|_{\mathbf{x}^*(t),\mathbf{p}^*} C_j & \text{otherwise.} \end{cases} \tag{6.13}$$

Thus, Equations 6.10 and 6.11 reduce to linear differential equations with a finite number of discontinuities fixed in time in the coefficient matrix multiplying the states. The assumption that the number of discontinuities is finite is valid for it is not unreasonable to assume that $u_i(t, \mathbf{x}, \mathbf{p})$ and $o_i(t, \mathbf{x}, \mathbf{p})$ are chosen such that their derivatives do not switch sign infinitely in a finite interval. That the discontinuities are fixed in time is clear because the discontinuities are defined by functions of the reference trajectories and are hence independent of the parameter. The solution to such a system of differential equations has been shown to be affine in $\mathbf{p}$ for fixed time in [56]; therefore, $\mathbf{c}(t, \mathbf{p})$ and $\mathbf{C}(t, \mathbf{p})$ are both concave and convex on P for each fixed $\underline{t} \in (t_0, t_f]$. □

Remark 6.17. If the initial condition depends on $\mathbf{p}$, then $\mathbf{c}_0$ and $\mathbf{C}_0$ may also depend on $\mathbf{p}$ provided they are both affine in $\mathbf{p}$ and underestimate and overestimate $\mathbf{x}(t_0, \mathbf{p})$ for each $\mathbf{p} \in P$, respectively. This condition can also be met via a linearization on a convex/concave outer approximation of the initial condition.

Following Corollary 6.5, a simplification was presented for bounding differential equations possessing at least a mixed monotone property. In an analogous fashion, monotonicity in $\mathbf{z}$ also simplifies the construction of $\mathbf{c}(t, \mathbf{p})$ and $\mathbf{C}(t, \mathbf{p})$. However, in this case, the monotonicity of the linearizations rather than the original functions

must be examined. Clearly, from Equation 6.7, only the coefficients multiplying the linearization of the state variables affect the monotonicity of the system. Essentially, the monotonicity of the linearizations is defined by the Jacobian with respect to the state of the relaxation of the right hand side of the Differential Equation 6.1. If the off-diagonal elements of the Jacobian are positive, the linearization is quasimonotone increasing with respect to a fixed $\mathbf{p}$. If the off-diagonal elements of the Jacobian are negative, the linearization is quasimonotone decreasing. Thus, if some monotonicity properties of the linearization are known, i.e., the sign of the partial derivative is known, then the min of Equation 6.12 and the max of Equation 6.13 are known a priori. When the linearization is known to be either quasimonotone increasing or quasimonotone decreasing in z_j for $j \neq i$, all of the minimums and maximums defining the right hand sides of Equations 6.10 and 6.11 are removed in the obvious manner.

An illustrative example demonstrating the application of Theorem 6.16 for constructing pointwise in time convex underestimators and concave overestimators for the solution of a differential equation is now presented. To emphasize the construction of the convex and concave relaxations rather than the state bounding mechanism, a system for which the state bounds are trivially computed was chosen.

Example 6.18. Consider the following first order, irreversible series reactions:

$$A \overset{k_1}{\rightarrow} B \overset{k_2}{\rightarrow} C.$$

Given the bounding set $K = [\mathbf{k}^L, \mathbf{k}^U]$, I wish to compute pointwise in time convex underestimators and pointwise in time concave overestimators on K for the concentrations of species A and B. The differential equations modeling the system are given by

$$\dot{x}_A = -k_1 x_A$$

$$\dot{x}_B = k_1 x_A - k_2 x_B$$

$$x_A(0) = x_{A0},\ x_B(0) = x_{B0}.$$

In order to apply Theorem 6.16 to Example 6.18, I must first identify convex

underestimators and concave overestimators on the set $[\mathbf{x}^L, \mathbf{x}^U] \times K$ for the right hand sides of the differential equations. A detailed analysis of the derivation of the set $[\mathbf{x}^L, \mathbf{x}^U]$ is not given, for I assume that this set is constructed via Corollary 6.6 in an analogous fashion to Example 6.11. From McCormick [65], the convex envelope of the bilinear term yz on $[y^L, y^U] \times [z^L, z^U]$ is given by

$$u(y,z) = \max\{y^L z + z^L y - y^L z^L,\ y^U z + z^U y - y^U z^U\},$$

and the concave envelope is given by

$$o(y,z) = \min\{y^U z + z^L y - y^U z^L,\ y^L z + z^U y - y^L z^U\}.$$

From the above formulae, $u_i(\underline{t}, \mathbf{x}(\underline{t}), \mathbf{k})$ and $o_i(\underline{t}, \mathbf{x}(\underline{t}), \mathbf{k})$ for Example 6.18 are given pointwise in time by

$$\begin{aligned}
u_A(\underline{t}, \mathbf{x}(\underline{t}), \mathbf{k}) &= -\min\{x_A^L(\underline{t})k_1 + k_1^U x_A(\underline{t}) - x_A^L(\underline{t})k_1^U,\ k_1^L x_A(\underline{t}) + x_A^U(\underline{t})k_1 - k_1^L x_A^U(\underline{t})\} \\
u_B(\underline{t}, \mathbf{x}(\underline{t}), \mathbf{k}) &= \max\{x_A^U(\underline{t})k_1 + k_1^U x_A(\underline{t}) - k_1^U x_A^U(\underline{t}),\ x_A^L(\underline{t})k_1 + k_1^L x_A(\underline{t}) - k_1^L x_A^L(\underline{t})\} \\
&\quad -\min\{x_B^L(\underline{t})k_2 + k_2^U x_B(\underline{t}) - x_B^L(\underline{t})k_2^U,\ k_2^L x_B(\underline{t}) + x_B^U(\underline{t})k_2 - k_2^L x_B^U(\underline{t})\}
\end{aligned}$$

$$\begin{aligned}
o_A(\underline{t}, \mathbf{x}(\underline{t}), \mathbf{k}) &= -\max\{x_A^U(\underline{t})k_1 + k_1^U x_A(\underline{t}) - k_1^U x_A^U(\underline{t}),\ x_A^L(\underline{t})k_1 + k_1^L x_A(\underline{t}) - k_1^L x_A^L(\underline{t})\} \\
o_B(\underline{t}, \mathbf{x}(\underline{t}), \mathbf{k}) &= \min\{x_A^L(\underline{t})k_1 + k_1^U x_A(\underline{t}) - x_A^L(\underline{t})k_1^U,\ k_1^L x_A(\underline{t}) + x_A^U(\underline{t})k_1 - k_1^L x_A^U(\underline{t})\} \\
&\quad -\max\{x_B^U(\underline{t})k_2 + k_2^U x_B(\underline{t}) - k_2^U x_B^U(\underline{t}),\ x_B^L(\underline{t})k_2 + k_2^L x_B(\underline{t}) - k_2^L x_B^L(\underline{t})\}.
\end{aligned}$$

Selecting the abstract reference trajectory $(\mathbf{x}^*(\underline{t}), \mathbf{k}^*)$, the following pointwise in time linearizations are derived from Equation 6.15:

$$\begin{aligned}
\mathcal{L}_{u_A}(\underline{t}, \mathbf{z}, \mathbf{k})\big|_{\mathbf{x}^*(\underline{t}), \mathbf{k}^*} &= u_A(\underline{t}, \mathbf{x}^*(\underline{t}), \mathbf{k}^*) + \alpha_1 (z_A - x_A^*(\underline{t})) + \beta_1 (k_1 - k_1^*) \\
\mathcal{L}_{u_B}(\underline{t}, \mathbf{z}, \mathbf{k})\big|_{\mathbf{x}^*(\underline{t}), \mathbf{k}^*} &= u_B(\underline{t}, \mathbf{x}^*(\underline{t}), \mathbf{k}^*) + \alpha_2 (z_A - x_A^*(\underline{t})) + \beta_2 (z_B - x_B^*(\underline{t})) \\
&\quad +\delta_1 (k_1 - k_1^*) + \gamma_1 (k_2 - k_2^*)
\end{aligned}$$

$$
\begin{aligned}
\mathcal{L}_{o_A}(\underline{t},\mathbf{z},\mathbf{k})\big|_{\mathbf{x}^*(\underline{t}),\mathbf{k}^*} &= o_A(\underline{t},\mathbf{x}^*(\underline{t}),\mathbf{k}^*) + \alpha_3\left(z_A - x_A^*(\underline{t})\right) + \beta_3\left(k_1 - k_1^*\right) \\
\mathcal{L}_{o_B}(\underline{t},\mathbf{z},\mathbf{k})\big|_{\mathbf{x}^*(\underline{t}),\mathbf{k}^*} &= o_B(\underline{t},\mathbf{x}^*(\underline{t}),\mathbf{k}^*) + \alpha_4\left(z_A - x_A^*(\underline{t})\right) + \beta_4\left(z_B - x_B^*(\underline{t})\right) \\
&\quad +\delta_2\left(k_1 - k_1^*\right) + \gamma_2\left(k_2 - k_2^*\right)
\end{aligned}
$$

where

$$
(\alpha_1,\beta_1) = \begin{cases} \left(-k_1^U, -x_A^L(\underline{t})\right) & \text{if } x_A^L(\underline{t})k_1^* + k_1^U x_A^*(\underline{t}) - x_A^L(\underline{t})k_1^U < \\ & \quad k_1^L x_A^*(\underline{t}) + x_A^U(\underline{t})k_1^* - k_1^L x_A^U(\underline{t}) \\ \left(-k_1^L, -x_A^U(\underline{t})\right) & \text{otherwise} \end{cases}
$$

$$
(\alpha_2,\delta_1) = \begin{cases} \left(k_1^U, x_A^U(\underline{t})\right) & \text{if } x_A^U(\underline{t})k_1^* + k_1^U x_A^*(\underline{t}) - k_1^U x_A^U(\underline{t}) > \\ & \quad x_A^L(\underline{t})k_1^* + k_1^L x_A^*(\underline{t}) - k_1^L x_A^L(\underline{t}) \\ \left(k_1^L, x_A^L(\underline{t})\right) & \text{otherwise} \end{cases}
$$

$$
(\beta_2,\gamma_1) = \begin{cases} \left(-k_2^U, -x_B^L(\underline{t})\right) & \text{if } x_B^L(\underline{t})k_2^* + k_2^U x_B^*(\underline{t}) - x_B^L(\underline{t})k_2^U < \\ & \quad k_2^L x_B^*(\underline{t}) + x_B^U(\underline{t})k_2^* - k_2^L x_B^U(\underline{t}) \\ \left(-k_2^L, -x_B^U(\underline{t})\right) & \text{otherwise} \end{cases}
$$

$$
(\alpha_3,\beta_3) = \begin{cases} \left(-k_1^U, -x_A^U(\underline{t})\right) & \text{if } x_A^U(\underline{t})k_1^* + k_1^U x_A^*(\underline{t}) - k_1^U x_A^U(\underline{t}) > \\ & \quad x_A^L(\underline{t})k_1^* + k_1^L x_A^*(\underline{t}) - k_1^L x_A^L(\underline{t}) \\ \left(-k_1^L, -x_A^L(\underline{t})\right) & \text{otherwise} \end{cases}
$$

$$
(\alpha_4,\delta_2) = \begin{cases} \left(k_1^U, x_A^L(\underline{t})\right) & \text{if } x_A^L(\underline{t})k_1^* + k_1^U x_A^*(\underline{t}) - x_A^L(\underline{t})k_1^U < \\ & \quad k_1^L x_A^*(\underline{t}) + x_A^U(\underline{t})k_1^* - k_1^L x_A^U(\underline{t}) \\ \left(k_1^L, x_A^U(\underline{t})\right) & \text{otherwise} \end{cases}
$$

$$
(\beta_4,\gamma_2) = \begin{cases} \left(-k_2^U, -x_B^U(\underline{t})\right) & \text{if } x_B^U(\underline{t})k_2^* + k_2^U x_B^*(\underline{t}) - k_2^U x_B^U(\underline{t}) > \\ & \quad x_B^L(\underline{t})k_2^* + k_2^L x_B^*(\underline{t}) - k_2^L x_B^L(\underline{t}) \\ \left(-k_2^L, -x_B^L(\underline{t})\right) & \text{otherwise.} \end{cases}
$$

The final step in deriving the differential equations defining the pointwise in time

convex and concave bounds is to perform the optimization on the right hand sides thus constructing the differential equations themselves. I therefore have

$$\dot{c}_A = \inf_{\substack{\mathbf{z}\in\mathcal{C} \\ z_A=c_A(t)}} \mathcal{L}_{u_A}(t,\mathbf{z},\mathbf{k})\big|_{\mathbf{x}^*(t),\mathbf{k}^*} = u_A(t,\mathbf{x}^*(t),\mathbf{k}^*) + \alpha_1\left(c_A - x_A^*(t)\right) + \beta_1\left(k_1 - k_1^*\right)$$

$$\dot{c}_B = \inf_{\substack{\mathbf{z}\in\mathcal{C} \\ z_B=c_B(t)}} \mathcal{L}_{u_B}(t,\mathbf{z},\mathbf{k})\big|_{\mathbf{x}^*(t),\mathbf{k}^*} = u_B(t,\mathbf{x}^*(t),\mathbf{k}^*) + \min\{\alpha_2 c_A, \alpha_2 C_A\} - \alpha_2 x_A^*(t)$$
$$+\beta_2\left(c_B - x_B^*(t)\right) + \delta_1\left(k_1 - k_1^*\right) + \gamma_1\left(k_2 - k_2^*\right)$$

$$\dot{C}_A = \sup_{\substack{\mathbf{z}\in\mathcal{C} \\ z_A=C_A(t)}} \mathcal{L}_{o_A}(t,\mathbf{z},\mathbf{k})\big|_{\mathbf{x}^*(t),\mathbf{k}^*} = o_A(t,\mathbf{x}^*(t),\mathbf{k}^*) + \alpha_3\left(C_A - x_A^*(t)\right) + \beta_3\left(k_1 - k_1^*\right)$$

$$\dot{C}_B = \sup_{\substack{\mathbf{z}\in\mathcal{C} \\ z_B=C_B(t)}} \mathcal{L}_{o_B}(t,\mathbf{z},\mathbf{k})\big|_{\mathbf{x}^*(t),\mathbf{k}^*} = o_B(t,\mathbf{x}^*(t),\mathbf{k}^*) + \max\{\alpha_4 c_A, \alpha_4 C_A\} - \alpha_4 x_A^*(t)$$
$$+\beta_4\left(C_B - x_B^*(t)\right) + \delta_2\left(k_1 - k_1^*\right) + \gamma_2\left(k_2 - k_2^*\right)$$

and initial conditions

$$c_A(0) = C_A(0) = x_{A0},\ c_B(0) = C_B(0) = x_{B0}.$$

In order to compute a numerical solution for Example 6.18, the following numerical values were assigned

$$x_{A0} = 1,\ x_{B0} = 0,\ (k_1,k_2) \in [5,10]^2,\ t \in [0,1],$$

where concentration is in mol/L, the rate constants are given in min^{-1}, and time is given in minutes. The final step is the determination of a reference trajectory. Obviously, an uncountable number of reference trajectories exist, each one potentially generating different convex and concave bounding functions. In general, three rules should be obeyed in the selection of the reference trajectory. First, the state reference trajectory must be independent of the parameter, for if it is not, no guarantee exists for

the convexity or concavity of the resulting integrated functions. Second, the reference trajectories must be chosen such that they lie within the state and parameter bounds for all time. This statement is obvious since the reference trajectories are the points at which the convex and concave relaxations of the right hand sides of the differential equations are linearized; selecting a reference trajectory outside the range of validity of the relaxations is not valid. Finally, the reference trajectories, if possible, should be chosen such that they do not upset the numerical integrator. For bilinear terms, the right hand side convex and concave relaxations are defined by the maximum and the minimum of two functions respectively. If the midpoint of the states, defined as

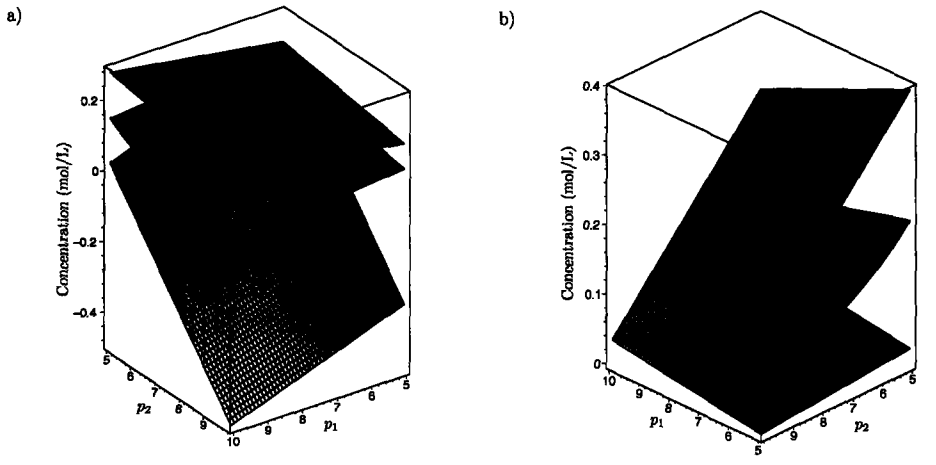
$$\mathbf{x}^{mid}(t) = (\mathbf{x}^{L}(t) + x^{U}(t))/2,$$

and the midpoint of the parameters are selected as the reference trajectory, then the derivatives of the relaxations for the bilinear terms do not exist. Effectively, this reference trajectory corresponds to the intersection of the two planes defining the relaxations. In principle, this problem is trivially remedied by arbitrarily selecting either the right or left hand derivative, for either choice yields a valid linearization. However, if instead, the residual routine for the numerical integration is defined with if statements as in the above analysis, chattering ensues and the integration fails. A remedy for this situation is addressed in detail in Chapter 7. For this example, I have chosen to present results from two different reference trajectories. Figure 6-6 illustrates the convex and concave bounds on the concentration of species B at $t = 0.5$ minutes.

6.3 Nonlinear Dynamic Relaxations and Branch-and-Bound Convergence

In this section, I combine the results of the previous two sections to illustrate that combining Corollary 3.7, Corollary 6.6, and Theorem 6.12 leads to a convergent branch-and-bound algorithm. As was discussed in Chapter 4, the branch-and-bound algo-

Figure 6-6: Convex underestimator and concave overestimator for Example 6.18. a) $(\mathbf{k}^*, \mathbf{x}^*) = (\mathbf{k}^L, \mathbf{x}^{mid})$ b) $(\mathbf{k}^*, \mathbf{x}^*) = (\mathbf{k}^U, \mathbf{x}^{mid})$



rithm possesses infinite convergence provided that the selection operation is bound improving, the bounding operation is consistent, and any node for which deletion by infeasibility is certain in the limit is fathomed (Theorem IV.3 [48]). Furthermore, I argued that the branch-and-bound algorithm was bound improving by the least lower bound heuristic and proving the bounding operation consistent reduced to proving that as the parameter space approaches degeneracy, the integral relaxation approaches the objective function pointwise. Given that the Euclidean space P is always refineable, the following theorem proves that the techniques established in this chapter lead to a consistent bounding operation for relaxing Problem 6.1 and hence an infinitely convergent branch-and-bound algorithm.

Theorem 6.19. *Consider the optimization problem given by*

$$\min_{\mathbf{p}\in P} L(\mathbf{p}) = \int_{t_0}^{t_f} \ell(t, \mathbf{x}(t,\mathbf{p}), \mathbf{p})\, dt$$

subject to

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{p})$$
$$\mathbf{x}(t_0, \mathbf{p}) = \mathbf{x}_0(\mathbf{p})$$

and the relaxation $U(\mathbf{p})$ defined by Corollary 3.7. Let the integrand $u(t, \mathbf{p})$ be defined

by Theorem 6.12, let $\mathbf{c}(t,\mathbf{p})$ *and* $\mathbf{C}(t,\mathbf{p})$ *be defined by Theorem 6.16, and let the state bounds be defined by Corollary 6.6. If the interval in any partition approaches degeneracy, then the lower bound in this partition (*$U(\mathbf{p})$*) converges pointwise to the objective function value (*$L(\mathbf{p})$*) in this same partition.*

Proof. Choose any partition and any fixed $\underline{t} \in (t_0, t_f]$ and let $[\mathbf{p}^L, \mathbf{p}^U]$ be the bounds on the parameter in this partition. From Corollary 6.6, as the interval $[\mathbf{p}^L, \mathbf{p}^U]$ approaches the degenerate value $\mathbf{p}^d$, the interval $[\mathbf{x}^L(\underline{t}), \mathbf{x}^U(\underline{t})]$ approaches the degenerate value of $\mathbf{x}^d(\underline{t})$. To be valid, the convex underestimator u_i and the concave overestimator o_i from Theorem 6.16 must themselves possess a consistent bounding operation and hence as $\mathcal{X}(\underline{t}) \times P$ shrinks to degeneracy, $u_i \uparrow f_i$ and $o_i \downarrow f_i$ for $i = 1, \ldots, n_x$. The right hand sides of the equations defining $\dot{c}_i$ and $\dot{C}_i$ are linearizations on u_i and o_i respectively. Since u_i and o_i are each approaching f_i, choosing $(\mathbf{x}^*(t), \mathbf{p}^*) = (\mathbf{x}^d(t), \mathbf{p}^d)$, I have that $g_{c,i}(\mathbf{c}, \mathbf{C}, \mathbf{p}) \uparrow f_i(\mathbf{x}, \mathbf{p})$ and $g_{C,i}(\mathbf{c}, \mathbf{C}, \mathbf{p}) \downarrow f_i(\mathbf{x}, \mathbf{p})$ since the linearization approaches the value of the function it approximates at the point of linearization. Now, suppose that at each step k, the interval $[\mathbf{p}^L, \mathbf{p}^U]_i$ is bisected (or reduced in some other manner) such that as $k \to \infty$, $[\mathbf{p}^L, \mathbf{p}^U]_k \to \mathbf{p}^d$ (the reduction is possible because the subdivision rule is exhaustive). By construction, I have the following sequence:

$$u_k \uparrow \ell \quad \text{as} \quad k \to \infty \quad \text{for } \underline{t} \in [t_0, t_f],$$

where the convergence arises because the McCormick underestimator u possesses a consistent bounding operation (with monotonic convergence) as $[\mathbf{p}^L, \mathbf{p}^U]$ approaches degeneracy. Because $\underline{t}$ was fixed arbitrarily, the convergence is true for all $t \in [t_0, t_f]$. By the monotone convergence theorem (Theorem 1, §2.3 [2]),

$$L(\mathbf{p}^d) = \int_{t_0}^{t_f} \ell \, dt = \lim_{k \to \infty} \int_{t_0}^{t_f} u_k \, dt = U(\mathbf{p}^d).$$

Because the partition was arbitrarily chosen, the convergence is applicable to any partition.

□

Proving the convergence of the branch-and-bound algorithm completes the development of the theory for optimizing Problem 6.1. The next three chapters address the implementation of the theory developed in this chapter and the application of the theory to solving numerical case studies.

Chapter 7

Implementation for Problems with Embedded Nonlinear Dynamics

In this chapter, I examine in detail the implementation I have developed for solving Problem 6.1. Because solving Problem 6.1 for all but the most trivial of problems is extremely computationally expensive, a great emphasis was placed on efficiency. For this reason, the entire program was developed in compiled, low level, systems programming languages (Fortran, C, and C++) as opposed to higher level, interpreted programming languages (e.g., Matlab, Java). When available, publicly available libraries were utilized, with preference given to libraries distributed with source code; the remainder of the code derives from custom designed and written source. The development platform was SuSE Linux 9.0 with kernel 2.4.21; gcc version 3.3.1 was utilized for compiling Fortran, C, and C++ source.

To promote code transparency, maintainability, and reuse, the numerical implementation is divided into four orthogonal software modules: branch-and-bound, local optimization, function evaluation, and ODE residual evaluation. The program combining these four distinct modules for solving Problem 6.1 has been termed the Global Dynamic Optimization Collection (GDOC). Branch-and-bound was performed utilizing libBandB, a fully functional branch-and-bound library I wrote for performing pure branch-and-bound and branch-and-bound with heuristics. Although I designed this module specifically for solving global dynamic optimization problems, the branch-

and-bound algorithm is by no means a new contribution. Therefore, the algorithm is not discussed here, and the reader is referred to Appendix A for the user's manual and application program interface (API) for libBandB. As with the linear implementation, the local optimization was performed exclusively with NPSOL [43]; a wrapper to NPSOL is distributed as part of libBandB. Numerical integration and sensitivity analysis were performed with CVODES [47], a dense, stiff ODE integration tool. However, CVODES does not explicitly handle integration events; thus, a discontinuity locking extension was written for CVODES. The final component of the numerical implementation is the ODE residual evaluator. The residual evaluator was implemented as a compiler that accepts problem input in a customized input language and outputs Fortran residual evaluation routines to be directly utilized with the discontinuity locking version of CVODES. The residual evaluator is responsible for implementing the theory developed in Chapter 6 for solving Problem 6.1. The remainder of this chapter is devoted to describing each component of GDOC in detail. The main emphasis is placed on the discussion of the numerical integration extensions to CVODES and the implementation of the nonlinear theory in the ODE residual evaluator.

7.1 Branch-and-Bound

At the inception of this project, I was unaware of any publicly available branch-and-bound libraries providing source code. Therefore, a custom branch-and-bound library, libBandB, was written in C++. The current version of the code is Version 3.2; the user's manual and API for this library are found in Appendix A. Because libBandB was designed to be an efficient, open source solution for implementing branch-and-bound rather than just a prototyping tool for implementing the nonlinear theory, the library provides several run-time configurable user options and heuristics to accelerate the convergence of the branch-and-bound algorithm. These heuristics represent the work of many researchers, in particular Sahinidis *et al.*, and the theoretical details of these heuristics are described thoroughly in [83]. However, unless otherwise noted, for comparison purposes, the case studies examined in this dissertation were

performed without using any branch-and-bound convergence heuristics. The method for selecting the next node on which to branch was always to select the node in the branch-and-bound tree with the least lower bound. At each node, the variable satisfying arg $\max_i |p_i^U - p_i^L|$ was selected for branching, and the branching was performed by bisection. The tolerances to which each problem was solved are unique to the individual problems and are noted in the chapters detailing the case studies.

7.2 Local Optimization

As with the linear implementation, local optimization of the original problem is utilized to provide upper bounds at each node, and local optimization of a convex relaxation is utilized to provide lower bounds at each node. In general, the most computationally expensive aspect to solving Problem 6.1 is the repeated numerical integrations required to provide both objective function values and gradients for the optimization. Thus, a sequential quadratic programming routine was chosen in order to limit the number of integration calls required to perform the optimization. Because most of the case studies examined in this thesis are small, dense problems, NPSOL version 5.0 [43] was utilized. NPSOL's optimization tolerance was always set at least two orders of magnitude smaller than the branch-and-bound tolerance and at least two orders of magnitude greater than the integration tolerance for each specific problem.

7.3 Function Evaluation

Conceptually, solving Problem 6.1 is identical to solving Problem 4.1. That is, I am confronted with the task of generating sequences of rigorous upper and lower bounds defined by differential equations. A branch-and-bound algorithm is then utilized to converge these two sequences. This section explains the numerical techniques employed to compute function and gradient evaluations for the upper and lower bounding problems.

7.3.1 Upper Bounding Problem

As previously stated, the upper bounding problem is defined as a local optimization of Problem 6.1 with the integral reformulated as an additional quadrature variable. I drop the ϕ term from the objective function for the remainder of the discussion, for I assume that this steady-state term is handled trivially via standard methods. A function evaluation for the upper bounding problem is given by

$$J(\mathbf{p}) = z_{ubp}(t_f, \mathbf{p})$$

where $z_{ubp}(t_f, \mathbf{p})$ is computed by numerically by solving the following system of ODEs:

$$\begin{aligned} \dot{z}_{ubp} &= \ell(t, \mathbf{x}(t, \mathbf{p}), \mathbf{p}) \\ z_{ubp}(0) &= 0 \\ \dot{\mathbf{x}} &= \mathbf{f}(t, \mathbf{x}, \mathbf{p}) \\ \mathbf{x}(t_0, \mathbf{p}) &= \mathbf{x}_0(\mathbf{p}). \end{aligned}$$

The gradient of $J(\mathbf{p})$ is computed via a sensitivity calculation with the staggered corrector [32] option of CVODES, where the right hand sides (RHS) of the sensitivity equations are computed via the CVODES finite differencing option. Theoretically, CVODES is capable of integrating any system of differential equations with Lipschitz continuous right hand sides. Except for the optimal control problems with piecewise constant control profiles, the upper bounding problems considered in this thesis all possess Lipschitz continuous RHS, and the integration proceeds smoothly. For optimal control problems, however, integration is explicitly reset at each discretization point, and the problem is numerically treated as a succession of Lipschitz continuous RHS differential equations. For problems truly possessing inherent discontinuities in the RHS of the differential equations, a discontinuity locking approach, as is employed for solving the lower bounding problem, may be applied to the upper bounding problem.

7.3.2 Lower Bounding Problem

In addition to the upper bounding problem, in order to solve Problem 6.1, I must solve a convex relaxation of the original optimization problem; the derivation of the convex relaxation is defined in the previous chapter. Again, the integral objective function is reformulated as a quadrature variable, and the lower bounding problem is given by

$$J(\mathbf{p}) = z_{lbp}(t_f, \mathbf{p})$$

where $z_{lbp}(t_f, \mathbf{p})$ is computed numerically by solving the following system of ODEs:

$$\begin{aligned}
\dot{z}_{lbp} &= \ell(t, \mathbf{x}^L(t), \mathbf{x}^U(t), \mathbf{c}(t, \mathbf{p}), \mathbf{C}(t, \mathbf{p}), \mathbf{p}) \\
z_{lbp}(0) &= 0 \\
\dot{\mathbf{x}}^L &= \mathbf{g}_{x^L}(t, \mathbf{x}^L, \mathbf{x}^U, \mathbf{p}) \\
\mathbf{x}^L(t_0, \mathbf{p}) &= \mathbf{x}_0^L \\
\dot{\mathbf{x}}^U &= \mathbf{g}_{x^U}(t, \mathbf{x}^L, \mathbf{x}^U, \mathbf{p}) \\
\mathbf{x}^U(t_0, \mathbf{p}) &= \mathbf{x}_0^U \\
\dot{\mathbf{c}} &= \mathbf{g}_{\mathbf{c}}(t, \mathbf{x}^L, \mathbf{x}^U, \mathbf{c}, \mathbf{C}, \mathbf{p}) \\
\mathbf{c}(t_0, \mathbf{p}) &= \mathbf{c}_0(\mathbf{p}) \\
\dot{\mathbf{C}} &= \mathbf{g}_{\mathbf{C}}(t, \mathbf{x}^L, \mathbf{x}^U, \mathbf{c}, \mathbf{C}, \mathbf{p}) \\
\mathbf{C}(t_0, \mathbf{p}) &= \mathbf{C}_0(\mathbf{p}),
\end{aligned}$$

where the subscript on $\mathbf{g}$ symbolizes an index, not a partial derivative. Strictly speaking, a lower bound can be generated without the original differential equations.

As with the upper bounding problem, the gradient of $J(\mathbf{p})$ is computed via a sensitivity calculation with the staggered corrector option of CVODES, where the right hand sides of the sensitivity equations are again computed via the CVODES finite differencing option. Numerical integration of the lower bounding problem is substantially more complicated because the RHS of the differential equations are, by construction, rarely Lipschitz continuous. The discontinuities in the RHS arise from

taking the partial derivatives of nonsmooth RHS convex and concave relaxations (in particular, from bilinear terms or relaxations requiring a mid function). For this reason, a discontinuity locking approach to numerical integration is employed.

7.3.3 Discontinuity Locking in CVODES

In this subsection, I discuss the higher level aspects of discontinuity locking that are performed in the integration routine itself, namely, event detection and event location. Here, I assume that both locked and unlocked residual evaluations are available to CVODES as necessary; the construction of locked and unlocked residual code is detailed below in Section 7.4. Rather than present examples in an arbitrary pseudocode, examples are presented in Fortran 77 and C++. The examples are relatively simple, however, and explicit knowledge of the language semantics should not be necessary to understand the examples.

Discontinuity locked integration and rigorous state event detection and location for arbitrary ODEs or differential-algebraic equations (DAEs) is a nontrivial task. The discussion here is not meant to provide a generalized method for state event location. Rather, the extensions to CVODES that I consider are specific to event location as it applies to numerically solving a lower bounding problem as constructed by the methods outline in Chapter 6. Park and Barton [72] provide an excellent overview of several different methods for state event location for DAEs and discuss a technique for rigorous state event location based on interval methods.

In numerical integration, an integration event occurs when some logical condition changes (i.e., which clause of a conditional statement in a program is executed changes), and the triggering of this logical condition alters the form of the RHS of the differential equations. I note that the events considered in this thesis are never conjunctive or disjunctive. That is, the logical expressions contain only one conditional and do not contain either an AND or an OR. If the RHS of the ODEs remain Lipschitz continuous through an event, then the error correction mechanism of the numerical integrator adjusts the integration stepsize to compensate for the change in the ODEs. However, if the RHS does not remain at least Lipschitz continuous, then

the numerical integrator often fails. Provided that the number of events that occurs is finite, if the location of the events can be accurately determined, then the system of differential equations can be integrated normally as a sequence of discontinuity locked differential equations that are at least Lipschitz continuous between the integration events. A discontinuity locked model is a realization of the original differential equations with all the conditional elements of the model (those parts of equations defined conditionally by the particular state of a logical operation) fixed in one particular clause. Each discontinuity locked realization is called a mode of the model. In any given mode, no conditionals exist, and the RHS of the ODEs are Lipschitz continuous by construction. The flipping of a logical condition between integration steps must then be detected, the event located, and the residual equations locked into a new mode corresponding to the appropriate realization of the model based on the new logical conditions. The integration routine is then reinitialized at the event, and integration resumes in the new discontinuity locked mode. Therefore, the numerical integrator itself must be capable of detecting events, locating events, and locking the residual routine into the appropriate mode.

Two classifications of events can occur: explicit time events and state events. Explicit time events are those events whose logical condition depends explicitly on the integration time. For example,

Example 7.1.

```
c An example of an explicit time event in Fortran 77
      if(t .ge. 1d0) then
        ! mode 1
        ...
      else
        ! mode 2
        ...
      end if
```

From the code snippet above, one sees that detection and location of explicit time

events is quite trivial. After CVODES has taken an integration step, if the current time, t_c, exceeds the event time, t_e, then an event has occurred. The current time is reset to $t_c = t_e + \varepsilon$ for some $\varepsilon > 0$, CVODES interpolation mechanism is utilized to reset the states at t_c (state continuity is always assumed over integration events), the residual routine is locked into the new mode, and integration resumes normally. t_c is set to $t_e + \varepsilon$ rather than t_e to avoid sticking at an event. That is, the integration resumes slightly after the event time to avoid repeatedly triggering the same event. An appropriate value for ε is left as an implementation detail.

The second classification of integration events, state events, are significantly more difficult to handle than explicit time events. As their name implies, state events are events whose logical condition depends upon the current state of the system. Unlike explicit time events, the timing of a state event is not known *a priori*. For example,

Example 7.2.

```
c An example of a state event in Fortran 77
      if(x(1) .ge. x(2)) then
        ! mode 1
        ...
      else
        ! mode 2
        ...
      end if
```

Because the location of a state event is not known at the outset of an integration, techniques must be developed to determine if an event has occurred between two integration mesh points, and if an event has occurred, locating its first occurrence. Further complicating the situation, in the general case, the timing of an event may even be parameter dependent, potentially causing jumps in the sensitivities at state events [37]. Fortunately, for the convex relaxations developed in Chapter 6, the proof of Theorem 6.16 demonstrates that parameter dependent state events do not occur for the lower bounding problem. Combined with state continuity at the events,

parameter independence of state event location is sufficient to establish that jumps in the sensitivity variables do not occur [37].

In addressing state events, researchers have typically taken two different approaches. In the first approach, the state dependent logical conditions are reformulated as new algebraic equations and appended to the system of ODEs or DAEs. Consider the logical condition of Example 7.2. The condition triggering the state event can be reformulated as a discontinuity function, $d = x(1) - x(2)$, and clearly, an event occurs when $d(t) = 0$. In general, a discontinuity function may be a function of time, states, state derivatives, algebraic variables, or parameters. After each integration step, an event detection routine determines if a zero crossing of the discontinuity function occurred during the step. If a zero crossing has occurred, then the event is located. For an integration method such as BDF [41], the interpolating polynomials approximating the state and algebraic variables between integration steps are available. Therefore, after completing an integration step, a polynomial approximation for $d(t)$ is reconstructed, and root finding methods are employed to locate the zero crossing. If multiple zero crossings occur, the earliest is sought. Elaborate techniques have been developed [72] to ensure rigorously proper event detection and location. Once the event's location has been determined, the event is "polished" to a time $t_e + \varepsilon$ (i.e., reinitialized at $t_e + \varepsilon$), the model is locked into the new mode, and integration resumes. The proper mode is determined by evaluation of an unlocked model to determine the active mode at the reinitialization time. An unlocked model evaluation, as opposed to a locked model evaluation, performs a residual evaluation without locking the conditionals into one of the integration modes. For DAEs, polishing is slightly more difficult than for ODEs because a consistent initialization calculation must be performed at the event. This method of state event location and detection has both advantages and disadvantages. One advantage is that the event location can be performed rigorously with the discontinuity equations themselves placed under error control. For typical engineering problems, the number of states vastly exceeds the number of discontinuity functions, and the method has been shown to be relatively cheap [72]. Unfortunately, Problem 6.1 tends to produce many

more discontinuities than states, and the cost of integrating the larger system becomes noticeable. Moreover, this method transforms ODEs with discontinuities into DAEs. CVODES cannot integrate DAEs; thus, a method for handling discontinuities without conversion to DAEs is preferred.

The second method for addressing state events does not require the addition of extra discontinuity functions. Instead, the state conditions are utilized directly. This method of event handling is attributed to Pantelides [70]. After each step in the locked integration, an unlocked residual evaluation is performed. If an event has occurred over the last integration step, then the mode resulting from the unlocked residual evaluation differs from the mode of the locked residual evaluation. That is, at least one of the logical conditions has changed over the last integration step. The location of the event is then computed via a bisection algorithm, the event is polished, and integration resumes in the new locked mode. This method of event location is not completely rigorous; however, the method is both efficient and satisfactory for the task at hand. The following example addresses event location and detection via bisection.

Example 7.3. Suppose, a system of differential equations possesses *nDisc* discontinuity functions, and for each discontinuity function, the associated condition of the RHS of the ODEs is representable by an integer (the details of this are discussed in Section 7.4). For each locked integration, I maintain an integer array, `int lockedMode[nDisc]`, storing the current mode and another integer array, `int unlockedMode[nDisc]`, containing the mode as determined by an unlocked evaluation. Furthermore, assume that an unlocked residual evaluation assigns the appropriate mode to the `unlockedMode` array. The following simplified C++ code enables event detection within this scheme (here, which discontinuity function triggers an event is unimportant):

```
bool event_occurred(int *lockedMode, int *unlockedMode){
  for(int i = 0; i < nDisc; i++){
    if(lockedMode[i] != unlockedMode[i]) return true;
```

```
  }
  return false;
}
```

Thus, an event is detected when the integration mode changes between integration steps. For efficiency, the function returns as soon as an event is detected in any of the discontinuity functions. The following simplified C++ subroutine locates the event via bisection. Here, I have assumed that the function `unlocked_model(double, int*)` returns the integration mode for the unlocked model at the specified time. Furthermore, I assume that `int nDisc` and `int tMidMode[nDisc]` are globally accessible without concern for memory allocation. In the actual implementation, `double event_time(double, double, int*)` is a member function, and `nDisc` and `tMidMode` are class members; memory management is handled by the class constructor and destructor.

```
double event_time(double tLeft, double tRight, int *tLeftMode){
  if(tRight < tLeft + tol) return tRight;
  double tMid = (tLeft+tRight)*0.5;
  unlocked_model(tMid, tMidMode);
  if(event_occurred(tLeftMode, tMidMode)){
    // event occurs between tLeft and tMid
    return event_time(tLeft, tMid, tLeftMode);
  }
  else{
    // event occurs between tMid and tRight
    for(int i = 0; i < nDisc; i++){
      tLeftMode[i] = tMidMode[i];
    }
    return event_time(tMid, tRight, tLeftMode);
  }
}
```

On first entry into function `event_time`, the integrator has already detected that an event has occurred between `tLeft` and `tRight`. The array `tLeftMode` holds the integration mode at `tLeft`. The algorithm terminates if `tRight` is less than `tLeft + tol`, where `tol` is the ε ensuring the integrator does not stick at the event. The function then computes the midpoint time and determines the integration mode at the midpoint. The function then computes if the event occurred between `tLeft` and `tMid` or between `tMid` and `tRight`. The function is called recursively until the location of the event is determined.

7.3.4 Event Chattering

The above methods for detecting and locating events are equipped to locate events where the discontinuity function equals zero only at a finite number of points. Unfortunately, the lower bounding problem often possesses discontinuity functions equal to zero over an interval of nonzero measure. For infinite precision calculations, no problems would arise, for the logical condition would always consistently choose one of the integration modes. For a finite precision machine, however, the discontinuity function is only zero within the integration tolerance. Therefore, at each successive integration step, the discontinuity function is computed as $d(t) = 0 \pm \delta$, where $\delta > 0$ is the absolute integration tolerance. Every time the sign of $d(t)$ changes, an event is triggered, and the integrator is forced to detect and locate the event. Integration then resumes at $t_e + \varepsilon$, and the same event is triggered again as soon as the sign of $d(t)$ changes again. The integrator can never take a step larger than ε, and the RHS of the differential equations may be arbitrarily locked into different modes at each integration step. Effectively, because of this chattering, the integration becomes numerically intractable.

For engineering problems possessing discontinuities deriving directly from the physics of the system, chattering may be more than just a numerical dilemma, for accurately modeling the physics of the system may demand selecting a specific integration mode. In solving Problem 6.1, however, the discontinuities in the lower bounding problem arise from the purely mathematical construct of deriving a convex

relaxation. By construction (Theorem 6.16), the RHS of the equations defining $\mathbf{c}$ and $\mathbf{C}$ are linearizations of the convex and concave relaxations, u_i and o_i respectively, of the RHS of the state differential equations. If $f_i(t, \mathbf{x}, \mathbf{p})$ possesses bilinear terms or univariate intrinsics such as the exponential function, then the functions u_i and o_i will possibly contain max, min, and mid functions. The linearizations defining the RHS of $\mathbf{c}$ and $\mathbf{C}$ will therefore involve derivatives of these nonsmooth functions. Depending upon the choice of reference trajectory, the logical conditions defining the partial derivatives of the max, min, and mid functions may cause chattering. Without loss of generality, I assume that I have the function $u = \max\{h_1, h_2\}$ deriving from the convex relaxation of a bilinear function f. Both h_1 and h_2 are themselves valid linear relaxations for f. Chattering ensues if the chosen reference trajectory implies that $h_1 = h_2$ (this is the intersection of the two planes). Because h_1 and h_2 are both valid, to prevent chattering, the integrator is simply locked into one mode selecting either h_1 or h_2. Which of the two modes is selected is arbitrary, but care must be taken to ensure that the selection is consistent over all occurrences of u. For the mid function, the equality of any two of the arguments implies that the middle value is either of these two equal arguments. Again, the selection is arbitrary provided consistency is maintained. Numerically, the integrator detects chattering when the same event is triggered on several successive integration steps in a small time interval. This method of preventing chattering is admittedly a heuristic. However, the heuristic is completely valid theoretically.

7.3.5 Exploiting Affine Structure

As previously mentioned, numerically solving the ODEs is the most computationally expensive aspect of solving Problem 6.1. Clearly, integrating the lower bounding problem is more expensive than integrating the upper bounding problem, for the lower bounding problem is both four times larger than the upper bounding problem (in number of state equations) and requires discontinuity locking. Each major iteration in the optimization routine requires an integration call because the lower bound is parameter dependent. However, for problems only requiring the objective function at

fixed time points (i.e., the objective function does not contain an integral), the affine structure of $\mathbf{c}$ and $\mathbf{C}$ may be exploited to reduce the number of required integrations. In order to compute the lower bound at a fixed time $\underline{t}$, I must be able to compute $\mathcal{X}(\underline{t})$, $\mathbf{c}(\underline{t}, \mathbf{p})$, and $\mathbf{C}(\underline{t}, \mathbf{p})$. By definition, the state bounds are parameter independent; therefore, they need not be recomputed for each major iteration of the optimizer. Furthermore, because $\mathbf{c}$ and $\mathbf{C}$ are both affine, they have the following structure (see Chapter 3):

$$\begin{aligned} \mathbf{c}(\underline{t}, \mathbf{p}) &= \mathbf{M}_c(\underline{t})\mathbf{p} + \mathbf{n}_c(\underline{t}) \\ \mathbf{C}(\underline{t}, \mathbf{p}) &= \mathbf{M}_C(\underline{t})\mathbf{p} + \mathbf{n}_C(\underline{t}). \end{aligned}$$

Thus, by integrating once and subsequently storing the state bounds, $\mathbf{M}_c(\underline{t})$, $\mathbf{M}_C(\underline{t})$, $\mathbf{n}_c(\underline{t})$, and $\mathbf{n}_C(\underline{t})$, $\mathbf{c}(\underline{t}, \mathbf{p})$ and $\mathbf{C}(\underline{t}, \mathbf{p})$ can be computed via linear algebra instead of numerical integration on future major iterations of the optimizer at a particular node. The computation of $\mathbf{M}_c(\underline{t})$ and $\mathbf{M}_C(\underline{t})$ is effectively free, for they are simply the parametric sensitivities, which are already computed for the calculation of the gradient of the lower bound. The values $\mathbf{n}_c(\underline{t})$ and $\mathbf{n}_C(\underline{t})$ can be computed either by augmenting the original system with an additional set of differential equations or via linear algebra. The error in $\mathbf{c}(\underline{t}, \mathbf{p})$ and $\mathbf{C}(\underline{t}, \mathbf{p})$ introduced by the linear algebra calculation is negligible if the problem is scaled such that $p_i^U - p_i^L$ is order one or less for $i = 1, \ldots, n_p$.

An interesting aspect of exploiting the affine nature of the relaxations for solving the lower bounding problem is that this technique also enables one to compute an upper bound for the problem without performing any integration. From Theorem 6.12, once $\mathbf{c}(\underline{t}, \mathbf{p})$ and $\mathbf{C}(\underline{t}, \mathbf{p})$ have been computed, the only additional information required for constructing a concave overestimator for a state composition is determining z_{max}. A rigorous upper bound could then be computed by maximizing the concave overestimator for the objective function, and I note simply that the upper bound converges to the objective function value in an analogous manner to the convergence of the lower bound to the objective function. Furthermore, because the state relaxations

are affine, depending on the structure of the objective function, Problem 6.1 may be completely reducible to solving a sequence of converging upper and lower bounding linear programs requiring only one integration per branch-and-bound node. However, because solving the original problem locally is relatively inexpensive computationally, periodically solving the original problem locally provides a better incumbent by which to fathom in the branch-and-bound algorithm hence accelerating the convergence of the algorithm.

7.4 Residual Evaluation

The residual evaluation is the most important implementation aspect of solving Problem 6.1, for the residual evaluation defines the construction of the upper bound, the state bounds, and the lower bound to the problem. Because the upper bound is obtained as a local optimization, the residual for the upper bound is trivially defined by the integrand (remembering that the integral has been reformulated as a quadrature variable) and the RHS of the embedded nonlinear differential equations. The construction of the residual for the lower bounding problem, however, is a much more difficult task, for differential equations must be derived defining $\mathbf{x}^L$, $\mathbf{x}^U$, $\mathbf{c}$, $\mathbf{C}$, and z_{lbp} as discussed theoretically in the previous chapter. One method for constructing the lower bounding differential equations is to derive analytically the necessary information and code the residual routine manually. However, the application of the theory discussed in Chapter 6 is both tedious and error-prone for all but the simplest problems thus making manual derivation of the lower bounding problem infeasible. Fortunately, the theory lends itself naturally to automation.

Two different approaches are viable for automating the theory required for relaxing Problem 6.1: operator overloading and code generation. In the operator overloading approach, the implementer defines class objects for which the standard operators of arithmetic and assignment are overloaded via special language facilities in order to perform various mathematical operations. The compiler is then responsible for applying the appropriate operations based on the runtime type identification (RTTI)

of the objects (class instantiations) participating in the equations. The following example illustrates C++ operator overloading for interval addition.

Example 7.4. Let the following be the definition of a very simple class for interval arithmetic:

```
class Interval{
public:
  inline Interval(double, double);
  inline Interval(Interval&);
  Interval& operator+(Interval&);
  double zL, zU;
};
```

Assume that suitable constructors `Interval(double, double)` and `Interval(Interval&)` have been implemented. `operator+` is given by

```
Interval& Interval::operator+(Interval &rhs){
  xL = xL + rhs.xL;
  xU = xU + rhs.xU;
  return *this;
}
```

By overloading `operator+` for the Interval class, the compiler interprets the symbol "+" to behave like interval addition when it identifies both operands of "+" to be Interval objects at runtime. The following program creates three Intervals `x`, `y`, and `z` and assigns `z` the result of the overloaded addition of Intervals `x` and `y`. At program completion, `z.xL` = 1.0 and `z.xU` = 11.0.

```
int main(int argc, char **argv){
  Interval x(-1.0, 4.0);
  Interval y(2.0, 7.0);
  Interval z = x + y;
  return 0;
```

```
}
```

Of course, all of the arithmetic operators could be overloaded to create a more complete Interval class.

For simple classes where arithmetic operations make intuitive sense, like the Interval class defined above, operator overloading is an invaluable tool for data abstraction. Unfortunately, the C++ language permits terrible abuses of operator overloading. Essentially, any operator may be overloaded to do whatever the programmer wishes, provided the overloaded operator matches the function prototype allowed by the language. For extremely complex operations that one would not naturally associate with a single given operator, operator overloading can lead to extremely obfuscated code maintainable by the original author at best. Furthermore, while operator overloading permits a high degree of data abstraction, this abstraction does not come without a price. First, for a small operation count, the RTTI overhead is unnoticeable. For millions upon millions of operations, as is required to solve Problem 6.1, the RTTI overhead can become appreciable. Second, each overloaded operation requires a function call rather than a primitive machine instruction. Third, the overhead of using a language implementing operator overloading is more expensive than simply writing the lower bounding equations in a simple language such as Fortran. Finally, symbolic manipulation of equations is extremely difficult, if not impossible, with an operator overloaded approach; therefore, discontinuity locking cannot be performed. For these reasons, a code generation technique was chosen for GDOC.

7.4.1 A Domain Specific, Declarative Minilanguage

One of the main differences between an operator overloaded approach and a code generation approach is the method of input. In an operator overloaded approach, the user writes the model in the compiled language that implements the overloading. In a code generation approach, the user writes the input in a customized input language, usually in an input file. The file is then scanned and parsed into internal data structures, symbolic manipulation is performed on the data structures, and the

output is written in a compiled language. The input to GDOC is a small input file in a customized input language, and the output of GDOC is the upper and lower bounding problems written in Fortran 77.

My intent in this subsection is not to explain compiler theory or computer language grammar; the interested reader is referred to [6] for an introduction to these topics. Instead, this subsection focuses on the particulars of the language, and briefly, the method utilized to scan and parse the input file. The input file is divided into five required input sections and three optional input sections. The required inputs are declaration, parameter values, equation, reference, and initial. In the declaration section, the state variables, the parameter variables, constant variables, and integration time are all defined. In the parameter values section, the bounds on the parameters are set, and an initial guess for the root node is given. In the equation section, the dynamics of the problem are defined by semicolon terminated equations. The reference section defines the reference trajectory for the problem in terms of the state variable and parameter bounds. Finally, the initial section defines the initial conditions for the state variables. The three optional sections are constant values, natural bounds, and integrand. The constant values section simply defines any numerical constants declared for the problem. The natural bounds section defines the set $\overline{\mathcal{X}}$ as defined in Chapter 6. The integrand section defines an integrand for problems with integral objective functions. Example 7.5 below illustrates the input language; the input file in the example is the actual input file utilized for solving the problem defined in Section 8.3.

Example 7.5.

```
# Input file for catalytic cracking problem
# The '#' symbol represents comments

# This section declares the state and its size, the parameter and its,
# size, and the integration time.
DECLARATION
  state: x(1:2)
```

```
  parameter: p(1:3)
  time: [0,0.95]
END

# This section defines the parameter ranges and an initial guess for
# the root node.
PARAMETER VALUES
  p(1)=0:[0,20]
  p(2)=4:[0,20]
  p(3)=7:[0,20]
END

# This section defines the dynamic system. The '$' symbol represents
# the time derivative.
EQUATION
  $x(1)=-(p(1)+p(3))*x(1)^2;
  $x(2)=p(1)*x(1)^2-p(2)*x(2);
END

# This section defines the reference trajectory.
REFERENCE
  xRef(1)=(xL(1)+xU(1))*0.5;
  xRef(2)=(xL(2)+xU(2))*0.5;
  pRef(1)=(pL(1)+pU(1))*0.5;
  pRef(2)=(pL(2)+pU(2))*0.5;
  pRef(3)=(pL(3)+pU(3))*0.5;
END

# This section defines the initial condition for the state.
INITIAL
```

```
  x(1)=1;
  x(2)=0;
END

# This problem does not contain an integral objective function. If the
# objective function were an integral, the objective function would
# be defined here.
#INTEGRAND
  # integrand here
#END

# This section defines the natural bounds. This section does not require
# nx equations. If only one bound is known, the symbol '--' may be used.
# For example, x(1):[--,1] is a valid input.
NATURAL BOUNDS
  x(1):[0,1]
  x(2):[0,1]
END
```

The input language also permits the declaration of symbolic constants and a corresponding section defining numerical values for these constants.

Rather than write a custom lexical analyzer and parser, the programs lex and yacc [63] (or rather their open source counterparts flex and bison) were utilized for this task. Lex is a scanner generator built on the concepts of regular expressions. The function of the scanner is to read the input file and generate tokens that the parser can utilize and recognize as a language grammar. Yacc is an LALR (look ahead left right) parser generator utilized for parsing a context free grammar (see [6]). A yacc generated parser accepts a tokenized input from lex, recognizes the grammar of the input language, and executes semantic routines to convert the input file into usable internal data structures. The yacc grammar is included in Appendix B. The GDOC compiler stores equations as parse trees; the symbolic operations necessary

to implement the relaxation theory from Chapter 6 are all performed on these parse trees and are explained in the following subsection.

7.4.2 Parse Trees and Symbolic Manipulation

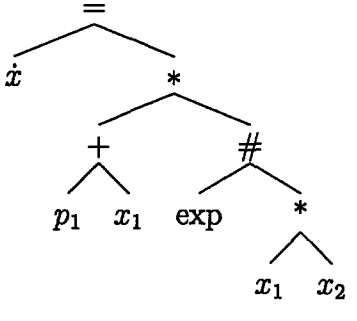
Although parse trees can be implemented for many different kinds of language statements, GDOC utilizes parse trees exclusively to store and symbolically manipulate equations. In this context, a parse tree for an equation is a binary tree representation that stores each token of an equation at a distinct node. The following example demonstrates the construction of a parse tree from a differential equation.

Example 7.6. Consider the following equation:

$$\dot{x} = (p + x_1)\exp(x_1 x_2). \tag{7.1}$$

The parse tree for this equation is illustrated in Figure 7-1. The # symbol represents the presence of a unary function, in this case, the exponential function.

Figure 7-1: Parse tree for Equation 7.1



In general, mathematical operations on parse trees are performed by recursively walking the binary tree and performing various operations at each node. Three different methods for walking a binary tree are implemented in GDOC: preorder traversal, inorder traversal, and postorder traversal. In a preorder traversal, an operation is performed at a node before walking either branch of the tree. In an inorder traversal,

an operation is performed at a node after walking the left branch of the tree but before walking the right branch of the tree. In a postorder traversal, the operation at a node is performed after walking both directions of the tree. The following code example illustrates these three methods for walking a binary tree.

Example 7.7. Consider the following binary tree class, where `left = right = NULL` for leaf nodes:

```
class BinaryTree{
public:
  inline BinaryTree(BinaryTree *left, BinaryTree *right);
  void do_something_at_node();
  BinaryTree *left;
  BinaryTree *right;
};
```

The following function implements a preorder tree traversal:

```
void preorder_traversal(BinaryTree *node){
  if(node != NULL){
    node->do_something_at_node();
    node->preorder_traversal(node->left);
    node->preorder_traversal(node->right);
  }
  return;
}
```

The following function implements an inorder tree traversal:

```
void preorder_traversal(BinaryTree *node){
  if(node != NULL){
    node->preorder_traversal(node->left);
    node->do_something_at_node();
    node->preorder_traversal(node->right);
```

```
  }
  return;
}
```

The following function implements a postorder tree traversal:

```
void preorder_traversal(BinaryTree *node){
  if(node != NULL){
    node->preorder_traversal(node->left);
    node->preorder_traversal(node->right);
    node->do_something_at_node();
  }
  return;
}
```

Of course, operating on a binary tree may not be quite as simple as stated above, for the `do_something_at_node()` member function may alter the state of the tree, or the tree walk itself may not be merely a void function. The following subsections illustrate how parse trees are utilized to construct convex relaxations for Problem 6.1.

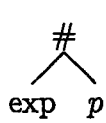
7.4.3 An Inheritance Hierarchy

The actual design of parse trees in GDOC is polymorphic in nature. Essentially, the base class for the parse tree implements the binary tree structure and the ability to walk the tree. The nodes themselves represent derived classes from the BinaryTree class. For example, Variables, Numbers, Operators, and Functions all represent different derived classes from the BinaryTree class. A further level of specialization yields Addition, Subtraction, Multiplication, etc. as derived classes of Operators; the complete implementation has several levels of derived classes. In this manner, the required mathematical operations on the binary trees (i.e., the function `do_something_at_node()`) can be implemented polymorphically through C++'s virtual function mechanism. This design promotes data abstraction and code transparency. The algorithms necessary for constructing convex relaxations for Problem

6.1 are implemented at the parse tree level, while the details of the implementation for each derived class are delegated to the derived class member functions. The following subsections explain the basic concepts underlying the implementation of the theory presented in Chapter 6; the actual implementation is substantially more sophisticated than the below presentation.

7.4.4 An Implementation of State Bounds

In GDOC, the state bounds are computed via Corollary 6.6. As stated in the discussion following Corollary 6.6, the parametric optimization problems defining the state bounding differential equations are overestimated utilizing interval arithmetic. Because interval arithmetic requires both upper and lower bounds simultaneously, the state bounding function computes both in parallel. The parse tree is walked in a postorder traversal, and at each node, `do_something_at_node()` is responsible for creating both the upper and lower bound for the current node. In any valid equation parse tree, the leaf nodes are either numbers, parameters, or state variables. For a number, both its upper and lower bounds are simply copies of the number. Given a parameter p, its bounds are simply given by new parameters p^L and p^U. Computing state bounds for variables is slightly more complex. First, from Corollary 6.6, in the ith equation, the $z_i = x_i^L$ constraint is active for the lower bounding differential equation, and the $z_i = x_i^U$ constraint is active for the upper bounding differential equation. Therefore, the variable x_i is treated as a constant rather than an interval. For the remaining state variables x_j $(j \neq i)$, the lower bound is given by $\max\{x_j^L, \overline{x}_j^L\}$, and the upper bound is given by $\min\{x_j^U, \overline{x}_j^U\}$. The set $\overline{X}$ from which $\overline{x}_j^L$ and $\overline{x}_j^U$ are derived is defined by the user in the input file. Each interior node of the parse tree is either a unary function or a binary operation. For a unary function, the lower bound is found by applying interval rules for the function. For example, the function $\exp(p)$ is represented by the following parse tree:



Because the bounds are generated by a postorder traversal, when operating on the node containing #, the left and right child nodes have already been bounded (obviously, no operation is necessary at the left node). Because exp is a monotone function, the bounds generated at # are

$$N_{\#}^{L} = \exp(R_{\#}^{L}) = \exp(p^{L}) \qquad \text{and} \qquad N_{\#}^{U} = \exp(R_{\#}^{U}) = \exp(p^{U})$$

where $R_{\#}$ represents the pointer to the right child node of node $N_{\#}$ (analogously, I use $L_{\#}$ to represent the pointer to the left child node of node $N_{\#}$). For binary operations, the rules for interval arithmetic are applied. For example, given the following parse tree:

the bounds for N_{-} are given by the standard rules of interval subtraction:

$$N_{-}^{L} = L_{-}^{L} - R_{-}^{U} = x_2^{L} - p^{U} \qquad \text{and} \qquad N_{-}^{U} = R_{-}^{U} - L_{-}^{L} = x_2^{U} - p^{L}.$$

In this manner, Corollary 6.6 is satisfied recursively, and the residuals of the state bounding differential equations are symbolically generated.

7.4.5 Convex and Concave Relaxations of the RHS

The first step in applying Theorem 6.16 to derive differential equations defining $\mathbf{c}$ and $\mathbf{C}$ is to derive convex and concave relaxations for $\mathbf{f}$ on the set $\mathcal{X}(\underline{t}) \times P$ pointwise in time. The set P is available directly from the branch-and-bound routine, and the set $\mathcal{X}(t)$ is obtained from solving the differential equations defining the state bounds. Theorem 6.12 is applied recursively in order to derive convex and concave relaxations for each f_i $(i = 1, \ldots, n_x)$. In general, applying Theorem 6.12 first requires recursively factoring f_i into sums and products of nested univariate compositions. By inspection, however, a parse tree representation of a function is a recursive factorization of the function. Each leaf node represents the innermost composition variables of the

factorization. Each parent node is then itself a univariate composition, the sum of composition variables, or the product of composition variables (or can trivially be reformulated as such). These univariate compositions, sums, or products are then in turn interpreted as composition variables, and the procedure recursively repeats until reaching the root node of the parse tree. Given that a parse tree representation is equivalent to a factorization of the equation, Equation 7 in [65] can be applied to the parse tree in a postorder traversal analogously to applying Corollary 6.6 as discussed above for deriving the residuals of the state bounding differential equations. I note only that at the leaf nodes, $c(x) = C(x) = x$ for state variables and $c(p) = C(p) = p$ for parameters.

One of the more complicated aspects of applying Equation 7 in [65] is that the functional form of the bilinear relaxation depends on the sign of the bounds. Unlike in the operator overloaded approach, relaxations are computed symbolically at compile time rather than numerically at runtime. However, the sign of the bounds is unknown until runtime. Therefore, special conditional variables must be utilized in this context. Furthermore, because the sign of the bounds may change as a function of time, these conditional variables must be discontinuity locked.

7.4.6 RHS Linearizations

In order to apply Theorem 6.16, linearizations of the convex and concave relaxation parse trees must be generated at a given reference trajectory. Generating linearizations can be broken into two tasks. The first task is computing the partial derivative of a parse tree, and the second task is the operation of substituting the reference trajectory. Variable substitution is trivially performed via any walk of the parse tree where `do_something_at_node()` is a polymorphic, comparison member function. Therefore, numerically implementing linearizations effectively reduces to symbolically computing the partial derivatives of a parse tree.

Taking the partial derivative of a parse tree is simply an application of the chain rule applied to a binary tree walked via a postorder traversal. Suppose I am taking the partial derivative of the function $u(\mathbf{x}, \mathbf{p})$ with respect to variable x_i. As previously

stated, the leaf nodes are either numbers, parameters, or state variables. In taking derivatives, no distinction is made between state variables and parameters, and I consider both to be variables in this context. When the leaf node is a number, the partial derivative is zero. For a variable, the following formula is applied:

$$\frac{\partial x_j}{\partial x_i} = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

For the interior nodes of the parse tree, the chain rule is applied. For example, suppose the interior node of parse tree were multiplication. Then, for N_*, the partial derivative with respect to variable x_i would be given by

$$\frac{\partial N_*}{\partial x_i} = R_* \frac{\partial L_*}{\partial x_i} + L_* \frac{\partial R_*}{\partial x_i}.$$

Because the derivative operation is performed recursively via a postorder tree traversal, both $\partial L_*/\partial x_i$ and $\partial R_*/\partial x_i$ are already available when taking the partial derivative of N_*. The following example examines taking the derivative of a relatively simple function.

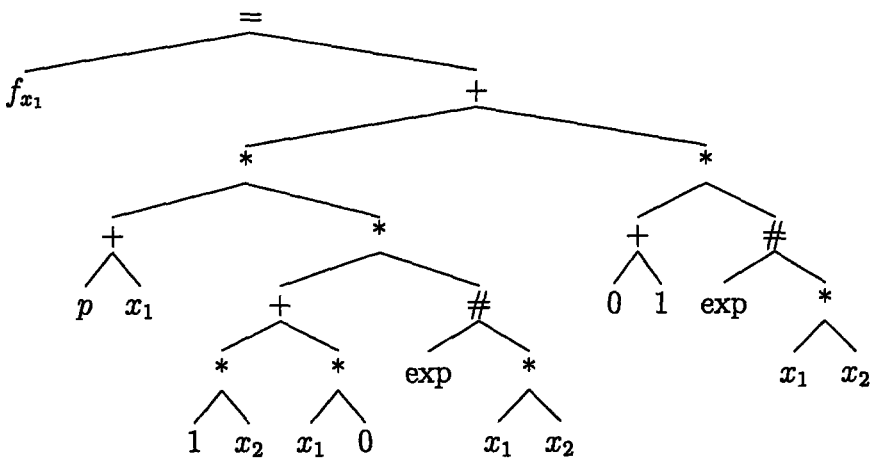
Example 7.8. Consider the function

$$f(\mathbf{x}, p) = (p + x_1) \exp(x_1 x_2)$$

with parse tree given in Figure 7-1. The above methodology is applied to taking the partial derivative of f with respect to x_1. The resulting parse tree of the partial derivative is found in Figure 7-2. Of course, in the actual implementation, terms such as $1 * x_2$ and $x_1 * 0$ are reduced to x_2 and 0 respectively. These reductions are performed recursively such that after the derivative operation, the resulting parse tree does not contain any redundant mathematical operations

Due to the inherent nonsmoothness of McCormick's relaxations, many of the functions for which the derivative is required are nonsmooth. For nonsmooth functions,

Figure 7-2: Partial derivative of f with respect to x_1 for the parse tree in Figure 7-1



the derivative is divided into cases. For example, the partial derivative of the term $f(\mathbf{x}) = \max\{u(\mathbf{x}),\ v(\mathbf{x})\}$ with respect to x_1 is given by

$$\frac{\partial f(\mathbf{x})}{\partial x_1} = \begin{cases} \partial u(\mathbf{x})/\partial x_1 & \text{if } u(\mathbf{x}) \geq v(\mathbf{x}) \\ \partial v(\mathbf{x})/\partial x_1 & \text{otherwise} \end{cases}$$

These discontinuous derivative terms are discontinuity locked.

7.4.7 Constructing c and C

Once methods have been established both for generating relaxations and for taking partial derivatives of these relaxations, applying Theorem 6.16 is a relatively trivial task. First, convex and concave relaxations for the RHS of each function f_i are generated. Given the user defined reference trajectory, a new parse tree is generated by applying Equation 6.7 to generate the RHS for the equations defining $\mathbf{c}$ and $\mathbf{C}$. Because the parametric optimization problems defining the RHS of these differential equations are linear programs, the infimum and supremum are replaced by the solution of the linear programs as is illustrated in the proof of Theorem 6.16. This completes the construction of differential equations defining $\mathbf{c}$ and $\mathbf{C}$.

7.4.8 Reducing the Number of Discontinuities

In order to compute the residual evaluation in the most efficient possible manner, the number of discontinuities in the system should be reduced to a minimum. Discontinuities arise due to two separate phenomena: generating convex and concave relaxations and taking derivatives of nonsmooth functions. Very frequently, discontinuities are generated that have identical logical conditions but different execution clauses. In particular, this situation arises because by Equation 6.7, the derivative of the same nonsmooth relaxation is taken several times with respect to different variables. Each time the derivative of a max, min, or mid function is taken, the logical conditions are duplicated. Rather than repeatedly evaluating and locking identical logical expressions, the execution clauses of discontinuities with identical logical expressions are symbolically combined before the residual code is generated. Additionally, any completely redundant discontinuity variables are eliminated at this time.

7.4.9 The Mechanism of Locking the Model

In order to implement discontinuity locking, two different approaches exist. In the first approach, only one residual evaluation routine exists, and this routine is responsible for returning both locked and unlocked model evaluations. In the second implementation, separate subroutines are provided for locked and unlocked residual evaluations. For the present implementation, I have chosen the latter approach. Although this method duplicates source code, the runtime efficiency of the code is faster because only the necessary information for a either a locked or unlocked model is computed. As alluded to in the discussion of discontinuity locking in CVODES, locking is implemented by an integer array, where each entry in the array represents one of the modes of the discontinuity function. In the unlocked evaluation, the lock array is an output, and the conditional statement evaluated is the original logical expression; the result of selecting a logical branch is simply to set the appropriate mode in the lock array. In a locked model evaluation, the locked array is an input containing the locked integration mode; the conditional for evaluating which logical branch of

a discontinuity function is evaluated is based solely on the lock array. The following example illustrates the locking of a discontinuity variable.

Example 7.9. Suppose discontinuity number seven is generated from the partial derivative with respect to p of the following equation fragment:

$$u = \max\{x^L p + p^U x - x^L p^U,\ x^U p + p^L x - x^U p^L\}.$$

The following code excerpt would be from the unlocked model:

```
c discontinuity function 7 (unlocked model)
c locks code into appropriate clause
      if(xL*pref+pU*xref-xL*pU .ge. xU*pref+pL*xref-xU*pL) then
        lockArray(7) = 0
      else
        lockArray(7) = 1
      end if
```

The corresponding code fragment in the locked model would be

```
c discontinuity function 7 (locked model)
c executes the appropriate clause to correctly evaluate the
c partial derivative
      if(lockArray(7) .eq. 0)
        tempVar(7) = xL
      else
        tempVar(7) = xU
      end if
```

I note that utilizing an integer array for the locking as opposed to a logical array is necessary in order to handle discontinuities with more than two logical branches. This situation arises from taking the derivative of a mid function.

7.4.10 The Mechanism for Preventing Chattering

As previously discussed, detecting chattering is a task assigned to the discontinuity handling component of the integrator. When the integrator detects chattering, the model is arbitrarily locked into one of the modes causing the chattering. However, once chattering is detected, the event should no longer be detected at successive integration steps. Naïvely, one might simply ignore any events occurring on a discontinuity known to be chattering. However, this ignores the possibility that the chattering might eventually cease and a real event might occur in the future for this discontinuity function. Therefore, in the unlocked model, a quantity an order of magnitude larger than the integration tolerance is added to one side of the logical condition when chattering is detected. Essentially, after chattering has been detected, the unlocked model is tricked into reporting no event occurring because the two sides of the logical argument will no longer be equal. However, if a real event occurs in the future, the unlocked model still detects this fact, the event is located, and the chatter locking mechanism is reset. In practice, one side of each conditional is augmented by the addition of an element of a chatter locking array of double precision variables. Under normal operating conditions, this chatter locking factor is set identically to zero. If chattering is detected, this value is set to a value an order of magnitude larger than the integration tolerance. The following example illustrates the chatter locking mechanism for the same discontinuity variable described above in Example 7.9.

Example 7.10. Consider the discontinuity variable as defined above in Example 7.9. The following code fragment for the unlocked model implements chatter locking.

```
c discontinuity variable 7 (unlocked model)
      if(xL*pref+pU*xref-xL*pU .ge. xU*pref+pL*xref-xU*pL
     &   + chatterLock(7)) then
        lockArray(7) = 0
      else
        lockArray(7) = 1
```

```
end if
```

As previously stated, under normal operation, `chatterLock(7)` is set identically to 0.0d0. After chattering has been detected, `chatterLock(7)` is set equal to `10*iTol`.

7.4.11 Limitations of the Current Compiler and the Next Generation

For simple to moderately difficult problems, the compiler described in this chapter performs adequately for generating residual routines to relax Problem 6.1; for problems with symbolically complex differential equations, a limitation of the method quickly emerges. Taking derivatives or deriving relaxations of individual equations via parse trees generates reasonably efficient code. However, when these operations must be repeated recursively several times on symbolically complex equations, the complexity of the generated code grows exponentially because common subexpressions are not exploited. Instead, identical subexpressions are repeated at each occurrence in a given binary tree, and identical subexpressions are repeated between different equations. In some instances, applying Theorem 6.16 has generated over one hundred lines of fully dense Fortran code for the relaxation of a single equation, and for very small input files, hundreds of thousands of total lines of Fortran have been generated! Clearly, an alternative technique is sought.

The explosion of operation count in utilizing binary parse trees to perform symbolic mathematics is not unique to applying Theorem 6.16. Fortunately, the problem has been largely addressed by algorithmic differentiation [44] and similar techniques for automatically generating convex relaxations [40]. In these two techniques, symbolic manipulations of the equations are performed via computational graphs. The methods exploit common subexpressions and provide theoretical upper bounds for the complexity of the generated code as a function of the complexity of the original equations. A second generation compiler implementing algorithmic differentiation techniques is currently in progress.

Chapter 8

Case Studies for Literature Problems with Nonlinear Dynamics

In this chapter, I apply the theory developed in Chapter 6 and the implementation developed in Chapter 7 to solve global optimization problems with nonlinear dynamics embedded that have appeared in the literature. Unless otherwise noted, no branch-and-bound heuristics are utilized to accelerate convergence. For point constrained problems, the linear structure of the convex relaxation is always exploited, and the lower bounding integration is only performed for the first optimization function call at a given node. The computations were all performed on an AMD Athlon XP2000+ operating at 1667 MHz, and all code was compiled with gcc 3.3.1.

One of the most challenging aspects of solving Problem 6.1 globally is the presence of nonquasimonotone differential equations in the embedded dynamics. As previously stated, applying standard techniques for generating state bounds ([45] and [93]) to nonquasimonotone differential equations leads to bounds that rapidly explode on short time scales (e.g., Example 6.11). That is, on typical time scales of interest, the state bounds generated by standard techniques exponentially approach infinity. Because generating convex relaxations for Problem 6.1 requires bounds on the states, state bounds approaching infinity lead to convex relaxations approaching

negative infinity. Although one can prove that even these exploding state bounds for nonquasimonotone systems approach degeneracy as the bounds on P approach degeneracy, the numerical implementation for solving Problem 6.1 requires finite bounds at all branch-and-bound nodes.

As has been thoroughly discussed in Chapter 6, given a natural bounding set $\overline{\mathcal{X}}$, Corollary 6.6 is utilized to ensure that the state bounds do not explode. In Chapter 7, I discussed the implementation of Corollary 6.6 in GDOC. A brief explanation of the application of Corollary 6.6 for the case studies in this chapter is provided here to enable the reproducibility of the results in this chapter in an implementation independent fashion.

In general, solving the parametric optimization problems defining the right hand sides of the state bounding differential equations is prohibitively expensive. Instead, the solution to the optimization problems is relaxed via interval extensions of the right hand sides of the differential equations. For the lower [upper] bounding differential equation i, the equality constraint $z_i = x_i^L$ $[z_i = x_i^U]$ is enforced. For all other state variables $j \neq i$, natural interval arithmetic is applied, where the bounds for variable j are given pointwise in time by

$$\mathcal{X}_j(\underline{t}) \cap \overline{\mathcal{X}}_j(\underline{t}) = \left[\max\{\overline{x}_j^L(\underline{t}),\ x_j^L(\underline{t})\},\ \min\{\overline{x}_j^U(\underline{t}),\ x_j^U(\underline{t})\}\right] \qquad \forall\ \underline{t} \in (t_0, t_f].$$

Additionally, the derivative of the state bound is set to zero when the state bound exceeds its natural bound. Therefore, by construction, $\mathcal{X} \subseteq \overline{\mathcal{X}}$. The set $\overline{\mathcal{X}}$ is defined for each of the case studies for which this technique is utilized. Finally, the notation x^* represents the reference trajectory for variable x; the reference trajectories for both states and parameters are given for each problem.

8.1 First-Order Irreversible Series Reaction

The first example is a simple parameter estimation problem first proposed by Tjoa and Biegler [86]. As with all parameter estimation problems, the objective is to

minimize the error between "experimental" data and the prediction of the model. For this problem, the kinetics are given by

$$A \xrightarrow{k_1} B \xrightarrow{k_2} C,$$

and the data were taken from [35]. The problem is mathematically formulated below.

$$\min_{\mathbf{k}} \sum_{\mu=1}^{10} \sum_{i=1}^{2} (\hat{x}_{\mu,i} - x_{\mu,i})^2$$

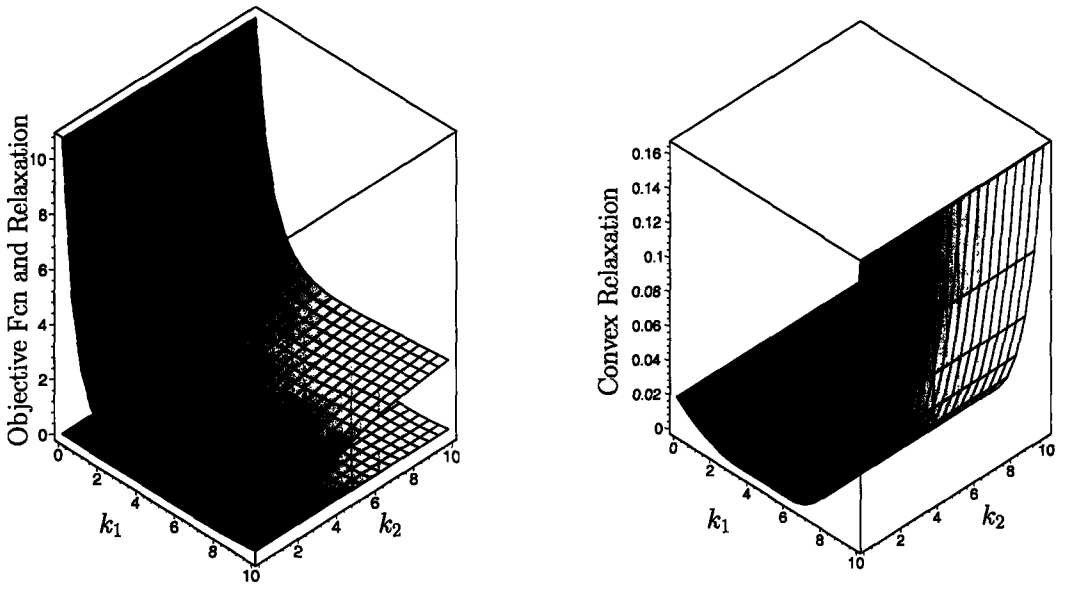
subject to

$$\begin{aligned} \frac{d\hat{x}_1}{dt} &= -k_1 \hat{x}_1 \\ \frac{d\hat{x}_2}{dt} &= k_1 \hat{x}_1 - k_2 \hat{x}_2 \end{aligned}$$

$$\begin{gathered} \hat{\mathbf{x}}(0) = (1, 0) \\ \mathbf{k} \in [0, 10]^2 \\ (\mathbf{k}^*, \mathbf{x}^*(\underline{t})) = (\mathbf{k}^L, \mathbf{x}^L(\underline{t})) \quad \forall\, \underline{t} \in (t_0, t_f]. \end{gathered}$$

GDOC was used to solve this problem to a global minimum within an absolute branch-and-bound tolerance of 10^{-4}. GDOC terminated in 0.036 CPU seconds with an objective function value of 1.22×10^{-6} at the point (5.0, 1.0), but the absolute value of the objective function at termination should be viewed with some skepticism considering the tolerance of the branch-and-bound code. Because the data in [35] differs from the exact prediction of the model only by roundoff error, the global minimum to this problem is effectively zero (within the propagation of the roundoff error) since the model exactly fits the data. Furthermore, although the problem is nonconvex, the objective function possesses only one local minimum. By construction, our algorithm guarantees that the convex relaxation for this problem is nonnegative. As expected, utilizing a local solution for the upper bound, the problem trivially solved at the root node. A picture of the objective function and its convex relaxation are found in Figure 8-1.

Figure 8-1: Objective function and convex relaxation for the problem in Section 8.1



8.2 First-Order Reversible Series Reaction

The second example problem presented, also attributed to Tjoa and Biegler [87], extends the first example by allowing reversibility of the kinetic equation:

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B \underset{k_4}{\overset{k_3}{\rightleftharpoons}} C.$$

The data for this problem were also obtained from [35]. Aside from simply possessing more degrees of freedom, solving the problem with reversible kinetics is slightly more difficult than solving the problem with irreversible kinetics because the dynamics are nonquasimonotone. However, this difficulty is overcome by utilizing Corollary 6.6 to generate state bounds. The problem is mathematically stated as

$$\min_{\mathbf{k}} \sum_{\mu=1}^{20} \sum_{i=1}^{3} (\hat{x}_{\mu,i} - x_{\mu,i})^2$$

subject to

$$\dot{\hat{x}}_1 = -k_1 \hat{x}_1 + k_2 \hat{x}_2$$

$$\begin{aligned}
\dot{\hat{x}}_2 &= k_1\hat{x}_1 - (k_2+k_3)\hat{x}_2 + k_4\hat{x}_3 \\
\dot{\hat{x}}_3 &= k_3\hat{x}_2 - k_4\hat{x}_3
\end{aligned}$$

$$\hat{\mathbf{x}}(0) = (1,0,0)$$
$$\overline{\mathcal{X}} = [0,1]^3$$
$$\mathbf{k} \in [0,10]^2 \times [10,50]^2$$
$$(\mathbf{k}^*, \mathbf{x}^*(\underline{t})) = (\mathbf{k}^L, \mathbf{x}^L(\underline{t})) \quad \forall\, \underline{t} \in (t_0, t_f].$$

For this problem, the state bounds are given by the following system of differential equations. Note that the hats have been dropped from the state variables for notational convenience and that the sets

$$\overline{\mathcal{X}} = [\overline{x}_1^L, \overline{x}_1^U] \times [\overline{x}_2^L, \overline{x}_2^U] \times [\overline{x}_3^L, \overline{x}_3^U]$$

and

$$K = [k_1^L, k_1^U] \times [k_2^L, k_2^U] \times [k_3^L, k_3^U] \times [k_4^L, k_4^U]$$

are utilized for clarity instead of the numerical values given in the problem statement.

$$\begin{aligned}
\dot{x}_1^L &= \min\{k_2^L \max\{x_2^L, \overline{x}_2^L\},\ k_2^L \min\{x_2^U, \overline{x}_2^U\},\ k_2^U \max\{x_2^L, \overline{x}_2^L\},\ k_2^U \min\{x_2^U, \overline{x}_2^U\}\} - \\
&\quad \max\{k_1^L x_1^L,\ k_1^U x_1^L\}
\end{aligned}$$

$$\begin{aligned}
\dot{x}_2^L &= \min\{k_1^L \max\{x_1^L, \overline{x}_1^L\},\ k_1^L \min\{x_1^U, \overline{x}_1^U\},\ k_1^U \max\{x_1^L, \overline{x}_1^L\},\ k_1^U \min\{x_1^U, \overline{x}_1^U\}\} - \\
&\quad \max\{k_2^L x_2^L, k_2^U x_2^L\} - \max\{k_3^L x_2^L, k_3^U x_2^L\} + \\
&\quad \min\{k_4^L \max\{x_3^L, \overline{x}_3^L\},\ k_4^L \min\{x_3^U, \overline{x}_3^U\},\ k_4^U \max\{x_3^L, \overline{x}_3^L\},\ k_4^U \min\{x_3^U, \overline{x}_3^U\}\}
\end{aligned}$$

$$\begin{aligned}
\dot{x}_3^L &= \min\{k_3^L \max\{x_2^L, \overline{x}_2^L\},\ k_3^L \min\{x_2^U, \overline{x}_2^U\},\ k_3^U \max\{x_2^L, \overline{x}_2^L\},\ k_3^U \min\{x_2^U, \overline{x}_2^U\}\} - \\
&\quad \max\{k_4^L x_3^L, k_4^U x_3^L\}
\end{aligned}$$

$$\begin{aligned}
\dot{x}_1^U &= \max\{k_2^L \max\{x_2^L, \overline{x}_2^L\},\ k_2^L \min\{x_2^U, \overline{x}_2^U\},\ k_2^U \max\{x_2^L, \overline{x}_2^L\},\ k_2^U \min\{x_2^U, \overline{x}^U\}\} -
\end{aligned}$$

$$\min\{k_1^L x_1^U, k_1^U x_1^U\}$$

$$\begin{aligned}\dot{x}_2^U &= \max\{k_1^L \max\{x_1^L, \overline{x}_1^L\},\ k_1^L \min\{x_1^U, \overline{x}_1^U\},\ k_1^U \max\{x_1^L, \overline{x}_1^L\},\ k_1^U \min\{x_1^U, \overline{x}_1^U\}\} \\ &\quad - \min\{k_2^L x_2^U, k_2^U x_2^U\} - \min\{k_3^L x_2^U, k_3^U x_2^U\} \\ &\quad + \max\{k_4^L \max\{x_3^L, \overline{x}_3^L\},\ k_4^L \min\{x_3^U, \overline{x}_3^U\},\ k_4^U \max\{x_3^L, \overline{x}_3^L\},\ k_4^U \min\{x_3^U, \overline{x}_3^U\}\}\end{aligned}$$

$$\begin{aligned}\dot{x}_3^U &= \max\{k_3^L \max\{x_2^L, \overline{x}_2^L\},\ k_3^L \min\{x_2^U, \overline{x}_2^U\},\ k_3^U \max\{x_2^L, \overline{x}_2^L\},\ k_3^U \min\{x_2^U, \overline{x}_2^U\}\} - \\ &\quad \min\{k_4^L x_3^U, k_4^U x_3^U\}\end{aligned}$$

As with the previous problem, using a branch-and-bound tolerance of 10^{-4}, this problem also solved at the root node because the literature data for this problem also contains no error. GDOC terminated in 0.044 CPU s with an objective function value of 7.69×10^{-5} at the point (4.00, 2.00, 40.0, 20.0).

8.3 Catalytic Cracking of Gas Oil

The following problem is another parameter estimation problem attributed to Tjoa and Biegler [86]. The problem is stated as

$$\min_{\mathbf{k}} \sum_{\mu=1}^{20} \sum_{i=1}^{2} (\hat{x}_{\mu,i} - x_{\mu,i})^2$$

subject to

$$\begin{aligned}\dot{\hat{x}}_1 &= -(k_1 + k_3)\hat{x}_1^2 \\ \dot{\hat{x}}_2 &= k_1 \hat{x}_1^2 - k_2 \hat{x}_2\end{aligned}$$

$$\hat{\mathbf{x}}(0) = (1, 0)$$
$$\overline{\mathcal{X}} = [0, 1]^2$$
$$\mathbf{k} \in [0, 20]^2$$
$$(\mathbf{k}^*, \mathbf{x}^*) = (\mathbf{k}^{mid}, \mathbf{x}^{mid}(\underline{t})) \quad \forall\, \underline{t} \in (t_0, t_f].$$

Unlike the previous two problems, the fitted data for this problem at the global minimum do not exactly fit the "experimental" data. Because the objective function value at termination for this problem is close to zero, I believe that solving the problem to an absolute error tolerance is the most sensible approach. Table 8.1 presents the results for solving this problem.

Table 8.1: Results for catalytic cracking of gas oil problem.

objective function	location	CPU s	nodes	absolute tolerance
2.66×10^{-3}	(12.2, 7.98, 2.22)	0.18	1	10^{-2}
2.66×10^{-3}	(12.2, 7.98, 2.22)	5.78	83	10^{-3}
2.66×10^{-3}	(12.2, 7.98, 2.22)	12.23	189	10^{-4}
2.66×10^{-3}	(12.2, 7.98, 2.22)	19.40	295	10^{-5}

8.4 Singular Control

The next problem I examine is the singular control problem originally formulated in [58]. The problem is given by

$$\min_{u(t)} \int_{t_0}^{t_f} x_1^2 + x_2^2 + 0.0005(x_2 + 16t - 8 - 0.1x_3u^2)^2 \, dt$$

subject to

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -x_3u + 16t - 8 \\
\dot{x}_3 &= u \\
\mathbf{x}_0 &= (0, -1, -\sqrt{5}) \\
t \in (0, 1] \qquad & -4 \leq u(t) \leq 10 \\
(\mathbf{p}^*, \mathbf{x}^*(\underline{t})) &= (\mathbf{p}^U, \mathbf{x}^U(\underline{t})) \quad \forall \, \underline{t} \in (t_0, t_f],
\end{aligned}$$

where the variable $\mathbf{p}$ in the reference trajectory derives from a piecewise constant control parameterization. Because this problem does not derive from a physical system,

no natural bounding set exists. Studying the singular control problem enables me to examine a very important tradeoff in implementing a solution strategy for problems with integral objective functions. The theoretical presentation of the algorithm has emphasized relaxing Problem 6.1 utilizing Corollary 3.7 to relax the integral directly. As previously stated, in applying Corollary 3.7, the relaxation for the integrand must be constructed for each fixed $\underline{t} \in (t_0, t_f]$. Under this requirement, the affine nature of the relaxations $\mathbf{c}(\underline{t}, \mathbf{p})$ and $\mathbf{C}(\underline{t}, \mathbf{p})$ cannot be exploited, and an integration must be performed for each major iteration of the optimizer. Trivially, however, the singular control problem can be reformulated as

$$\min_{u(t)} z(t_f)$$

subject to

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -x_3 u + 16t - 8 \\
\dot{x}_3 &= u \\
\dot{z} &= x_1^2 + x_2^2 + 0.0005(x_2 + 16t - 8 - 0.1 x_3 u^2)^2 \\
z(0) &= 0 \\
\mathbf{x}_0 &= (0, -1, -\sqrt{5}) \\
t \in (0, 1] \quad & \quad -4 \leq u(t) \leq 10 \\
(\mathbf{p}^*, \mathbf{x}^*(\underline{t})) &= (\mathbf{p}^U, \mathbf{x}^U(\underline{t})) \quad \forall\, \underline{t} \in (t_0, t_f],
\end{aligned}$$

where the integrand has simply been replaced by a quadrature variable; this transformation is quite common in the literature. Now, the affine nature of the state relaxations can be exploited, for I only need relaxations at a fixed time. The problem was solved by GDOC in both the original formulation and the quadrature variable reformulation; the results are presented in Table 8.2 and Table 8.3 below. For each formulation, the problem was solved to an absolute tolerance of 10^{-3} with piecewise constant control profiles on equally spaced meshes with a varying number of time

intervals. Additionally, each problem was solved with and without post-processing, Lagrangian reduce heuristics [78].

Table 8.2: Results for singular control problem in original formulation.

No. time intervals	obj. fcn.	location	CPU s	nodes	heuristics
1	0.497	(4.07)	2.0	21	no
1	0.497	(4.07)	1.8	15	yes
2	0.277	(5.57, -4.00)	26.5	89	no
2	0.277	(5.57, -4.00)	22.5	47	yes
3	0.146	(8.05, -1.85, 6.09)	814.3	1225	no
3	0.146	(8.05, -1.85, 6.09)	540.3	489	yes

Table 8.3: Results for singular control problem in quadrature variable reformulation.

No. time intervals	obj. fcn.	location	CPU s	nodes	heuristics
1	0.497	(4.07)	5.2	33	no
1	0.497	(4.07)	3.4	15	yes
2	0.277	(5.57, -4.00)	55.1	193	no
2	0.277	(5.57, -4.00)	28.8	49	yes
3	0.146	(8.05, -1.85, 6.09)	1929.0	3931	no
3	0.146	(8.05, -1.85, 6.09)	816.3	789	yes

As expected, the number of nodes required to solve the integral formulation is less than the number of nodes required to solve the quadrature formulation. This is a common feature for problems with nonlinear dynamics, particularly in cases where the integrand is highly nonlinear, for adding additional nonlinearity in the dynamics decreases the tightness of the state bounds and the tightness of the state relaxations. For the problem considered, the CPU time required for solution is also less for the integral formulation than for the quadrature formulation. However, one quickly notes that the cost per node for the quadrature formulation is smaller than the cost per node for the integral formulation; furthermore, the relative cost per node for the quadrature formulation decreases with the increasing number of parameters because the cost of the integration increases slightly with an increasing number of parameters. Thus, this tradeoff must be considered individually for each problem. In

general though, I expect that an integral formulation will almost always outperform a quadrature reformulation, especially for small problems or problems with highly nonlinear integrands.

8.5 Oil Shale Pyrolysis

The next problem examined is a fixed final time formulation of the Oil Shale Pyrolysis problem originally posed in [58]. The problem is stated below.

$$\min_{u(t)} -x_2(t_f)$$

$$\begin{aligned}
\dot{x}_1 &= -(k_1 x_1 + k_3 x_1 x_2 + k_4 x_1 x_2 + k_5 x_1 x_2) \\
\dot{x}_2 &= k_1 x_1 - k_2 x_2 + k_3 x_1 x_2 \\
k_i &= a_i \exp\left[\frac{-b_i/R}{698.15 + 50u}\right], \qquad i = 1, \ldots, 5 \\
\mathbf{x}(t_0) &= (1, 0) \\
\overline{\mathcal{X}} &= [0, 1]^2 \\
t \in (0, 1] \qquad & 0 \leq u(t) \leq 1 \\
(\mathbf{p}^*, \mathbf{x}^*(\underline{t})) &= (\mathbf{p}^U, \mathbf{x}^U(\underline{t})) \quad \forall\, \underline{t} \in (t_0, t_f],
\end{aligned}$$

where the variable $\mathbf{p}$ in the reference trajectory derives from a piecewise constant control parameterization, and the values for a_i and b_i/R are defined in [35]. This problem was solved within an absolute error of 10^{-3} for a piecewise constant control profile. The problem was solved with both one and two equally spaced time intervals. The results are found in Table 8.4 below.

From the relative change in the objective function between using a one stage and two stage constant profile, I conclude that increasing the number of control stages beyond two is unnecessary. Furthermore, this problem clearly demonstrates the worst case exponential complexity of the branch-and-bound algorithm. This problem does not converge rapidly because the state bounds tighten very slowly with the branching

Table 8.4: Results for oil shale pyrolysis problem.

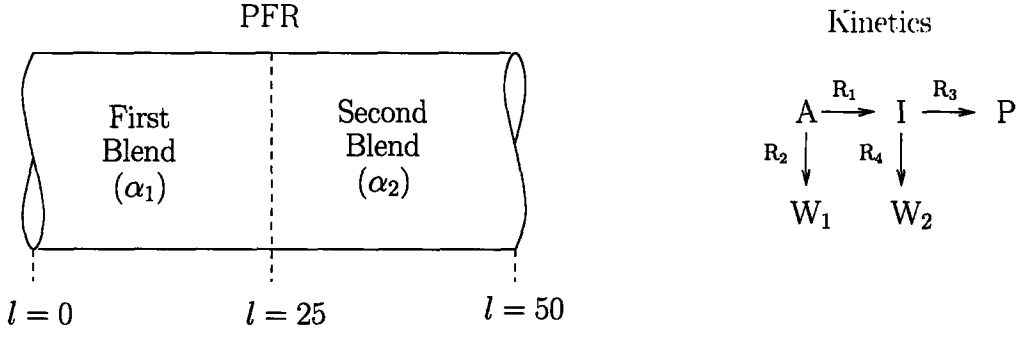
No. time intervals	objective function	location	CPU s	nodes	heuristics
1	-0.348	(0.231)	27.3	127	no
1	-0.348	(0.231)	26.2	115	yes
2	-0.351	(0.431, 0.00)	1720.7	5807	no
2	-0.351	(0.431, 0.00)	1597.3	4933	yes

in the parameter space. This in turn implies weaker state relaxations and hence slower overall convergence.

8.6 PFR Catalyst Blending

The final problem is a plug flow reactor catalyst blending problem. A plug flow reactor is divided into two regions of equal length, and each region is loaded with a different volume blend (α_i, $i = 1, 2$) of two catalysts. The reactor configuration and chemical kinetics [25] are shown in Figure 8-2.

Figure 8-2: Reactor configuration and kinetics for the PFR catalyst blending problem



The objective of the optimization is to maximize the profit of operating the reactor given a simple profit function that accounts for the sale of product P and the disposal of wastes W_1 and W_2. The problem is mathematically stated as

$$\max_{\alpha_1,\ \alpha_2} \quad x_P(50) - 0.1x_{W_1}(50) - 0.1x_{W_2}(50)$$

subject to

$$\dot{x}_A = -[\alpha k_1^1 + (1-\alpha)k_1^2 + \alpha k_2^1 + (1-\alpha)k_2^2]x_A$$

$$\dot{x}_{W_1} = [\alpha k_2^1 + (1-\alpha)k_2^2]x_A$$

$$\dot{x}_I = [\alpha k_1^1 + (1-\alpha)k_1^2]x_A - [\alpha k_3^1 + (1-\alpha)k_3^2 + \alpha k_4^1 + (1-\alpha)k_4^2]x_I$$

$$\dot{x}_{W_2} = [\alpha k_4^1 + (1-\alpha)k_4^2]x_I$$

$$\dot{x}_P = [\alpha k_3^1 + (1-\alpha)k_3^2]x_I$$

$$x_A(0) = 1000$$

$$x_{W_1}(0) = x_{W_2}(0) = x_I(0) = x_P(0) = 0$$

$$\alpha = \begin{cases} \alpha_1 & 0 \le l \le 25 \\ \alpha_2 & 25 < l \le 50 \end{cases}$$

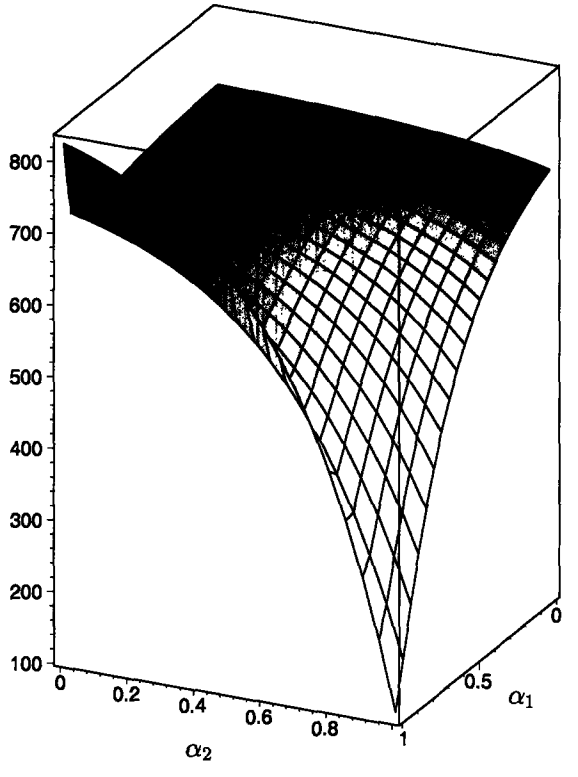
where k_j^i is the kinetic rate constant for catalyst i in reaction j and x represents species concentration. The objective function is pictured in Figure 8-3.

Although the objective function appears nonsmooth from Figure 8-3, this illusion is merely an effect of the scaling of the picture; the objective function is smooth. Additionally, one observes from the figure that the problem has two local maxima, and operating at the global maximum corresponds to roughly a 12.5% greater profit margin than operating at the local maximum. GDOC was utilized to solve the PFR catalyst blending problem to within 10^{-3} relative branch-and-bound error. The results found in Table 8.5 below are for a reference trajectory of $\mathbf{x}^*(t) = \mathbf{x}^L(t)$ and $\alpha^* = \alpha^L$.

Table 8.5: Results for the PFR catalyst blending problem

objective function	location	CPU s	nodes	heuristics
823	(1, 0)	66.94	813	no
823	(1, 0)	29.44	191	yes

Figure 8-3: Objective function for the PFR catalyst blending problem



Chapter 9

Chemical Engineering Case Study: Direct Measurement of the Fast, Reversible Reaction of Cyclohexadienyl Radicals with Oxygen in Nonpolar Solvents

In postulating a new kinetic mechanism, one must always validate the accuracy of the hypothesis against experimental data. In general, the kinetic model is expressed mathematically as a system of nonlinear, parameter dependent differential equations, where the embedded parameters represent chemically significant quantities such as kinetic rate constants, pre-exponential factors, or activation energies. Validation of the proposed mechanism is performed by estimating parameter values that yield model predictions consistent with experimentally collected data. Although the methodology is simplistic, the practicality is difficult, for determining the best parameter estimates is a highly nontrivial task. Typically, quantum chemistry, thermodynamics, or some other physical insight leads to estimates accurate within a few orders of magnitude. From these order of magnitude estimates, many researchers employ a trial-and-error

simulation approach to validation in which the researcher refines his or her choice of parameter values by repeatedly simulating the kinetic model and comparing these predictions to experimental data. While this method may provide the researcher insight into the physical behavior of the model, this technique provides absolutely no mathematical method for quantifying the goodness of fit of the model to the data; therefore, deterministic methods for estimating kinetic parameters are sought.

A vast improvement over trial-and-error methods for estimating kinetic parameter values is to employ deterministic optimization techniques. Typically, the objective of the optimization problem is to minimize the sum of squared errors (or a weighted variant) between the model prediction and the experimental data at many predefined points over the time horizon of the experiment. Historically, gradient descent optimization methods have been employed, in conjunction with numerical integration, to solve the optimization problem. Unfortunately, even trivial kinetic mechanisms yield nonlinear differential equations, which subsequently imply a nonconvex optimization problem. The dilemma associated with nonconvex optimization problems is that these problems can possess multiple local minima (sometimes pathologically), and gradient descent optimization methods cannot distinguish between local and global minima. That is, because kinetic parameter estimation problems are nonconvex, at termination of an optimization routine, the kinetic parameter values obtained may not represent the best possible fit of the model to the data. While repeating the optimization from a different initial guess may identify estimates with improved objective function value, no theoretical result exists to assure the researcher that restarting the optimization from another initial guess will not improve his or her solution even further. Hence, the process repeats *ad infinitum*.

At this point, one might inquire why a local optimum to a problem is insufficient. First, although a local solution might empirically fit the data reasonably well, because other minima may exist, other researchers may experience difficulty in reproducing the results, even utilizing the same experimental data. In particular, if a second researcher obtains a better solution, one may question the conclusions drawn by the original researchers. Second, because a local solution may not fit the experimental

data well at all, the researcher may abandon the model feeling that the experimental data invalidates the mechanism. Quite possibly, however, the mechanism fits the data quite well at the global solution. An example of this phenomena is exhibited later in this paper. Finally, knowing that the optimum obtained may only be local, a researcher may cling to an incorrect mechanism that truly has been invalidated by comparison to experimental data because he or she believes that a better solution is likely to exist. Essentially, by not guaranteeing a global solution to an optimization problem, the conclusions that can be drawn simply remain subjective interpretations of the mathematical results.

In this chapter, I examine the application of the theory developed in Chapter 6 and the implementation developed in Chapter 7 to a parameter estimation problem taken from a research problem of current interest to several colleagues at MIT. Their research focuses on studying the liquid phase reaction of resonantly stabilized cyclohexadienyl radicals with molecular oxygen to form two isomeric cyclohexadienylperoxyl radicals; this reaction is important early in flame ignition. Because of its importance, this reaction has been studied in both the liquid and the gas phase. In the gas phase, the reaction is unusually slow for radical reactions with oxygen, while in the liquid phase, the reaction is diffusion limited. In their research, Taylor *et al.* [?] have experimentally measured the absorption bands of the cyclohexadienyl radical at 316 nm to infer the rate constant for the reaction of interest. Their findings indicate that the rate constant measured in the liquid phase is two orders of magnitude greater than the rate constant in the gas phase as measured by other researchers. Rather than propose distinct mechanisms for the liquid and gas phases, Taylor *et al.* [?] have proposed an equilibrium mechanism consistent in both phases that explains the discrepancy between the overall reaction rates in each phase. They have justified their mechanism by estimating intermediate rate constants via quantum mechanical calculations. The objective of their study is to validate the proposed mechanism experimentally. Parameter estimation from time series concentration data is utilized to compute the rate constants of the intermediate reactions in the proposed mechanism. These experimentally obtained values can then be compared to those obtained via

the quantum mechanical calculations. In this chapter, I mainly concentrate on the mathematical aspects of their problem. The interested reader is referred to the paper by Taylor *et al.* [?] for details concerning the chemistry of the reaction and the experimental setup.

9.1 The Proposed Mechanism

Table 9.1 below describes the kinetic mechanism proposed by Taylor *et al.* Bounds on the rate constants based on physical insight are given for each elementary step. Reaction 1 and Reaction 5 are well-studied, and the values for their rate constants are considered constant and are given by literature values.

Table 9.1: Proposed Kinetic Mechanism

No.	Reactions	k_{298} ($M^{-1}\mu s^{-1}$ or μs^{-1})
1	$(CH_3)_3CO\cdot + 1,4C_6H_8 \longrightarrow C_6H_7\cdot + (CH_3)_3COH$	5.3×10^1
2_f	$C_6H_7\cdot + O_2 \longrightarrow p\text{-}C_6H_7OO\cdot$	$[10^1, 1.2 \times 10^3]$
2_r	$p\text{-}C_6H_7OO\cdot \longrightarrow O_2 + C_6H_7\cdot$	$[10^{-3}, 2 \times 10^{-1}]$
3_f	$C_6H_7\cdot + O_2 \longrightarrow o\text{-}C_6H_7OO\cdot$	$[10^1, 1.2 \times 10^3]$
3_r	$o\text{-}C_6H_7OO\cdot \longrightarrow O_2 + C_6H_7\cdot$	$[10^{-3}, 2 \times 10^{-1}]$
4	$o\text{-}C_6H_7OO\cdot \longrightarrow C_6H_6 + HO_2\cdot$	$[10^{-3}, 4 \times 10^1]$
5	$2\ C_6H_7\cdot \longrightarrow$ products	1.2×10^3

In their experiments, the concentration of $C_6H_7\cdot$ was measured. The objective of the optimization is to minimize the least square error between the experimentally measured concentration and the concentration of $C_6H_7\cdot$ as predicted by differential equations describing the evolution of the concentrations of the species; the differential equations are derived from the proposed mechanism given in Table 9.1. The following substitutions are introduced for notational convenience:

$$
\begin{aligned}
C_6H_7\cdot &\Leftrightarrow A \\
(CH_3)_3CO\cdot &\Leftrightarrow Z \\
1,4C_6H_8 &\Leftrightarrow Y \\
o\text{-}C_6H_7OO\cdot &\Leftrightarrow B \\
C_6H_6 &\Leftrightarrow C \\
p\text{-}C_6H_7OO\cdot &\Leftrightarrow D.
\end{aligned}
$$

Differential equations for the concentrations of the relevant species are given from elementary kinetics:

$$\dot{x}_A = k_1 x_Z x_Y - x_{O_2}(k_{2_f} + k_{3_f})x_A + k_{2_r} x_D + k_{3_r} x_B - k_5 x_A^2 \quad (9.1a)$$

$$\dot{x}_Z = -k_1 x_Z x_Y \quad (9.1b)$$

$$\dot{x}_Y = -k_1 x_Z x_Y \quad (9.1c)$$

$$\dot{x}_D = k_{2_f} x_A x_{O_2} - k_{2_r} x_D \quad (9.1d)$$

$$\dot{x}_B = k_{3_f} x_{O_2} x_A - (k_{3_r} + k_4)x_B. \quad (9.1e)$$

In the experiment, the initial concentration of $(CH_3)_3CO\cdot$ is given by $x_{Z,0}$, the initial concentration of $1,4C_6H_8$ is given by $x_{Y,0}$, and all other initial concentrations are taken as zero. Experimentally, the concentration of oxygen is in large excess; therefore, this concentration is treated as a constant and is not modeled by a differential equation. Rather than model the equilibrium reactions with independent forward and reverse rate constants, k_{2_r} and k_{3_r} are eliminated from the mechanism by employing equilibrium constants and the following equilibrium relations:

$$k_{2_r} = k_{2_f}/K_2$$

$$k_{3_r} = k_{3_f}/K_3$$

Although the values for the equilibrium constants are not known exactly, they are treated as fixed, known parameters for the optimization problem. The base model utilized in the optimization therefore contains five state variables and three degrees of freedom.

To complete the model of the system, I must mathematically formulate the objective function. As previously stated, the objective function is to minimize the sum of square errors between the measured concentration of $C_6H_7\cdot$ and the concentration of $C_6H_7\cdot$ as predicted by the model. Experimentally, however, the absorbance of the radical is measured, and the concentration must be inferred from the Beer-Lambert Law. Moreover, a correction factor must be added to account for the absorbance at

the same wavelength for o-$C_6H_7OO\cdot$ and p-$C_6H_7OO\cdot$. The following equation relates the concentration to the absorbance:

$$I = 2100x_A + 200(x_B + x_D),$$

where I is the absorbance. Because absorbance is the measured quantity in this experiment, the objective function is formulated in terms of absorbance to yield:

$$J(\mathbf{k}) = \sum_{i=1}^{N}(I_{E,i} - I_{M,i})^2,$$

where $I_{E,i}$ is the absorbance measured from the experiment at the time corresponding to measurement i, $I_{M,i}$ is the absorbance as computed from the model at the time corresponding to measurement i, and N is the number of measurements.

9.2 Scaling of the Model

In general, reaction kinetics parameter estimation problems are known to have severe scaling issues and are frequently very difficult to solve even as local optimization problems. In the current formulation, the optimization problem is not numerically stable for several reasons. First, the magnitude of the parameters is very large relative to the magnitude of the objective function. As a rough approximation, the objective function has an optimum on the order of 5×10^{-2}, while the first two parameters are bounded above by the order of 10^3. Typically, problems on these scales tend to be difficult to optimize because large changes in the parameters only effect small changes in the objective function value. Second, the difference between the upper and the lower bounds on the parameters is several orders of magnitude. The convergence of the branch-and-bound algorithm depends upon shrinking the parameter space to degeneracy (e.g., via bisection). Because the range of the parameters is logarithmic, by not scaling, bisection converges disproportionately quickly for larger parameter values. Finally, as discussed in Chapter 7, in order to exploit the affine structure of the

relaxations, the upper bound for the parameters should be around order one. If not, numerical error from linear algebra calculations possibly causes numerical instability in the evaluation of the convex relaxation. Therefore, the following exponential scaling is introduced

$$k_i' = \ln(k_i), \qquad i = 2f, 3f, 4. \tag{9.2}$$

A natural logarithmic scaling factor, as opposed to a base 10 logarithmic scaling factor, is selected simply because the current implementation of GDOC supports relaxing functions of the form $f(z) = \exp(z)$ but not functions of the form $f(z) = 10^z$. The final formulation of the optimization problem is given by the following:

Problem 9.1.

$$\min_{\mathbf{k}'} J(\mathbf{k}') = \sum_{i=1}^{N} [I_{E,i} - 2100 x_{A,i} - 200(x_{B,i} + x_{D,i})]^2$$

subject to

$$\begin{aligned}
\dot{x}_A &= k_1 x_Z x_Y - [\exp(k_{2_f}') + \exp(k_{3_f}')] x_A x_{O_2} + x_D \exp(k_{2_f}')/K_2 + x_B \exp(k_{3_f}')/K_3 - k_5 x_A^2 \\
\dot{x}_Z &= -k_1 x_Z x_Y \\
\dot{x}_Y &= -k_1 x_Z x_Y \\
\dot{x}_D &= \exp(k_{2_f}') x_A x_{O_2} - x_D \exp(k_{2_f}')/K_2 \\
\dot{x}_B &= \exp(k_{3_f}') x_{O_2} x_A - [\exp(k_{3_f}')/K_3 + \exp(k_4')] x_B \\
x_Z(0) &= x_{Z,0},\ x_Y(0) = x_{Y,0},\ x_A(0) = x_B(0) = x_D(0) = 0, \qquad t \in [0, t_N] \\
2.303 &= \ln(k_{2_f}^L) \le k_{2_f}' \le \ln(k_{2_f}^U) = 7.090 \\
2.303 &= \ln(k_{3_f}^L) \le k_{3_f}' \le \ln(k_{3_f}^U) = 7.090 \\
-6.908 &= \ln(k_4^L) \le k_4' \le \ln(k_4^U) = 3.689 \\
K_1 &= 46 \exp(6500/T - 18), \quad K_3 = 2K_2, \quad k_1 = 53, \quad k_5 = 1200
\end{aligned}$$

where $x_{Z,0}$, $x_{Y,0}$, x_{O_2}, t_N, and $I_{E,i}$ $(i = 1, \ldots, N)$ are all data set dependent, and T is given in Kelvin. In this chapter, I consider three different data sets from experiments conducted at three different temperatures. The experimental data for the problems in the chapter are located in Appendix C.

The implementation of the scaling can be performed by several different techniques; here, I consider two methods. In the first technique, the problem is implemented exactly as it is presented above in Problem 9.1. In particular, this formulation implies that the exponential term appears explicitly in the RHS dynamics. Because the quality of the state relaxations depends indirectly on the quality of $\mathbf{u}$ and $\mathbf{o}$, the ODE RHS underestimators and overestimators, tighter RHS relaxations yield tighter state relaxations. By introducing the exponential into the RHS of the ODEs, I have weakened the relaxations involving bilinear and trilinear terms. Therefore, a scaling method for which the optimization is performed in the scaled space but the symbolic analysis of relaxing the dynamics in the unscaled space is preferred. The second method of implementing the scaling accomplishes this task. For both the branch-and-bound and the local optimizations of the problem, the parameters are considered in the scaled space. However, at each function call in the local optimization, the parameter space is transformed by applying the inverse of Equation 9.2. That is, the current values of the parameters and their bounds are transformed back to their original unscaled values, and the numerical integration is performed in the original unscaled space. Therefore, the differential equations are relaxed in the original space, and tighter relaxations for their RHS are obtained because exponential terms no longer appear in the RHS of the ODEs. Of course, the objective function value returned to the optimizer is not affected by this method of scaling. However, the ith component of the gradient must be multiplied by $\exp(k_i')$, $i = 2_f, 3_f, 4$.

9.3 The Accuracy of the Objective Function

For the cases studies considered previous to this chapter, the selection of the convergence tolerance was rather arbitrary since the data did not derive from experimental measurement. For the problems in this chapter, however, the precision to which the objective function is known is limited by the precision of the experimental data. If the data were assumed to be exact to machine precision, a multistart analysis indicates the problems possess hundreds of local minima. This is clearly nonsense, for many

of the local minima first differ in objective value after the fourth or fifth decimal place. Because the data are not this precise, these local minima cannot be considered distinguishable. Data from another experiment performed under the same conditions would likely yield different local minima. Therefore, an order of magnitude estimate of the error propagated into the objective function by the experimental uncertainty must be computed in order to determine the precision to which the objective function is known. A tolerance based on this analysis will be utilized to distinguish distinct local minima in a multistart analysis and will be utilized as the convergence criterion for the global optimization routine.

Although the model itself contains uncertain parameters, I consider the model to be exact relative to error propagation. Any error introduced due to the uncertainty of the model parameters is interpreted as a flaw of the model and not as uncertainty in the measurements. Additionally, because the model predictions are derived from the numerical solution of an ODE, the predicted concentrations are subject to calculational error. Because I can set the integration tolerance arbitrarily small (to within machine precision), I assume that the experimental error dominates the computational error in the calculation of the residual (the difference in the model and experimental data). Therefore, I completely ignore the error introduced from the numerical solution of the ODEs. The preceeding arguments justify considering the following reduced equation to compute the propagation of the error into the objective function:

$$J = \sum_{i=1}^{N} (I_{E,i} - I_{M,i})^2,$$

where $I_{E,i}$ is subject to the experimental error $\pm\Delta I$ and $I_{M,i}$ is considered to be exact. From statistics, the following equation yields an order of magnitude estimate of the propagated uncertainty in J:

$$\Delta J = \pm 2\Delta I \sum_{i=1}^{N} |I_{E,i} - I_{M,i}| \approx \pm \frac{2\Delta I N \sqrt{J}}{\sqrt{N}} \approx$$
$$\pm \frac{2(5 \times 10^{-5})(10^3)\sqrt{5 \times 10^{-2}}}{\sqrt{10^3}} \approx \pm 7 \times 10^{-4}.$$

Therefore, 10^{-3} is chosen as the absolute tolerance for the branch-and-bound algorithm.

9.4 The Presence of Local Minima

Before solving the optimization problems, a multistart algorithm was applied to demonstrate the existence of local minima for the problems. For each problem, a local optimization was performed starting from one thousand random initial guesses. The random initial guesses were derived from the following equation:

$$p_i = p_i^L + r_i(p_i^U - p_i^L), \qquad i = 1, \ldots, 3$$

where r_i is a random, nonnegative, double precision number uniformly distributed over the interval [0.0,1.0]; r_i was generated by the standard C library function drand48(). In order to consider a local minimum unique, its objective function had to differ in value by more than 10^{-3} from any other previously located local minimum. The results are found below in Tables 9.2, 9.3, and 9.4.

Table 9.2: Multistart Results for Problem 9.1 for data at 273 K

J	k'_{2_f}	k'_{3_f}	k'_4	frequency
0.1829	2.445	6.890	1.463	1
0.1786	2.352	6.835	3.592	2
0.1253	7.090	2.539	-6.313	2
0.0933	2.632	7.090	-6.706	8
0.0923	2.714	7.090	-5.766	4
0.0832	5.103	7.059	-6.561	2
0.0805	6.091	6.786	-6.441	113
0.0793	5.904	6.867	-5.024	11
0.0778	5.884	6.874	-4.410	2
0.0642	2.486	7.090	-1.845	4
0.0631	3.293	7.090	-1.848	14
0.0615	4.017	7.090	-1.872	3
0.0599	6.348	6.588	-1.207	421
0.0584	6.727	5.983	1.941	414
Total CPU time (s): 104.34				

Table 9.3: Multistart Results for Problem 9.1 for data at 298 K

J	k'_{2_f}	k'_{3_f}	k'_4	frequency
0.1311	5.872	6.569	-6.455	119
0.1300	5.867	6.571	-5.773	19
0.1290	5.858	6.576	-5.396	3
0.0422	3.269	6.931	-1.167	112
0.0406	4.887	6.798	-1.171	70
0.0391	6.282	6.018	1.934	678
Total CPU time (s): 75.35				

Table 9.4: Multistart Results for Problem 9.1 for data at 323 K

J	k'_{2_f}	k'_{3_f}	k'_4	frequency
2.0386	6.029	6.746	-6.867	13
2.0371	6.016	6.735	-6.639	19
2.0358	6.024	6.732	-6.469	8
0.0598	6.254	6.554	1.963	374
0.0588	6.155	6.724	1.193	18
0.0573	5.852	6.955	0.679	569
Total CPU time (s): 95.38				

From the multistart results, several important pieces of information concerning Problem 9.1 are immediately evident. First and foremost, for each data set, distinct local minima occur, and a global optimization method is certainly warranted. Second, the optimization is reasonably insensitive to k'_4. For example, in Table 9.3, for the minima located near 0.13, the location of these three minima differ by no more than 0.02 in each of k'_{2_f} and k'_{3_f}. However, the location of these minima differ by more than 1 in k_4. Finally, because the objective value for several of the local minima with parameter values very near each other differ by only slightly more than 10^{-3}, these minima are probably not actually distinct. This empirically validates the choice for the branch-and-bound tolerance. The error estimate obtained in the previous section was only an order of magnitude estimate and is probably a little smaller than the actual experimental error propagated into the objective function. In selecting a tolerance slightly smaller than the true error, I am likely to identify spurious local minima; however, I am unlikely to converge to a minimum that is not truly global. That is,

the tolerance is set tightly enough that I am confident that the minimum obtained by GDOC is the global minimum within the precision of the objective function.

9.5 Global Optimization Results

GDOC was utilized to solve Problem 9.1 for the data sets generated at 273 K, 298 K, and 323 K. The computer utilized was an AMD Athlon XP2000+ operating at 1667 MHz, and all code was compiled with gcc 3.3.1. As previously discussed, the absolute branch-and-bound tolerance was set to 0.001 (no relative tolerance was applied). Because the magnitude of the concentrations is quite small, 10^{-10} was utilized for the absolute integration tolerance. The tolerance for NPSOL was set to 10^{-5}. The least reduced axis rule was utilized to select the next branching variable, and branching was performed by the incumbent rule. By the incumbent rule, the branching location is selected as the location of the incumbent if the incumbent location is in the current partition. If the incumbent location is not in the current partition, then bisection is applied. This branching strategy was selected to balance the effects of branching on a logarithmically scaled parameter space. Post-processing Lagrangian reduce heuristics were utilized at each node to accelerate convergence. That is, the bounds were updated by Tests 1 and 2 in [78], and the domain reduction step was repeated (up to a maximum of 10 repeats) if the previous bounds reduction step improved any bound by at least 10%.

Table 9.5 below contains the results for optimizing Problem 9.1 for the first scaling method, Table 9.6 below contains the results for optimizing Problem 9.1 for the second scaling method, and Table 9.7 below contains the results for optimizing Problem 9.1 without scaling. In each table, results are included for the experiment at all temperatures. Of course, all of the scaling methods result in the same objective function value. However, the location of this value varies slightly. As previously discussed, this behavior is expected, for I have already established that many false local minima occur below the absolute tolerance threshold. As expected, the second scaling method substantially outperforms the first scaling method. This performance differ-

ence in the required number of nodes arises because the RHS relaxations, and hence the state relaxations, are tighter for the second scaling method. The performance difference in time per node arises because the relaxed differential system without the exponential term is easier to integrate. For this example, the unscaled system outperforms the second scaling method at all three temperatures. The differential systems for both of these methods are identical, and the performance difference is attributed to taking different paths through the branch-and-bound tree derived from branching on different sets. As previously stated, however, whether or not scaling improves the numerical results is very problem dependent. Because not scaling possibly introduces numerical instability, scaling is still generally recommended.

Table 9.5: Global Optimization Results for Problem 9.1 utilizing the first scaling method

T (K)	J	k'_{2_f}	k'_{3_f}	k'_4	nodes	t (CPU s)
273	0.058 ± 0.001	6.718	5.977	2.711	617	209.37
298	0.039 ± 0.001	6.270	5.997	3.198	555	169.74
323	0.057 ± 0.001	5.857	6.949	0.691	7107	2333.63

Table 9.6: Global Optimization Results for Problem 9.1 utilizing the second scaling method

T (K)	J	k'_{2_f}	k'_{3_f}	k'_4	nodes	t (CPU s)
273	0.058 ± 0.001	6.720	5.976	2.592	177	12.04
298	0.039 ± 0.001	6.272	5.998	3.025	233	14.24
323	0.057 ± 0.001	5.856	6.951	0.686	5833	338.26

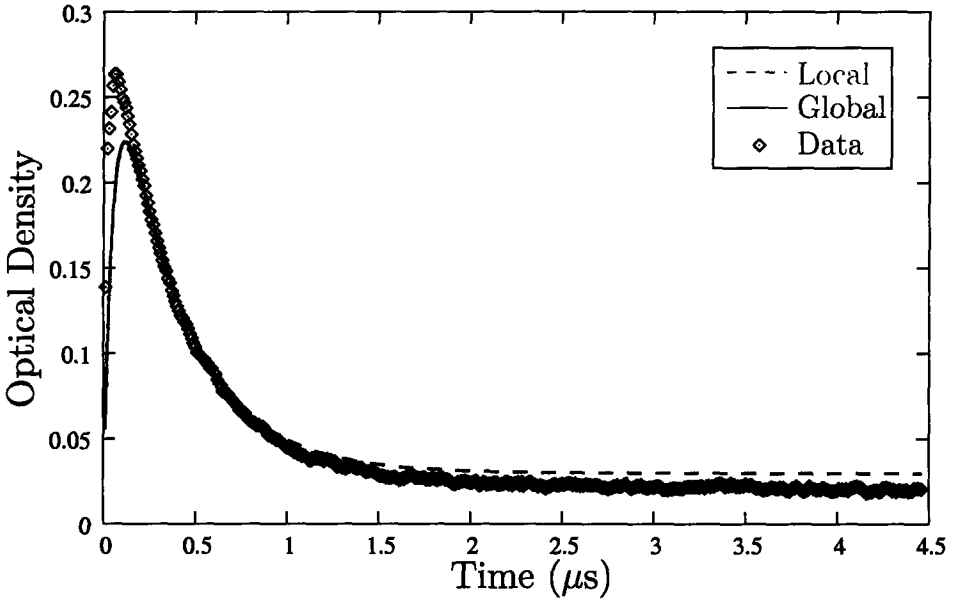
Table 9.7: Global Optimization Results for Problem 9.1 without scaling

T (K)	J	k_{2_f}	k_{3_f}	k_4	nodes	t (CPU s)
273	0.058 ± 0.001	828	394	13.5	157	11.45
298	0.039 ± 0.001	531	403	19.2	75	6.14
323	0.057 ± 0.001	354	1040	1.99	4967	286.75

To validate the fit generated via the parameters obtained via global optimization, the model with the globally optimal parameters was plotted with the experimental

data (the plots derive from the data in Table 9.5). For comparison, at 273 K, the local minimum with objective value of 0.0805 is plotted with the global solution. At 298 K, the local minimum with objective value of 0.1311 is plotted with the global solution. These graphs are pictured below in Figures 9-1, 9-2, and 9-3.

Figure 9-1: Optimized Model and Experimental Data for T = 273 K

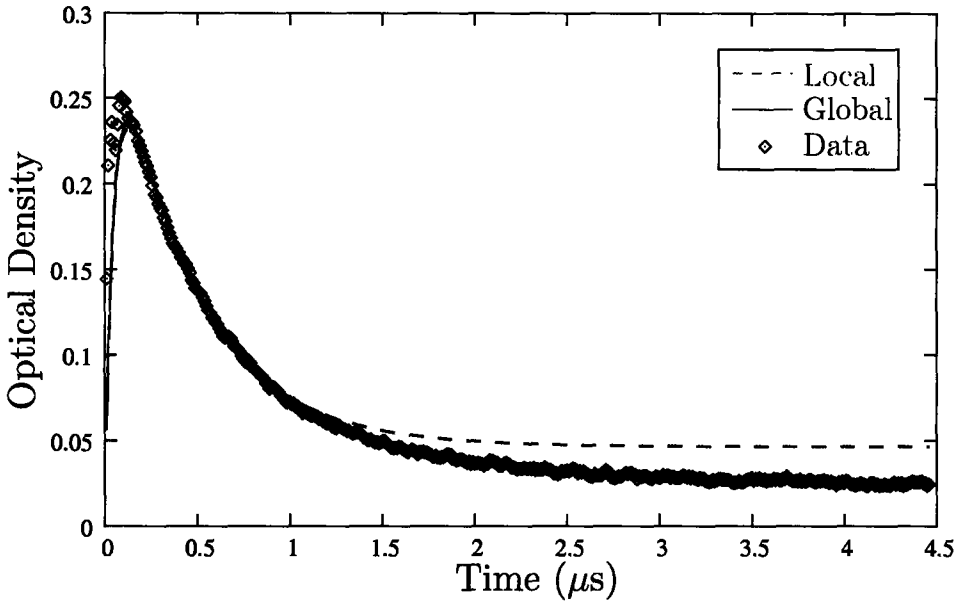


From Figures 9-1 and 9-2, one sees that except for at very early times, the models fitted by global optimization almost perfectly fit the data. Numerically, the reason that the models do not fit the data as well at the peaks is because relatively few data points were taken at very early times. Therefore, the objective function is dominated by the data taken at later times. In order to correct this problem, the objective function could be reformulated as

$$\min_{\mathbf{k}'} J(\mathbf{k}') = \sum_{i=1}^{N} w_i [I_{E,i} - 2100 x_{A,i} - 200(x_{B,i} + x_{D,i})]^2,$$

where w_i is a correction factor to give more weight to data collected at earlier times and less weight to data collected at later times. From discussions with James Taylor, due to the very small time scale of the experiment, the first data points are subject to substantially more experimental error than the later data points. Goodness of fit

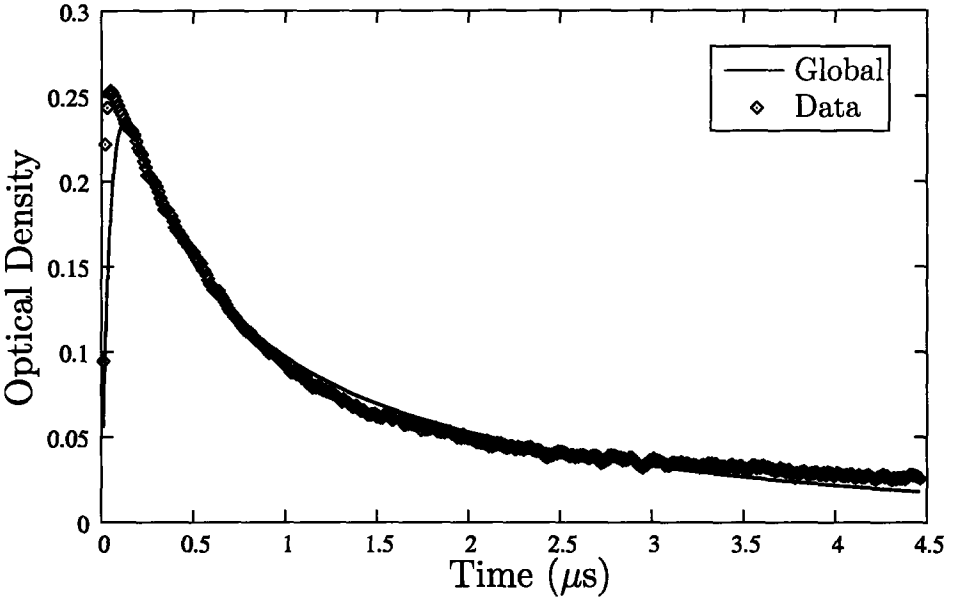
Figure 9-2: Optimized Model and Experimental Data for T = 298 K



at short times is considered less important than goodness of fit for the duration of the species decay. Experimentally, the deviations of the model from the experimental data at short times was expected and is deemed acceptable. Mr. Taylor suggested not repeating the optimization with a weighting factor.

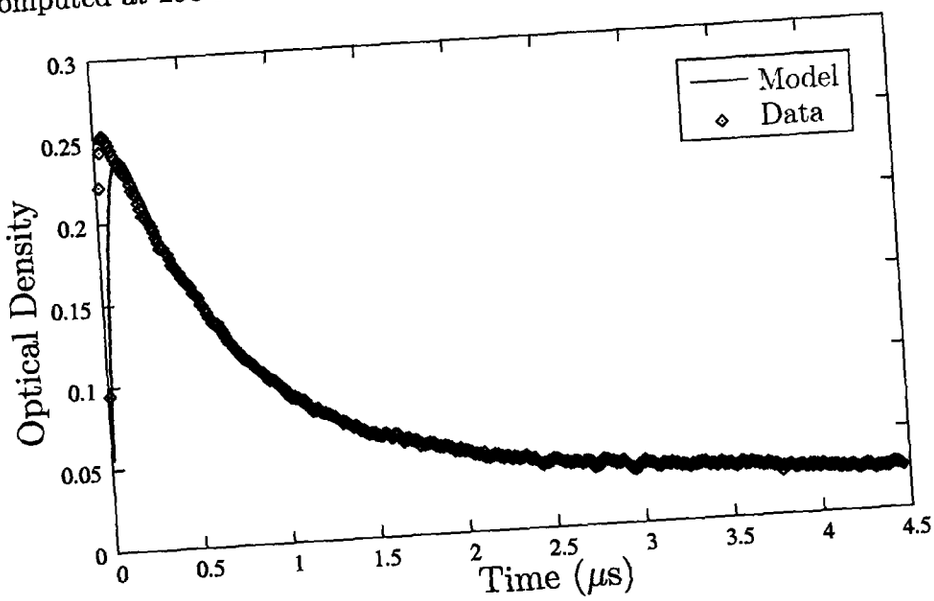
The numerical analysis of this problem demonstrates one of the most important reasons for utilizing global optimization for kinetic parameter estimation problems. Figures 9-1 and 9-2 each display the results of a local minimum along with the global minimum. If an experimentalist were to draw conclusions from these figures based solely on the local minima, one would likely conclude incorrectly that the model demonstrated systematic error relative to the experimental data. From the global minimum at each temperature, this conclusion is shown to be invalid, and I know that the model indeed fits the data well. Conversely, however, from Figure 9-3, because the minimum in the figure is global, I know that the discrepancy between the model and the data is not due to a local minimum. That is, I know a systematic error truly exists for this model. To investigate a possible cause of this discrepancy, the global optimization trial was repeated with the equilibrium calculated at 298

Figure 9-3: Optimized Model and Experimental Data for T = 323 K



K; the result is illustrated below in Figure 9-4. From the figure, one sees that the fit of the model to the experimental data is similar to the fits observed from the trials performed at 273 K and 298 K. This analysis indicates that a problem may exist in the correlation utilized to compute the equilibrium constant as a function of temperature. From discussions with Mr. Taylor, this result is not unexpected, for the error in the physical constants in the equilibrium correlation is large. However, the global optimization results indicate that reexamining the calculations utilized to derive the equilibrium correlation may be warranted.

Figure 9-4: Optimized Model and Experimental Data for T = 323 K with equilibrium values computed at 298 K



Chapter 10

Necessary and Sufficient Conditions for Convex Relaxations in an Infinite Dimensional Space

Until now, I have only considered optimization problems with parameter dependent differential equations where the optimization has been performed in a finite dimensional, Euclidean space. In this chapter, I now consider an objective functional that maps an infinite dimensional function space to a real value. Of particular interest are integral functionals on linear spaces of functions. Integral functionals are functionals that map a linear space of functions to a real value via a definite integral. Thus, rather than seeking parameter values that minimize the objective function, I am seeking a function that minimizes the objective function. Variational problems are much more difficult to solve than their finite dimensional analogs, both numerically and theoretically. Numerically, the necessary and sufficient conditions equate to solving two point boundary value problems rather than solving for KKT conditions constrained by parameter dependent ordinary differential equations. Theoretically, I am aware of no method for applying branch-and-bound in an infinite dimensional decision space. Thus, the methods developed in this chapter only provide a rigorous technique for constructing a lower bound on a variational problem.

10.1 Convexity

At the root of developing convex underestimators for variational problems is the notion of convexity for real valued functions on $\mathbb{R}^d$. For the purposes of this chapter, only continuously differentiable functions are considered. This restriction implies the existence of the gradient of the function and permits convexity to be defined in the following manner.

Definition 10.1. The real valued function f is convex on $D \subset \mathbb{R}^d$ if it has continuous partial derivatives on D and satisfies the inequality

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0), \qquad \forall\, \mathbf{x}, \mathbf{x}_0 \in D$$

Moreover, f is strictly convex on D when the above inequality holds at each $\mathbf{x}_0 \in D$ with equality if and only if $\mathbf{x} = \mathbf{x}_0$.

Remark 10.2. An equivalent definition for convexity on $D \subset \mathbb{R}^d$ is

$$f(\mathbf{x}+\mathbf{v}) - f(\mathbf{x}) \geq \nabla f(\mathbf{x}) \cdot \mathbf{v}, \qquad \forall\, \mathbf{v} \in \mathbb{R}^d \qquad \text{for which } \mathbf{x},\ \mathbf{x}+\mathbf{v} \in D.$$

Moreover, when $\mathbf{v}$ is taken as an arbitrary vector, $\delta f(\mathbf{x};\mathbf{v}) = \nabla f(\mathbf{x}) \cdot \mathbf{v}$, where $\delta f(\mathbf{x};\mathbf{v})$ is the Gâteaux variation, or directional derivative, of f.

The purpose of expressing convexity as in the above remark is merely to illustrate a direct analogy between convexity in an Euclidean space $\mathbb{R}^d$ and convexity in a linear space of functions $\mathcal{X}$, the definition of which is given below.

Definition 10.3. A functional J on a set $\mathcal{D} \subset \mathcal{X}$ is said to be [strictly] convex on $\mathcal{D}$ provided that when x and $x + v \in \mathcal{D}$ then $\delta J(x;v)$ is defined and $J(x+v) - J(x) \geq \delta J(x;v)$ [with equality if and only if v=$\mathcal{O}$, where $\mathcal{O}$ is the unique vector such that $c\mathcal{O} = 0x = \mathcal{O},\ \forall x \in \mathcal{X},\ c \in \mathbb{R}$].

Convexity is an important notion in conventional optimization schemes because convexity implies that any stationary point found is a global minimum (the unique

global minimum for strict convexity). As expected, an analogous result exists for the minimization of convex functionals, as stated in the following proposition.

Proposition 10.4. *If J is [strictly] convex on $\mathcal{D} \subset \mathcal{X}$ then each $x_0 \in \mathcal{D}$ for which $\delta J(x_0; v) = 0$, $\forall\ x_0 + v \in \mathcal{D}$ minimizes J on $\mathcal{D}$ [uniquely].*

The proof of Proposition 10.4 is immediately evident from Definition 10.3 and is thus omitted. The interested reader is referred to Troutman [89, pp. 54-55] for the details of the proof. Note that the above Proposition illustrates the fundamental principle that minimization of convex functionals occurs for the function that makes the Gâteaux variation equal to zero.

The following defines convexity of functions on a subset $S \subset \mathbb{R}^3$. Specifically, functions of this form comprise the class of functions from which the integrands of functionals will be selected. Note that underlined variables are held fixed in the inequality hence only requiring the partial derivatives of f of the remaining variables.

Definition 10.5. $f(\underline{t}, x, \dot{x})$ is said to be [strongly] convex on a set $S \subset \mathbb{R}^3$ if $f = f(t, x, \dot{x})$ and its partial derivatives f_x and $f_{\dot{x}}$ are defined and continuous on this set and satisfy the inequality:

$$f(t, x+v, \dot{x}+\dot{v}) - f(t, x, \dot{x}) \geq f_x(t, x, \dot{x})v + f_{\dot{x}}(t, x, \dot{x})\dot{v},$$
$$\forall\ (t, x, \dot{x}) \text{ and } (t, x+v, \dot{x}+\dot{v}) \in S$$

[with equality at $(t, x, \dot{x})$ only if $v = 0$ or $\dot{v} = 0$].

10.2 Convex Underestimators

The existence of tight convex underestimators for real functions of special form such as univariate concave, bilinear, and trilinear functions has long been established in the field of optimization. Adjiman *et al.* [5] have developed a method for deriving convex underestimators for twice continuously differentiable real functions. This section develops a rigorous analytical method for extending these known theories of convex

relaxations for real functions to variational problems. At the heart of this analysis is the following fundamental theorem, which links convexity and relaxation of real functions to convexity and relaxation of integral functionals. The astute reader will recognize that this is simply the variational analog of Corollary 3.7.

Theorem 10.6. *Let the functionals*

$$F(x) = \int_a^b f(t, x(t), \dot{x}(t))\ dt \qquad \text{and} \qquad G(x) = \int_a^b g(t, x(t), \dot{x}(t))\ dt$$

be defined on the set

$$\mathcal{D} = \{x \in C^1[a,b]; (x(t), \dot{x}(t)) \in D \subset \mathbb{R}^2\}.$$

If $g(\underline{t}, x, \dot{x})$ is convex on $[a,b] \times D$ in the sense of Definition 10.5 and

$$g(\underline{t}, x, \dot{x}) \leq f(\underline{t}, x, \dot{x}), \qquad \forall \text{ fixed } t \in [a,b] \text{ and } x(t), \dot{x}(t) \in D$$

then $G(x)$ is convex on $\mathcal{D}$ and $G(x) \leq F(x)$. That is, if $g(t, x, \dot{x})$ is a convex underestimator for $f(t, x, \dot{x})$ on $[a,b] \times D$, then $G(x)$ is a convex underestimator for $F(x)$ on $\mathcal{D}$.

Proof. When x, $x + v \in \mathcal{D}$, then Definition 10.5 shows that at each $t \in (a,b)$, the convexity of g yields

$$g(t, x+v, \dot{x}+\dot{v}) - g(t, x, \dot{x}) \geq g_x(t, x, \dot{x})v + g_{\dot{x}}(t, x, \dot{x})\dot{v}.$$

The above equation is integrated to yield

$$\int_a^b g(t, x(t)+v(t), \dot{x}(t)+\dot{v}(t)) - g(t, x(t), \dot{x}(t))\ dt \geq \int_a^b g_x(t, x(t), \dot{x}(t))v(t) + g_{\dot{x}}(t, x(t), \dot{x}(t))\dot{v}(t)\ dt.$$

However, this is equivalent to

$$G(x+v) - G(x) \geq \delta G(x;v),$$

which by Definition 10.3 shows that $G(x)$ is convex. It remains to show that $G(x) \leq F(x)$, but this is evident from integral monotonicity because, by assumption, I have that $g(\underline{t}, x, \dot{x}) \leq f(\underline{t}, x, \dot{x})$, $\forall$ fixed $t \in [a,b]$, and $x(t)$, $\dot{x}(t) \in D$. $\square$

Remark 10.7. This theorem enables any known method for convex underestimation of real functions to be harnessed in the convex underestimation of integral functionals.

At this point, a short example is in order to demonstrate the construction of a convex underestimator for an integral functional.

Example 10.8. Suppose I wish to minimize the following functional (with respect to $x(t)$):

$$F(x) = \int_0^1 [\dot{x}(t)]^2 - [x(t)]^2 \, dt$$

on the set

$$D = \{x(t) \in C^1[0,1] : \; x(0) = x_0, \; x(1) = x_1, \; 0 \leq x(t) \leq 1 \; \forall t \in [0,1]\}.$$

Clearly, $F(x)$ is nonconvex; therefore, a convex underestimator will be derived by finding a convex underestimator for the integrand. The following formula [5] is appropriate for underestimating the integrand. Note that $\dot{x}^2$ is already convex. Since the sum of convex functions is still convex, only the nonconvex term, $-x^2$ needs to be convexified. That x and $\dot{x}$ are functions of t has been dropped to emphasize that for convexification, x and $\dot{x}$ are treated as elements of $\mathbb{R}$ rather than as elements of $C^n[0,1]$ (where $n = 1$ for $x(t)$ and $n = 0$ for $\dot{x}(t)$). In the following formula, x^L and x^U respectively represent the lower and upper bounds on x.

$$f(x^L) + \frac{f(x^U) - f(x^L)}{x^U - x^L}(x - x^L) = f(0) + \frac{f(1) - f(0)}{1 - 0}(x - 0) = -x.$$

Hence, a valid convex underestimator, denoted as $G(x)$, for this function is

$$G(x) = \int_0^1 [\dot{x}(t)]^2 - x(t)\, dt.$$

Theorem 10.6 provides a method for constructing a convex underestimating integral functional by constructing a convex underestimator for the integrand. However, Theorem 10.6 does not address the problem of determining a minimum for this convex underestimator. At first glance, Proposition 10.4 appears to offer a method by which to solve the underestimating integral; however, Proposition 10.4 is applicable only to unconstrained variational problems. The generation of convex underestimators requires constraints on the nonconvex variables [5], as illustrated by Example 10.8. Necessary and sufficient conditions for optimizing these constrained variational problems are discussed in the following section.

10.3 Necessary and Sufficient Conditions for Constrained Variational Problems

Many algorithms designed for solving variational problems focus exclusively on satisfying necessary conditions for optimality. For variational problems, this technique is synonymous with finding stationary functions, or those functions which satisfy the Euler-Lagrange equations. The downfall of this approach is that stationarity does not necessarily even imply local minimality. However, using the convexity inherent to the underestimators, a complementary sufficiency condition has been discovered thus guaranteeing that any function satisfying the Euler-Lagrange equation is a minimum [a unique minimum under strong convexity]. For this discussion, existence conditions are not discussed. Rather, it is assumed that a feasible minimum for the problem exists.

The sufficiency condition is developed in two stages. First, a lemma is presented that illustrates a method for transforming a constrained problem into an unconstrained problem, the minimum of which implies the minimum of the original con-

strained problem. Second, the optimality condition for a constrained, convex variational problem is stated. Note that the hat symbol is used to denote piecewise continuity or piecewise continuous differentiability.

Lemma 10.9. *Suppose* $\hat{f} = \hat{f}(t, x(t), \dot{x}(t))$ *and* $\hat{g} = \hat{g}(t, x(t), \dot{x}(t))$ *are continuous on* $[a, b] \times \mathbb{R}^2$ *and there exists a function* $\hat{\lambda}(t) \in \hat{C}[a, b]$, *for which* x_0 *minimizes* $\tilde{F}(x) = \int_a^b \tilde{f}(t, x(t), \dot{x}(t))\ dt$ *on* $\hat{\mathcal{D}} \subseteq \hat{C}^1[a, b]$ *where* $\tilde{f} \stackrel{\text{def}}{=} \hat{f} + \hat{\lambda}\hat{g}$. *Then* x_0 *minimizes* $F(x) = \int_a^b \hat{f}(t, x(t), \dot{x}(t))\ dt$ *on* $\hat{\mathcal{D}}$ *under the following constraints:*

1. $\hat{g}(t, x(t), \dot{x}(t)) \leq 0, \quad t \in (a, b)$
2. $\hat{\lambda}(t) \geq 0, \quad t \in (a, b)$
3. $\hat{\lambda}(t)\hat{g}(t, x_0(t), \dot{x}_0(t)) \equiv 0$

Proof. If $x \in \hat{\mathcal{D}}$, the following will be true by the definition of a minimum:

$$\begin{aligned} \tilde{F}(x) &\geq \tilde{F}(x_0) \\ F(x) + \int_a^b \hat{\lambda}(t)\hat{g}(t, x(t), x'(t))\ dt &\geq F(x_0) + \int_a^b \hat{\lambda}(t)\hat{g}(t, x_0(t), x_0'(t))\ dt \\ F(x) - F(x_0) &\geq \int_a^b (\hat{\lambda}(t)\hat{g}(t, x_0(t), x_0'(t)) - \hat{\lambda}(t)\hat{g}(t, x(t), x'(t)))\ dt \end{aligned}$$

By Constraint 3,

$$F(x) - F(x_0) \geq -\int_a^b (\hat{\lambda}(t)\hat{g}(t, x(t), x'(t)))\ dt.$$

Due to the nonnegativity of $\hat{\lambda}(t)$, Constraint 1 may be multiplied by $\hat{\lambda}(t)$ without altering the sign of the inequality

$$\hat{\lambda}(t)\hat{g}(t, x(t), x'(t)) \leq 0\ .$$

It immediately follows that

$$F(x) \geq F(x_0).$$

□

Remark 10.10. Although the product $\hat{\lambda}(t)\hat{g}(t, x(t), \dot{x}(t))$ may only be piecewise continuous, this poses no difficulty to integration provided there exist only finitely many points of discontinuity (cf. Rudin [77]).

The chapter concludes with the following Theorem stating the conditions of optimality for minimizing a bounded convex functional. Only sufficiency for the optimality condition is proven; the reader is directed to Hestenes [46] for the proof of necessity.

Theorem 10.11. *For a domain D of $\mathbb{R}^2$ suppose that $f(\underline{t}, x(t), \dot{x}(t)) \in C^1([a,b] \times D)$ is convex, and we wish to minimize*

$$F(\hat{x}) = \int_a^b f(t, \hat{x}(t), \hat{x}'(t))\ dt$$

on

$$\hat{\mathcal{D}} = \{\hat{x}(t) \in \hat{C}^1[a,b] : \hat{x}(a) = a_1, \hat{x}(b) = b_1\},$$

subject to the inequality constraint

$$g(t, \hat{x}(t)) \leq 0, \qquad t \in (a,b),$$

where $g(\underline{t}, \hat{x})$ is also convex. For any $\hat{x}_0 \in \mathcal{D}$ satisfying the inequality, the following conditions are necessary and sufficient to guarantee a minimum: There exists a $\lambda(t) \in \hat{C}[a,b]$ such that $\lambda(t) \geq 0$ and $\lambda(t)g(\underline{t}, x_0) \equiv 0$. Additionally, for all intervals excluding corner points the following equation must be satisfied:

$$\frac{d}{dt} f_{\dot{x}}(t, \hat{x}_0(t), \hat{x}_0'(t)) - f_x(t, \hat{x}_0(t), \hat{x}_0'(t) = \lambda(t) g_x(\underline{t}, \hat{x}_0(t)).$$

At any corner point c, the following condition must hold:

$$f_{\dot{x}}(c-, \hat{x}_0(c-), \hat{x}_0'(c-)) - f_{\dot{x}}(c+, \hat{x}_0(c+), \hat{x}_0'(c+)) = k g_x(t, \hat{x}_0(t))$$

for a constant k (for necessity, $k \leq 0$ and for sufficiency, $k = 0$). Currently, a small

gap exists between the necessary and sufficient conditions.

Proof of sufficiency. First, construct the functional

$$\tilde{F}(\hat{x}) = \int_a^b f(t, \hat{x}(t), \hat{x}'(t) + \lambda(t) g(\underline{t}, \hat{x}(t))\, dt \quad \text{on } \hat{\mathcal{D}}.$$

Lemma 10.9 above establishes that a minimizing solution to $\tilde{F}(\hat{x})$ will also be a minimizing solution for $F(\hat{x})$ under the given inequality constraint. Therefore, it will be sufficient to demonstrate that $\hat{x}_0(t)$ minimizes $\tilde{F}(\hat{x})$. By hypothesis, we have $\hat{x} \in \hat{C}^1[a, b]$, and we also have the following condition of stationarity at non-corner points:

$$\frac{d}{dt} f_{x'}(t, \hat{x}(t), \hat{x}'(t) - f_x(t, \hat{x}(t), \hat{x}'(t) = \lambda(t) g_x(t, \hat{x}(t)).$$

Moreover, by assumption, we also have

$$\tilde{f}(\underline{t}, \hat{x}_0(t), \hat{x}_0'(t) = f(\underline{t}, \hat{x}_0(t), \hat{x}_0'(t) + \lambda(\underline{t}) g(\underline{t}, \hat{x}),$$

which is convex at non-corner points. Additionally by hypothesis, $\hat{x}_0$ satisfies the stated corner condition at any corner point. Therefore, by Theorem 7.12 in Troutman, $\hat{x}_0$ provides a minimum for $\tilde{F}$ and hence F. □

I note that the differential equation defining optimality is the first Euler-Lagrange equation for the modified function $\tilde{F}(\hat{x})$. For a detailed discussion of stationarity, the reader is referred to Troutman [89]. Corner conditions for constrained variational problems are discussed in detail in both Troutman [89] and Hestenes [46]. Additionally, while the above theorem is stated only for one constraint, it should be obvious that the same theorem can trivially be extended to multiple constraints provided the constraints are nonintersecting. This follows because the minimum $\hat{x}_0(t)$ cannot exist at multiple distinct points simultaneously. Thus, when $\hat{x}_0(t)$ lies on one constraint, the multiplier for any other constraint is simply 0.

Chapter 11

Conclusions and Future Work

The objective of this dissertation was to expound a technique for efficiently solving to global optimality problems with an integral objective function subject to ordinary differential equations. In particular, the main emphasis of the thesis was placed on efficiently solving Problem 2.4, a formulation restricting the decision space to finite dimension. Building on an existing framework for global optimization of steady-state problems, a branch-and-bound algorithm was employed for rigorously guaranteeing termination at global optimality. Because the optimization was always performed in a finite dimensional parameter space, no modifications of the standard branch-and-bound algorithm were required, and the theoretical emphasis of solving Problem 2.4 became generating rigorous bounds for the problem. Local optimization was systematically utilized to generate upper bounds. Analogously to generating lower bounds for steady-state problems, a method for generating convex relaxations for dynamic problems was developed.

In Chapter 3, two principles were developed that guided the remainder of the theoretical developments in the thesis. First, a general method for relaxing an integral was presented. The result stated was that an integrand, $u(t, \mathbf{p})$, convex on P for each fixed $\underline{t} \in (t_0, t_f]$ implies an integral, $U(\mathbf{p})$, convex on P. By integral monotonicity, this immediately implied that a partially convex relaxation of an integrand implies a convex relaxation for an integral. In dynamic optimization problems, the objective function is never a simple function of $\mathbf{p}$ and t. Instead, the objective function is

defined via compositions with state variables defined only by differential equations. Typically, $\mathbf{p}$ appears embedded within the differential equations, and the compositions are defined implicitly via the numerical solution of the ODEs. Therefore, in order to apply the integral relaxation result, techniques were required which addressed the convexity of composite functions. The second important theoretical development presented in Chapter 3 was the affine nature of the solution to a system of linear differential equations. In general, very little can be stated concerning the convexity of the solution of a system of ODEs from mere structural knowledge of the differential equations themselves. The exception to this rule is linear differential equations. For a system of linear differential equations, the solution of the differential equations for fixed $\underline{t} \in (t_0, t_f]$ is affine in the parameter. By definition, an affine function is both convex and concave. This structural information provided a technique to address the difficulty imposed by the composition of the objective function with the solution of the parameter dependent differential equations.

In general, to apply Corollary 3.7 to relax an integral, two issues must be addressed: convex composition with the solution to the ODEs and deriving state bounds. In order to generate the tightest possible relaxations, special structure should always be exploited whenever it is available. Following this maxim, the development of convex relaxations for Problem 2.4 led to separate techniques for problems with linear ODEs and problems with nonlinear ODEs. Several chapters were devoted to each topic. For linear problems, the composition issue was addressed by the standard result that the composition of an affine function and a convex function remains convex. Exact state bounds were derived for both the states and the derivatives of the states via interval arithmetic techniques and the affine solution to a system of linear ODEs. Implementation issues were discussed, and in particular, the construction of exact state bounds was emphasized. Utilizing the developed implementation, several problems were numerically solved. For nonlinear problems, both the theoretical and computational issues required more sophisticated methods. McCormick's relaxation technique provided a method for handling the nonconvex composition of the state variables in the objective function. However, in order to utilize McCormick's relax-

ation technique, convex and concave relaxations for the numerical solution of the ODEs had to be derived. This led to a theory for deriving rigorous linear relaxations of the solutions of ODEs based on knowledge only of the right hand sides of the differential equations. Additionally, in order to successfully apply McCormick's relaxation technique, a technique was derived to bound nonquasimonotone differential equations. The technique was based on coupling physical insight into the behavior of differential equations with the theory of differential inequalities. Because application of the nonlinear theory was substantially more difficult than application of the linear theory, a separate implementation was developed for solving nonlinear problems. The unique features of the implementation included a specialized language for the problem statement, a compiler to apply the theory and generate residual files for the integrator, and an event location and detection wrapper for the CVODES numerical ODE solver. By exploiting the special affine structure of the convex relaxations, the implementation was shown to be computationally inexpensive. Several literature case studies were examined, and an entire chapter was devoted to studying a parameter estimation problem taken from a research lab at MIT. The thesis concluded with a short chapter describing a variational approach to solving Problem 2.4.

Although a substantial amount of work has already been completed, a substantial amount of work still remains. The bulk of the outstanding issues all lie within the scope of solving the nonlinear problem. In particular, I believe that four issues require additional research. First, although Corollary 6.6 addresses constructing nonexploding state bounds for nonquasimonotone differential equations, the method often yields relatively slow convergence. Additionally, the technique is very limited by the quality of the natural bounds. Because the tightness of the generated relaxations is coupled to the quality of the state bounds, any improvements in the bounding procedure could yield profound effects on the overall convergence of the branch-and-bound algorithm. Second, no systematic method currently exists for selecting an optimal reference trajectory to satisfy Theorem 6.16. In general, any reference trajectory yields a convergent state relaxation. However, for some problems, the choice of reference trajectory can lead to solving a sizeable number of additional branch-and-bound nodes.

The selection of an optimal reference trajectory can be formulated as an optimization problem. However, the optimization problem is itself a nonconvex dynamic optimization problem, and determining the optimal reference trajectory problem is potentially more difficult than solving the original problem. An alternative to selecting an optimal reference trajectory is simply to include simultaneously relaxations generated from multiple reference trajectories. Third, although the branch-and-bound algorithm has worst case exponential complexity, heuristics need to be developed to accelerate the expected running time of the branch-and-bound algorithm. Because Problem 2.4 is an optimization problem in a finite dimensional space, many of the heuristics currently employed for solving nondynamic problems can be directly applied. Unfortunately, many standard branch-and-bound heuristics cannot be immediately applied because the functions defining the problem are known only implicitly via the numerical solution of the ODEs. However, I believe that domain specific heuristics can be developed that exploit the dynamic structure of Problem 2.4 to accelerate the overall convergence to a global minimum. Finally, as mentioned in the concluding remarks of Chapter 7, the current implementation for solving nonlinear problems is somewhat limited by the complexity of the code generated by the compiler. This problem is currently being addressed by the development of a second generation compiler that implements algorithmic differentiation techniques in applying the nonlinear theory.

The limitations of the current theory outlined in the previous paragraph only begin to discuss future areas in which to continue research into global dynamic optimization. Already, substantial work has begun on applying the theory and implementation I have developed for solving Problem 2.4 to solving optimization problems with hybrid systems and to solving mixed-integer dynamic optimization problems. Additionally, the field is completely open to extend this work to problems with differential algebraic equations and to problems with partial differential equations. As these theories mature, practitioners will certainly wish to apply these techniques to industrially-sized problems, and this will almost certainly open entirely new endeavors into scaling the results to large systems. Therefore, while this chapter concludes the research in my doctoral thesis, I believe enough outstanding issues exist for this chapter to represent

the beginning of many theses yet to come.

Appendix A

LibBandB User's Manual and API

A.1 Introduction

LibBandB is a callable library written in C++ containing the branch-and-bound component of a global optimization algorithm (the user must provide upper and lower bounding information him/herself). The code itself is a derivative work from the global NLP solver written by Dr. Edward P. Gatzke in his postdoctoral appointment at MIT in the academic year 2000-2001. This original code, itself a component of KABAGE, was written in a combination of C and Fortran. In early 2001, I simply modified Ed's code to suit my purposes for solving global dynamic optimization problems and libBandB v. 1.0 was born (although at the time, it was only one source file called bandb.c). At the end of 2001, I completely rewrote large portions of bandb.c and added very simple heuristics to the branch-and-bound algorithm (v. 2.0). Eventually, the code, while functional, became a complete mess and needed a complete rewrite from scratch. Additionally, I felt the code would benefit greatly (at least in organization) from an object oriented design; thus, in November 2002, I began designing and writing libBandB v. 3.0. Version 3.1 added many heuristics and user options not yet implemented in 3.0. Version 3.2 placed the entire library in the libBandB namespace and added fathoming based on feasibility of the upper and lower bounding problems.

There are certainly other branch-and-bound options available in the community

(pretty good ones at that such as BARON from which this code does indeed draw some inspiration), so why did I choose to write my own? Well, I had several reasons. First, our research group did not have any other branch-and-bound software for linux available at the time I began this project. Second, I enjoy writing code and used version 2 as an introduction to writing in C and version 3 as an introduction to writing in C++ (and no, for those of you who know me personally and are wondering, there will never be a Perl version). Finally, of the software currently available, to the best of my knowledge, none are available with source. Personally, I feel that academic work, particularly that funded by taxpayer dollars, should not be designed for commercialization. Specifically though, I fervently believe that academics should openly share their knowledge to help promote scientific work simply for the sake of science. Closed source software and research group "trade secrets" seem very counterintuitive to the ideals of academia and leave other researchers wasting time reinventing the wheel (not to mention the unreproducibility of published work). I also think that if someone wants to use a piece of code in academic work, he/she should know exactly what the code is doing. Plus, I have been frustrated to no end trying to use software that does not work properly (e.g. seg faults) with no recourse to fix the errors myself. Of course, anyone using someone else's work ought naturally to properly acknowledge the original author **cough, cough**. Therefore, my intention is that this code should NEVER be distributed without source. I am not particularly concerned with what MIT may choose to do with licensing fees and such provided the source is always included. All that said, while anyone is free to make whatever modifications they like to the code, I absolutely will NOT fix bugs in source code modified by someone other than myself. I will, however, fix legitimate bugs myself and entertain any comments about improving the code or adding new features (although this does not mean that I will necessarily accept and incorporate every feature suggested).

This document is strictly a user's manual. To this end, I will explain the organization of the code and how to use the code. I do not intend to explain branch-and-bound algorithms, reduce heuristics, global optimization, or any other such notions in this

manual. Instead, the interested reader is referred to the articles and books at the end of the document for further reading on theoretical topics.

A.2 Organization of the Code

As previously stated, the code itself is written in C++. However, one can easily use the code with only a working knowledge of C (or maybe even Fortran). More or less, the library may be called with no knowledge of C++ beyond the C subset. In fact, given suitable wrappers accommodating the C linkage style, the library could itself be directly called from C or Fortran. The only compiler on which the code has been tested is gcc in linux (version 3.2 and higher). However, the library should compile with any ANSI compliant C++ compiler. Because of its use of namespaces, I am uncertain if it will compile properly with older versions of gcc. With the -Wall flag, the code should compile without any compiler warnings. If this is not the case and a compiler older than gcc 3.2 is being used, I recommend a compiler upgrade.

The code itself is divided into two directories. libBandB contains the files belonging to the library proper, while wrappers contains a wrapper illustrating the calling of libBandB and a wrapper illustrating a call to the optimizer NPSOL. The optimization wrapper is written for dual use for computing both the upper and lower bounding problems; the user may wish to provide separate routines for each. Due to licensing, NPSOL itself cannot be shipped with libBandB. In order to build the library itself, one should type 'make' in the libBandB directory. In order to build the wrappers and include them into the library, the user should type 'make insert' in either the libBandB directory or the wrappers directory. The components of each directory are now examined in more detail; the interface for individual function calls is explained in Section A.3.

As previously stated, the main components of libBandB are found in the libBandB directory. The files comprising the library are auxiliary.cc, Node.cc, Options.cc, BandB.cc, and their associated header files. The file auxiliary.cc contains the built-in functions used for output handling; this includes cli_output and the not

yet implemented gui_output. A third option, user_output, may be provided by the user if a custom output routine is desired. The file options.cc contains the member functions for the Options class required in order to scan and parse the options file. The options are made available to the user by declaring

```
extern class Options UserOpts;
```

in the libBandB namespace and including the Options.h header file. The options can then be accessed via the public member functions defined in the Options.h header file. Node.cc contains the internal workings of each node in the branch-and-bound tree. Nodes are created through a class inheritance. RootNode and RegularNode are both derived classes from the base class Node; however, polymorphism is not implemented in this library. The definition of the Node class is found in the header file Node.h. BandB.cc contains the data structures, memory management, timing utilities, and initialization procedures (initializing static class members, reading the options file, etc.). BandB.cc stores the nodes in memory in a minimum heap data structure implemented via a vector container built using the C++ standard template library (STL). This provides an efficient mechanism for selecting the next node by the lowest lower bound. Because of the modular design, a different data structure could be chosen for node storage (maybe a linked list) without changing the inner workings of the nodes themselves. BandB.h contains the definition of some global variables and functions used throughout the entire library.

The wrapper directory contains two files: main.cc and npsolWrapper.cc. main.cc is simply an example of a main program that calls the branch-and-bound function BandB with the default argument of "options.bnb" as the options file (more about this in the Application Program Interface Section). The npsolWrapper.cc file contains a wrapper for computing the upper and lower bounding arguments with NPSOL for an unconstrained optimization problem. Additionally, several supporting functions are also provided. The user should note the use of statically declared pointers to dynamically allocated memory. Because the optimizer is likely called many times in the branch-and-bound routine, this prevents the overhead of allocating and deallocating

memory unnecessarily. If for some reason the size of the problem is changed within the execution of the user's program, the size of the memory should be declared to be the largest necessary or simply reallocated with the problem size at each function call. A return value of false from any of the solvers causes the program to exit with a user defined fatal error.

A.3 Application Program Interface

In order to use libBandB, the user must be able to call the branch-and-bound solver and provide upper and lower bounds for the branch-and-bound solver. Optionally, the user may wish to provide his/her own output routine. These three topics are discussed below. Remember, the entire library is contained within the libBandB namespace. For notational convenience, the libBandB:: prefix has been dropped from the declarations in the API, but appear appropriately in the sample code. The user must remember to use the namespace appropriately.

A.3.1 Function BandB

The function BandB is the main function responsible for controlling the branch-and-bound solver. The function prototype is given by

```
double BandB(const int np, double *lowerBounds, double *upperBounds,
             double *initialPoint,
             const string inputFileName = "options.bnb")
```

Input:

	np:	the number of parameters in the optimization problem
	lowerBounds:	array of size np containing the lower bounds on the parameters
	upperBounds:	array of size np containing the upper bounds on the parameters
	initialPoint:	array of size np containing the initial guess for the upper bounding problem

inputFileName: optional string providing the name of the options file–if this argument is not provided, the default name of options.bnb is used

Output: initialPoint: (pointer to an) array of size np containing the point at which the global solution is obtained

Return Value: Global optimum

A.3.2 Upper and Lower Bounding Functions

The following functions must be provided by the user for computing the upper and lower bounds at each node. The user is given the choice of three different routines: solver1, solver2, solver3. Which solver is used for the upper and lower problems is determined at run time via the options file. This permits flexibility to use different optimizers to solve problems without the need for recompiling. If only one function is ever employed, the others must be provided as dummy routines for linking purposes. The wrappers directory provides file npsolWrapper.cc that provides a wrapper for NPSOL for unconstrained problems. The prototype for the optimizers is given below where X=1,2,3. Note that the parenthetical expressions next to the integer return types are enumerated types that can be used instead of the integer value. The enumeration is defined in BandB.h.

```
int solverX(const int np, const int probe, const int probeVar,
            const int problemType, double *lowerBounds,
            double *upperBounds, double *parameters, double *objf,
            double *lagrangeMultipliers)
```

Input:

np: the number of parameters in the optimization problem

probe: -1 if probing lower bound, +1 if probing upper bound, 0 if probing not active

	probeVar:	if probing active, index of the variable being probed
	problemType:	0 for lower bounding problem, +1 for upper bounding problem
	lowerBounds:	array of size np containing current lower bounds
	upperBounds:	array of size np containing current upper bounds
	parameters:	array of size np containing guess for the optimizer
Output:	parameters:	array of size np containing point at which minimum occurred
	objf:	value of the objective function at minimum
	lagrangeMultipliers:	lagrange multipliers (does not need to be set if not using optimality based reduce heuristics) Positive for a variable at a lower bound and negative for a variable at an upper bound.
Return Value:	-3 (FATAL):	User defined failure. Program terminates.
	0 (FAILURE)	The user defines that the optimizer has failed. For the upper bounding problem, the node is retained without updating the incumbent. For the lower bounding problem, this is considered an error, and the program terminates.
	1 (FEASIBLE)	User defines that the optimizer has performed normally.
	2 (INFEASIBLE)	User defines that the node is infeasible; the

node is fathomed.

A.3.3 Output Functions

The program provides output functions cli_output and gui_output (gui not yet implemented) and the option for the user to write his/her own output routine in the user defined function user_output. The output routine is chosen at run time via the options file. If BandB.h is included in the user's file, then the output function can be accessed via a function pointer (*BandB_output). Additionally, intermediate output may be compiled out of the program by setting the preprocessor flag SUPPRESS_OUTPUT in BandB.h and recompiling the entire library. The prototype is given below (gui_output and user_output have the same prototype).

```
void cli_output(const int code, const string &outputMessage)
```

Input:	code:	integer code–defined by an enumeration in BandB.h
	outputMessage	contains a string output message to be displayed

A.4 The Options File

By default, the options file for which BandB looks is options.bnb in the directory from which the program was executed; however, the user may change this via the optional fifth argument to BandB. Additionally, the program may be executed without an options file in which case all variables will be set to their default values. The options file possesses the following format:

keyword : option

Whitespace is ignored, and the file is case insensitive except for the name of the output file. The # symbol denotes a comment line; any remarks following a # until the end of the line are ignored. The order of the options is irrelevant, but only one option

may be set per line. The options are set at run time, and the options file is read only once immediately after BandB is called. The options themselves are contained in a global Options class object. By including the Options.h header file and declaring

```
extern class Options UserOpts;
```

in the libBandB namespace, the user can access the options via the public member functions of the Options class. The options are now examined in more detail.

absolute tolerance **default: 0.001**

relative tolerance **default: 0.001**

Both an absolute and relative tolerance are always active as fathoming criterion. A node is retained if both of the following tests are true:

$$L < U - \varepsilon_a$$

and

$$L < U - |U| \cdot \varepsilon_r$$

where L is the lower bound in a node and U is the current incumbent value. The following member functions return the absolute and relative tolerances respectively.

```
UserOpts.absTol()
UserOpts.relTol()
```

optimizer tolerance **default:** 1×10^{-5}

This is a value that the user may wish to set for use in the optimizer routines. BandB does not use this value (solver3 in npsolWrapper.cc uses this value). It is returned to the user with the member function

```
UserOpts.optimizer_tolerance()
```

lower optimizer **default: solver1**

upper optimizer **default: solver1**

The options for each of these are solver1, solver2, and solver3. The member functions return a pointer to a function.

```
UserOpts.plower_solver()
UserOpts.pupper_solver()
```

npsol function precision **default:** 1×10^{-8}

This is a value specifically used for solver3 in npsolWrapper.cc. The user can use this for any double precision value he/she desires.

```
UserOpts.npsol_function_precision()
```

domain reduce **default: false**

maximum reduce loops **default: 3**

reReduce percent **default: 0.25**

reduce round precision **default: 4**

The above four options all pertain to post-processing optimality based domain reduction. The first option simply turns domain reduction on and off. The second option specifies how many times domain reduction will be performed at a given node. By specifying 0, domain reduction is performed only after the initial solving of the lower bounding problem. By specifying a number greater than 0, domain reduction may "loop" and repeat the calculation of the lower bounding problem and domain reduction steps up to the specified maximum. Domain reduction, at least with 0 maximum loops, is highly recommended because it is very effective and relatively cheap. The reReduce percent is the percentage reduction on any individual variable required to enter a reduction loop. The reduce round precision is the number of decimal places to

which quantities are rounded in the domain reduction step. This should be set conservatively based on the optimality tolerance so that not too much of the optimization space is removed by domain reduction.

```
UserOpts.domainReduce()
UserOpts.maxReduceLoops()
UserOpts.reReducePercent()
UserOpts.rndPrec()
```

violation scale factor one **default: 1.0**

violation scale factor two **default: 0.0**

violation scale factor three **default: 0.0**

violation scale factor four **default: 0.0**

ignore bounds violation **default: false**

bounds tolerance **default:** 1×10^{-5}

relative bounds tolerance **default: 0.0**

relative bounds style **default: root**

Which variable is selected for branching next is computed based on a compound violation computed by the following formula:

$$branchVar = \arg\max \left\{ \alpha_1 \left(ubd[i] - lbd[i]\right) + \alpha_2 \left(\frac{ubd[i] - lbd[i]}{ubd_r[i] - lbd_r[i]} \right) + \alpha_3 \left| p_U - p_L \right| + \alpha_4 \left(\frac{\left| p_U - p_L \right|}{ubd_r[i] - lbd_r[i]} \right) \right\}$$

where α_i is the ith violation scale factor, $ubd[i]$ is the ith upper bound at the current node, $ubd_r[i]$ is the ith upper bound at the root node, $lbd[i]$ is the ith lower bound at the current node, $lbd_r[i]$ is the ith lower bound at the root node, $p_U[i]$ is the ith parameter value at the location of the upper bound for the current node, and $p_L[i]$ is the ith parameter value at the location of the lower bound for the current node.

By default, if the parameter value for the lower bound is at one of its bounds, this variable will not be selected for branching. However, this violation can be ignored by setting the ignore bounds violation option to true. Finally, for fine tuning, the user can control how close to the bound is considered at the bound. This is controlled by both the bounds tolerance option and the relative bounds tolerance option. The relative option may be set to be relative to the bounds at the root node or the current node. This option is useful for preventing too much branching on one parameter or never branching on a given parameter.

```
UserOpts.scaleOne()
UserOpts.scaleTwo()
UserOpts.scaleThree()
UserOpts.scaleFour()
UserOpts.ignoreBoundsViolation()
UserOpts.bdsTol()
UserOpts.relBdsTol()
UserOpts.relBdsStyle()
```

branch location **default: bisection**

bisection limit **default: 10**

Once the branch variable is chosen, the branching location must be set. The options are bisection, incumbent, and omega. Bisection sets the branch location as the mid point of the current bounds. Incumbent sets the branch location to the location of the incumbent if it is inside the current bounds else defaults to bisection. Omega sets the branch location to the location of the solution of the lower bounding problem. For either incumbent or omega branching, to ensure finiteness of the algorithm, the search must be exhaustive. Therefore, every once in a while, bisection must be employed for the branching location. This is controlled by the bisection limit option. That is, bisection is employed every X times branching occurs, where X is the value set by the bisection limit option.

```
UserOpts.branchLocation()
UserOpts.bisectionLimit()
```

root node probing **default: false**

This option turns root node probing on and off. Root node probing probes at all np variables.

```
UserOpts.rootNodeProbing()
```

color **default: true**

This option turns color on and off for the default cli_output routine.

```
UserOpts.color()
```

output file **default: no output file**

By default, output is not directed to a file. By including this option, output will be directed to the file given by the user. The file name is case sensitive.

```
UserOpts.outputToFile()
UserOpts.outputFileName()
```

suppress terminal output **default: false**

This option suppresses the terminal output (except fatal errors) from the cli_output routine.

```
UserOpts.suppressOutput()
```

output routine **default: cli_output**

This option sets the output routine to either cli_output or user_output. gui_output is also reserved; however, no gui is currently implemented.

```
extern pBandB_output BandB_output;
```

Appendix B

YACC grammar for GDOC

```
%{
/* header information */
%}
%union{
  int ival;
  double dval;
  char *pcval;
  char cval;
  class ptNode *ptNode;
}
%token PARAMETER
%token STATE
%token SINE
%token COSINE
%token EXPONENTIAL
%token INTEGRAND
%token NAT_LOG
%token LOG10
%token SQUARE_ROOT
%token ARRHENIUS
```

```
%token END
%token DECLARATION
%token EQUATION
%token INITIAL_CONDITION
%token TIME
%token REFERENCE
%token VALUES
%token BOUNDS
%token CONSTANT
%token CONSTANT_VALUES
%token <dval> DOUBLE
%token <ival> INTEGER
%token <pcval> NAME
/* ptNode is the base class for the parse trees */
%type <ptNode> Name
%type <ptNode> DeclareVar
%type <ival> Subscript
%type <ptNode> Number
%type <ptNode> Variable
%type <ptNode> LHS
%type <ptNode> Expression
%type <ptNode> SimpleEquation
%type <ptNode> Function
%left '+' '-'
%nonassoc UMINUS
%left '*' '/'
%right '^'
%%
ValidFile          : TdeclareSection TvalueSection TeqnSection TrefSection
                     TinitSection
```

```
                | TdeclareSection TvalueSection TeqnSection TrefSection
                  TinitSection OptionalSection
                ;
OptionalSection : OptionalUnit
                | OptionalUnit OptionalSection
                ;
OptionalUnit    : TboundsSection
                | TconstantSection
                | TintegrandSection
  ;
TintegrandSection : INTEGRAND Expression ';' END
                ;
TdeclareSection : DECLARATION DeclareSection END
                ;
DeclareSection  : DeclareUnit
                | DeclareUnit DeclareSection
                ;
DeclareUnit     : DeclareVariable
                | DeclareParameter
                | DeclareTime
                | DeclareConstant
                ;
DeclareConstant : CONSTANT ':' DeclareVar
                ;
DeclareTime     : TIME ':' TimeRange
                ;
DeclareVariable : STATE ':' DeclareVar
                ;
DeclareParameter : PARAMETER ':' DeclareVar
                ;
```

```
DeclareVar        : Name VarRange
                  ;
TvalueSection     : VALUES ValueSection END
                  ;
ValueSection      : Tvalue
                  | ValueSection Tvalue
                  ;
Tvalue            : SimpleEquation ':' '[' Expression ',' Expression ']'
                  ;
TconstantSection  : CONSTANT_VALUES ConstantSection END
                  ;
ConstantSection   : Cvalue
                  | ConstantSection Cvalue
                  ;
Cvalue            : SimpleEquation ';'
                  ;
TboundsSection    : BOUNDS BoundsSection END
                  ;
BoundsSection     : Bvalue
                  | BoundsSection Bvalue
                  ;
Bvalue            : Variable ':' BRange
                  ;
TimeRange         : '(' NumRange ')'
                  | '[' NumRange ']'
                  ;
VarRange          : '(' IntRange ')'
                  | '[' IntRange ']'
                  ;
BRange            : '[' NumRange ']'
```

```
                | '[' LRange ']'
                | '[' RRange ']'
                ;
RRange          : '-' '-' ',' Expression
                ;
LRange          : Expression ',' '-' '-'
                ;
NumRange        : Expression ',' Expression
                ;
IntRange        : INTEGER ':' INTEGER
                ;
TeqnSection     : EQUATION EqnSection END
                ;
EqnSection      : Teqn
                | EqnSection Teqn
                ;
Teqn            : SimpleEquation ';'
                ;
TrefSection     : REFERENCE RefSection END
                ;
RefSection      : Tref
                | RefSection Tref
                ;
Tref            : SimpleEquation ';'
                ;
TinitSection    : INITIAL_CONDITION InitSection END
                ;
InitSection     : Tinit
                | InitSection Tinit
                ;
```

```
Tinit           : SimpleEquation ';'
                ;
SimpleEquation  : LHS '=' Expression
                ;
LHS             : Variable
                | '$' Variable
                ;
Expression      : Expression '+' Expression
                | Expression '-' Expression
                | Expression '*' Expression
                | Expression '/' Expression
                | Expression '^' Expression
                | '+' Expression %prec UMINUS
                | '-' Expression %prec UMINUS
                | Function '(' Expression ')'
                | '(' Expression ')'
                | Number
                | Variable
                ;
Function        : SINE
                | COSINE
                | EXPONENTIAL
                | NAT_LOG
                | LOG10
                | SQUARE_ROOT
                | ARRHENIUS
                ;
Number          : DOUBLE
                | INTEGER
                ;
```

```
Variable          : Name
                  | Name Subscript
                  ;
Name              : NAME
                  ;
Subscript         : '(' INTEGER ')'
                  ;
%%
/* Semantic routines and error handling */
```

Appendix C

Experimental Data for Chapter 9

Appendix C contains the experimental data for the problems presented in Chapter 9. The data were collected by James W. Taylor via laser flash photolysis at a wavelength of 316 nm. The experimental setup is detailed in [84].

C.1 Experimental data at T = 273 K

Initial Conditions: $x_{Z,0} = 1.40 \times 10^{-4}$ M, $x_{Y,0} = 0.400$ M, $x_{O_2} = 2.00 \times 10^{-3}$ M

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
0.01	13.88	0.28	17.05	0.55	9.640	0.82	5.953
0.02	22.00	0.29	16.57	0.56	9.466	0.83	5.784
0.03	23.17	0.30	16.18	0.57	9.308	0.84	5.761
0.04	24.13	0.31	15.85	0.58	9.117	0.85	5.797
0.05	25.67	0.32	15.43	0.59	9.144	0.86	5.707
0.06	26.34	0.33	15.08	0.60	8.896	0.87	5.712
0.07	26.34	0.34	14.84	0.61	8.786	0.88	5.551
0.08	25.93	0.35	14.36	0.62	8.450	0.89	5.410
0.09	25.47	0.36	14.14	0.63	8.217	0.90	5.210
0.10	24.96	0.37	13.67	0.64	8.145	0.91	5.196
0.11	24.71	0.38	13.38	0.65	7.787	0.92	5.100
0.12	24.39	0.39	13.03	0.66	7.760	0.93	5.076
0.13	23.91	0.40	12.78	0.67	7.682	0.94	4.949
0.14	23.43	0.41	12.52	0.68	7.619	0.95	4.945
0.15	22.85	0.42	12.24	0.69	7.423	0.96	4.913
0.16	22.25	0.43	12.07	0.70	7.327	0.97	4.813
0.17	21.85	0.44	11.86	0.71	7.175	0.98	4.685
0.18	21.43	0.45	11.72	0.72	7.071	0.99	4.579
0.19	21.03	0.46	11.45	0.73	6.923	1.00	4.470
0.20	20.69	0.47	11.13	0.74	6.687	1.01	4.453
0.21	20.21	0.48	10.89	0.75	6.624	1.02	4.377
0.22	19.82	0.49	10.59	0.76	6.541	1.03	4.293
0.23	19.25	0.50	10.37	0.77	6.508	1.04	4.201
0.24	18.81	0.51	10.07	0.78	6.291	1.05	4.187
0.25	18.30	0.52	9.948	0.79	6.118	1.06	4.095
0.26	17.82	0.53	9.848	0.80	6.045	1.07	4.172
0.27	17.52	0.54	9.762	0.81	5.973	1.08	3.976

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
1.09	4.019	1.38	3.271	1.67	2.896	1.96	2.520
1.10	3.982	1.39	3.298	1.68	2.889	1.97	2.536
1.11	3.895	1.40	3.320	1.69	2.854	1.98	2.502
1.12	3.760	1.41	3.290	1.70	2.723	1.99	2.470
1.13	3.729	1.42	3.130	1.71	2.726	2.00	2.422
1.14	3.892	1.43	3.063	1.72	2.706	2.01	2.430
1.15	3.859	1.44	3.098	1.73	2.713	2.02	2.506
1.16	3.855	1.45	3.099	1.74	2.733	2.03	2.538
1.17	3.849	1.46	3.145	1.75	2.732	2.04	2.406
1.18	3.838	1.47	3.052	1.76	2.601	2.05	2.236
1.19	3.863	1.48	3.056	1.77	2.637	2.06	2.313
1.20	3.695	1.49	2.943	1.78	2.592	2.07	2.463
1.21	3.812	1.50	2.862	1.79	2.679	2.08	2.423
1.22	3.821	1.51	2.823	1.80	2.663	2.09	2.306
1.23	3.748	1.52	2.828	1.81	2.634	2.10	2.330
1.24	3.761	1.53	2.797	1.82	2.559	2.11	2.370
1.25	3.676	1.54	2.716	1.83	2.686	2.12	2.406
1.26	3.675	1.55	2.756	1.84	2.637	2.13	2.463
1.27	3.352	1.56	2.677	1.85	2.697	2.14	2.479
1.28	3.404	1.57	2.716	1.86	2.635	2.15	2.306
1.29	3.398	1.58	2.745	1.87	2.640	2.16	2.302
1.30	3.296	1.59	2.745	1.88	2.599	2.17	2.350
1.31	3.299	1.60	2.680	1.89	2.519	2.18	2.343
1.32	3.247	1.61	2.609	1.90	2.406	2.19	2.489
1.33	3.349	1.62	2.628	1.91	2.373	2.20	2.505
1.34	3.360	1.63	2.712	1.92	2.342	2.21	2.329
1.35	3.401	1.64	2.640	1.93	2.453	2.22	2.422
1.36	3.384	1.65	2.807	1.94	2.440	2.23	2.276
1.37	3.306	1.66	2.778	1.95	2.500	2.24	2.390

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
2.25	2.409	2.54	2.389	2.83	2.219	3.12	2.116
2.26	2.467	2.55	2.353	2.84	2.337	3.13	2.104
2.27	2.280	2.56	2.276	2.85	2.359	3.14	2.092
2.28	2.470	2.57	2.359	2.86	2.193	3.15	2.089
2.29	2.235	2.58	2.259	2.87	2.119	3.16	2.071
2.30	2.350	2.59	2.293	2.88	2.085	3.17	2.155
2.31	2.396	2.60	2.223	2.89	2.162	3.18	2.175
2.32	2.467	2.61	2.249	2.90	2.165	3.19	2.195
2.33	2.463	2.62	2.297	2.91	2.277	3.20	2.179
2.34	2.310	2.63	2.349	2.92	2.226	3.21	2.188
2.35	2.299	2.64	2.312	2.93	2.304	3.22	2.179
2.36	2.282	2.65	2.303	2.94	2.196	3.23	2.192
2.37	2.216	2.66	2.282	2.95	2.155	3.24	2.178
2.38	2.297	2.67	2.273	2.96	2.121	3.25	2.158
2.39	2.247	2.68	2.316	2.97	2.109	3.26	2.243
2.40	2.220	2.69	2.287	2.98	2.162	3.27	2.195
2.41	2.088	2.70	2.274	2.99	2.158	3.28	2.106
2.42	2.175	2.71	2.206	3.00	2.145	3.29	2.209
2.43	2.200	2.72	2.142	3.01	2.192	3.30	2.250
2.44	2.269	2.73	2.148	3.02	2.229	3.31	2.185
2.45	2.320	2.74	2.125	3.03	2.217	3.32	2.240
2.46	2.357	2.75	2.048	3.04	2.153	3.33	2.199
2.47	2.284	2.76	2.101	3.05	2.004	3.34	2.304
2.48	2.273	2.77	2.111	3.06	1.993	3.35	2.316
2.49	2.316	2.78	2.200	3.07	1.996	3.36	2.396
2.50	2.350	2.79	2.193	3.08	1.989	3.37	2.384
2.51	2.303	2.80	2.276	3.09	2.044	3.38	2.306
2.52	2.357	2.81	2.188	3.10	2.024	3.39	2.183
2.53	2.384	2.82	2.197	3.11	2.092	3.40	2.266

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
3.41	2.227	3.70	2.111	3.99	1.946	4.28	2.099
3.42	2.399	3.71	2.151	4.00	1.977	4.29	2.070
3.43	2.263	3.72	1.980	4.01	2.082	4.30	1.929
3.44	2.293	3.73	1.956	4.02	2.018	4.31	2.030
3.45	2.180	3.74	1.932	4.03	2.050	4.32	1.891
3.46	2.233	3.75	2.018	4.04	1.993	4.33	1.958
3.47	2.188	3.76	2.128	4.05	2.070	4.34	2.003
3.48	2.252	3.77	2.105	4.06	1.939	4.35	1.955
3.49	2.229	3.78	2.227	4.07	2.047	4.36	1.943
3.50	2.343	3.79	2.135	4.08	2.129	4.37	1.977
3.51	2.277	3.80	2.018	4.09	2.141	4.38	2.044
3.52	2.273	3.81	1.958	4.10	2.202	4.39	1.914
3.53	2.327	3.82	1.905	4.11	2.205	4.40	1.935
3.54	2.219	3.83	1.962	4.12	2.172	4.41	1.946
3.55	2.166	3.84	2.007	4.13	2.183	4.42	2.091
3.56	2.082	3.85	2.220	4.14	2.095	4.43	2.085
3.57	2.118	3.86	2.151	4.15	1.977	4.44	2.097
3.58	2.111	3.87	2.105	4.16	1.833	4.45	2.077
3.59	2.018	3.88	2.070	4.17	1.870	4.46	2.051
3.60	1.962	3.89	2.010	4.18	1.907		
3.61	2.050	3.90	2.040	4.19	1.911		
3.62	2.041	3.91	2.026	4.20	1.986		
3.63	2.099	3.92	2.121	4.21	1.996		
3.64	2.169	3.93	2.131	4.22	1.972		
3.65	2.153	3.94	2.053	4.23	1.884		
3.66	2.131	3.95	2.033	4.24	1.928		
3.67	2.079	3.96	1.960	4.25	2.131		
3.68	2.079	3.97	1.909	4.26	2.106		
3.69	2.169	3.98	1.928	4.27	2.170		

C.2 Experimental data at T = 298 K

Initial Conditions: $x_{Z,0} = 1.40 \times 10^{-4}$ M, $x_{Y,0} = 0.400$ M, $x_{O_2} = 1.90 \times 10^{-3}$ M

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
0.01	14.45	0.28	19.20	0.55	12.87	0.82	9.023
0.02	21.05	0.29	18.83	0.56	12.62	0.83	8.974
0.03	22.53	0.30	18.60	0.57	12.49	0.84	8.828
0.04	23.59	0.31	18.43	0.58	12.19	0.85	8.764
0.05	22.36	0.32	18.04	0.59	12.15	0.86	8.603
0.06	21.93	0.33	17.81	0.60	11.89	0.87	8.359
0.07	23.44	0.34	17.45	0.61	11.79	0.88	8.240
0.08	24.54	0.35	17.13	0.62	11.55	0.89	8.042
0.09	25.03	0.36	16.85	0.63	11.34	0.90	8.160
0.10	24.92	0.37	16.53	0.64	11.11	0.91	8.130
0.11	24.81	0.38	16.36	0.65	11.02	0.92	8.081
0.12	24.19	0.39	16.24	0.66	11.05	0.93	7.962
0.13	23.81	0.40	15.99	0.67	11.06	0.94	7.849
0.14	23.54	0.41	15.69	0.68	10.97	0.95	7.648
0.15	23.59	0.42	15.59	0.69	10.75	0.96	7.574
0.16	23.42	0.43	15.36	0.70	10.50	0.97	7.403
0.17	23.07	0.44	15.31	0.71	10.42	0.98	7.273
0.18	22.51	0.45	15.05	0.72	10.34	0.99	7.250
0.19	22.21	0.46	14.82	0.73	10.19	1.00	7.255
0.20	21.89	0.47	14.38	0.74	9.952	1.01	7.127
0.21	21.57	0.48	14.20	0.75	9.786	1.02	7.152
0.22	21.25	0.49	13.90	0.76	9.618	1.03	6.997
0.23	21.03	0.50	13.83	0.77	9.744	1.04	7.001
0.24	20.70	0.51	13.72	0.78	9.660	1.05	6.817
0.25	20.39	0.52	13.61	0.79	9.530	1.06	6.710
0.26	19.90	0.53	13.39	0.80	9.267	1.07	6.602
0.27	19.38	0.54	13.17	0.81	9.170	1.08	6.669

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
1.09	6.643	1.38	5.400	1.67	4.399	1.96	3.699
1.10	6.573	1.39	5.351	1.68	4.321	1.97	3.800
1.11	6.495	1.40	5.176	1.69	4.194	1.98	3.641
1.12	6.438	1.41	5.115	1.70	4.333	1.99	3.736
1.13	6.488	1.42	5.031	1.71	4.343	2.00	3.651
1.14	6.444	1.43	5.039	1.72	4.361	2.01	3.668
1.15	6.415	1.44	5.058	1.73	4.301	2.02	3.690
1.16	6.273	1.45	4.958	1.74	4.288	2.03	3.644
1.17	6.269	1.46	4.903	1.75	4.242	2.04	3.639
1.18	6.134	1.47	4.910	1.76	4.178	2.05	3.536
1.19	6.154	1.48	4.859	1.77	4.078	2.06	3.648
1.20	5.989	1.49	4.907	1.78	4.047	2.07	3.639
1.21	6.073	1.50	4.957	1.79	4.107	2.08	3.854
1.22	5.961	1.51	4.995	1.80	4.020	2.09	3.811
1.23	5.865	1.52	4.872	1.81	3.993	2.10	3.804
1.24	5.767	1.53	4.795	1.82	4.049	2.11	3.708
1.25	5.729	1.54	4.673	1.83	4.044	2.12	3.717
1.26	5.834	1.55	4.592	1.84	4.121	2.13	3.543
1.27	5.868	1.56	4.568	1.85	4.206	2.14	3.540
1.28	5.769	1.57	4.573	1.86	4.192	2.15	3.543
1.29	5.693	1.58	4.578	1.87	4.171	2.16	3.659
1.30	5.671	1.59	4.651	1.88	3.950	2.17	3.697
1.31	5.580	1.60	4.618	1.89	3.938	2.18	3.645
1.32	5.586	1.61	4.548	1.90	3.854	2.19	3.581
1.33	5.473	1.62	4.444	1.91	3.880	2.20	3.426
1.34	5.414	1.63	4.366	1.92	3.921	2.21	3.440
1.35	5.448	1.64	4.354	1.93	3.897	2.22	3.399
1.36	5.479	1.65	4.409	1.94	3.847	2.23	3.332
1.37	5.532	1.66	4.467	1.95	3.792	2.24	3.370

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
2.25	3.367	2.54	3.347	2.83	3.049	3.12	2.891
2.26	3.376	2.55	3.270	2.84	3.092	3.13	2.973
2.27	3.429	2.56	3.085	2.85	3.110	3.14	2.896
2.28	3.362	2.57	3.233	2.86	3.007	3.15	2.852
2.29	3.362	2.58	3.147	2.87	3.003	3.16	2.903
2.30	3.405	2.59	3.217	2.88	2.749	3.17	2.791
2.31	3.327	2.60	3.119	2.89	2.917	3.18	2.891
2.32	3.383	2.61	3.042	2.90	2.882	3.19	2.797
2.33	3.270	2.62	2.965	2.91	2.915	3.20	2.776
2.34	3.317	2.63	2.912	2.92	2.950	3.21	2.722
2.35	3.291	2.64	2.896	2.93	2.941	3.22	2.821
2.36	3.327	2.65	3.040	2.94	2.843	3.23	2.757
2.37	3.315	2.66	2.944	2.95	2.805	3.24	2.807
2.38	3.431	2.67	2.952	2.96	2.763	3.25	2.652
2.39	3.344	2.68	2.962	2.97	2.830	3.26	2.618
2.40	3.297	2.69	2.985	2.98	2.893	3.27	2.591
2.41	3.277	2.70	3.175	2.99	2.855	3.28	2.613
2.42	3.202	2.71	3.262	3.00	2.825	3.29	2.725
2.43	3.085	2.72	3.200	3.01	2.836	3.30	2.612
2.44	3.096	2.73	3.101	3.02	2.992	3.31	2.667
2.45	3.024	2.74	2.849	3.03	2.914	3.32	2.733
2.46	3.075	2.75	2.831	3.04	2.967	3.33	2.734
2.47	3.232	2.76	2.803	3.05	2.944	3.34	2.703
2.48	3.187	2.77	2.848	3.06	2.813	3.35	2.721
2.49	3.224	2.78	2.905	3.07	2.900	3.36	2.700
2.50	3.256	2.79	2.887	3.08	2.872	3.37	2.706
2.51	3.205	2.80	2.955	3.09	2.803	3.38	2.676
2.52	3.274	2.81	3.057	3.10	2.810	3.39	2.558
2.53	3.373	2.82	3.105	3.11	2.809	3.40	2.540

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
3.41	2.540	3.70	2.763	3.99	2.601	4.28	2.458
3.42	2.561	3.71	2.713	4.00	2.504	4.29	2.515
3.43	2.637	3.72	2.624	4.01	2.640	4.30	2.411
3.44	2.588	3.73	2.616	4.02	2.612	4.31	2.512
3.45	2.551	3.74	2.645	4.03	2.512	4.32	2.344
3.46	2.606	3.75	2.622	4.04	2.435	4.33	2.423
3.47	2.688	3.76	2.728	4.05	2.449	4.34	2.429
3.48	2.734	3.77	2.595	4.06	2.359	4.35	2.502
3.49	2.752	3.78	2.637	4.07	2.418	4.36	2.540
3.50	2.821	3.79	2.594	4.08	2.447	4.37	2.613
3.51	2.659	3.80	2.646	4.09	2.522	4.38	2.557
3.52	2.767	3.81	2.722	4.10	2.464	4.39	2.546
3.53	2.749	3.82	2.709	4.11	2.424	4.40	2.552
3.54	2.799	3.83	2.624	4.12	2.365	4.41	2.568
3.55	2.734	3.84	2.576	4.13	2.349	4.42	2.607
3.56	2.698	3.85	2.583	4.14	2.494	4.43	2.507
3.57	2.745	3.86	2.633	4.15	2.519	4.44	2.506
3.58	2.628	3.87	2.622	4.16	2.500	4.45	2.350
3.59	2.689	3.88	2.651	4.17	2.377	4.46	2.442
3.60	2.689	3.89	2.701	4.18	2.463		
3.61	2.707	3.90	2.624	4.19	2.365		
3.62	2.794	3.91	2.628	4.20	2.377		
3.63	2.700	3.92	2.537	4.21	2.378		
3.64	2.727	3.93	2.625	4.22	2.467		
3.65	2.685	3.94	2.566	4.23	2.381		
3.66	2.649	3.95	2.518	4.24	2.390		
3.67	2.763	3.96	2.402	4.25	2.371		
3.68	2.760	3.97	2.563	4.26	2.455		
3.69	2.920	3.98	2.519	4.27	2.574		

C.3 Experimental data at T = 323 K

Initial Conditions: $x_{Z,0} = 1.40 \times 10^{-4}$ M, $x_{Y,0} = 0.400$ M, $x_{O_2} = 1.40 \times 10^{-3}$ M

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
0.01	94.73	0.28	2.004	0.55	1.482	0.82	1.100
0.02	2.219	0.29	1.992	0.56	1.473	0.83	1.090
0.03	2.435	0.30	1.971	0.57	1.451	0.84	1.073
0.04	2.522	0.31	1.942	0.58	1.426	0.85	1.065
0.05	2.534	0.32	1.903	0.59	1.401	0.86	1.057
0.06	2.521	0.33	1.874	0.60	1.389	0.87	1.048
0.07	2.498	0.34	1.836	0.61	1.365	0.88	1.042
0.08	2.472	0.35	1.829	0.62	1.360	0.89	1.025
0.09	2.443	0.36	1.831	0.63	1.345	0.90	1.012
0.10	2.414	0.37	1.807	0.64	1.358	0.91	9.935
0.11	2.394	0.38	1.796	0.65	1.332	0.92	9.994
0.12	2.367	0.39	1.765	0.66	1.315	0.93	9.972
0.13	2.345	0.40	1.731	0.67	1.296	0.94	9.879
0.14	2.329	0.41	1.711	0.68	1.276	0.95	9.840
0.15	2.307	0.42	1.704	0.69	1.253	0.96	9.656
0.16	2.302	0.43	1.686	0.70	1.237	0.97	9.639
0.17	2.296	0.44	1.673	0.71	1.227	0.98	9.421
0.18	2.274	0.45	1.646	0.72	1.208	0.99	9.406
0.19	2.238	0.46	1.640	0.73	1.201	1.00	9.273
0.20	2.195	0.47	1.620	0.74	1.182	1.01	9.112
0.21	2.174	0.48	1.612	0.75	1.173	1.02	8.973
0.22	2.160	0.49	1.597	0.76	1.154	1.03	8.802
0.23	2.118	0.50	1.587	0.77	1.138	1.04	8.793
0.24	2.083	0.51	1.554	0.78	1.139	1.05	8.873
0.25	2.038	0.52	1.548	0.79	1.123	1.06	8.717
0.26	2.031	0.53	1.537	0.80	1.118	1.07	8.740
0.27	2.024	0.54	1.519	0.81	1.115	1.08	8.668

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
1.09	8.510	1.38	6.831	1.67	5.782	1.96	4.959
1.10	8.619	1.39	6.846	1.68	5.919	1.97	5.135
1.11	8.545	1.40	6.735	1.69	5.752	1.98	4.994
1.12	8.346	1.41	6.562	1.70	5.754	1.99	4.984
1.13	8.168	1.42	6.553	1.71	5.744	2.00	4.930
1.14	8.012	1.43	6.571	1.72	5.765	2.01	4.940
1.15	7.922	1.44	6.449	1.73	5.726	2.02	4.888
1.16	8.019	1.45	6.318	1.74	5.411	2.03	4.845
1.17	7.926	1.46	6.320	1.75	5.580	2.04	4.734
1.18	8.045	1.47	6.343	1.76	5.474	2.05	4.698
1.19	7.861	1.48	6.370	1.77	5.460	2.06	4.733
1.20	7.819	1.49	6.354	1.78	5.479	2.07	4.658
1.21	7.769	1.50	6.202	1.79	5.513	2.08	4.707
1.22	7.767	1.51	6.200	1.80	5.565	2.09	4.631
1.23	7.672	1.52	6.148	1.81	5.513	2.10	4.769
1.24	7.667	1.53	6.152	1.82	5.555	2.11	4.464
1.25	7.602	1.54	6.295	1.83	5.355	2.12	4.514
1.26	7.534	1.55	6.295	1.84	5.358	2.13	4.449
1.27	7.506	1.56	6.234	1.85	5.276	2.14	4.507
1.28	7.407	1.57	6.159	1.86	5.334	2.15	4.553
1.29	7.283	1.58	6.022	1.87	5.269	2.16	4.641
1.30	7.206	1.59	6.027	1.88	5.332	2.17	4.560
1.31	7.078	1.60	6.246	1.89	5.318	2.18	4.615
1.32	7.217	1.61	6.195	1.90	5.318	2.19	4.440
1.33	7.131	1.62	6.193	1.91	5.196	2.20	4.519
1.34	7.034	1.63	5.986	1.92	5.213	2.21	4.380
1.35	6.980	1.64	5.858	1.93	5.102	2.22	4.425
1.36	6.864	1.65	5.719	1.94	4.982	2.23	4.474
1.37	6.786	1.66	5.896	1.95	5.036	2.24	4.469

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
2.25	4.332	2.54	3.981	2.83	3.814	3.12	3.494
2.26	4.300	2.55	3.887	2.84	3.650	3.13	3.465
2.27	4.426	2.56	3.932	2.85	3.741	3.14	3.459
2.28	4.351	2.57	3.934	2.86	3.679	3.15	3.457
2.29	4.360	2.58	3.836	2.87	3.702	3.16	3.543
2.30	4.267	2.59	3.893	2.88	3.788	3.17	3.457
2.31	4.327	2.60	3.870	2.89	3.810	3.18	3.417
2.32	4.370	2.61	3.924	2.90	3.699	3.19	3.470
2.33	4.377	2.62	3.925	2.91	3.558	3.20	3.425
2.34	4.402	2.63	3.914	2.92	3.439	3.21	3.320
2.35	4.262	2.64	3.870	2.93	3.348	3.22	3.347
2.36	4.233	2.65	3.799	2.94	3.276	3.23	3.355
2.37	4.180	2.66	3.782	2.95	3.228	3.24	3.340
2.38	4.247	2.67	3.849	2.96	3.357	3.25	3.270
2.39	4.197	2.68	3.873	2.97	3.353	3.26	3.221
2.40	4.099	2.69	3.890	2.98	3.541	3.27	3.306
2.41	3.929	2.70	3.863	2.99	3.617	3.28	3.358
2.42	3.873	2.71	3.802	3.00	3.578	3.29	3.303
2.43	3.817	2.72	3.476	3.01	3.728	3.30	3.353
2.44	3.914	2.73	3.521	3.02	3.643	3.31	3.447
2.45	3.912	2.74	3.549	3.03	3.600	3.32	3.377
2.46	3.949	2.75	3.640	3.04	3.591	3.33	3.442
2.47	4.037	2.76	3.704	3.05	3.549	3.34	3.377
2.48	4.054	2.77	3.924	3.06	3.516	3.35	3.445
2.49	4.158	2.78	3.934	3.07	3.370	3.36	3.303
2.50	4.146	2.79	3.910	3.08	3.398	3.37	3.253
2.51	4.157	2.80	3.897	3.09	3.333	3.38	3.293
2.52	4.102	2.81	3.837	3.10	3.392	3.39	3.285
2.53	4.037	2.82	3.797	3.11	3.491	3.40	3.253

t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)	t (μs)	I ($\times 10^2$)
3.41	3.185	3.70	3.186	3.99	2.724	4.28	2.749
3.42	3.280	3.71	3.075	4.00	2.798	4.29	2.611
3.43	3.367	3.72	3.093	4.01	2.912	4.30	2.571
3.44	3.388	3.73	2.969	4.02	2.858	4.31	2.502
3.45	3.201	3.74	3.037	4.03	2.793	4.32	2.602
3.46	3.265	3.75	3.087	4.04	2.760	4.33	2.537
3.47	3.211	3.76	2.901	4.05	2.731	4.34	2.602
3.48	3.298	3.77	2.896	4.06	2.792	4.35	2.604
3.49	3.300	3.78	2.625	4.07	2.734	4.36	2.589
3.50	3.171	3.79	2.909	4.08	2.817	4.37	2.640
3.51	3.136	3.80	2.838	4.09	2.813	4.38	2.548
3.52	3.090	3.81	2.921	4.10	2.675	4.39	2.655
3.53	3.180	3.82	2.970	4.11	2.639	4.40	2.731
3.54	3.188	3.83	2.995	4.12	2.706	4.41	2.803
3.55	3.338	3.84	2.926	4.13	2.762	4.42	2.785
3.56	3.337	3.85	2.813	4.14	2.665	4.43	2.757
3.57	3.178	3.86	2.833	4.15	2.650	4.44	2.718
3.58	3.250	3.87	2.866	4.16	2.665	4.45	2.670
3.59	3.198	3.88	2.886	4.17	2.731	4.46	2.591
3.60	3.285	3.89	2.929	4.18	2.764		
3.61	3.301	3.90	2.944	4.19	2.726		
3.62	3.221	3.91	2.822	4.20	2.630		
3.63	3.248	3.92	2.826	4.21	2.589		
3.64	3.230	3.93	2.856	4.22	2.545		
3.65	3.141	3.94	2.916	4.23	2.622		
3.66	3.123	3.95	2.785	4.24	2.683		
3.67	3.028	3.96	2.956	4.25	2.736		
3.68	3.063	3.97	2.879	4.26	2.800		
3.69	3.087	3.98	2.876	4.27	2.673		

Bibliography

[1] O. Abel and Wolfgang Marquardt. Scenario-integrated modeling and optimization of dynamic systems. *AIChE Journal*, 46(4):803–823, 2000.

[2] Malcolm Adams and Victor Guillemin. *Measure Theory and Probability.* Birkhäuser, Boston, 1996.

[3] C.S. Adjiman, I.P. Androulakis, and C.A. Floudas. Global optimization of mixed-integer nonlinear problems. *AIChE Journal*, 46(9):1769–1797, 2000.

[4] C.S. Adjiman, S. Dallwig, and C.A. Floudas. A global optimization method, αBB, for general twice-differentiable constrained NLPs – II. Implementation and computational results. *Computers and Chemical Engineering*, 22(9):1159–1179, 1998.

[5] C.S. Adjiman, S. Dallwig, C.A. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs – I. Theoretical advances. *Computers and Chemical Engineering*, 22(9):1137–1158, 1998.

[6] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques, and Tools.* Addison-Wesley Publishing Company, Reading, 1986.

[7] R.J. Allgor and P.I. Barton. Mixed-integer dynamic optimization. *Computers and Chemical Engineering*, 21(S):S451–S456, 1997.

[8] R.J. Allgor and P.I. Barton. Mixed-integer dynamic optimization I: Problem formulation. *Computers and Chemical Engineering*, 23:567–584, 1999.

[9] Ioannis P. Androulakis. Kinetic mechanism reduction based on an integer programming approach. *AIChE Journal*, 46(2):361–371, 2000.

[10] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, Philadelphia, 1998.

[11] M.J. Bagajewicz and V. Manousiouthakis. On the generalized Benders decomposition. *Computers and Chemical Engineering*, 15(10):691–700, 1991.

[12] J.R. Banga and W.D. Seider. Global optimization of chemical processes using stochastic algorithms. In C.A. Floudas and P.M. Pardalos, editors, *State of the Art in Global Optimization: Computational Methods and Applications.* Kluwer Academic Publishing, Dordrecht, The Netherlands, 1996.

[13] P.I. Barton. Hybrid systems: A powerful framework for the analysis of process operations. In *PSE Asia 2000: International Symposium on Design, Operation, and Control of Next Generation Chemical Plants*, pages 1–13, Kyoto, Japan, 2000.

[14] P.I. Barton, J.R. Banga, and S. Galán. Optimization of hybrid discrete/continuous dynamic systems. *Computers and Chemical Engineering*, 4(9/10):2171–2182, 2000.

[15] Mokhtar S. Bazaraa, Hanif D. Sherali, and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms.* John Wiley & Sons, Inc., New York, second edition, 1993.

[16] Leonard D. Berkovitz. On control problems with bounded state variables. *Journal of Mathematical Analysis and Applications*, 5:488–498, 1962.

[17] H.G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. In *Springer Series in Chemical Physics*, pages 102–125. Springer Verlag, 1981.

[18] R.G. Brusch and R.H. Schappelle. Solution of highly constrained optimal control problems using nonlinear programming. *AIAA Journal*, 11(2):135–136, 1973.

[19] Michael D. Canon, Jr. Clifton D. Cullum, and Elijah Polak. *Theory of Optimal Control and Mathematical Programming.* McGraw-Hill, Inc., New York, 1970.

[20] E.F. Carrasco and J.R. Banga. Dynamic optimization of batch reactors using adaptive stochastic algorithms. *Industrial & Engineering Chemistry Research*, 36(6):2252–2261, 1997.

[21] M. B. Carver. Efficient integration over discontinuities in ordinary differential equation simulations. *Mathematics and Computers in Simulation*, XX:190–196, 1978.

[22] S.S.L. Chang. Optimal control in bounded phase space. *Automatica*, 1:55–67, 1962.

[23] M.S. Charalambides. *Optimal Design of Integrated Batch Processes.* PhD thesis, University of London, 1996.

[24] Early A. Coddington and Norman Levinson. *Theory of Ordinary Differential Equations.* McGraw-Hill, New York, 1955.

[25] K. G. Denbigh. Optimum temperature sequences in reactors. *Chemical Engineering Science*, 8:125–132, 1958.

[26] V.D. Dimitriadis, N. Shah, and C.C. Pantelides. Modeling and safety verification of discrete/continuous processing systems. *AIChE Journal*, 43(4):1041–1059, 1997.

[27] Stuart Dreyfus. Variational problems with inequality constraints. *Journal of Mathematical Analysis and Applications*, 4:297–308, 1962.

[28] M.A. Duran and I.E. Grossmann. An outer approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[29] William R. Esposito and Christodoulos A. Floudas. Deterministic global optimization in nonlinear optimal control problems. *Journal of Global Optimization*, 17:97–126, 2000.

[30] William R. Esposito and Christodoulos A. Floudas. Global optimization for the parameter estimation of differential-algebraic systems. *Industrial and Engineering Chemical Research*, 39:1291–1310, 2000.

[31] James E. Falk and Richard M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15(9):550–569, 1969.

[32] W.F. Feehery, J.E. Tolsma, and P.I. Barton. Efficient sensitivity analysis of large-scale differential-algebraic systems. *Applied Numerical Mathematics*, 25(1):41–54, 1997.

[33] M. Fikar, M.A. Latifi, and Y. Creff. Optimal changeover profiles for an industrial depropanizer. *Chemical Engineering Science*, 54(13):2715–2120, 1999.

[34] Roger Fletcher and Sven Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[35] Christodoulos A. Floudas, Panos M. Pardalos, Claire S. Adjiman, William R. Esposito, Zeynep H. Gumus, Stephen T. Harding, John L. Klepeis, Clifford A. Meyer, and Carl A. Schweiger. *Handbook of Test Problems in Local and Global Optimization.* Kluwer Academic Publishers, Dordrecht, 1999.

[36] S. Galán and P.I. Barton. Dynamic optimization of hybrid systems. *Computers and Chemical Engineering*, 22(S):S183–S190, 1998.

[37] S. Galán, W.F. Feehery, and P.I. Barton. Parametric sensitivity functions for hybrid discrete/continuous systems. *Applied Numerical Mathematics*, 31(1):17–48, 1999.

[38] E.A. Galperin and Q. Zheng. Nonlinear observation via global optimization methods: Measure theory approach. *Journal of Optimization Theory and Applications*, 54(1):63–92, 1987.

[39] Efim A. Galperin and Quan Zheng. Variation-free iterative method for global optimal control. *International Journal of Control*, 50(5):1731–1743, 1989.

[40] Edward P. Gatzke, John E. Tolsma, and Paul I. Barton. Construction of convex relaxations using automated code generation techniques. *Optimization and Engineering*, 3:305–326, 2002.

[41] C. W. Gear. Simultaneous numerical solution of differential-algebraic equations. *IEEE Transactions on Circuit Theory*, CT-18:89–95, 1971.

[42] A.M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237, 1972.

[43] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. User's guide for NPSOL 5.0: A fortran package for nonlinear programming. Technical report, Stanford University, July 1998.

[44] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentation.* SIAM, Philadelphia, 2000.

[45] G.W. Harrison. Dynamic models with uncertain parameters. In X.J.R. Avula, editor, *Proceedings of the First International Conference on Mathematical Modeling*, volume 1, pages 295–304, 1977.

[46] Magnus R. Hestenes. *Calculus of Variations and Optimal Control Theory.* John Wiley & Sons, Inc., New York, 1983.

[47] Alan C. Hindmarsh and Radu Serban. User documentation for CVODES, an ODE solver with sensitivity analysis capabilities. Technical report, Lawrence Livermore National Laboratory, July 2002.

[48] Reiner Horst and Hoang Tuy. *Global Optimization.* Springer-Verlag, Berlin, 1993.

[49] Reiner Horst and Hoang Tuy. *Global Optimization: Deterministic Approaches.* Springer, New York, third edition, 1996.

[50] David H. Jacobson and Milind M. Lele. A transformation technique for optimal control problems with a state variable inequality constraint. *IEEE Transactions on Automatic Control*, AC-14(5):457–464, 1969.

[51] D.H. Jacobson, M.M. Lele, and J.L. Speyer. New necessary conditions of optimality for control problems with state-variable inequality constraints. *AIAA Journal*, 6(8):1488–1491, 1968.

[52] Padmanaban Kesavan, Russell J. Allgor, Edward P. Gatzke, and Paul I. Barton. Outer approximation algorithms for separable nonconvex mixed-integer nonlinear programs. *Mathematical Programming*, December 2003. In press.

[53] Padmanaban Kesavan and Paul I. Barton. Decomposition algorithms for nonconvex mixed-integer nonlinear programs. *AIChE Symposium Series*, 96(323):458–461, 2000.

[54] Padmanaban Kesavan and Paul I. Barton. Generalized branch-and-cut framework for mixed-integer nonlinear programs. *Computers and Chemical Engineering*, 24(2/7):1361–1366, 2000.

[55] G.R. Kocis and Ignacio E. Grossmann. A modelling and decomposition strategy for the MINLP optimization of process flowsheets. *Computers and Chemical Engineering*, 13(7):797–819, 1989.

[56] Cha Kun Lee, Adam B. Singer, and Paul I. Barton. Global optimization of linear hybrid systems with explicit transitions. *Systems & Control Letters*, 51:363–375, 2004.

[57] R. Luus, J. Dittrich, and F.J. Keil. Multiplicity of solutions in the optimization of a bifunctional catalyst blend in a tubular reactor. *Canadian Journal of Chemical Engineering*, 70:780–785, 1992.

[58] Rein Luus. Optimal control by dynamic programming using systematic reduction in grid size. *International Journal of Control*, 5:995–1013, 1990.

[59] Rein Luus. *Iterative Dynamic Programming.* Chapman & Hall/CRC, Boca Raton, 2000.

[60] T. Maly and L.R. Petzold. Numerical methods and software for sensitivity analysis of differential-algebraic systems. *Applied Numerical Mathematics*, 20:57–79, 1996.

[61] C.D. Maranas and C.A. Floudas. Global minimum potential energy conformations of small molecules. *Journal of Global Optimization*, 4:135–170, 1994.

[62] R.B. Martin. Optimal control drug scheduling of cancer chemotherapy. *Automatica*, 28(6):1113–1123, 1992.

[63] Tony Mason, John Levine, and Doug Brown. *Lex & Yacc.* O'Reilly & Associates, Sebastopol, 2nd edition, 1992.

[64] Arthur Mattuck. *Introduction To Analysis.* Prentice Hall, New Jersey, 1999.

[65] Garth P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I – convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[66] M. Jezri Mohideen, John D. Perkins, and Efstratios N. Pistikopoulos. Optimal design of dynamic systems under uncertainty. *AIChE Journal*, 42(8):2251–2272, 1996.

[67] M.J. Mohideen, J.D. Perkins, and E.N. Pistikopoulos. Towards an efficient numerical procedure for mixed integer optimal control. *Computers and Chemical Engineering*, 21:S457–S462, 1997.

[68] Ramon E. Moore. *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, 1979.

[69] C.P. Neuman and A. Sen. A suboptimal control algorithm for constrained problems using cubic splines. *Automatica*, 9:601–613, 1973.

[70] C. C. Pantelides. SpeedUp – Recent advances in process simulation. *Computers and Chemical Engineering*, 12(7):745–755, 1988.

[71] Ioannis Papamichail and Claire S. Adjiman. A rigorous global optimization algorithm for problems with ordinary differential equations. *Journal of Global Optimization*, 24:1–33, 2002.

[72] Taeshin Park and Paul I. Barton. State event location in differential algebraic models. *ACM Transactions on Modelling and Computer Simulation*, 6(2):137–165, 1996.

[73] Linda R. Petzold and Wenjie Zhu. Model reduction for chemical kinetics: An optimization approach. *AIChE Journal*, 45(4):869–886, 1999.

[74] L.L. Raja, R.J. Kee, R. Serban, and Linda R. Petzold. Computational algorithm for dynamic optimization of chemical vapor deposition processes in stagnation flow reactors. *Journal of the Electrochemical Society*, 147(7):2718–2726, 2000.

[75] D.W.T. Rippin. Simulation of single and multiproduct batch chemical plants for optimal design and operation. *Computers and Chemical Engineering*, 7:137–156, 1983.

[76] R. Tyrrell Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, 1970.

[77] Walter Rudin. *Principles of Mathematical Analysis.* McGraw-Hill, Inc., New York, third edition, 1976.

[78] H.S. Ryoo and N.V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with application in process design. *Computers and Chemical Engineering*, 19(5):551–566, 1995.

[79] R.S. Ryoo and N.V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 2:107–139, 1996.

[80] Carl A. Schweiger and Christodoulos A. Floudas. Interaction of design and control: Optimization with dynamic models. *Optimal control: Theory, Algorithms, and Applications*, pages 1–48, 1997.

[81] M. Sharif, N. Shah, and C.C. Pantelides. On the design of multicomponent batch distillation columns. *Computers and Chemical Engineering*, 22:S69–S76, 1998.

[82] Jason L. Speyer and Arthur E. Bryson Jr. Optimal programming problems with a bounded state space. *AIAA Journal*, 6(8):1488–1491, 1968.

[83] Mohit Tawarmalani and Nikolaos V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming : Theory, Algorithms, Software, and Applications.* Kluwer Academic Publishers, Dordrecht, 2002.

[84] James W. Taylor, Gerhard Ehlker, Hans-Heinrich Carstensen, Leah Ruslen, Robert W. Field, and William H. Green. Direct measurement of the fast, reversible reaction of cyclohexadienyl radicals with oxygen in nonpolar solvents. Journal of Physical Chemistry A. In Press, 2004.

[85] James W. Taylor, Gerhard Ehlker, Hans-Heinrich Carstensen, Leah Ruslen, Robert W. Field, and William H. Green. Direct measurement of the fast, reversible reaction of cyclohexadienyl radicals with oxygen in nonpolar solvents. Submitted, 2004.

[86] K. Teo, G. Goh, and K. Wong. *A Unified Computational Approach to Optimal Control Problems.* Pitman Monographs and Surveys in Pure and Applied Mathematics. John Wiley & Sons, Inc., New York, 1991.

[87] I.B. Tjoa and L.T. Biegler. Simultaneous solution and optimization strategies for parameter-estimation of differential-algebraic equation systems. *Industrial and Engineering Chemistry Research*, 30(2):376–385, 1991.

[88] John Tolsma and Paul I. Barton. DAEPACK: An open modeling environment for legacy models. *Industrial & Engineering Chemistry Research*, 39(6):1826–1839, 2000.

[89] John E. Tolsma and Paul I. Barton. DAEPACK: A symbolic and numeric library for open modeling. http://yoric.mit.edu/daepack/daepack.html.

[90] John L. Troutman. *Variational Calculus and Optimal Control: Optimization with Elementary Convexity.* Springer-Verlag, New York, second edition, 1996.

[91] T.H. Tsang, D.M. Himmelblau, and T.F. Edgar. Optimal control via collocation and nonlinear programming. *International Journal of Control*, 21:763–768, 1975.

[92] V. I. Utkin. *Sliding Modes in Control and Optimization.* Springer-Verlag, Berlin, 1992.

[93] Frederick Albert Valentine. *Contributions to the Calculus of Variations.* PhD thesis, University of Chicago, 1937.

[94] Wolfgang Walter. *Differential and Integral Inequalities.* Springer-Verlag, Berlin, 1970.

[95] Lotfi A. Zadeh and Charles A. Desoer. *Linear System Theory: The State Space Approach.* McGraw-Hill, New York, 1963.