Evaluation of Process Systems Operating Envelopes

by

Matthew David Stuber

B.Ch.E., University of Minnesota - Twin Cities (2007)

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

February 2013

Author .....
Department of Chemical Engineering
November 14, 2012

Certified by .....
Paul I. Barton
Lammot du Pont Professor of Chemical Engineering
Thesis Supervisor

Accepted by .....
Patrick S. Doyle
Professor of Chemical Engineering
Chairman of the Committee for Graduate Students

Evaluation of Process Systems Operating Envelopes

by

Matthew David Stuber

Submitted to the Department of Chemical Engineering
on November 14, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Chemical Engineering

Abstract

This thesis addresses the problem of worst-case steady-state design of process systems under uncertainty, also known as robust design. Designing for the worst case is of great importance when considering systems for deployment in extreme and hostile environments, where operational failures cannot be risked due to extraordinarily high economic and/or environmental expense. For this unique scenario, the cost of "over-designing" the process far outweighs the cost associated with operational failure. Hence, it must be guaranteed that the process is sufficiently robust in order to avoid operational failures.

Many engineering, economic, and operations research applications are concerned with worst-case scenarios. Classically, these problems give rise to a type of leader-follower game, or Stackelberg game, commonly known as the "minimax" problem, or more precisely as a max-min or min-max optimization problem. However, since the application here is to steady-state design, the problem formulation results in a more general nonconvex equality-constrained min-max program, for which no previously available algorithm can solve effectively. Under certain assumptions, the equality constraints, which correspond to the steady-state model, can be eliminated from the problem by solving them for the state variables as implicit functions of the control variables and uncertainty parameters. This approach eliminates explicit functional dependence on the state variables, and in turn reduces the dimensionality of the original problem. However, this embeds implicit functions in the program, which have no explicit algebraic form and can only be approximated using numerical methods. By doing this, the max-min program can be reformulated as a more computationally tractable semi-infinite program, with the caveat that there are embedded implicit functions.

Semi-infinite programming with embedded implicit functions is a new approach to modeling worst-case design problems. Furthermore, modeling process systems—especially those associated with chemical engineering—often results in highly non-convex functions. The primary contribution of this thesis is a mathematical tool for solving implicit semi-infinite programs and assessing robust feasibility of process systems using a rigorous model-based approach. This tool has the ability to determine,

with mathematical certainty, whether or not a physical process system based on the proposed design will fail in the worst case by taking into account uncertainty in the model parameters and uncertainty in the environment.

Thesis Supervisor: Paul I. Barton
Title: Lammot du Pont Professor of Chemical Engineering

To My Family

Acknowledgments

I only get one opportunity to write the acknowledgments section of my doctoral thesis. I've always been a very social person who deeply values his relationships and so it is my intention to thank everyone who has made a contribution to getting me where I am today. The reader should take this as a warning; this section is long and maybe a bit rambling as I reflect on the past. There are so many important people who I owe thanks for their contributions in one way or another. Please don't take offense if I don't mention you specifically. It in no way reflects a lack of gratitude but simply that at this point, my brain is fried.

First, I would like to thank my thesis advisor, Professor Paul Barton. Not because its obligatory, but because of him, I rather enjoyed my experience as a graduate student at MIT. The amount of personal attention he was able to devote, as well as the level of patience he was able to exhibit with regards to my education and research is astounding. His attention to detail, albeit annoying at times, gave me confidence that the final product would be of the highest standard of quality. His strong emphasis on exposure within the scientific community enabled me to be able to travel often and have some of my most cherished experiences while being a graduate student, such as white knuckled driving through the Schwarzwald (Black Forest) in the early morning mist and drag limiting a rental car on the Autobahn to München to gorge on Käsekrautspätzle and Laugenbrezel larger than my head at Oktoberfest...and of course present my research to the community. I am forever grateful of his understanding nature and sensitivity with regards to personal issues and life outside of the lab. In particular, there was a six-month period where he was incredibly flexible (with deadlines etc.) so that I could act as the primary caregiver for my wife while she was undergoing chemotherapy. Because of his flexibility and sensitivity, he helped reduce a lot of stress which enabled me to stay rather productive and develop what is likely the largest contribution of this thesis. Overall, he fosters a great environment for success and I am proud to have had the opportunity to work for him, be mentored by him, and call him my friend.

I would also like to thank my thesis committee members Professor William Green and Professor Alexander Mitsos. Their advice and wisdom did not fall on deaf ears and helped structure this project into something I am proud to have worked on. I owe a special thanks to Professor Mitsos for providing me with the algorithm framework for Chapter 7. I also owe a special thanks to Dr. Benoit Chachuat for being so responsive to emails regarding bugfixes and implementing new features into his `MC++` code, which I relied upon heavily. I am also indebted to my PSE lab brethren for without them, this thesis would not be possible. In particular, I owe Dr. Joe Scott many thanks for the discussions regarding everything from our current research problems to general maths and sciences all the way to deeper discussions of society, politics, and religion. Furthermore, I am grateful for the countless hours spent helping me clarify and refine my ideas into presentable and high quality contributions, especially regarding the material in Chapter 4. I would like to thank Achim Wechsung for always being open and willing to help solve my programming problems and eliminate

bugs. Furthermore, I'd like to acknowledge his contributions to Chapter 8 where his awesome coding on the constraint propagation tool played a pivotal role in solving the subsea separator problem. I owe Dr. Arul Sundaramoorthy and Achim Wechsung many thanks for contributing the constraint propagation material to Chapter 8. I owe Spencer Schaber many thanks for his help with the kinetic mechanism example in Chapter 4. I'd like to thank the other PSE labmates I haven't mentioned specifically for making the lab a place that I enjoyed coming to day after day and an enjoyable and stimulating place to work.

I am forever grateful to my wife Whitney Bogosian. Besides listening to countless presentation rehearsals and incessant complaining about my work and being a grad student, I am grateful for her unconditional love and constant support while accompanying me on this endeavor that accounts for almost 20% of my life. Before our paths converged that day in E51, I was just an unhappy "first year" struggling with the anxiety of failure brought on by the weight and stress of being a new PhD student at MIT. She not only made life in Boston bearable, she made it comfortable and bright; an environment I could succeed in. This thesis has been the primary reason for our almost entirely uneventful summer and the excuse for not taking her on the vacation she well deserves.

I'd like to thank all of my friends for supporting me on this adventure in one way or another. My Twin Cities friends deserve to know how grateful I am for their friendship and support and how much I value how close we've remained in spite of my absence. Specifically, I'd like to mention my best man Nathan Bond, Bill Foley, Scott Elmgren, Lindsey Jader, Brian and Tammy Blechinger, Tim and Mandy Carroll, Dan and Steffanie Corning, and Paul and Emily Morrison, for various rides to and from the airport, visiting me in Boston, hosting/attending parties while I was in town, or simply calling me to catch up because I've been MIA. Paul Morrison was especially awesome at calling me and keeping in touch. I'd like to mention my closest Boston friends (besides my wife) Dr. Christopher Pritchard, Eric Holihan, Adam Serafin, and John Martin. Having these guys around to share hobbies with was invaluable to my happiness and sanity in grad school. Our 3 hour lunches of intense intellectual conversation, skiing/snowboarding, shredding tires racing cars, and consuming large portions of lentils at Haveli and Punjabi Dhaba were invaluable to my success. I was lucky to have my dear friend Dr. Torren Carlson (Torrey) working on his PhD at Amherst while I was here at MIT. Although two hours apart, we still managed to carry on with our favorite past times such as going to hardcore shows, including driving to NJ for a reunion show of The Movielife. Watching him defend his thesis gave me the much needed motivation to stay productive in my final two years.

I am lucky to have such an amazing family who, without their unconditional support, I could not have succeeded this far. First, I owe so much to my grandma Victoria (Vicki) Stuber, for being such a positive role model and for playing the foundational role in the family. She embodies "Minnesota Nice" as a loving, compassionate, and tolerant woman who is always going out of her way to help others. She is one of my greatest sources of inspiration. Every day I try to follow her example and better myself by living a positive lifestyle, seeking out knowledge, and maintaining an open mind; which has surely helped me succeed in the diverse environment of academia. I

also want to acknowledge my grandpa Marvin Stuber who is never shy to convey how proud he is of "his grandson at MIT." It is an honor to make him proud. I owe many thanks to my parents Nikki Black and Dave and Renee Stuber. Besides the various forms of financial support and emotional support they gave me throughout the years, perhaps what I am most thankful for is the freedom they granted me to discover who I am and forge my own path. Because of this, I was able to find a discipline to pursue that excites me and makes me truly happy. Although it meant moving away and possibly only seeing me once per year, they supported and encouraged me to pursue a graduate education. This helped me so much to confront my inhibitions about moving away to another city and out of my comfort zone. It's amazing to have parents whose metric for success is simply my own happiness. I'd like to thank my sister Trisha Griebenow who has helped me on this journey in so many ways. She was always available and willing to listen to my seemingly endless rants, offer words of advice when needed, reflect on the pursuit of knowledge and deeper philosophical questions, and always give me perspective. I am deeply grateful to have her support and friendship and I will be forever indebted for her role in getting me to this point and beyond. I owe many thanks to my in-laws Wayne and Sandy Bogosian for their love and support, welcoming me warmly into their family, and always offering their homes as getaway destinations away from the stresses of the city and work. I am indebted to them for including me in the tropical destination vacations and the countless Celtics and Red Sox games, and allowing me to disassemble cars in their driveway and garage; all of which were incredibly therapeutic activities that were much needed on this journey.

I think it's important at this point to thank all of the teachers and instructors in the public schools that I attended in Fridley and Chaska who survive on meager salaries in order to educate the less-than-appreciative youth. I am truly sorry that they had to put up with my young self. They cared more about my education and future than I did at the time. Because of their dedication, somewhere along the way, they awakened my deep passion for math and the sciences. A passion that, in spite of the enormous workloads and sacrifices involved in the pursuit of knowledge, is still present today as I am finishing this doctoral thesis. I hope someday I can make a contribution in motivating younger generations to pursue math and science.

Lastly, I'd like to acknowledge Chevron Corporation for funding this research through a partnership with the MIT Energy Initiative.

Thinking must never submit itself, neither to a dogma, nor to a party, nor to a passion, nor to an interest, nor to a preconceived idea, nor to anything whatsoever, except to the facts themselves, because, for it to submit to anything else would be the end of its existence.

-Henri Poincaré

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation: Designing for the Worst Case

Engineering design has long been practiced as a trade-off between economics and reliability/safety. Probabilities of destructive environmental events as well as probabilities of operational failures are typically studied along with their impact on production and safety, in order to assess risk associated with a proposed design.[1] It is in the best interest of the design engineer to minimize risk while also minimizing the cost of the design, in order to make it more economically feasible. For instance, it would be economically unwise to invest extra money to increase the seismic performance of a skyscraper that is to be constructed in a region having a 0% chance of a destructive earthquake. Such a design would be overly conservative. Alternatively, if that same skyscraper were to be constructed in a region with a significantly higher probability of a destructive earthquake occurring, the cost associated with increasing the structure's seismic performance will be outweighed by the cost of structural damage or even collapse in the event of seismic activity.

Chemical process systems, especially those related to energy products such as liquid fuels, are often considered to be inherently risky. This is because although reliability and safety records may be impeccable (in many cases they are not), the

[1] Here, the standard definition of risk is being used: risk $\equiv$ (probability of failure)$\times$(impact of failure).

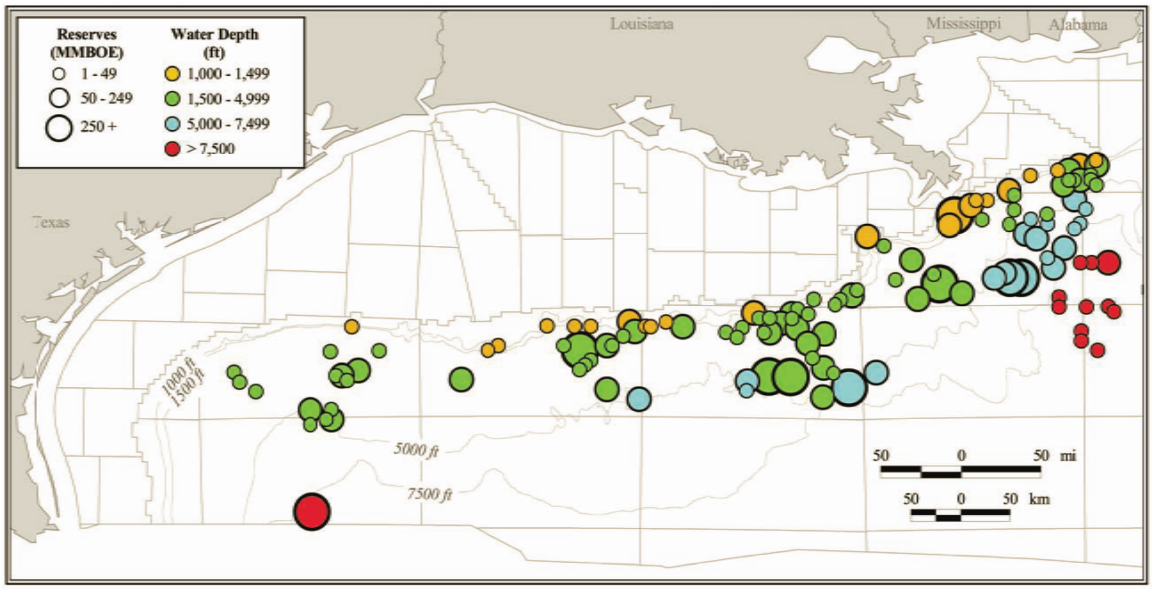

Figure 1-1: The active oil field on the outer continental shelf of the Gulf of Mexico as of 2009. (Photo credit: [103])

impact of an operational failure on production and safety can be extraordinarily high. Deployment of chemical processes in extreme and hostile environments increases risk even more by (potentially) drastically increasing the impact of operational failures, even though the probability of such a failure may not increase. One such chemical process system of interest is relevant to deepwater oil and gas production.

The depletion of petroleum reserves from traditional on-shore and shallow-water off-shore fields, coupled with political pressure for reduced dependence on petroleum from foreign sources, has motivated exploration into increasingly more extreme environments. One promising frontier is in ultra deepwater,[2] where in 2004, a vast deposit of petroleum, known as the "lower tertiary trend", containing 3-15 billion barrels (120-600 billion gal.) of petroleum, was discovered by Chevron geologists [81]. Figure 1-1 is a map of the "outer continental shelf", containing the lower tertiary trend, depicting proven wells, and their estimated volume of oil, as of 2009 [103]. However, in April 2010, BP sufficiently demonstrated that pursuing oil reserves in ultra deepwater environments[3] comes with inherently high risk exacerbated by a lack

[2]Defined here as depths $\geq$ 7500ft.

[3]The BP disaster actually occurred in only 5000ft of water, demonstrating that failure in ultra deepwater will be at least as catastrophic, if not significantly more.

Figure 1-2: In contrast to floating platforms, subsea production facilities, shown here, perform all upstream processing on the seabed locally near the wellhead. (Photo credit: FMC Technologies)

of sufficient technology. In the BP incident, commonly referred to as the Deepwater Horizon oil spill, a catastrophic failure of the leased ultra deepwater drilling platform named *Deepwater Horizon* resulted in 11 human lives lost,[4] an estimated $30 billion in expenses, 5 million barrels of oil spilled, and untold ecological fallout which is still being investigated, including the economic impact on commercial fishing and the fish value chain [16, 21, 57, 135]. In this environment, the costs associated with operational failures far outweigh the costs associated with "over-designing" the process, and so extreme effort must be made to avoid failures altogether.

Industry engineers have suggested that the application of traditional floating platforms to ultra deepwater production is too risky. They propose that novel remote compact subsea production facilities, as depicted in Figure 1-2, are the key enabling technology for ultra deepwater oil and gas production. Thus posing the question:

How can one design a novel process system that is guaranteed to be ro-

[4]The 11 lives mentioned were lost during the accident on the Deepwater Horizon alone. This doesn't include lives lost as an indirect result of the spill such as health effects of long-term exposure, etc.

bust to operational failures given the high level of uncertainty in extreme and hostile environments which cannot be accurately reproduced in the laboratory?

Since field conditions cannot be successfully recreated in the laboratory, and similarly since pilot plant systems can only be tested under a finite number of conditions, experimental approaches to addressing this question are inadequate. In order to successfully address this question, a mathematically rigorous model-based approach—taking into account uncertainty in the environment as well as uncertainty that is inherent to the model[5]—must be taken and the system must be designed for the worst-case realization of uncertainty. Thus, however improbable, the novel process system design will be robust to operational failures in the face of the worst-case scenario(s). Since a deterministic approach must be taken, an implicit result is that all uncertain events are considered to be independent of one another. Therefore, the worst case may be the realization of cascading events.

One common example of the worst case being the realization of cascading events is nuclear reactor meltdowns, such as the most recent incident at the Fukushima Daiichi reactor in 2011. In the case of Fukushima, a magnitude 9 earthquake triggered a plant-wide reactor shutdown, forcing the cooling system to be powered by emergency generators [45]. A subsequent 14-meter tsunami wiped out the emergency generators, which were designed to withstand a 5.7-meter tsunami [45]. By design, the cooling system had redundant emergency power supplies in the event of a failure of the emergency generators. Eventual failure of the redundant (battery) supply proved to be catastrophic, overheating the reactor and causing a meltdown. A robust design of the system would have included a reactor that could never produce a runaway scenario, even under the condition of zero coolant flow, and/or a backup power system that could withstand *all* sizable tsunamis. These are the type of reactors being designed most recently and are commonly referred to as "inherently safe." Of course, this discussion is only to illustrate when and why worst-case design strategies are necessary. In the case of a tsunami, the larger it is, the greater impact on the process

[5]Uncertainty in the model must be known or estimated here.

it will have, and so the worst case is simply the largest tsunami imaginable. In this case, the worst-case strategy would be to design the process that would be unaffected by any tsunami.

In summary, worst-case design strategies should be applied when:

1. there are extraordinarily high costs associated with operational failures,[6]
2. there is a high level of uncertainty associated with the environment and/or the technology, or
3. there are requirements for technology qualification such as rigorous performance and safety verification.

The problem of *design under uncertainty* has stimulated a large effort in research, especially within the chemical engineering community. In the next section, the previous research and the existing approaches are discussed.

1.2 Existing Approaches to Robust Simulation and Design

Since the widespread adoption of modern computers, engineers have been designing and simulating more advanced and complex process systems. With continuous advancements in mathematics and development and improvement of numerical methods, engineers have been able to address a wide variety of concerns from performance to controllability of novel systems. Going back to the design trade-off mentioned previously, design engineers have been able to address how to design a complex process that maximizes performance and safety while minimizing cost, by taking more rigorous mathematical approaches, as opposed to the heuristic approaches of the early design engineers. It is because of this that mathematical programming, or optimization, has become the mathematical workhorse for engineering design.

[6]These can be in the form of environmental damage, economic losses, loss of life, loss of confidence in an entire technology, etc.

The existence of uncertainty in the environment as well as uncertainty introduced by inaccurate models of real-world systems, has motivated engineers to address design in the presence of uncertainty, using a variety of optimization approaches.

1.2.1 Stochastic Programming

In [122], an extensive overview and summary of stochastic programming approaches to optimization under uncertainty is given. Stochastic programming is commonly implemented as a two-stage (or multistage) decision problem where the first-stage decision is made before the realization of the uncertainty parameters and the second stage problem, or recourse problem, is solved after the random events have been presented [17, 67]. In stochastic programming, uncertainty as a family of events is modeled through probability measures in either a discrete or continuous manner [17, 67, 122]. Therefore, for any particular event from a family of events, the probability of that event occurring is known and thus the probability of an uncertain parameter taking any particular value is known [17, 67].

One example where stochastic programming was applied to optimal design of chemical processes under uncertainty was given in [84]. In this case, the authors motivation was to eliminate "excessive overdesign" [84]. The primary objective is then to produce a design that is flexible enough to "allow adjustment for the most *important* uncertainties" [84]. To do this, they formulated a two-stage stochastic program with the first stage corresponding to the design stage, where decisions are made regarding the actual equipment sizing, and the second stage as the operating stage, where operating conditions are subsequently adjusted for optimal performance following the uncertainty realization [84]. The uncertain variables were considered as random with associated probability distributions, and corresponded to things such as the yield prediction or kinetic rate constants [84]. The authors then considered six different design formulations for comparison.

The idea of robust (stochastic) optimization was introduced in [95] to address real-world operations research problems in which "noisy, erroneous, or incomplete data" are inherent. The robust optimization approach produces a series of solutions that are

progressively less sensitive, and therefore more robust, to realizations of uncertainty [95]. The authors of [95] define two types of robustness: solution robust and model robust. Solution robust means that the optimal solution of the optimization problem is robust with respect to optimality in the sense that it will remain *close* to optimal for *any* realization of uncertainty [95]. Model robust means that the optimal solution of the optimization problem is robust with respect to feasibility in the sense that it will remain *almost* feasible for *any* realization of uncertainty [95]. The authors make an explicit claim that their robust optimization formalism is superior to deterministic worst-case strategies because they yield "very conservative and potentially expensive solutions" [95]. However, when designing processes under uncertainty classified as begin extraordinarily risky, robustness with respect to the worst-case must be verified rigorously, with absolute certainty. For such systems, the conservativeness of deterministic worst-case strategies is precisely what is desired.

In order for stochastic approaches to give accurate results, probability distributions must be known with high accuracy for each uncertain variable. Furthermore, stochastic methods fail to capture improbable events sufficiently—even though such events may have an extraordinarily large impact on the performance and safety of a design—and thus for a highly improbable worst-case scenario, robustness with respect to uncertainty cannot be guaranteed with certainty. By characterizing uncertainty by known or estimated intervals, all possible realizations are treated equally, eliminating the need for probability distributions and stochastic programming altogether. Because of this, stochastic programming approaches to design under uncertainty are not applicable to robust simulation and design problems, and will not be considered further.

1.2.2 Dynamic Programming

Dynamic programming is a term introduced by Bellman [8] to describe a mathematical program that focuses on multi-stage decision making [8, 122], which has most commonly been applied using a stochastic approach to characterize uncertainty [27, 64, 71, 122, 131, 142]. Dynamic programming is essentially the same idea as

stochastic programming described previously, except instead of a two-stage formulation, dynamic programming resolves multi-stage decision problems. For some current state of the system, there is an associated probability distribution of uncertain parameters for which an uncertain value is chosen and an allowable control chosen from subsequent knowledge of the state of the system. This hierarchy of decision-making is always carried out so as to minimize the expected cost over all states [8].

Dynamic programming is largely applicable to finite-time systems, such as designing batch processes or planning problems with varying operational and/or market conditions [30]. For instance, in [9], a discrete-time ballistic trajectory control application is discussed. The main idea is to ensure that the optimal design or optimal operating policy is implemented throughout the project lifetime as environment evolves [30]. Steady-state process design can be handled using a dynamic programming approach, for instance, if varying realizations of uncertainty present themselves during operation. In this case, the design objective is for the process to be robust with respect to uncertainty.

Similar to stochastic programming, since uncertainty is handled stochastically, this approach is inadequate for worst-case design strategies. Recalling the discussion in Section 1.1, the worst-case realization of uncertainty may be that associated with cascading events. Without the ability to enumerate an infinite number of uncertainty realizations, capturing the worst-case behavior is impossible. In addition, due to their complexity, solving dynamic programming formulations can be very computationally intensive and are often intractable for relatively simple nonlinear problems [122]. Because of this, dynamic programming cannot be applied to robust simulation and design problems with the intent of providing a rigorous certificate of feasibility.

1.2.3 Fuzzy Programming

Fuzzy programming is similar to stochastic (and dynamic) programming except that uncertainty is modeled using fuzzy numbers instead of probability measures and performance and safety constraints are treated as fuzzy sets [10]. A fuzzy set has no sharp boundaries and has an associated membership function that defines the "de-

gree of membership" of, in this case, a number [10]. Since constraints are handled in this way, there are no hard guarantees of satisfaction and feasibility; only a "degree of satisfaction" [10]. Fuzzy numbers are a special case of fuzzy set whose membership function equals 1 at precisely one value. Fuzzy intervals are then a fuzzy set containing an interval of numbers whose membership function equals 1. Again, there is a degree of membership associated with these sets and so modeling uncertainty in engineering systems using fuzzy numbers or fuzzy intervals may be inadequate since values outside of certain intervals may be nonphysical, and therefore may lead to nonphysical solutions. In light of this, fuzzy programming is incapable of giving hard guarantees of robustness and will not be considered as a viable approach to design under uncertainty in the context of this thesis.

1.2.4 Deterministic Approaches

One of the earliest deterministic attempts at the robust design problem focused on optimal process design under uncerainty [102]. In optimal design under uncertainty, the system must be designed such that it performs optimally while satisfying all performance and safety specifications for every possible realization of uncertainty. In [102], the authors focus was to design a process that performed optimally while being robust to the worst-case realization of uncertainty. The authors formulate the equality-constrained min-max program with the model equations exhibiting an explicit relationship between the process inputs and outputs [102]. They required that their uncertainty set is a bounded finite set and the functions involved are twice continuously differentiable. The authors solve this problem locally using an iterative gradient-based technique.

In [78], Kwak and Haug addressed optimal design with a more general optimization formulation. The application that is given in [78] is a military missile system that must operate optimally for every temperature within a given interval. The authors [78] formulated this problem as a *bilevel* program that minimizes some relevant design objective varying design parameters subject to an inner program that attempts to find the worst-case realization of environmental parameters subject to some steady-

state model of the design parameters, environmental parameters, and state variables. This formulation is intractable and extremely difficult to solve in the general case, however. In [78], the authors solve an approximate problem by considering only first-order (affine) approximations of the objective function and constraints. Another important contribution is the discussion of the relationship between the more general bilevel formulation and the worst-case "minimax" formulation [78].

Grossmann and Sargent [51] present a bilevel formulation that is a slight modification of [78] and the special case in [102]. Their approach treats the design variables as having the ability to be partitioned into a fixed design variable and a control variable that can be chosen during operation according to realizations of uncertainty [51]. Their focus is to determine the optimal design variables, minimizing an economic objective, so that there exists a control setting such that for any realization of uncertainty, the design meets all performance and safety specifications [51]. They conclude by discussing special cases for which solutions to their formulation can be obtained using previously developed optimization theory [51].

Halemane and Grossmann [52] extended the ideas of [51] and offered a more general optimization formulation for optimal design under uncertainty and what is called the feasibility problem, or the problem of verifying feasible operation in the face of the worst-case realization of uncertainty. In light of uncertainty in model parameters as well as disturbances to the process, the authors state that "it is clearly very important to consider at the design stage the effect that uncertain parameters can have on both the optimality and feasibility of operation of the plant" [52]. The authors solve the model equations as implicit functions of the controls, design variables, and uncertainty parameters and then formulate the problem as a reduced-dimension semi-infinite program (SIP). They analyze the problem and offer two solution methods that rely on convexity, amongst other assumptions [52].

The SIP problem formulation presented in [52] is then used in [136] to assess the performance of a process system in terms of *flexibility*, which is stated as "the ability to operate over a range of conditions while satisfying performance specifications". Swaney and Grossmann then describe a quantitative index of flexibility [136]. The

flexibility problem is then solved in [137] under certain quasiconvexity assumptions on the feasible set. However, barring satisfaction of the quasiconvexity assumption or the optimal solution lying on a vertex of the hyperrectangle of feasible parameter values, global optimality cannot be guaranteed [136].

Deterministic robust optimization is surveyed in [11]. As mentioned in Section 1.2.1, solutions to these problems are conservative in that they are guaranteed to be feasible/optimal for all realizations of uncertainty. The authors formulate the problem as an SIP equivalent to the worst-case min-max problem [11]. Robust optimization of linear programs (LP), semi-definite programs (SDP), and conic quadratic problems—all of which are convex with explicit constraints—were considered [11]. Robust optimization of convex least-squares problems with explicit constraints was considered in [37]. Since the considered problems were all explicitly constrained convex programs, their methodology is inadequate for solving robust simulation and design problems.

In [43], a method is presented that provides a rigorous approach to the design under uncertainty and flexibility problems, without relying on convexity assumptions. The authors rely on the assumption that the performance constraints and model equations are twice-differentiable [43]. Contrasting [52, 136, 137], Floudas *et al.* do not eliminate the model equations from the formulation and therefore must solve the full-space constrained "max-min problem" [43]. However, even for relatively simple examples, their bilevel formulation can be computationally intractable.

In Chapter 2, the robust design problem will be formulated mathematically as the (implicit) SIP presented in [52]. The rest of this thesis will focus on solving this SIP in the general case, using a rigorous approach, without relying on convexity assumptions. The method for solving this problem relies on: (1) the ability to bound implicit functions, discussed in Chapter 3, and (2) the ability to solve nonconvex nonlinear programs, having embedded implicit functions, to global optimality, discussed in Chapter 4. In Chapter 7, these developments are put together within an SIP algorithm to solve the robust design problem. Finally, the application to ultra deepwater subsea production facilities is covered in Chapter 8 with a model and case

study.

Chapter 2

Robust Simulation and Design Using Semi-Infinite Programming with Implicit Functions

2.1 Problem Formulation

Process systems operating at steady state can be modeled in general as the following system of (nonlinear) algebraic equations:

$$\mathbf{h}(\mathbf{z}, \mathbf{u}, \mathbf{d}, \boldsymbol{\pi}) = \mathbf{0},\ \mathbf{h} : D_x \times D_u \times D_d \times D_\pi \to \mathbb{R}^{n_x} \tag{2.1}$$

with $D_x \subset \mathbb{R}^{n_x}, D_u \subset \mathbb{R}^{n_u}, D_d \subset \mathbb{R}^{n_d}, D_\pi \subset \mathbb{R}^{n_\pi}$ as open sets. The variables $\mathbf{z} \in X \subset D_x$ represent the process state variables, $\mathbf{u} \in U \subset D_u$ are the control variables, $\mathbf{d} \in D \subset D_d$ represent the disturbance uncertainty, and $\boldsymbol{\pi} \in \Pi \subset D_\pi$ as the model uncertainty, as depicted in Figure 2-1.

Model uncertainty will be characterized in the usual manner as the discrepancy between the model and the physical system. Two types of model uncertainty can be characterized: parametric uncertainty and structural uncertainty [29, 33]. Parametric uncertainty is the uncertainty in the model parameters that arise from statistical errors in measurements and the propagation through calculations such as regression

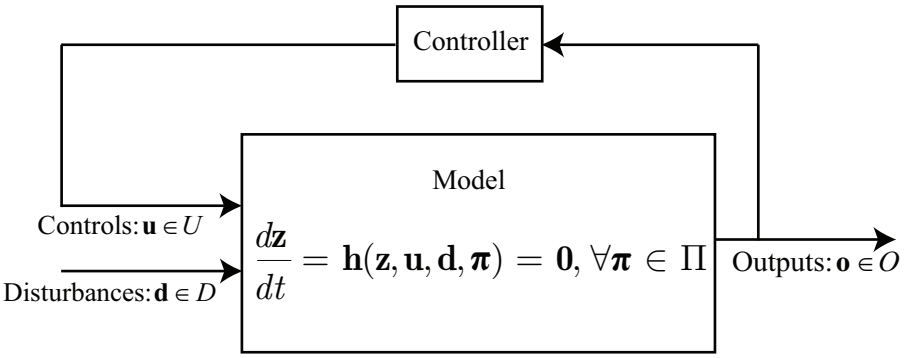

Figure 2-1: Steady-state process systems model representation.

or parameter estimation. Structural uncertainty arises from the inability of the model equations to represent the physical system accurately. In other words, the model cannot sufficiently capture the physics of the system. Structural uncertainty cannot be handled in this framework except to the extent to which it can be characterized as parametric uncertainty. Throughout this thesis, model uncertainty will refer to parametric model uncertainty.

For simplicity of notation, disturbance and model uncertainty will be represented as the uncertain parameters:

$$\mathbf{p} \equiv (\mathbf{d}, \boldsymbol{\pi}), \quad \mathbf{p} \in P \subset D_p \equiv D_d \times D_\pi.$$

The uncertain parameters can take any realization from the uncertainty set, which, using physical knowledge of the system, can be represented as a connected compact set[1]:

$$P \equiv \{\mathbf{p} \in \mathbb{R}^{n_p} : \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\},$$

with $\mathbf{p}^L, \mathbf{p}^U \in \mathbb{R}^{n_p}$ known *a priori*. Furthermore, since control actions are bounded,[2] the control set will also be represented as a connected compact set:

$$U \equiv \{\mathbf{u} \in \mathbb{R}^{n_u} : \mathbf{u}^L \leq \mathbf{u} \leq \mathbf{u}^U\},$$

[1] In practice, each uncertain parameter will not take values from arbitrarily large intervals.

[2] Controls can only take values within the interval corresponding to, for example, fully-opened and fully-closed control valves.

with $\mathbf{u}^L, \mathbf{u}^U \in \mathbb{R}^{n_u}$. The design question that must be addressed is [134]:

> Given a process model, and taking into account uncertainty in the model and disturbances to the inputs of the system, do there exist control settings such that, at steady state, the physical system will always meet the performance and/or safety specification?

Letting $g : D_x \times D_u \times D_p \to \mathbb{R}$ be the performance and/or safety specification, this question can be stated formally as the feasibility problem[3]:

$$\forall \mathbf{p} \in P, \exists \mathbf{u} \in U : g(\mathbf{z}, \mathbf{u}, \mathbf{p}) \leq 0, \mathbf{h}(\mathbf{z}, \mathbf{u}, \mathbf{p}) = \mathbf{0}.$$

In order to formulate this problem mathematically, consider for the moment a single realization of uncertainty. The question that must be addressed is whether or not there exists a control setting such that the performance/safety specification is satisfied, for that particular realization of uncertainty. This problem can be formulated mathematically as the following nonlinear program (NLP):

$$\begin{aligned} \psi(\mathbf{p}) = \min_{\mathbf{z} \in X, \mathbf{u} \in U} \; & g(\mathbf{z}, \mathbf{u}, \mathbf{p}) \\ \text{s.t.} \; & \mathbf{h}(\mathbf{z}, \mathbf{u}, \mathbf{p}) = \mathbf{0}. \end{aligned} \tag{2.2}$$

Upon solving (2.2), if $\psi(\mathbf{p}) \leq 0$, this establishes (with mathematical certainty) that there exists a control such that the performance/safety specification (and the model equations) are satisfied, for that particular realization of uncertainty. In other words, $\psi(\mathbf{p})$ can be thought of as a measure of feasibility (or infeasibility) of the design with respect to a particular realization of uncertainty $\mathbf{p}$ [52]. Of course, the next step is to consider *every* realization of uncertainty; in particular, the worst-case realization

[3]Satisfying this constraint will also be referred to as meeting robust feasibility.

of uncertainty. This amounts to solving the constrained max-min program[4]:

$$\begin{aligned} \eta^* = & \max_{\mathbf{p}\in P, \eta\in\mathbb{R}} \eta \\ \text{s.t. } & \eta \leq \min_{\mathbf{z}\in X, \mathbf{u}\in U} g(\mathbf{z},\mathbf{u},\mathbf{p}) \\ & \text{s.t. } \mathbf{h}(\mathbf{z},\mathbf{u},\mathbf{p}) = \mathbf{0}, \end{aligned} \tag{2.3}$$

by introducing the auxiliary variable $\eta \in \mathbb{R}$. In a similar fashion to the interpretation of ψ, upon solving (2.3), if $\eta^* \leq 0$, this establishes robust feasibility of the design (i.e. for the worst-case realization of uncertainty, there exists a control such that the performance/safety specification is not violated). In other words, η^* can be thought of as a measure of robust feasibility (or infeasibility) of the design.

From here on, it will be assumed that the model function, $\mathbf{h}$, is continuously differentiable on D_x. Conditions under which this assumption may not be necessary will be discussed in later chapters. If for some U and P, unique $\mathbf{z} \in X$ exist that satisfy $\mathbf{h}(\mathbf{z},\mathbf{u},\mathbf{p}) = \mathbf{0}$ at each $(\mathbf{u},\mathbf{p}) \in U \times P$, then they define an implicit function of the controls and uncertainty parameters, that will be expressed as $\mathbf{x} : U \times P \to X$, by asserting the Implicit Function Theorem. Details of this will be discussed more thoroughly in Chapters 3 and 4. The significance of the set X will be discussed in the next section. By representing the state variables as implicit functions of the controls and uncertainty parameters, explicit dependence on them is eliminated and there is a potentially significant reduction in the number of optimization variables as compared to (2.3). The following optimization problem, equivalent to (2.3) and the original feasibility problem, results:

$$\begin{aligned} \eta^* = & \max_{\mathbf{p}\in P, \eta\in\mathbb{R}} \eta \\ & \text{s.t. } \eta \leq \min_{\mathbf{u}\in U} g(\mathbf{x}(\mathbf{u},\mathbf{p}),\mathbf{u},\mathbf{p}). \end{aligned}$$

[4]Program (2.3) is referred to as a constrained max-min program since it is equivalent to: $\eta^* = \max_{\mathbf{p}\in P} \min_{\mathbf{u}\in U, \mathbf{z}\in X} \{g(\mathbf{z},\mathbf{u},\mathbf{p}) : \mathbf{h}(\mathbf{z},\mathbf{u},\mathbf{p}) = \mathbf{0}\}$.

Furthermore, the inner-minimization constraint can be expressed as

$$\eta \leq \min_{\mathbf{u}\in U} g(\mathbf{x}(\mathbf{u},\mathbf{p}),\mathbf{u},\mathbf{p}) \Leftrightarrow \eta \leq g(\mathbf{x}(\mathbf{u},\mathbf{p}),\mathbf{u},\mathbf{p}),\ \forall \mathbf{u} \in U. \tag{2.4}$$

The following optimization problem can then be formulated:

$$\begin{aligned} \eta^* = \max_{\mathbf{p}\in P,\eta\in\mathbb{R}}\ & \eta \\ \text{s.t. }\ & \eta \leq g(\mathbf{x}(\mathbf{u},\mathbf{p}),\mathbf{u},\mathbf{p}),\ \forall \mathbf{u} \in U, \end{aligned} \tag{2.5}$$

which is an SIP since it has a finite number of decision variables, $\mathbf{p}$, and an infinite number of constraints[5] indexed by the set U. The SIP (2.5) will be referred to as the robust simulation SIP which will in turn be referred to as an implicit SIP since it has implicit functions embedded. The solution of the robust simulation SIP will be the primary focus of this work. In the next section, an intuitive picture of the operational envelope and how it relates to the feasibility problem and the implicit SIP will be discussed.

2.2 The Feasibility Problem and the Operating Envelope

The *operating envelope* is an important concept to design engineers. In the context of this work, the operating envelope is simply the region in which the steady-state process operates given all realizations of uncertainty and controls.[6] Figure 2-2 depicts the operating envelope of a process for a single control setting. The design corresponding to the operating envelope in the figure is said to be feasible since for every realization of uncertainty, there exists a control setting such the operating envelope does not

[5]There is a constraint corresponding to each control realization $\mathbf{u}$, for which there are infinitely many realizations from the interval U.

[6]The concept of flexibility, mentioned earlier, is related to the concept of the operating envelope. Again, the index of flexibility is a measure of the size of the uncertainty interval P for which there exists a control setting such that the operating envelope doesn't violate the performance/safety constraint (i.e. the plant operates within the feasible region and the design is said to be feasible).

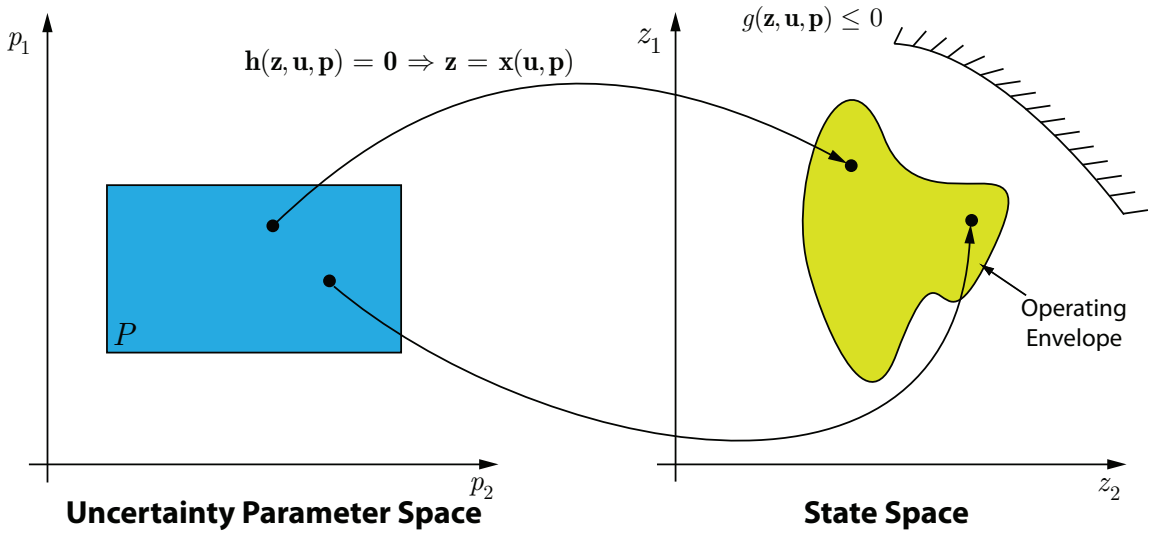

Figure 2-2: The uncertainty parameters are mapped through the model equations to state space, defining the operating envelope, which in this case, is within the region satisfying the performance/safety constraint.

violate the performance/safety specification. With reference to the robust simulation SIP (2.5), this corresponds to an optimal solution value $\eta^* \leq 0$. In short, solving the feasibility problem requires a rigorous evaluation of the operating envelope.

Since the explicit enumeration of uncertainty parameters and controls is an inadequate procedure for evaluating the operating envelope (there are infinitely many points), global information is required. This can be interpreted as needing rigorous and conservative bounds on the operating envelope, or more precisely, on the implicit function $\mathbf{x}$. Figure 2-3 shows such bounds, depicted as the interval X. In essence, evaluating the operating envelope boils down to the ability to calculate rigorous bounds on the image of the control and uncertainty sets under the mapping of the implicit function $\mathbf{x}$. Stated more precisely, rigorous bounds on the image set $\mathbf{x}(U, P)$ are required. In Figure 2-3, rigorous interval bounds on the operating envelope are depicted such that they do not violate the performance/safety specification. This illustrates how robust feasibility of a design can be determined using only the global bounding information. However, it also illustrates how overly-conservative bounds on the operating envelope can generate a situation where the bounds violate the per-

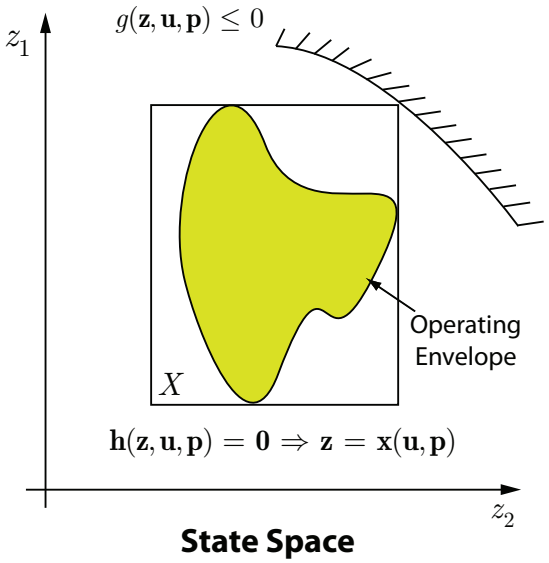

Figure 2-3: The operating envelope enclosed by the interval X.

formance/safety specification when the operating envelope is actually feasible. Of course, in order to determine robust feasibility of a design, there must be a procedure for reconciling the latter situation.

2.2.1 Objectives

Since a design engineer actually only cares about the worst-case realization of uncertainty, most of the uncertainty set does not need to be considered. Therefore, the operating envelope that needs to be evaluated may be considerably smaller than that corresponding to the entire uncertainty set. In turn, this *pruning* of the uncertainty set may lead to the ability to calculate much *tighter* and less conservative bounds on the operating envelope, which in turn leads to the better chances of guaranteeing robust feasibility of the design. However, in the general case, the operating envelope is nonconvex since process systems models often exhibit complex nonlinear behavior. In this case, simply ensuring feasibility for a single realization of uncertainty requires solving an NLP with embedded implicit functions to global optimality. This leads to the following objectives of this thesis:

1. develop a method to calculate rigorous and convergent global bounding information on implicit functions over a range of parameter values,

2. develop a method to solve NLPs with embedded implicit functions to global optimality, and

3. apply these developments to solve SIPs with embedded implicit functions—the so-called robust feasibility problem—to global optimality.

In Chapter 3, a method for bounding implicit functions using interval analysis is developed. In Chapter 4, the interval bounds are used to calculate convex underestimating and concave overestimating functions of implicit functions which are potentially refinements on the interval bounds. In the same chapter, these convex and concave bounding functions are used within the *global optimization of implicit functions* algorithm. These chapters essentially accomplish objectives (1) and (2) above. In Chapter 7, the global solution of SIPs with embedded implicit functions is presented, accomplishing objective (3).

Chapter 3

Bounding Implicit Functions Using Interval Analysis

In this chapter the global root-finding problem is considered for systems of parameter-dependent nonlinear algebraic equations. Solutions of such systems are implicit functions of the parameters and therefore this problem amounts to finding real function *branches*, rather than solution points. In this chapter, a bisection algorithm is presented that relies on (parametric) interval Newton-type methods to calculate interval boxes that are guaranteed to each enclose a locally unique solution branch. A test for existence and uniqueness of enclosed solution branches is presented that is sharper than the classical tests from interval-Newton methods applied to parametric systems. Furthermore, a method for partitioning the parameter space is presented that intelligently searches for a partition that leads to subinterval boxes that are more likely to pass the existence and uniqueness tests. A number of numerical examples are presented with results illustrating the effectiveness of the algorithm.

3.1 Introduction

Enclosing the locally unique solutions of systems of nonlinear equations of the form

$$\mathbf{h}(\mathbf{z}) = \mathbf{0}, \quad \mathbf{h} : D_x \subset \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}, \tag{3.1}$$

with D_x open, using interval analysis, has been addressed extensively in the past. However, it is common that variable coefficients are parameter-dependent, giving rise to parametric nonlinear systems of equations. The form of these equations is:

$$\mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{0}, \quad \mathbf{h} : D_x \times D_p \to \mathbb{R}^{n_x}, \tag{3.2}$$

with $D_p \subset \mathbb{R}^{n_p}$ open. Parametric dependence of coefficients commonly arises when the equations model real-world systems and therefore are inherently inaccurate and uncertain. Model formulations such as (3.2) are therefore applicable across a wide variety of disciplines. In this case, the problem one wishes to solve can be formulated as finding a nontrivial $\Xi \subset \mathbb{R}^{n_x}$ such that

$$\forall \mathbf{p} \in P, \exists \mathbf{z} \in \Xi : \mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{0}, \tag{3.3}$$

with $P \subset D_p$ a compact interval. If there exist such $\mathbf{z}$ satisfying (3.3), then they define an implicit function $\mathbf{x} : P \to \Xi \subset D_x$. Such an $\mathbf{x}$ may not be unique on P, in which case, if there are a finite number of solutions, each $\mathbf{x}_i$ is called a solution *branch*. If $\mathbf{x}_i$ is continuous on P, its image $\mathbf{x}_i(P)$ is a connected compact set. Thus, Ξ encloses the union of a collection of connected compact sets that are the image sets of the locally unique solutions. Under appropriate assumptions, continuity of $\mathbf{x}_i$ is guaranteed by the Implicit Function Theorem.

The new applications in this thesis, in the realms of *Robust Simulation* and *Global Optimization*, require efficient calculations of valid, tight, and convergent enclosures of the solutions of parameter-dependent nonlinear equations. That is, algorithms are required that can efficiently calculate rigorous and tight interval enclosures that are guaranteed to each contain a solution branch that is locally unique on an interval $P^l \subset P$, where the parameter interval P is known or chosen *a priori*, and P^l has nontrivial width.

In [99], Neumaier describes the "covering method" in which the parametric solution set of polynomial systems in the form of (3.2) is covered with interval boxes which then are refined. The application [99] considered for this method is in the

realm of computer-aided geometric design. Thus, the accuracy of the algorithm is simply the accuracy it takes to represent the solution set as an image on a computer screen [99]. He uses the enclosure property of inclusion monotonic interval extensions to test whether a current interval may enclose a solution of (3.2). Together with a generalized bisection approach, an interval will be refined or discarded, where bisections are made in the coordinate with the largest width (in both the $\mathbf{z}$ direction *and* $\mathbf{p}$ direction). The so-called covering method makes use of an interval-Newton method for refining intervals which may contain parts of a solution branch and excluding regions guaranteed not to contain any part of a solution branch. However, the method cannot actually use the inclusion tests that are inherent to interval-Newton methods and therefore, existence and uniqueness of enclosed solutions cannot be guaranteed except in the limit of infinite partitioning. Due to the scope of its application, the covering method is simply not applicable to the problem which this chapter intends to address.

Bounding the solution branch of parameter-dependent linear systems, of the form $\mathbf{A}(\mathbf{p})\mathbf{z} = \mathbf{b}(\mathbf{p})$, has been discussed previously in the literature. In [106], the solution of parameterized linear systems is discussed which makes use of "Rump's fixed-point iteration method for bounding the hull of the solution set [119]." This technique makes it essentially an interval version of the fixed-point method for solving linear systems that relies on inner-approximations of the hull as well as outer-approximations. In [107, 108], the method's key results and implementation as packages for commercially available software are discussed.

The parametric linear system methods have specific importance when solving sensitivity analysis problems. Applications of (3.3) to sensitivity analysis have been considered in [46, 47, 75, 100, 118, 119]. The problem (3.3) is referred to as the *perturbed* problem. In [100], linearizations of (3.2) were considered to calculate rigorous bounds on the solution(s), with [118] offering some improvements. The authors of [100, 118] present rigorous methods using linear interval enclosures of the nonlinear parametric solution. These methods reduce to solving a linear interval system of equations. In [75], the authors make use of rigorous affine interval enclosures, introduced in [74], to

calculate (outer) bounds on the interval hull of the image set. The problem in (3.3) then reduces to solving a linear interval system of equations as well. Although, in theory, the methods produce rigorous enclosures of a locally unique solution to (3.2) over P, they still rely on linear approximations to the nonlinear system, whereas the classic interval Newton-type methods do not.

The interval Newton-type methods provide inherent tests for the existence of solutions in a given interval. In [46], Gay makes use of the parametric extensions of two common "existence tests": the *Krawczyk* [76] test and the Kioustelidis and Moore [93] test based on Miranda [86]. He claims that "it is easy to generalize such existence tests to account for problems in the form of (3.2)" [46]. He also states the "the results of Moore [91] extend immediately to (3.2)" [46]. Although Gay's generalizing statement is true, because he is only interested in verifying the existence of (not necessarily unique) solutions of (3.3), he never discusses the non-triviality associated with making the parametric extension of the interval-method based uniqueness tests.

In [77], Krawczyk uses the formulation introduced in [46], and considers bounding the parameterized function with a so-called *function strip*. The function strip approach uses an interval-valued function that takes the real vector-valued argument $\mathbf{z}$, and outputs an interval enclosure of $\mathbf{h}$ evaluated at $\mathbf{z}$ valid for all $\mathbf{p} \in P$. The problem he then wishes to solve is to find a valid enclosure of the locally unique solution $\mathbf{x}$ on P. The Krawczyk operator and the interval-Newton operator were then generalized for the use with function strips. The idea of a function strip is analogous in many ways to what how the interval methods will be used in this chapter, except no modifications to the already well understood interval Newton-type operators will be made, except to incorporate parameter dependence.

In [56], Hansen and Walster briefly discuss some theory and application of the interval Newton method to parametric nonlinear problems, including some analysis. A one-dimensional parametric example is presented in [56, 54] in which a modified parametric interval-Newton method is applied to calculate tight bounds on the locally unique solution branch. The presented approach calculates the parametric interval-Newton operator from multiple *points of expansion*, as opposed to just one, in which

the standard parametric extension to the interval-Newton operator is calculated. In [54], they also generalize the approach to multi-dimensional cases noting its inherent inefficiency. However, the discussion in [56] and [54] is incomplete as they fail to address thoroughly the many important results concerning the parametric extension of interval Newton-type methods. A more thorough analysis of interval methods for parameter-dependent nonlinear systems of equations is given in [101].

Interval arithmetic and (non-parametric) interval Newton-type methods have been applied within various algorithms for bounding *all* solutions of systems of equations, such as (3.1), that exist within a *large* initial box. In [68], the authors propose applying *generalized bisection* to the solve *global root-finding problem*. Coupled with interval methods, in which tests for existence and uniqueness of solutions in a given box are inherent, all real solutions of (3.1) can be found. In [70] this exact strategy was applied to bound all solutions of (3.1). In [53], *extended interval arithmetic* was developed and applied in a similar manner to reduce the initial space into subintervals, known to enclose solutions, in which the interval Jacobian matrix is nonsingular, providing uniqueness of enclosed solutions. This technique is rather appealing because even without bisection, new information may be calculated that allows for regions of the original box to be excluded.

An extension of generalized bisection [68] to parametric nonlinear systems, as in (3.2) will be proposed in this work. The objective of such an algorithm is to generate boxes $X^l \times P^l \subset X \times P$, with P^l having substantial width, such that X^l is guaranteed to enclose a locally-unique solution branch on all of P^l. One subtle difference in the parameterized generalized bisection procedure is the processing of interval boxes taking into account the idea of *partial enclosures*. In other words, simply applying generalized bisection to a parameterized problem and blindly bisecting in X (and not P) is prone to produce boxes that enclose a solution branch for some, but not all $\mathbf{p} \in P$. The parameterized generalized bisection algorithm will apply the standard theory developed for interval methods as well as incorporate some new results to, first and foremost, avoid partial enclosures while bisecting X. Similarly, if no "safe" bisection of X can be guaranteed, the algorithm applies a procedure for partitioning the P

interval box. Since it is desired that this algorithm bounds solution branches for all $\mathbf{p} \in P^l \subset P$, with sufficiently wide P^l, passing the classical existence/uniqueness tests of the interval Newton-type methods becomes an issue. Since the classical existence and uniqueness tests are often too strong to pass for parameter-dependent problems, the parameterized generalized bisection algorithm relies on a sharper existence and uniqueness test, developed in Section 3.4.

Alternatively, homotopy continuation [4, 129] has been applied to find all solution branches of the system (3.2). In particular, it has commonly been applied to (small) polynomial systems of a single parameter [94]. This is done by *tracing* a parametric solution curve starting at the lowest value of the parameter and continuously solving the system of equations successively for increasing parameter values. Various applications to systems with multiple parameters has been discussed [113, 114, 115, 129]. However, as discussed, homotopy continuation can only handle a single parameter. Extensions to multiple parameter problems requires a reformulation into a single parameter problem. Equivalence of the reduced problem to the higher dimension problem is then guaranteed only on a given path on the solution surface to the multiple parameter problem [114]. Thus, multiple parameter problems are not only inefficient to solve, but generating accurate pictures of multidimensional solution surfaces from a one-dimensional approach is inherently problematic [114]. Furthermore, continuation does not offer the ability to enclose solutions branches rigorously, only approximate their critical boundaries [113] to finite precision.

In Section 3.2, the mathematical notation and nomenclature used throughout this chapter and beyond, is presented. Section 3.3 presents two classical interval methods, discusses the idea of interval iteration, and presents the standard, well-established results regarding existence and uniqueness of enclosed solutions. In Section 3.4, some theoretical results regarding partial enclosures are given as well as a stronger existence and uniqueness result that provides a sharper test than the classical tests. Furthermore, in Section 3.4, the convergence properties of interval enclosures under partitioning the parameter interval is discussed. These results have important implications when using the interval enclosures as bounding information in global

optimization applications. In Section 3.5, the parameterized generalized bisection algorithm is formalized and the heuristic cutting strategies are discussed. In Section 3.6, the computer implementation is discussed and some numerical examples are given that illustrate the performance of the algorithm. Finally, in Section 3.7, some concluding remarks are given.

3.2 Background

This section provides the reader with the background mathematical concepts, results, and nomenclature, with respect to interval analysis, used throughout this thesis.

Assumption 3.2.1. Unless otherwise stated, there exists at least one implicit function $\mathbf{x} : P \subset D_p \to D_x$ such that $\mathbf{h}(\mathbf{x}(\mathbf{p}), \mathbf{p}) = \mathbf{0}$ holds for every $\mathbf{p} \in P$.

Remark 3.2.2. If the closed convex hull of the Jacobian matrix of $\mathbf{h}$ with respect to $\mathbf{z}$ (denoted $\mathbf{J}_\mathbf{z}$) on the set $X \times P \subset D_x \times D_p$ does not contain any singular matrices, an implicit function satisfying Assumption 3.2.1 is unique in X and it is continuous on P. This is a consequence of Proposition 5.1.4 in [101] and the semilocal implicit function Theorem 5.1.3 in [101].

3.2.1 Interval Analysis

The notation and some concepts from interval analysis are presented in this section. The reader is directed to [92, 101] for a more complete background on the concepts of interval analysis.

Definition 3.2.3. An interval $Z \subset \mathbb{R}^m$ is defined as the nonempty connected compact set:

$$Z = \{\mathbf{z} \in \mathbb{R}^m : \mathbf{z}^L \leq \mathbf{z} \leq \mathbf{z}^U\},$$

with $\mathbf{z}^L \in \mathbb{R}^m$ and $\mathbf{z}^U \in \mathbb{R}^m$ as the lower and upper bounds of the interval Z, respectively. The i^{th} component of Z will be denoted Z_i.

Definition 3.2.4. The *interior* of an interval $Z \subset \mathbb{R}^m$ is defined as the connected open (possibly empty) set:

$$\text{int}(Z) = \{\mathbf{z} \in \mathbb{R}^m : \mathbf{z}^L < \mathbf{z} < \mathbf{z}^U\}.$$

Definition 3.2.5. The set of interval subsets of $\mathbb{R}$ is denoted $\mathbb{IR}$.

Definition 3.2.6. Let $Z \subset \mathbb{R}^m$. The set $\{Y \in \mathbb{IR}^m : Y \subset Z\}$ is denoted as $\mathbb{I}Z$.

With this definition, it is clear that $\mathbb{I}Z \subset \mathbb{IR}^m$.

Definition 3.2.7 (Midpoint)**.** The *midpoint* or median, of an interval $Z \in \mathbb{IR}$ is defined as:

$$m(Z) \equiv \frac{z^L + z^U}{2}, \quad i = 1, \ldots, m. \tag{3.4}$$

For $Z \in \mathbb{IR}^m$, the midpoint will be a real vector $m(Z)$ whose i^{th} component is $m(Z_i)$. Similarly, for $Z \in \mathbb{IR}^{m \times n}$, the midpoint will be a real matrix $m(Z)$ whose $(i,j)^{th}$ element is $m(Z_{ij})$.

Definition 3.2.8 (Image)**.** The *image* of the set $Z \in \mathbb{IR}^m$ under the mapping $\mathbf{f} : A \subset \mathbb{R}^m \to \mathbb{R}^n$, with $Z \in \mathbb{I}A$ is denoted as $\hat{\mathbf{f}}(Z)$.

Note that the image is not necessarily an interval.

Definition 3.2.9 (Interval Hull)**.** Let $A \subset \mathbb{R}^m$ be bounded. $\square A \in \mathbb{IR}^m$ is called the *interval hull* of A if $A \subset \square A$ and for any $Z \in \mathbb{IR}^m$ such that $A \subset Z$, $\square A \subset Z$.

Definition 3.2.10. The *interval hull* of the image of $Z \in \mathbb{I}A$ under $\mathbf{f} : A \subset \mathbb{R}^m \to \mathbb{R}^n$ is denoted as $\square\hat{\mathbf{f}}(Z) = \left[\hat{\mathbf{f}}^L(Z), \hat{\mathbf{f}}^U(Z)\right]$.

Definition 3.2.11. An interval-valued function $F : \mathbb{I}A \to \mathbb{IR}^n$, evaluated at any $Z \in \mathbb{I}A \subset \mathbb{IR}^m$, is denoted as $F(Z)$.

Definition 3.2.12 (Interval Extension)**.** Let $Z \subset \mathbb{R}^m$. An interval-valued function $F : \mathbb{I}Z \to \mathbb{IR}^n$ is called an *interval extension* of the real-valued function $\mathbf{f} : Z \to \mathbb{R}^n$ on Z, if

$$\mathbf{f}(\mathbf{z}) = \mathbf{y} = [\mathbf{y}, \mathbf{y}] = F([\mathbf{z}, \mathbf{z}]), \quad \forall \mathbf{z} \in Z.$$

It should be noted that this definition of an interval extension is the same as that in [92]. It is more general than the definition of an interval extension in [101] which also requires that F is an inclusion function of f, which is defined below. For the purposes of this thesis the more general definition of an interval extension is used.

Definition 3.2.13. An interval extension, $F(Z)$, of the function $\mathbf{f} : A \subset \mathbb{R}^m \to \mathbb{R}^n$, at $Z \in \mathbb{I}A$, is called *exact* at $Z \in \mathbb{I}A$ if $F(Z) = \square\hat{\mathbf{f}}(Z)$.

Definition 3.2.14 (Inclusion Monotonic [92, 101])**.** Let $Z \subset \mathbb{R}^m$. An interval-valued function $F : \mathbb{I}Z \to \mathbb{IR}^n$ is called *inclusion monotonic* on Z if for every $A, B \in \mathbb{I}Z$,

$$B \subset A \Rightarrow F(B) \subset F(A). \tag{3.5}$$

Definition 3.2.15 (Inclusion Function)**.** An interval-valued function $F : \mathbb{I}Z \to \mathbb{IR}^n$ is called an *inclusion function* of $f : Z \to \mathbb{R}^n$ on Z if

$$\hat{\mathbf{f}}(A) \subset F(A), \ \ \forall A \in \mathbb{I}Z.$$

Theorem 3.2.16 ([92, 101])**.** *Let $Z \subset \mathbb{R}^m$. and let $F : \mathbb{I}Z \to \mathbb{IR}^n$ be an inclusion monotonic interval extension of $\mathbf{f} : Z \to \mathbb{R}^n$ on Z. Then F is an inclusion function of $\mathbf{f}$ on Z.*

Proof. Proof can be found in [92] page 21. □

This so-called inclusion property is also known as *The Fundamental Theorem of Interval Analysis.*

Definition 3.2.17 (Nested Sequences)**.** A sequence of intervals $\{Z^k\}$, with $Z^k \in \mathbb{IR}^m$ is said to be nested if $Z^{k+1} \subset Z^k$ for every k.

Theorem 3.2.18 (Finite Convergence [92])**.**

1. *Every nested sequence of intervals is convergent and has the limit*

$$Z^* = \bigcap_{k=1}^{\infty} Z^k.$$

2. *For some real vector* $\mathbf{z}$ *such that* $\mathbf{z} \in Z^k$ *for all* k*, the sequence of intervals* $\{Y^k\}$ *defined by* $Y^1 = Z^1$ *and* $Y^{k+1} = Z^{k+1} \cap Y^k$ *for* $k = 1, 2, \ldots$ *is a nested sequence with limit* Y^* *and*

$$\mathbf{z} \in Y^* \subset Y^k.$$

3. *Using outward-rounded interval arithmetic, there exists a* $K \in \mathbb{N}$ *such that* $Y^k = Y^K$ *for* $k \geq K$ *and the sequence* $\{Y^k\}$ *is said to converge in* K *steps.*

Proof. The proof can be found in [92] on page 36. □

Definition 3.2.19 (Interval Width)**.** The *width* of an interval $Z \in \mathbb{IR}$ is defined as the distance between its upper and lower bounds:

$$w(Z) = z^U - z^L.$$

Definition 3.2.20 (Interval Vector Width)**.** The width of an interval vector $Z \in \mathbb{IR}^m$ is the vector, $w(Z)$, whose i^{th} component is $w(Z_i)$.

Note that this definition is consistent with that of [3] and differs from that of [92].

Definition 3.2.21 (Radius)**.** Let $A \in \mathbb{IR}$. The *radius* of A is defined as one-half its width:

$$rad(A) = w(A)/2.$$

For $A \in \mathbb{IR}^{m \times n}$, $rad(A)$ is the $m \times n$-dimensional real-valued matrix whose $(i, j)^{\text{th}}$ element is given by $rad(A_{ij})$.

The metric used in interval analysis is the Hausdorff metric.

Definition 3.2.22 (Hausdorff Metric [92, 101, 109])**.** Distances in the *Hausdorff metric* are defined as

$$d_H(Z, Y) = \max_i \{|z_i^L - y_i^L|, |z_i^U - y_i^U|\} \tag{3.6}$$

with $Z, Y \in \mathbb{IR}^m$.

Note that the Hausdorff metric induces the max-norm $||\cdot||_\infty$ [109].

Theorem 3.2.23 (Completeness, [3])**.** *The space $\mathbb{IR}^m$ equipped with the Hausdorff metric is a complete metric space.*

Proof. Proof can be found in [3]. □

Definition 3.2.24. An inclusion monotonic interval extension of the partial derivative of the continuously-differentiable function $f_i : A \subset \mathbb{R}^m \to \mathbb{R}$ with respect to z_j evaluated at $Z \in \mathbb{I}A$ is denoted as

$$\frac{\partial F_i}{\partial z_j}(Z).$$

Definition 3.2.25. An inclusion monotonic interval extension of the Jacobian matrix, of a vector-valued function $\mathbf{f} : A \subset \mathbb{R}^m \to \mathbb{R}^n$, evaluated at $Z \in \mathbb{I}A$ is denoted as

$$J_{\mathbf{z}}(Z) \equiv \begin{bmatrix} \frac{\partial F_1}{\partial z_1}(Z) & \cdots & \frac{\partial F_1}{\partial z_m}(Z) \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial z_1}(Z) & \cdots & \frac{\partial F_n}{\partial z_m}(Z) \end{bmatrix}. \tag{3.7}$$

The subscript $\mathbf{z}$ becomes essential when dealing with parameter dependent functions. For instance, given a function $\mathbf{f} : D_x \times D_p \to \mathbb{R}^{n_x}$, the notation $\mathbf{J_x}$ refers to the matrix of partial derivatives with respect to the first vector of arguments (in D_x).

Definition 3.2.26 (Singularity)**.** Let $A \in \mathbb{IR}^{n\times n}$. A is said to be *singular* if there exists a singular matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ such that $\mathbf{A} \in A$. Similarly, A is said to be nonsingular if no such $\mathbf{A}$ exists.

In [101], Neumaier introduces a more general concept applicable to rectangular interval matrices $A \in \mathbb{IR}^{m\times n}$. A matrix A is said to be regular if every $\mathbf{A} \in A$ has rank n. Thus, when $m = n$, the notions of non-singular and regular are equivalent.

Definition 3.2.27. Let $A \in \mathbb{IR}^{n\times n}$ be nonsingular. Then

$$A^{-1} \equiv \square\{\mathbf{A}^{-1} : \mathbf{A} \in A\}$$

is the inverse of an interval matrix.

Theorem 3.2.28. *Let $A \in \mathbb{IR}^{n_x \times n_x}$, $m(A)$ be nonsingular, and define $\mathbf{Z} \equiv |m(A)^{-1}|\, rad(A)$, where $|m(A)^{-1}|$ is the elementwise absolute value of $m(A)^{-1}$. Let $\lambda_{max} = \max_i\{|\lambda_i|\}$ be the magnitude of the extremal eigenvalue(s) of $\mathbf{Z}$. If $\lambda_{max} < 1$, then A is nonsingular.*

Proof. Follows from Proposition 4.1.1 and Corollary 4.1.3 in [101]. □

3.2.2 Extended Interval Arithmetic

Extended interval arithmetic is an interval arithmetic that, for the concerns of this thesis, defines division by intervals enclosing 0. Such cases may arise when applying parametric interval Newton-type iterations to bound parametric solutions of (3.2). For example, when an interval Jacobian matrix is calculated that encloses a singular matrix, an interval division by zero may be encountered and no new information can be obtained using standard interval division. However, applying extended interval arithmetic to the parametric interval Newton-type iteration may provide a way to calculate new information and circumvent the interval divide by zero scenario. It is apparent that if multiple solutions exist in a box the interval Jacobian matrix will be singular. Similarly, if an interval Jacobian matrix is singular, then uniqueness of enclosed solutions cannot be verified. The application of extended interval arithmetic to parametric systems is analogous to non-parametric systems. Divisions with intervals containing zero are defined in the following.

Definition 3.2.29. Let $A \in \mathbb{IR}$ such that $0 \in A = [a^L, a^U]$. Then

$$\frac{1}{A} = \begin{cases} \left[1/a^U, +\infty\right) & \text{if } \ a^L = 0, \\ \left(-\infty, 1/a^L\right] & \text{if } \ a^U = 0, \\ \left(-\infty, 1/a^L\right] \cup \left[1/a^U, +\infty\right) & \text{otherwise.} \end{cases}$$

Thus, extended interval arithmetic returns either an unbounded interval or the union of two disjoint unbounded intervals. If the division by an interval containing

zero results in the union of two disjoint intervals, each interval in the union is then separately returned to the parametric interval Newton-type iteration for refinement and/or the application of the inclusion/exclusion tests.

3.3 Interval Methods

The interval methods presented in this section have been studied extensively in the past. For a more thorough analysis, the reader is directed to [101]. This section will simply establish the parameterized notation and state the generalized inclusion results and the existence/uniqueness tests for each parametric interval method considered. The following assumptions must be made.

Assumption 3.3.1.

(a) The function $\mathbf{h} : D_x \times D_p \to \mathbb{R}^{n_x}$ is continuously differentiable on $D_x \times D_p$.

(b) The functions $\mathbf{h}$ and $\mathbf{J_x}$ have inclusion monotonic interval extensions, H and $J_{\mathbf{x}}$.

Remark 3.3.2. Continuous differentiability is only required for the interval methods as they are presented in this thesis. However, strictly speaking, this assumption is not necessary since *generalized derivative* information may be used so long as $\mathbf{h}$ is Lipschitz.

The parametric extension to the interval-Newton method has been discussed in [56, 101] and a slightly modified form in [77]. The parametric interval-Newton operator is defined, in its Gauss-Seidel form, in the following.

Definition 3.3.3. Let $X^k \in \mathbb{IR}^{n_x}$, $P \in \mathbb{IR}^{n_p}$, $\mathbf{x}^k \in X^k$, and $\mathbf{Y}^k \in \mathbb{R}^{n_x \times n_x}$. Define $A^k \in \mathbb{IR}^{n_x \times n_x}$ and $B^k \in \mathbb{IR}^{n_x}$ as $A^k \equiv \mathbf{Y}^k J_{\mathbf{x}}(X^k, P)$ and $B^k \equiv \mathbf{Y}^k H(\mathbf{x}^k, P)$, respectively. the parametric interval-Newton operator $N : \mathbb{R}^{n_x} \times \mathbb{IR}^{n_x} \times \mathbb{IR}^{n_p} \to \mathbb{IR}^{n_x}$ is

defined as

`for` $i = 1, \ldots, n_x$ `do`

$$N_i(\mathbf{x}^k, X^k, P) := x_i^k - \left[B_i^k + \sum_{j=1}^{i-1} A_{ij}^k \left(X_j^{k+1} - x_j^{k+1} \right) + \sum_{j=i+1}^{n_x} A_{ij}^k \left(X_j^k - x_j^k \right) \right] / A_{ii}^k \tag{3.8}$$

$X_i^{k+1} := N_i^k(\mathbf{x}^k, X^k, P) \cap X_i^k$

`end`.

A common preconditioning matrix is the *midpoint inverse*: $\mathbf{Y}^k = [m(J_{\mathbf{x}}(X^k, P))]^{-1}$, and thus $\mathbf{Y}^k = \mathbf{Y}^k(X^k, P)$. Note that by preconditioning with $\mathbf{Y}^k$, the case in which $0 \in A_{ii}^k$ is more likely to be avoided than without preconditioning. However, if $0 \in A_{ii}^k$, extended interval arithmetic may be employed. The next operator avoids interval division altogether.

Definition 3.3.4. For $X^k \in \mathbb{IR}^{n_x}$, $P \in \mathbb{IR}^{n_p}$, $\mathbf{x}^k \in X^k$, and $\mathbf{Y}^k \in \mathbb{R}^{n_x \times n_x}$, the parametric interval Krawczyk operator $K : \mathbb{R}^{n_x} \times \mathbb{IR}^{n_x} \times \mathbb{IR}^{n_p} \to \mathbb{IR}^{n_x}$ is defined as

$$K(\mathbf{x}^k, X^k, P) \equiv \mathbf{x}^k - \mathbf{Y}^k H(\mathbf{x}^k, P) + (\mathbf{I} - \mathbf{Y}^k J_{\mathbf{x}}(X^k, P))(X^k - \mathbf{x}^k), \tag{3.9}$$

where $\mathbf{I}$ is the $n_x \times n_x$-dimensional identity matrix.

A more efficient and practical calculation of the parametric Krawczyk operator can be done by using a sequential, componentwise strategy, analogous to the parametric interval-Newton operator.

Definition 3.3.5. Let $X^k \in \mathbb{IR}^{n_x}$, $P \in \mathbb{IR}^{n_p}$, $\mathbf{x}^k \in X^k$, and $\mathbf{Y}^k \in \mathbb{R}^{n_x \times n_x}$. Define $A^k \in \mathbb{IR}^{n_x \times n_x}$ and $B^k \in \mathbb{IR}^{n_x}$ as $A^k \equiv \mathbf{I} - \mathbf{Y}^k J_{\mathbf{x}}(X^k, P)$ and $B^k \equiv \mathbf{Y}^k H(\mathbf{x}^k, P)$,

respectively. The parametric Krawczyk operator is defined as

$$\begin{aligned}
&\texttt{for } i = 1, \ldots, n_x \texttt{ do} \\
&K_i(\mathbf{x}^k, X^k, P) := x_i^k - B_i^k + \sum_{j=1}^{i-1} A_{ij}^k \left(X_j^{k+1} - x_j^k\right) + \sum_{j=i}^{n_x} A_{ij}^k \left(X_j^k - x_j^k\right) \qquad (3.10) \\
&X_i^{k+1} := K_i(\mathbf{x}^k, X^k, P) \cap X_i^k \\
&\texttt{end.}
\end{aligned}$$

Definition 3.3.6 (Parametric Interval Method)**.** Let $X^0 \in \mathbb{I}D_x$, $P \in \mathbb{I}D_p$, $\mathbf{x}^0 \in X^0$, and $\mathbf{Y}^0 \in \mathbb{R}^{n_x \times n_x}$. Then, the iteration

$$X^{k+1} := \Phi(\mathbf{x}^k, X^k, P), \quad k \in \mathbb{N} \qquad (3.11)$$

with $\Phi \in \{N, K\}$ defined as in Definitions 3.3.3 or 3.3.5 will be referred to as a parametric interval method.

The sequence of intervals produced by the iteration (3.11) will, by construction, be a nested sequence of intervals. By Theorem 3.2.18, these sequences are convergent.

3.3.1 General Results on Parametric Interval Methods

The results presented in this section are simply statements of the results in [101] formalized for parameter-dependent systems.

Theorem 3.3.7 (Exclusion)**.** *Let $X^0 \in \mathbb{I}D_x$, $P \in \mathbb{I}D_p$, and $\mathbf{x}^0 \in X^0$. Let $\{X^k\}$ be a nested sequence of intervals generated by a parametric interval method* (3.11) *starting from X^0. If for some $k \in \mathbb{N}$, and for some component i, $\Phi_i(\mathbf{x}^k, X^k, P) \cap X_i^k = \emptyset$, then there are no solutions of* (3.2) *in X^0 (or X^k) for any $\mathbf{p} \in P$.*

Theorem 3.3.8 (Interval Enclosure)**.** *Let $X^0 \in \mathbb{I}D_x$, $P \in \mathbb{I}D_p$, and $\mathbf{x}^0 \in X^0$. Let $\{X^k\}$ be a nested sequence of intervals generated by a parametric interval method* (3.11) *starting from X^0. Suppose $\mathbf{x} : P \to \mathbb{R}^{n_x}$ is a continuous solution branch of* (3.2)*. If $\hat{\mathbf{x}}(P) \subset X^0$. Then:*

1. $\hat{\mathbf{x}}(P) \subset X^k, \ \ \forall k \in \mathbb{N}$,

2. $\hat{\mathbf{x}}(P) \subset \Phi(\mathbf{x}^k, X^k, P), \ \ \forall k \in \mathbb{N}$.

Corollary 3.3.9. *Let $X^0 \in \mathbb{I}D_x$, $P \in \mathbb{I}D_p$, and $\mathbf{x}^0 \in X^0$. Let $\{X^k\}$ be a nested sequence of intervals generated by a parametric interval method* (3.11) *starting from X^0. Suppose $\mathbf{x} : P \to \mathbb{R}^{n_x}$ is a continuous solution branch of* (3.2)*. Then the following holds:*

1. $\mathbf{x}(\mathbf{p}) \in X^0 \Rightarrow \mathbf{x}(\mathbf{p}) \in X^k, \ \forall k \in \mathbb{N}$,

2. $\mathbf{x}(\mathbf{p}) \notin X^0 \Rightarrow \mathbf{x}(\mathbf{p}) \notin X^k, \ \forall k \in \mathbb{N}$.

Theorem 3.3.10 (Existence and Uniqueness)**.** *Let $X \in \mathbb{I}D_x$, $P \in \mathbb{I}D_p$, $\mathbf{z} \in \text{int}(X)$, and $\Phi \in \{N, K\}$ with N and K defined as in Definitions 3.3.3 and 3.3.5, respectively. If $\Phi(\mathbf{z}, X, P) \subset \text{int}(X)$, then $J_{\mathbf{x}}(X, P)$ is nonsingular and there exists a solution branch $\mathbf{x} : P \to D_x$ of* (3.2) *in X and it is unique.*

Although applying the existence and uniqueness test of Theorem 3.3.10 is trivial computationally, due to the width of the parameter interval P, overestimation becomes an issue, potentially causing the test to be rather difficult to pass.

3.4 Theoretical Development

3.4.1 Existence and Uniqueness of Enclosed Solutions

In this section, useful results for verifying existence and uniqueness of enclosed solutions are presented. The first result is simply a generalization of the converse of Miranda's theorem (Cor. 5.3.8 in [101]) for existence of enclosed solutions.

Theorem 3.4.1. *Let $X \in \mathbb{I}D_x$ and $P \in \mathbb{I}D_p$. Let $H \equiv [\mathbf{h}^L, \mathbf{h}^U]$ be an inclusion function, and an interval extension, of $\mathbf{h}$ on $X \times P$. For some i, choose $\tilde{x}_i \in X_i$ and set $Z_i := [\tilde{x}_i, \tilde{x}_i]$ and $Z_j := X_j$ for $j \neq i$. If for some l, $h_l^L(Z, P)h_l^U(Z, P) > 0$ holds, then there does not exist a solution $\mathbf{z}$ to $\mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{0}$ in Z for any $\mathbf{p} \in P$.*

Proof. Suppose not. Then there exists a solution $\mathbf{z}$ to $\mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{0}$ in Z for some $\mathbf{p} \in P$. Since $Z_i = \tilde{x}_i$, the point $\mathbf{x} = [x_1, x_2, \ldots, \tilde{x}_i, \ldots, x_{n_x}]^{\mathrm{T}}$ must satisfy $\mathbf{h}(\mathbf{x}, \mathbf{p}) = \mathbf{0}$ for some $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n_x}, \mathbf{p}) \in X_1 \times \ldots \times X_{i-1} \times X_{i+1} \times \ldots \times X_{n_x} \times P$, since H is an inclusion function and an interval extension of $\mathbf{h}$. This implies $\mathbf{0} \in H(Z, P)$ which implies $h_l^L(Z, P) h_l^U(Z, P) \leq 0$ for every $l = 1, 2, \ldots, n_x$, a contradiction. □

The previous result offers an efficient way to rule out a partial enclosure scenario. For instance, if a partial enclosure is suspected in dimension i, then it is expected that x_i crosses a boundary of X_i; either its upper bound x_i^U, lower bound x_i^L, or both. Therefore, setting $\tilde{x}_i := x_i^L$ or $\tilde{x}_i := x_i^U$ in Theorem 3.4.1 allows one to verify if x_i does not leave the lower or upper bound of X_i, respectively. Of course, in order to verify this for all $i = 1, 2, \ldots, n_x$, Theorem 3.4.1 must be applied $2n_x$ times. A simpler way to exclude the possibility of a partial enclosure is given in the next result.

Theorem 3.4.2. *Let $X^0 \in \mathbb{I}D_x, P \in \mathbb{I}D_p$ and $\{X^k\}$ be a sequence of intervals generated by a parametric interval method* (3.11)*, starting at X^0. Suppose that a continuous solution branch $\mathbf{x} : P \to D_x$ of* (3.2) *exists. Then the quantities referred to exist and for $k \in \mathbb{N}$:*

$$\hat{x}_i^U(P) \geq x_i^{0,L} \geq \hat{x}_i^L(P) \Rightarrow x_i^{k,L} = x_i^{0,L}, \quad i = 1, 2, \ldots, n_x.$$

Proof. The minimum and maximum of a continuous function on a compact set exist and define the image set. By hypothesis $\hat{x}_i^U(P) \geq x_i^{0,L} \geq \hat{x}_i^L(P)$ so that by continuity of the solution, $x_i^{0,L}$ forms part of a solution for some $\hat{\mathbf{p}} \in P$. Also, $x_i^{0,L} \in X_i^0$ by definition. Hence, applying Theorem 3.3.8 on $[\hat{\mathbf{p}}, \hat{\mathbf{p}}]$ yields $x_i^{0,L} \in X_i^k$, $\forall k$ and thus $x_i^{k,L} \leq x_i^{0,L}$, $\forall k$ must hold. However, by construction $x_i^{k,L} \geq x_i^{0,L}$, $\forall k$. Thus $x_i^{k,L} = x_i^{0,L}$, $\forall k$ for $i = 1, 2, \ldots, n_x$. □

Corollary 3.4.3. *Suppose the hypotheses of Theorem 3.4.2 are satisfied. Then for $k \in \mathbb{N}$:*

1. $x_i^{k,L} > x_i^{0,L} \Rightarrow x_i(\mathbf{p}) \neq x_i^{0,L}$, $\forall \mathbf{p} \in P$,
2. $x_i^{k,U} < x_i^{0,U} \Rightarrow x_i(\mathbf{p}) \neq x_i^{0,U}$, $\forall \mathbf{p} \in P$

hold for $i = 1, 2, \ldots, n_x$.

The practical application of this result is that when applying a parametric interval Newton-type method, if any improvement is observed on any of the bounds, it is known that a solution curve does not cross that bound. Thus, if it is not known whether X^0 is a partial enclosure of a solution curve, one can apply a parametric interval Newton-type method and potentially guarantee no partial enclosure. As we will see in Section 3.5, Theorem 3.4.1 and Corollary 3.4.3 will be applied within Algorithm 3.1 to find a position to bisect the current interval so as to avoid a partial enclosure situation altogether. The following result provides a much sharper, computationally verifiable test for existence and uniqueness of enclosed solutions as compared to the standard test of Theorem 3.3.10.

Theorem 3.4.4 (Existence and Uniqueness)**.** *Let* $Z \in \mathbb{I}D_x$, $P \in \mathbb{I}D_p$, *and* $\mathbf{z} \in Z$. *Suppose* $\mathbf{Y}^{-1} \equiv m(J_{\mathbf{x}}(Z, P))$ *is nonsingular. Let* $\Phi \in \{N, K\}$ *be defined as in Definitions 3.3.3 or 3.3.5. Let* $\mathbf{A} = |\mathbf{Y}| rad(J_{\mathbf{x}}(Z, P))$. *Let* $\lambda_{max} = \max_i\{|\lambda_i|\}$ *be the magnitude of the extremal eigenvalue(s) of* $\mathbf{A}$. *Suppose it is guaranteed (say by Theorem 3.4.1 or Corollary 3.4.3) that no solution branch intersects the bounds of* Z. *If* $\lambda_{max} < 1$, $\Phi(\mathbf{z}, Z, \bar{\mathbf{p}}) \subset \text{int}(Z)$, $\mathbf{z} \in \text{int}(Z)$, *and* $\bar{\mathbf{p}} \in P$, *then there exists a unique solution branch in* Z *for every* $\mathbf{p} \in P$.

Proof. From the hypotheses, the following hold:

1. if solution branches exist in Z, they are contained in its interior,
2. since $\Phi(\mathbf{z}, Z, \bar{\mathbf{p}}) \subset \text{int}(Z)$, a unique solution exists in Z at $\bar{\mathbf{p}}$
3. since $\lambda_{max} < 1$, by Theorem 3.2.28 it follows that $J_{\mathbf{x}}(Z, P)$ is nonsingular.

Therefore, the hypotheses of Proposition 5.1.4 and Theorem 5.1.3 in [101] are satisfied. It follows that there exists a unique solution branch in Z. □

Remark 3.4.5. Guaranteeing that no solution branch intersects the bounds of Z can be done by either applying Theorem 3.4.1 $2n_x$ times (at each bound) or by applying a parametric interval method and verifying $Z^* \subset Z^0$ strictly, where Z^* is the converged interval and Z^0 is the initial interval; a result of Corollary 3.4.3.

3.4.2 Convergence

An important property of enclosures of locally unique solution branches of (3.2) generated by parametric interval methods, is their convergence behavior under partitioning P. The results in this section have important implications when using this bounding information within global optimization applications such as in the next chapter and in [128, 134]. The reader should be aware that the results presented in this section may be not entirely complete and are the topic of future research. The following result will be important in later continuity arguments.

Lemma 3.4.6. *Let $A \in \mathbb{IR}^{n\times n}$ with $m(A)$ nonsingular. If $m(A)^{-1}A$ is nonsingular, then for $B \in \mathbb{I}A$, $m(B)^{-1}B$ is nonsingular and has diagonal elements that do not contain zero.*

Proof. By Corollary 4.1.3 in [101], since $m(A)^{-1}A$ is nonsingular, $m(B)^{-1}B$ is nonsingular. By Theorem 4.1.1 in [101], diagonal elements of $m(B)^{-1}B$ do not contain 0. □

Lemma 3.4.7. *Let $m[J_{\mathbf{x}}(X^0, P)]$ be nonsingular for some $(X^0, P) \in \mathbb{I}D_x \times \mathbb{I}D_p$. Suppose $\mathbf{Y}^0 J_{\mathbf{x}}(X^0, P)$ is nonsingular with $=m[J_{\mathbf{x}}(X^0, P)]^{-1}$, then the interval-valued functions $N, K : D_x \times \mathbb{I}D_x \times \mathbb{I}D_p \to \mathbb{IR}^{n_x}$, defined in Definitions 3.3.3, 3.3.4, and 3.3.5, are continuous on their domains.*

Proof. By continuous differentiability of $\mathbf{h}$ on $D_x \times D_p$, $\mathbf{h}$ and $\mathbf{J_x}$ are continuous. By Lemma 3.4.6, if $\mathbf{Y}^0 J_{\mathbf{x}}(X^0, P)$ is nonsingular for some $(X^0, P) \in \mathbb{I}D_x \times \mathbb{I}D_p$, and $\mathbf{Y}^0 = m[J_{\mathbf{x}}(X^0), P]^{-1}$, then $\mathbf{Y}^k J_{\mathbf{x}}(X^k, P)$ is nonsingular for every k and its diagonal elements do not enclose 0. By Theorem 2.1.1 in [101], $K : D_x \times \mathbb{I}D_x \times \mathbb{I}D_p$ as in Def. 3.3.4 and Def. 3.3.5 and $N : D_x \times \mathbb{I}D_x \times \mathbb{I}D_p$ as in Def. 3.3.3 are continuous. □

Theorem 3.4.8. *Let $X^0 \in \mathbb{I}D_x$ be such that there exists a locally unique solution $\mathbf{x}(\mathbf{p}) \in X^0$ for every $\mathbf{p} \in P^1 \in \mathbb{I}D_p$ with $J_{\mathbf{x}}(X^0, P^1)$ nonsingular. Let $\{P^l\} \in \mathbb{I}P^1$ define a nested sequence of parameter intervals such that $\cap_{l=1}^{\infty} P^l = [\hat{\mathbf{p}}, \hat{\mathbf{p}}]$. Let $\{X^k\}$ be the nested sequence of intervals generated by the parametric interval method,*

Def. 3.3.6, with $\Phi = N$ as in Definition 3.3.3, using finite-precision rounded interval arithmetic starting at the initial interval X^0. Then $\lim_{l\to\infty}\lim_{k\to\infty}\Phi(\mathbf{x}^k, X^k, P^l) = \lim_{k\to\infty}\lim_{l\to\infty}\Phi(\mathbf{x}^k, X^k, P^l) = [\mathbf{x}(\hat{\mathbf{p}}), \mathbf{x}(\hat{\mathbf{p}})]$.

Proof. Let $G^k(P^l) = X^k = \Phi(\mathbf{x}^{k-1}, X^{k-1}, P^l)$ and let ϵ_{tol} be the precision to which intervals are rounded. Consider $\lim_{k\to\infty}\lim_{l\to\infty} G^k(P^l)$. Since $J_{\mathbf{x}}(X^0, P)$ is nonsingular, $m(J_{\mathbf{x}}(X^0, P))$ is nonsingular. Therefore $m(J_{\mathbf{x}}(X^k, P))^{-1}J_{\mathbf{x}}(X^k, P)$ is nonsingular for every k from Lemma 3.4.6. Furthermore, from the same Lemma, the diagonal elements of $m(J_{\mathbf{x}}(X^k, P))^{-1}J_{\mathbf{x}}(X^k, P)$ do not contain 0. From Lemma 3.4.7, N is continuous. It is clear from continuity of $N(\mathbf{x}^k, X^k, \cdot\,)$ on $\mathbb{I}D_p$ that $\lim_{k\to\infty}\lim_{l\to\infty} G^k(P^l) = \lim_{k\to\infty} G^k(\hat{\mathbf{p}})$, which is a simple reduction of the parametric case to the non-parametric case. Since $J_{\mathbf{x}}(X^0, \hat{\mathbf{p}})$ is nonsingular, we have $m(J_{\mathbf{x}}(X^k, \hat{\mathbf{p}}))^{-1}J_{\mathbf{x}}(X^k, \hat{\mathbf{p}})$ nonsingular and its diagonals do not contain 0. Then, from Theorem 5.2.6 in [101], $G^k(\hat{\mathbf{p}}) \to G^*(\hat{\mathbf{p}}) = \Phi(\mathbf{x}^*, X^*(\hat{\mathbf{p}}), \hat{\mathbf{p}}) = X^*(\hat{\mathbf{p}}) = [\mathbf{x}(\hat{\mathbf{p}}), \mathbf{x}(\hat{\mathbf{p}})] = \mathbf{x}(\hat{\mathbf{p}})$. By Theorem 3.2.18, since $\{G^k(P^l)\}$ is a nested sequence of intervals, it is convergent. Since $\mathbb{IR}^{n_x}$ is a complete metric space by Theorem 3.2.23, $\{G^k(P^l)\}$ is a Cauchy sequence in $\mathbb{IR}^{n_x}$. Thus, for every $\epsilon_{\text{tol}} > 0$, there exists an M such that for every $n, m \geq M$, $P^l \in \mathbb{I}D_p$ we have $d_H(G^n(P^l), G^m(P^l)) < \epsilon_{\text{tol}}$. Therefore, by Theorem 7.8 in [117], $\{G^k(P^l)\}$ converges uniformly on $\mathbb{I}D_x$. It follows directly that $\lim_{l\to\infty}\lim_{k\to\infty} G^k(P^l) = \lim_{k\to\infty}\lim_{l\to\infty} G^k(P^l) = \mathbf{x}(\hat{\mathbf{p}})$. □

An analogous result holds for the parametric interval method using the K operator.

Theorem 3.4.9. *Let $X^0 \in \mathbb{I}D_x$ be such that there exists unique solution $\mathbf{x}(\mathbf{p}) \in X^0$ for every $\mathbf{p} \in P^1 \in \mathbb{I}D_p$. Let $\{P^l\} \in \mathbb{I}P$ define a nested sequence of parameter intervals such that $\cap_{l=1}^{\infty} P^l = [\hat{\mathbf{p}}, \hat{\mathbf{p}}]$. Let $\{X^k\}$ be the nested sequence of intervals generated by the parametric interval method, Def. 3.3.6 with $\Phi = K$ as in Definition 3.3.5, using finite-precision rounded interval arithmetic with $\mathbf{x}^k = m(X^k)$. Let $\lambda_{max} = \max_i\{|\lambda_i|\}$ be the magnitude of the extremal eigenvalue(s) of the matrix $|\mathbf{Y}^0 J_{\mathbf{x}}(X^0, P) - \mathbf{I}|$. If $\lambda_{max} < 1$, then* $\lim_{l\to\infty}\lim_{k\to\infty}\Phi(\mathbf{x}^k, X^k, P^l) = \lim_{k\to\infty}\lim_{l\to\infty}\Phi(\mathbf{x}^k, X^k, P^l) = [\mathbf{x}(\hat{\mathbf{p}}), \mathbf{x}(\hat{\mathbf{p}})]$.

Proof. Let $G^k(P^l) = X^k = \Phi(\mathbf{x}^{k-1}, X^{k-1}, P^l)$ and let ϵ_{tol} be the precision to which intervals are rounded. Similar to Theorem 3.4.8, from Lemma 3.4.6, A^k in (3.10) is nonsingular for each k and its diagonal elements do not contain 0. Therefore by Lemma 3.4.7, we have continuity of K on P. It immediately follows that $\lim_{k\to\infty}\lim_{l\to\infty} G^k(P^l) = \lim_{k\to\infty} G^k(\hat{\mathbf{p}})$, which is the reduction to the non-parametric case. Since $J_{\mathbf{x}}(X^0, \hat{\mathbf{p}})$ is nonsingular and $\lambda_{max} < 1$, from Theorem 5.2.2 in [101], it follows that $G^k(\hat{\mathbf{p}}) \to G^*(\hat{\mathbf{p}}) = K(\mathbf{x}^*, X^*(\hat{\mathbf{p}}), \hat{\mathbf{p}}) \cap X^*(\hat{\mathbf{p}}) = X^*(\hat{\mathbf{p}}) = [\mathbf{x}(\hat{\mathbf{p}}), \mathbf{x}(\hat{\mathbf{p}})] = \mathbf{x}(\hat{\mathbf{p}})$. In an analogous argument to Theorem 3.4.8, it follows that $\lim_{l\to\infty}\lim_{k\to\infty} G^k(P^l) = G^*(\hat{\mathbf{p}}) = \mathbf{x}(\hat{\mathbf{p}})$. □

3.5 Bounding All Solutions of Parameter-Dependent Nonlinear Systems of Equations

An algorithm for bounding all real solution branches of parameter-dependent nonlinear systems of equations is presented in this section. In order to streamline the presentation of the algorithm, the classical existence and uniqueness test will be implemented as the following subroutine.

Subroutine 3.5.1 (`incTest`)**.**

```
incTest(Z, Φ){
 for  i = 1, ..., n_x do
    if (Φ_i = ∅) then
        return I_flag := −1
    elseif (φ_i^L ≤ z_i^L or φ_i^U ≥ z_i^U) then
        return I_flag := 0 endif
 end
 return I_flag := 1
}
```

For the algorithm below, the function Φ will be as in Definition 3.3.6.

Algorithm 3.1 (Parameterized Generalized Bisection)**.**

1. **(Initialization)**

(a) Pick box (X^0, P^0)*, initialize solution set* $\Xi = \emptyset$ *and stack* $\mathscr{S} = \{(X^0, P^0)\}$*, set* $l := 0$*.*

2. **(Termination)**

(a) Check stack. Stack empty? ($\mathscr{S} = \emptyset$*?)*

i. Yes. Algorithm terminates.

ii. No. Pop and delete a box (Z, P) *from stack* $\mathscr{S}$*, set* $I_{flag} := 0$*,* $l := l+1$*.*

(b) If $\mathbf{0} \in H(Z, P)$*, go to 4. Else, go to 2 (*(Z, P) *has been fathomed).*

3. **(Refinement)**

(a) Apply parametric interval Newton-type method iteratively with $Z^0 = Z$*.*

i. If at any point in any iteration k*,* $\Phi(\mathbf{z}^k, Z^k, P) = Z^L \cup Z^R$ *with* $Z^L \cap Z^R = \emptyset$ *(by extended interval arithmetic), place* (Z^L, P) *and* (Z^R, P) *on* $\mathscr{S}$*. Go to 2.*

ii. At every iteration k*, if* $\Phi(\mathbf{z}^k, Z^k, P)$ *is not an unbounded interval (by extended interval arithmetic),* $I_{flag} :=$ `incTest`$(Z^k, \Phi(\mathbf{z}^k, Z^k, P))$*.*

A. If $I_{flag} = -1$*, go to 2,* (Z^k, P) *has been fathomed.*

iii. At iteration k*, if* $\Phi(\mathbf{z}^k, Z^k, P) = Z^k$ *and* $I_{flag} = 0$*, go to 4. Else if* $I_{flag} = 1$ *place* (Z^k, P) *in solution set* Ξ*, go to 2.*

4. **(New Existence and Uniqueness Test)**

(a) If any bounds of Z^k *are not improved from* Z^0*, apply Theorem 3.4.1 at each of the unimproved bounds in order to verify that no solutions intersect these bounds. If this cannot be guaranteed, go to 6.*

(b) Calculate $\mathbf{Y}^{-1} := m(J_{\mathbf{z}}(Z^k, P))$*. If nonsingular, continue. Else, go to 5.*

(c) *Calculate* λ_{max}*, the maximum eigenvalue of* $|\mathbf{Y}|rad(J_{\mathbf{z}}(Z^k, P))$*. If* $\lambda_{max} <$ 1 *continue. Else, go to 5.*

(d) *Choose* $\bar{\mathbf{p}} \in P$ *and check* $\Phi(\mathbf{z}^k, Z^k, \bar{\mathbf{p}}) \subset \text{int}(Z^k)$*. If true, set* $I_{flag} = 1$*, place* (Z^k, P) *in solution set* Ξ *and go to 2. Else, go to 5.*

5. **(Partition** Z**)**

(a) *Partition* Z *using some strategy to avoid creating partial enclosures and add resulting boxes to the stack* $\mathscr{S}$*.*

(b) *If no such partition can be found, go to 6.*

6. **(Partition** P**)**

(a) *Partition* P *using some strategy. Add resulting boxes to the stack* $\mathscr{S}$*.*

The algorithm was designed such that the new existence and uniqueness test (Step *4*) is only applied once the interval iteration converges but it is not known whether or not the interval contains solution branches. This is because the classical exclusion test and existence and uniqueness test is applied naturally at each iteration with no additional computational cost. Only when the iteration can no longer improve the interval in question and it is still not known if it encloses a solution branch, is the sharper existence and uniqueness test called. It is expected that this strategy is more efficient than applying the sharper test at each iteration since intuitively, it is more likely to pass on the smaller, refined intervals.

It should be noted that Step *4(a)* is required in order to apply Theorem 3.4.4 properly. In that step, it is only required to apply Theorem 3.4.1 at each of the bounds that didn't improve during the interval iteration, if there were any. This is because Corollary 3.4.3 guarantees that if a bound happens to improve during the interval iteration, no solution could have intersected it. Therefore, Corollary 3.4.3 is applied implicitly within Algorithm 3.1.

3.5.1 Partitioning Strategies

Steps 5 and 6 of Algorithm 3.1 were stated generically in order to emphasize the possibility of the user supplying any appropriate partitioning strategy. In this section, the strategies implemented in the algorithm and used on the examples will be discussed. First, the strategy for partitioning the X space is discussed.

X Partitioning Strategies

The intuitive picture of partitioning the X space is to take a box that may include multiple solution branches and generate new boxes that are likely to contain either no solution branches or a single solution branch. In other words, the objective of partitioning X is to separate solution branches into their own boxes. This is essentially the standard bisection strategy, except with one subtlety: the X interval is cut such that the newly generated intervals either contain solution branches on all of P or no solution branches, excluding the partial enclosure scenario altogether. Since bisecting X at the midpoint of the i^{th} component will likely produce a partial enclosure scenario, this strategy cannot be employed blindly, which is common in classical generalized bisection algorithms. The simplest way to avoid making a cut that generates partial enclosures is to search for a cut position that satisfies Theorem 3.4.1. The strategy that was implemented in this thesis determines a component with the maximum width $i \in \arg\max_j w(X_j)$ and searches for a $\nu \in (0,1)$, if one exists, such that

$$\tilde{x}_i := \nu \left(x_i^L + \frac{rad(X_i)}{\mu} \right) + (1-\nu) \left(x_i^U - \frac{rad(X_i)}{\mu} \right) \tag{3.12}$$

satisfies Theorem 3.4.1, with $\mu \in \{m \in \mathbb{R} : m > 1\}$. That is, $\tilde{x}_i$ is chosen such that it does not intersect a solution branch, guaranteed by Theorem 3.4.1, and therefore avoiding a partial enclosure scenario. It is clear that the purpose of ν in (3.12) is to easily evaluate many candidate values of $\tilde{x}_i$ that are the convex combination of some relevant interval bounds. The parameter μ is included to give the user freedom over the interval from which $\tilde{x}_i$ can be chosen. It is a parameter that determines how much of the interior of X_i should be considered as containing a possible bisection position.

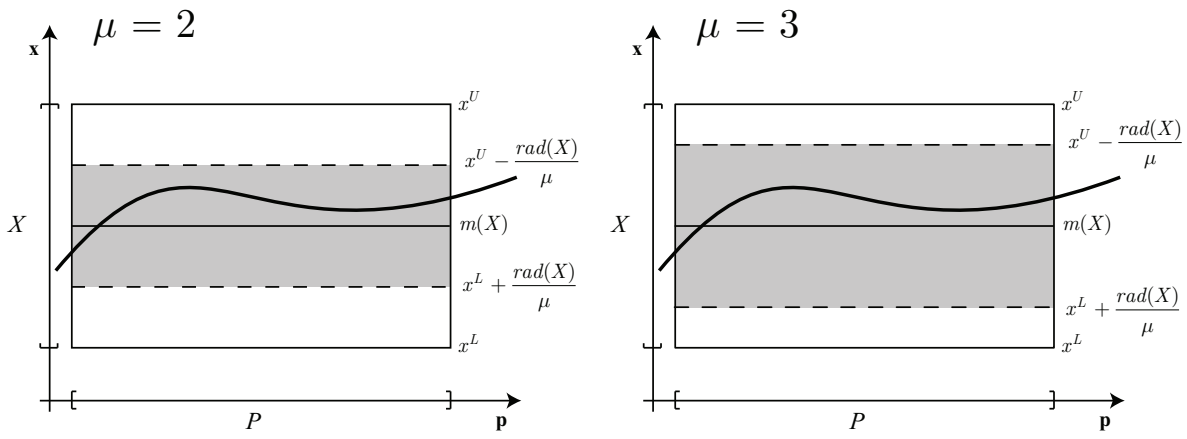

Figure 3-1: The effects of μ on the region considered for partitioning X in one dimension.

The value for μ can be chosen freely and *tuned* according to the system being bounded. As μ gets very large, the interval of candidate cut positions approaches X_i and thus there is a larger potential to find bisections very close to the original bounds of X_i therefore producing a very narrow interval and a wide interval nearly the width of X_i. Although this is conservative and poses no theoretical problems for the algorithm, it may be quite inefficient. Alternatively, values of μ close to 1 can pose problems for the algorithm since they limit the candidate $\tilde{x}_i$ values to very close to the midpoint of X_i. In general, it will be very difficult to find proper partitions of X_i that avoid partial enclosures. The effect of μ is illustrated in Figure 3-1 and the implementation of this strategy is discussed in Section 3.6.

If a partition for X_i cannot be identified, again with $i \in \arg\max_j(w(X_j))$ the strategy is to increase the μ parameter value and begin searching for a partition in all components of X. If a partition for X still cannot be identified, it is determined that the P interval must be partitioned. Figure 3-2(a) illustrates a scenario in which an interval X encloses multiple solution branches but there does not exist a partition (candidate positions represented as dashed lines) that separates them while avoiding the partial enclosure scenario. Figure 3-2(b) illustrates how, after finding a position to partition P, there exists partitions of X so that the solution branches are separated and no partial enclosures are generated. The next section discusses a strategy for

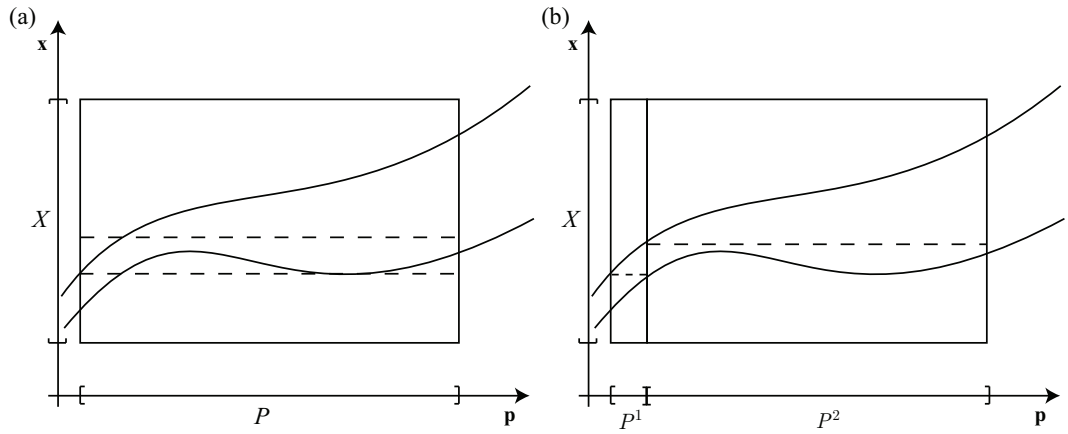

Figure 3-2: (a) A box $X \times P$ in which there does not exist a position to cut X (dashed lines) such that no partial enclosures are produced. (b) After cutting P, there exists positions to cut X avoiding partial enclosures.

cutting P.

P Partitioning Strategies

Due to the width of the parameter interval P, verifying existence and uniqueness of enclosed solution branches may be difficult or impossible, as illustrated in Fig. 3-2, even with the sharper existence and uniqueness test of Theorem 3.4.4. Therefore, a strategy for partitioning P must also be considered. An efficient strategy is to simply bisect down the middle of the widest dimension. The problem with this strategy is that, in general, each parameter may not contribute equally to overestimation and the inability to pass the existence and uniqueness test of Theorem 3.4.4. Therefore, it is desirable to have a method for determining which parameter dimension has the largest influence on the extremal eigenvalue(s) of $\mathbf{A}$, as defined in Theorem 3.4.4. With such information, the parameter interval P can be partitioned in an intelligent manner with the objective of generating boxes that pass the existence and uniqueness test of Theorem 3.4.4.

By treating λ_{max} of Theorem 3.4.4 as a real-valued function of the bounds of X and P, if one were able to calculate *pseudo*-derivative information of λ_{max} with respect to the bounds of P, the influence of the parameter interval on passing the existence and uniqueness test of Theorem 3.4.4 presents itself. Furthermore, this

information can be used to provide intelligent positions to cut P that will yield boxes that are more likely to pass the existence and uniqueness test of Theorem 3.4.4. As was illustrated in Figure 3-2(b), how one cuts P may have a large impact on how the X box is subsequently partitioned and on the ability to pass existence and uniqueness tests. The following results establish how this pseudo-derivative information can be calculated and how it is used in the partitioning strategy.

Definition 3.5.2 (Piecewise Continuous Differentiable [40])**.** A continuous function $\mathbf{g} : D \subset \mathbb{R}^n \to \mathbb{R}^m$ is said to be *piecewise continuously differentiable* near $\mathbf{z} \in D$ if there exists an open neighborhood $N \subset D$ of $\mathbf{z}$ and a finite family of continuously differentiable functions $\mathbf{g}^1, \mathbf{g}^2, \ldots, \mathbf{g}^k : N \to \mathbb{R}^m$, for $k \in \mathbb{N}$ $(k > 0)$, such that $\mathbf{g}(\mathbf{y})$ is an element of $\{\mathbf{g}^1(\mathbf{y}), \mathbf{g}^2(\mathbf{y}), \ldots, \mathbf{g}^k(\mathbf{y})\}$ for all $\mathbf{y} \in N$.

Definition 3.5.3 ([127])**.** Let $D \subset \mathbb{IR}^{n\times n}$ be open and let $\mathbf{F} : D \to \mathbb{R}^{n\times n}$. $\mathbf{F}$ will be called piecewise continuous differentiable on D if for every piecewise continuous differentiable function $M : E \subset \mathbb{R}^{2nn} \to \mathbb{IR}^{n\times n}$, the mapping $\mathbf{F}(M(\cdot)) : E \to \mathbb{R}^{n\times n}$ is piecewise continuous differentiable on the open set $E_D = \{\mathbf{a} \in E : M(\mathbf{a}) \in D\}$.

Definition 3.5.4 ([127])**.** Let $D \subset \mathbb{R}^{n\times n}$ be open and let $f : D \to \mathbb{R}$. f will be called piecewise continuous differentiable on D if for every piecewise continuous differentiable function $\mathbf{M} : E \subset \mathbb{R}^{nn} \to \mathbb{R}^{n\times n}$, the mapping $f(\mathbf{M}(\cdot)) : E \to \mathbb{R}$ is piecewise continuous differentiable on the open set $E_D = \{\mathbf{a} \in E : \mathbf{M}(\mathbf{a}) \in D\}$.

Lemma 3.5.5. *Let $D_A \subset \mathbb{IR}^{n\times n}$ be open such that for every $A \in D_A$, $m(A)$ is nonsingular. Define the real matrix-valued function $\mathbf{Z} : D_A \to \mathbb{R}^{n\times n}$ as $\mathbf{Z}(A) \equiv |mid(A)^{-1}|\, rad(A), \quad \forall A \in D_A$. Then $\mathbf{Z}$ is piecewise continuously differentiable on D_A.*

Proof. By definition, $rad(\cdot)$ and $m(\cdot)$ are continuously differentiable on $\mathbb{D}_A$. Since $m(A)$ is nonsingular $\forall A \in D_A$, $m(\cdot)^{-1}$ exists and can be expressed using only elementary arithmetic operations, and is therefore continuously differentiable on D_A. By definition, $|\cdot|$ is piecewise continuously differentiable on D_A. It immediately follows that $\mathbf{Z}$ is piecewise continuously differentiable on D_A. □

Assumption 3.5.6. Let $D \subset \mathbb{R}^{n\times n}$ be open. Let $\mathbf{Z}$ and D_A be as in Lemma 3.5.5 such that $\mathbf{Z}(A) \in D$ for every $A \in D_A$. Let $\lambda_i(\mathbf{Z}(A))$ be the i^{th} real eigenvalue of $\mathbf{Z}(A)$. It will be assumed that $\lambda_i : D \to \mathbb{R}$ is piecewise continuously differentiable on D.

Theorem 3.5.7. *Suppose Assumption 3.5.6 holds. Define the function* $\lambda_{max} : \mathbb{R}^{n\times n} \to \mathbb{R}$ *as* $\lambda_{max}(\mathbf{Z}(A)) = \max_i\{|\lambda_i(\mathbf{Z}(A))|\}$, *the magnitude of the extremal eigenvalue(s) of* $\mathbf{Z}(A)$, $\forall A \in D_A$. *Then* λ_{max} *is differentiable on* D *at all points outside of a Lebesgue nullset.*

Proof. By Assumption 3.5.6, λ_i is piecewise continuously differentiable on D. Therefore, λ_i is locally Lipschitz by Corollary 4.1.1 in [126]. By Proposition 4.1.2 in [126], λ_{max} is locally Lipschitz. From Theorem 3.1.1 in [40], λ_{max} is differentiable on D at all points outside of a Lebesgue nullset. □

From Theorem 3.5.7, λ_{max} is differentiable almost everywhere, provided λ_i is piecewise continuously differentiable on D. The Lipschitz result of λ_{max} may be too restrictive since, in general, λ_{max} is not Lipschitz for non-symmetric matrices. However pseudo-derivative information may still be available since it may be possible to calculate subgradient information of λ_{max} (e.g. see [23]). Now, λ_{max} will be defined more precisely as a function $\lambda_{max} : D \subset \mathbb{R}^{n_x\times n_x} \to \mathbb{R}$ with $\lambda_{max}(\mathbf{A}(X^l, P^l))$ as the extremal eigenvalue(s) of the matrix $\mathbf{A}(X^l, P^l) \equiv |m(J_{\mathbf{x}}(X^l, P^l))^{-1}|rad(J_{\mathbf{x}}(X^l, P^l))$ with $(X^l, P^l) \in \mathbb{I}D_x \times \mathbb{I}D_p$. By expressing the parameter space P as the real vector $\mathbf{p}^B = (p_1^L, p_1^U, \ldots, p_{n_p}^L, p_{n_p}^U)^{\mathrm{T}}$, the gradient vector of λ_{max} with respect to $\mathbf{p}^B$ evaluated at $\mathbf{A}(X^l, P^l)$, if it is defined, can be expressed as

$$\nabla_{\mathbf{p}^B}\lambda_{max}(\mathbf{A}(X^l, P^l)) = \left(\frac{\partial\lambda_{max}}{\partial p_1^L}(\mathbf{A}(X^l, P^l)), \frac{\partial\lambda_{max}}{\partial p_1^U}(\mathbf{A}(X^l, P^l)), \right.$$
$$\left. \ldots, \ldots, \frac{\partial\lambda_{max}}{\partial p_{n_p}^L}(\mathbf{A}(X^l, P^l)), \frac{\partial\lambda_{max}}{\partial p_{n_p}^U}(\mathbf{A}(X^l, P^l))\right).$$

Computationally, λ_{max} can be approximated using a fixed-point iteration such as the Arnoldi iteration or the power iteration which is how the implementation of

the algorithm calculates it. Furthermore, $\nabla_{\mathbf{p}^B}\lambda_{max}$ can be evaluated using forward automatic differentiation [50], which was performed for this chapter using an in-house `C++` library. The following procedure defines the strategy for partitioning P.

Subroutine 3.5.8.

1. *For a box* $(X^l, P^l) \in \mathbb{I}X^0 \times \mathbb{I}P^0$, *evaluate* $\nabla_{\mathbf{p}^B}\lambda_{max}(\mathbf{A}(X^l, P^l))$.

2. *Choose the component of* $\nabla_{\mathbf{p}^B}\lambda_{max}$ *with the largest magnitude and determine its corresponding component of* P^l. *Store this component as* j_{max}. *If* $w(P^l_{j_{max}}) < \epsilon_P$, $P^l_{j_{max}}$ *is too narrow to be cut. Choose the component with the next largest magnitude and repeat until a* j_{max} *is found with* $w(P^l j_{max}) > \epsilon_P$. *If no such* j_{max} *exists, terminate.* P *cannot be partitioned further.*

3. *Let* $\mathbf{p}_{j_{max}} = \left(p^{l,L}_{j_{max}}, p^{l,U}_{j_{max}}\right)$ *and* $\mathbf{d} = \left(\left(\nabla_{\mathbf{p}^B}\lambda_{max}(\mathbf{A}(X^l, P^l))\right)_{2j_{max}-1}, \left(\nabla_{\mathbf{p}^B}\lambda_{max}(\mathbf{A}(X^l, P^l))\right)_{2j_{max}}\right)$ *and take a steepest-descent step* $\mathbf{p}^{new}_{j_{max}} := \mathbf{p}_{j_{max}} - \alpha\mathbf{d}$ *with* $\alpha > 0$.

4. *Check if* $(\mathbf{p}^{new}_{j_{max}})_1 < (\mathbf{p}^{new}_{j_{max}})_2$ *and that* $(\mathbf{p}^{new}_{j_{max}})_1$ *and* $(\mathbf{p}^{new}_{j_{max}})_2$ *are separated at least by* ϵ_P. *If so, continue, else bisect* $P^l_{j_{max}}$ *down the middle and return two new boxes to the main algorithm to be placed on the stack.*

5. *Set* $P^*_i := P_i$ *for* $i \neq j_{max}$ *and* $P^*_{j_{max}} := [(\mathbf{p}^{new}_{j_{max}})_1, (\mathbf{p}^{new}_{j_{max}})_2]$. *Check if* $\lambda_{max}(\mathbf{A}(X^l, P^*)) < 1.5$ *and* $w(P^*_{j_{max}}) > \max\{\frac{1}{3}w(P^l_{j_{max}}), \frac{1}{4}w(P^0_{j_{max}})$. *If true, partition* $P^l_{j_{max}}$ *at* $(\mathbf{p}^{new}_{j_{max}})_1$ *and* $(\mathbf{p}^{new}_{j_{max}})_2$ *and return three new boxes to the main algorithm to be placed on the stack.*

6. *Set* $P^l_{j_{max}} := [(\mathbf{p}^{new}_{j_{max}})_1, (\mathbf{p}^{new}_{j_{max}})_2]$ *and go to step 3.*

Both the standard partitioning strategy of bisection in the widest dimension as well as that of Subroutine 3.5.8 were implemented as part of Algorithm 3.1.

3.6 Implementation and Numerical Examples

In this section, the implementation of the algorithm and its performance in solving a number of numerical examples is discussed.

3.6.1 Computer Implementation

X Partitioning The strategy for partitioning X was implemented in two stages, as described in Section 3.5.1. In the first stage, a value of $\mu = 4$ is set and ν is incremented from 0 to 0.5 in 20 steps until a safe bisection position in the widest dimension is found, using Theorem 3.4.1. If no safe bisection position can be found, ν is decremented from 1 to 0.5 until a safe bisection position is found. If no safe bisection position can be found, the strategy enters the second stage. In the second stage, a value of $\mu = 16$ is set and the same strategy is followed as previously except using 80 steps of ν from 0 to 0.5 and 80 from 1 to 0.5. The idea behind this implementation is that the first stage is a coarse, less computationally expensive procedure, while the second stage is a much finer, more computationally expensive search for a safe bisection position.

P Partitioning The strategy for partitioning the parameter space was implemented as Subroutine 3.5.8. For Subroutine 3.5.8, j_{max} is chosen according to Step 2 such that $w(P^l_{j_{max}}) > 0.2w(P^0_{j_{max}})$ if such a dimension exists. Else, j_{max} is taken to be the widest dimension of P^l such that $w(P^l_{j_{max}}) > 1E-3$, if such a dimension exists. Using a value of $\alpha = 0.9$, steps 3-6 are iterated. The procedure stops at Step 5 if the following conditions are met: $\lambda_{max} < 1.5$ and $(\mathbf{p}^{new}_{j_{max}})_2 - (\mathbf{p}^{new}_{j_{max}})_1 > \max\{\frac{1}{3}w(P^l_{j_{max}}), \frac{1}{4}w(P^0_{j_{max}})$. The purpose of this is an attempt at seeking out a partition that is sufficiently close to passing the new existence and uniqueness test while ensuring that it has substantial width. If these conditions are not met, the procedure stops at Step 4 and $P^l_{j_{max}}$ is simply bisected.

Algorithm 3.1 was applied to each of the following numerical examples. For each example, the performance of the algorithm was compared between each interval method (interval-Newton or Krawczyk) as well as how the parameter interval was partitioned (standard bisection or Subroutine 3.5.8). Each variation of the algorithm will be denoted AΦ-p where $\Phi \in \{N, K\}$ with N denoting the interval-Newton method and K denoting the Krawczyk method, and p$\in \{1, 2\}$ where 1 denotes the partitioning scheme of Subroutine 3.5.8 and 2 denotes the strategy of bisecting the

Algorithm Efficiency (sec.)				
Ex.	AN-1	AN-2	AK-1	AK-2
1	8.70E-3	8.00E-3	8.89E-3	8.04E-3
2	2.12E-3	2.13E-3	2.38E-3	2.41E-3
3	6.58E-3	6.90E-3	4.81E-1	4.83E-1
4	2.22E-2	5.95E-3	2.39E-2	7.35E-3
5	8.87E-2	1.18E-1	1.23E-1	1.53E-1

Table 3.1: The performance of the algorithm in terms of solution time for each example.

Algorithm Efficiency (iterations)				
Ex.	AN-1	AN-2	AK-1	AK-2
1	141	169	130	131
2	1	1	1	1
3	55	55	329	329
4	270	270	295	295
5	1993	2392	2189	2599

Table 3.2: The performance of the algorithm in terms of the number of iterations taken to solve each example.

widest dimension. The performance of each variation of the algorithm on solving the examples below is summarized in Tables 3.1, 3.2, and 3.3 in terms of the overall computation time, the number of iterations (or the number of intervals examined), and the number of interval boxes produced that cover the solution set, respectively.

Algorithm Efficiency (box count)				
Ex.	AN-1	AN-2	AK-1	AK-2
1	4	5	4	3
2	1	1	1	1
3	1	1	1	1
4	15	15	15	15
5	58	80	58	80

Table 3.3: The performance of the algorithm in terms of the number of interval boxes produced to enclose all locally unique solution branches.

3.6.2 Numerical Examples

Example 3.6.1. Consider

$$\mathbf{h}(\mathbf{z},\mathbf{p}) = \begin{pmatrix} z_1^2 + z_2^2 + p_1 z_1 + 4 \\ z_1 + p_2 z_2 \end{pmatrix} = \mathbf{0}$$

with $P^0 = [5,7]^2$ and $X^0 = [-10,10] \times [-2,2]$. There are two solution branches for this problem.

Example 3.6.2. Taken from Kolev *et al.* [75]:

$$\mathbf{h}(\mathbf{z},\mathbf{p}) = \begin{pmatrix} (3.25 - z_1)/p_1 - z_3 \\ z_1/p_2 - z_3 \\ z_2 - z_1^2/(1+z_1^2) \end{pmatrix} = \mathbf{0}$$

with $P^0 = [1800,2200] \times [900,1100]$ and $X^0 = [-30,30]^3$. There is one solution branch for this system.

Example 3.6.3. Taken from Kolev *et al.* [75], this example models an electric circuit having two resistors, a transistor, and a diode:

$$\mathbf{h}(\mathbf{z},\mathbf{p}) = \begin{pmatrix} 10^{-9}(\exp 38 z_1 - 1) + p_1 z_1 - 1.6722 z_2 + 0.6689 z_3 - 8.0267 \\ 1.98 \times 10^{-9}(\exp 38 z_2 - 1) + 0.6622 z_1 + p_2 z_2 + 0.6622 z_3 + 4.0535 \\ 10^{-9}(\exp 38 z_3 - 1) + z_1 - z_2 + p_3 z_3 - 6 \end{pmatrix} = \mathbf{0}$$

with $P^0 = [0.6020, 0.7358] \times [1.2110, 1.4801] \times [3.6, 4.4]$ and $X^0 = [-30,30]^3$. There is one solution branch for this system.

Example 3.6.4. Consider

$$h(z,p) = -z^3 + pz = 0,$$

with $P^0 = [0.25, 20]$ and $X^0 = [-10,10]$. This system has three solution branches. Although this problem has simple analytical solutions of $x(p) = 0, \pm\sqrt{p}$, it illustrates

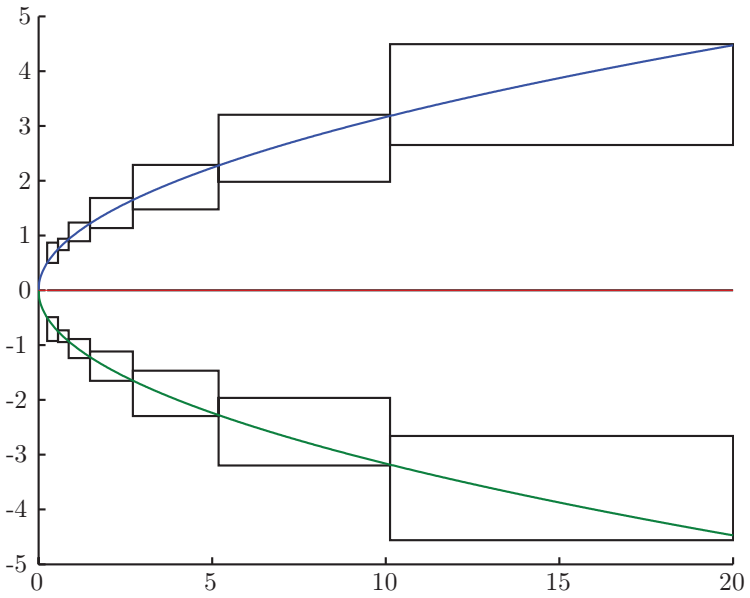

Figure 3-3: The three solutions of Ex. 3.6.4 and the interval boxes computed by algorithm AN-1 for $P^0 = [0.25, 20]$. It should be noted that the red solution branch (middle) has an interval box enclosing it but it is exact within machine precision.

the parameterized generalized bisection algorithm nicely. The result of algorithm AN-1 is shown in Figure 3-3. In order to demonstrate how the algorithm handles bifurcation points, the parameter interval $P^0 = [0, 20]$ was also considered. The result of algorithm AN-1 applied to the larger P is shown in Figure 3-4.

Example 3.6.5. This example originally appeared in [55] without parameter dependence. Parameter dependence was added for the purposes of this chapter:

$$\mathbf{h}(\mathbf{z}, \mathbf{p}) = \begin{pmatrix} p_1 z_1^5 - 25.2 z_1^3 + 6 p_1 z_1 - p_2 z_2 \\ 2 p_2 z_2 - p_1 z_1 \end{pmatrix} = \mathbf{0}$$

with $P^0 = [3, 5] \times [4.25, 7.75]$ and $X^0 = [-10, 10]^2$. There are five solution branches for this system.

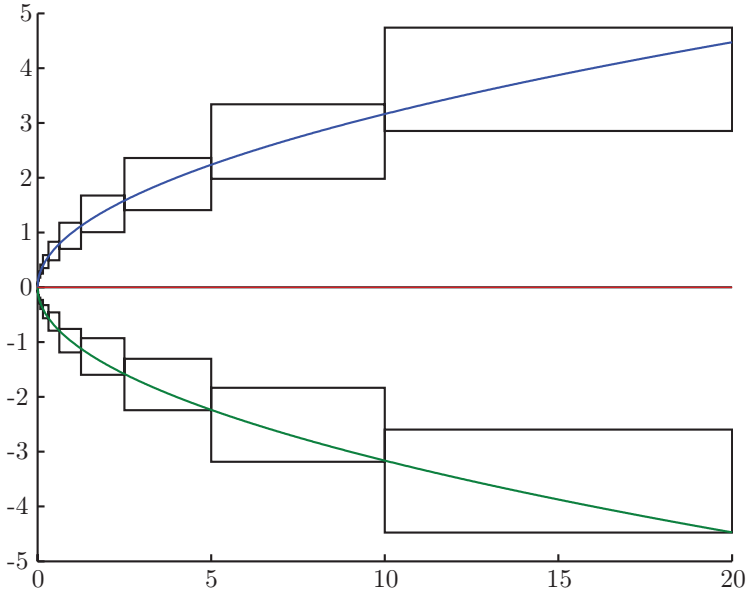

Figure 3-4: The three solutions of Ex. 3.6.4 and the interval boxes computed by algorithm AN-1 for $P^0 = [0, 20]$. It should be noted that the red solution branch (middle) has an interval box enclosing it but it is exact within machine precision.

3.7 Concluding Remarks

A generalized bisection-type algorithm for bounding all real solutions of parameter-dependent systems of equations was presented that addresses Objective (1) listed at the end of Chapter 2. The algorithm makes use of a stronger, computationally verifiable existence and uniqueness result, as compared to the classical existence and uniqueness tests of the interval-Newton and Krawczyk methods. Unlike standard global root-finding algorithms based on the generalized bisection framework, the search space cannot be bisected naively. While considering a partitioning strategy for the X space, the idea of partial enclosures must be taken into account. That is, X can only be partitioned in such a way that yields new intervals that are either valid enclosures of solution branches or do not enclose any solution branches. In order to do this, a separate procedure for bisecting P was developed that intelligently seeks positions to partition P that yields new interval boxes that are more likely to pass existence and uniqueness tests, while also maintaining that the parameter inter-

vals should be as wide as possible. Coupling the two partitioning strategies yields a method for calculating valid enclosures of locally unique solution branches on entire parameter intervals of nontrivial width. These enclosures have specific importance in global optimization applications, such as in Chapter 4.

The performance of the algorithm was demonstrated by solving five numerical examples; three of which have multiple solution branches. In every case, the algorithm using the parametric interval-Newton method was the most efficient in terms of total running time. The examples in which P was partitioned (Ex. 1, 4, 5) allow one to compare the effectiveness of the partitioning strategy and therefore demonstrate the trade-off between computational efficiency and algorithm sophistication.

For Example 1, the implementations in which P was partitioned using Subroutine 3.5.8 (i.e. AN-1, AK-1) completed in fewer iterations than just using the bisection strategy (i.e. AN-2, AK-2). However, in each case, the algorithm had longer running times, due to the higher cost-per-iteration. Furthermore, it should be noted that AN-1 computes 4 interval boxes that are guaranteed to contain locally unique solution branches whereas AN-2 computes 5. Unexpectedly, AK-2 computes only 3 boxes whereas AK-1 computed 4. This demonstrates the effects of partitioning the X interval on the partitioning strategy for P.

Example 4 demonstrated that, for some problems, a standard bisection strategy for partitioning P can be a superior strategy as compared to Subroutine 3.5.8. In this example, AN-1 and AN-2 completed after the same number of iterations after computing 15 interval boxes. However, the running time of AN-1 was almost three times longer than AN-2. A similar result was observed for AK-1 and AK-2, with the running time of AK-1 being more than 2 times longer than that of AK-2. For this example, Subroutine 3.5.8 never terminates at Step 5, and since there is only one parameter dimension, each partitioning procedure yields the same partition positions. Therefore, both partitioning strategies yield identical results in terms of the total number of iterations taken by the algorithm and the number of boxes produced. Therefore, it is clear why the partitioning strategy of Subroutine 3.5.8 is a poor choice for this example.

Example 5 demonstrates that Subroutine 3.5.8 can be very effective and, for some examples, be the superior partitioning strategy as compared to standard bisection. In every metric, AN-1 outperformed AN-2, taking less time to complete, fewer iterations, and computing fewer boxes. A similar result was observed for AK-1 versus AK-2. This result suggests that although Subroutine 3.5.8 is computationally more expensive as compared to simply bisecting P, the pseudo-derivative information that it computes can be so useful to the algorithm that the overall computational effort can be (drastically) reduced.

One problem area in which the proposed algorithm may have issues is in the face of bifurcations. In this case, for some parameter value(s) the Jacobian matrix $\mathbf{J_x}$ is singular. Therefore, the existence and uniqueness tests can never be passed. In turn, the algorithm will continue to partition X and P producing boxes that enclose the bifurcation point to some predetermined precision, or minimum width. The implementation of Algorithm 3.1 will continue to partition X and P until P is sufficiently narrow and no safe bisection direction in X can be found. In this case, the algorithm labels the box as "indeterminate" and it gets deleted from the stack. Example 4 has a bifurcation at $p = 0$, and hence, for the purposes of testing the performance of the algorithm, an initial parameter interval of $P^0 = [0.25, 20]$ was considered. However, the interval $P^0 = [0, 20]$ was also considered. In the case of AN-1, with the minimum allowable width of P set to 1E-3, the algorithm completes in 1.88E-2 seconds after 460 iterations, producing 45 boxes enclosing locally-unique solution branches and one box $X^n \times P^n = [-0.0252, 0.0252] \times [0, 6.1\text{E-}4]$ enclosing the bifurcation point and small sections of each of the three solution branches originating from it. Similar results were observed for AN-2, AK-1, and AK-2. The question of how to deal with bifurcations more effectively is still an active area of study.

In the next chapter, the interval bounds of implicit functions will be used to construct convex and concave relaxations of implicit functions for use in a novel reduced-space global optimization algorithm.

Chapter 4

Global Optimization of Implicit Functions

In this chapter, an algorithm for solving nonconvex NLPs globally using a reduced-space approach is presented. These problems are encountered when real-world models are involved as equality constraints and the decision variables include the state variables of the system. By solving the model equations for the dependent (state) variables as implicit functions of the independent (decision) variables, a significant reduction in dimensionality can be obtained. As a result, the inequality constraints and objective function themselves are implicit functions of the independent variables, which can be estimated via a fixed-point iteration. Relying on the recently developed ideas of generalized McCormick relaxations and McCormick-based relaxations of algorithms and subgradient propagation, the development of McCormick relaxations of implicit functions is presented. Using these ideas, the reduced space, implicit optimization formulation can be relaxed directly. When applied within a branch-and-bound framework, finite convergence to ϵ-optimal global solutions is guaranteed.

4.1 Introduction

Nonconvex NLPs of the form:

$$\begin{aligned} \min_{\mathbf{y}\in Y\subset\mathbb{R}^{n_y}} & f(\mathbf{y}) \\ \text{s.t. } & \mathbf{g}(\mathbf{y}) \leq \mathbf{0} \\ & \mathbf{h}(\mathbf{y}) = \mathbf{0} \end{aligned} \tag{4.1}$$

can be formulated to solve a wide variety of problems from diverse disciplines ranging from operations research to engineering design. Local algorithms, such as sequential quadratic programming (SQP) [48], are not guaranteed to find the desired global optima. Thus deterministic global optimization algorithms such as branch-and-bound (B&B) [42] and branch-and-reduce [120] have been developed. However, all currently known deterministic global optimization algorithms suffer from worst-case exponential run time. Therefore, if the original program (4.1) can be reformulated as an equivalent program with reduced dimensionality, there is potential for a significant reduction in computational cost. The primary framework of the algorithm presented in this chapter is based on the B&B algorithm.

"Selective branching" strategies (i.e., where only a subset of the variables are branched on) have been developed due to the worst-case exponential run time for global optimization. The works [61, 62, 98, 105] all require very special problem structures that can be exploited to reduce the number of variables that are branched on. A more general reduced-space B&B approach was first introduced in [38]. Their work builds on the previous selective branching ideas. In the work of [38], the variables $\mathbf{y}$ are partitioned into two sets of variables and the nonconvex functions are factored according to which set of variables the factors are dependent upon. It is required that all functions of the first set of variables are convex and all functions of the second set of variables are continuous. Under some additional assumptions, convergence of the B&B algorithm is guaranteed while only branching on the second set of variables. The authors of [38] state that this type of factorization and partitioning is applicable to

most practical problems. However, the method was developed largely with inequality constraints in mind. The authors of [38] state that equality constraints can be handled using a pair of opposing inequality constraints. However, given the requirements of their algorithm for selective branching, it can be shown that this restricts the equality constraints that can be handled to parametric linear systems (i.e., (4.10)).[1] Therefore, general nonlinear systems of equations cannot be addressed. Furthermore, in [88], the authors compared their method of relaxing implicit functions directly with selective branching and experienced a significant performance benefit from relaxing implicit functions directly.

Consider the equality constraints of (4.1) as the system of equations:

$$\mathbf{h}(\mathbf{y}) = \mathbf{0}, \tag{4.2}$$

where $\mathbf{h} : D_y \to \mathbb{R}^{n_x}$ is continuously differentiable, with $D_y \subset \mathbb{R}^{n_y}$ open. Here, it is assumed that the vector $\mathbf{y} \in D_y$ can be separated into *dependent* and *independent* variables $\mathbf{z} \in \mathbb{R}^{n_x}$ and $\mathbf{p} \in \mathbb{R}^{n_p}$, respectively, with $\mathbf{y} = (\mathbf{z}, \mathbf{p})$ such that $\mathbf{h}$ can be solved for $\mathbf{z}$ in terms of $\mathbf{p}$, with $(\mathbf{z}, \mathbf{p}) \in D_y$. Under this assumption, (4.2) can be written as:

$$\mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{0}. \tag{4.3}$$

If for some n_p-dimensional interval $P \subset \mathbb{R}^{n_p}$, such $\mathbf{z}$ exist that satisfy (4.3) at each $\mathbf{p} \in P$, then they define an implicit function of $\mathbf{p}$, that will be expressed as $\mathbf{x}(\mathbf{p})$. Such a partition of the vector $\mathbf{y}$ is valid for many practical "real-world" problems. For instance, consider the original application with $\mathbf{h}$ as a steady-state model of a chemical process. The variables $\mathbf{z}$ would again correspond to the process state variables and $\mathbf{p}$ would correspond to the model parameters and/or parametric uncertainty. Unless otherwise stated, it will be assumed that for some $X \subset \mathbb{R}^{n_x}$, there exists at least one continuously differentiable implicit function $\mathbf{x} : P \to X$ such that $\mathbf{h}(\mathbf{x}(\mathbf{p}), \mathbf{p}) = \mathbf{0}$ holds for every $\mathbf{p} \in P$. Conditions under which $\mathbf{x}$ is unique in X are given by the so-called *semilocal implicit function theorem* [101] discussed in Chapter 3. Continuous

[1] This result is illustrated in Appendix A.

differentiability follows from the same result.

Just as $\mathbf{y}$ was partitioned into $(\mathbf{z}, \mathbf{p})$, the search space of the optimization problem (4.1) is partitioned as $Y = X \times P$. The program (4.1) may then be reformulated as the following program

$$\begin{aligned} &\min_{\mathbf{p}\in P} f(\mathbf{x}(\mathbf{p}), \mathbf{p}) \\ &\text{s.t. } \mathbf{g}(\mathbf{x}(\mathbf{p}), \mathbf{p}) \leq \mathbf{0}. \end{aligned} \tag{4.4}$$

It can readily be deduced that if $n_y - n_x$ is small ($n_x >> n_p$), the formulation (4.4) offers a significant reduction in dimensionality.

In order to solve (4.4) to global optimality with branch-and-bound, a method for calculating convex relaxations of $f(\mathbf{x}(\cdot), \cdot)$ and $\mathbf{g}(\mathbf{x}(\cdot), \cdot)$ on P is required. The major complication is that $\mathbf{x}$ is not known explicitly and may not even have a closed algebraic form, but can only be approximated using a fixed-point algorithm, for instance. Thus, the objective function, $f(\mathbf{x}(\cdot), \cdot)$, and the inequality constraint(s), $\mathbf{g}(\mathbf{x}(\cdot), \cdot)$, are implicitly defined and must be evaluated with embedded fixed-point iterations. Because of this, the involved functions no longer have a *factorable representation*. Therefore relaxation techniques that rely on explicit algebraic and/or factorable functions, such as standard McCormick relaxations [85] or αBB [2], are no longer applicable. However, if relaxations of the implicit function $\mathbf{x}$ were made available by some method, the functions f and $\mathbf{g}$ could be composed with them, using a generalization of the ideas of McCormick [128], and relaxations of f and $\mathbf{g}$ could be calculated.

In [88], Mitsos and coworkers laid the foundations for calculating relaxations of implicit functions $\mathbf{x}$ evaluated by an algorithm with a fixed number of iterations known *a priori*. They outline the automatic construction of McCormick convex/concave relaxations of factorable functions and automatic subgradient calculation. The automatic construction of McCormick relaxations and subgradient calculation was done using `libMC`, a predecessor of the currently available `C++` library `MC++` [26]. The types of algorithms considered in their work, however, only included algorithms in which the

number of iterations is known *a priori*, such as Gauss elimination, thus their methods are not applicable to problems in which $\mathbf{x}$ is evaluated by more general fixed-point algorithms, such as Newton's method.

In [128], the concept of *generalized* McCormick relaxations is presented. This generalization of McCormick relaxations allows for the application of McCormick relaxations to a much broader class of functions [128]. One focus of that paper was on the relaxation of the successive substitution fixed-point iteration. In relaxing the fixed-point iteration, the authors show that relaxations of the sequence of approximations of $\mathbf{x}$ could be calculated [128]. However, in order to relax f and $\mathbf{g}$ rigorously, valid relaxations of $\mathbf{x}$ are required, not of approximations of $\mathbf{x}$. This will be the primary focus of the theoretical developments contained in this chapter.

In the next section, the necessary background information will be discussed. In Section 4.3, new ideas and results involved in relaxing implicit functions and calculating subgradients are presented, followed by the global optimization algorithm in Section 4.4.

4.2 Background

This section contains the definitions and previously developed material from the literature required for the development of global optimization of implicit functions.

4.2.1 Fixed-Point Iterations

The term *fixed-point iteration* applies to a general class of iterative methods, for which the iteration count required to satisfy a given convergence tolerance is not typically known *a priori*. They are commonly employed to solve systems of equations such as (4.2). The general ideas are introduced here. For the focus of this thesis, fixed-point iterations will be used to evaluate the embedded implicit functions in (4.4). For a more in-depth look at these iterative methods, the reader is directed to [104].

Definition 4.2.1 (Fixed-Point)**.** Let $\mathbf{f} : Z \subset \mathbb{R}^m \to \mathbb{R}^m$. A point $\mathbf{z} \in Z$ is a fixed

point of $\mathbf{f}$ if $\mathbf{z} = \mathbf{f}(\mathbf{z})$.

An iteration will be referred to as a *fixed-point iteration* if it takes the form

$$\mathbf{z}^{k+1} := \boldsymbol{\phi}(\mathbf{z}^k), \quad k \in \mathbb{N},$$

with $\boldsymbol{\phi} : A \subset Z \to \mathbb{R}^m$. The name suggests that the iteration will be used to find a fixed-point of $\boldsymbol{\phi}$. However, this is ambitious in the sense that these iterations are not guaranteed to do so except under certain conditions. One such condition is if $\boldsymbol{\phi}$ is a *contraction mapping*.

Definition 4.2.2 (Contraction Mapping [117])**.** Let Z be a metric space with metric d. A function $\boldsymbol{\phi} : A \subset Z \to Z$ is said to be a contraction mapping or contractive on a set $B \subset A$ if $\boldsymbol{\phi}(B) \subset B$ and there exists an $\alpha \in (0, 1)$ such that

$$d(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y})) \leq \alpha d(\mathbf{x}, \mathbf{y}), \ \forall \mathbf{x}, \mathbf{y} \in B.$$

Definition 4.2.3 $(\mathbf{J}_\mathbf{x}, \nabla_\mathbf{x})$**.** Let $A \subset \mathbb{R}^m$ and $B \subset \mathbb{R}^n$ be open. Suppose $\mathbf{h} : A \times B \to \mathbb{R}^m$ is differentiable on $A \times B$. Then for each $\mathbf{b} \in B$, let $\mathbf{J}_\mathbf{x}(\mathbf{z}, \mathbf{b})$ denote the $m \times m$ Jacobian matrix of $\mathbf{h}(\cdot\,, \mathbf{b})$ evaluated at $\mathbf{z} \in A$. Similarly, $\nabla_\mathbf{x} h_i(\mathbf{z}, \mathbf{b})$ denotes the $m \times 1$ gradient vector of $h_i(\cdot\,, \mathbf{b})$ evaluated at $\mathbf{z} \in A$.

Newton-type methods for (4.2) are based on the form $\mathbf{z} := \boldsymbol{\phi}(\mathbf{z}) = \mathbf{z} - \mathbf{Y}(\mathbf{z})\mathbf{h}(\mathbf{z})$, where it is not guaranteed that $\boldsymbol{\phi}$ is contractive on *any* set. Taking $\mathbf{Y}(\mathbf{z})$ to be the inverse of the (nonsingular) Jacobian matrix $\mathbf{J}_\mathbf{x}$ evaluated at the current iterate $\mathbf{z}^k$ gives the standard Newton's method, which under mild assumptions is guaranteed to be contractive. Likewise, taking $\mathbf{Y}(\mathbf{z})$ to be a (nonsingular) constant matrix results in the parallel-chord method [104]. In [104], the authors present an in-depth analysis of the theoretical results on fixed-point iterations including conditions for guaranteed convergence, etc. The key result on which Newton-type methods rely is the mean-value theorem. A slightly modified form of that stated in [96] is presented here.

Theorem 4.2.4 (Mean-Value Theorem)**.** *Let $A \in \mathbb{R}^m$ be open and connected and let $f : A \to \mathbb{R}$ be differentiable on A. If A contains the line segment with endpoints* $\mathbf{a}$

and $\mathbf{b}$*, then there exists a point* $\mathbf{c} = \lambda\mathbf{a} + (1-\lambda)\mathbf{b}$ *with* $\lambda \in (0,1)$ *such that*

$$f(\mathbf{b}) - f(\mathbf{a}) = \nabla f(\mathbf{c})^{\mathrm{T}}(\mathbf{b} - \mathbf{a}). \tag{4.5}$$

The result that we rely upon is the parametric extension of the mean-value theorem.

Corollary 4.2.5 (Parametric Mean-Value Theorem)**.** *Let* $A \in \mathbb{R}^m$ *be open and connected and let* $P \subset \mathbb{R}^{n_p}$*, and let* $f : A \times P \to \mathbb{R}$ *be differentiable on* A *for every* $\mathbf{p} \in P$*. Let* $\mathbf{v}, \mathbf{w} : P \to A$*. Suppose that, for every* $\mathbf{p} \in P$*, the set* A *contains the line segment with endpoints* $\mathbf{v}(\mathbf{p})$ *and* $\mathbf{w}(\mathbf{p})$*. Then there exists* $\mathbf{y} : P \to A$ *such that, for each* $\mathbf{p} \in P$*,* $\mathbf{y}(\mathbf{p}) = \lambda(\mathbf{p})\mathbf{v}(\mathbf{p}) + (1 - \lambda(\mathbf{p}))\mathbf{w}(\mathbf{p})$ *for some* $\lambda : P \to (0,1)$*, and*

$$f(\mathbf{w}(\mathbf{p}), \mathbf{p}) - f(\mathbf{v}(\mathbf{p}), \mathbf{p}) = \nabla_{\mathbf{x}} f(\mathbf{y}(\mathbf{p}), \mathbf{p})^{\mathrm{T}}(\mathbf{w}(\mathbf{p}) - \mathbf{v}(\mathbf{p})). \tag{4.6}$$

Proof. Choose a $\mathbf{p}^* \in P$, then (4.6) reduces to (4.5). To see this, let $\mathbf{v}(\mathbf{p}^*) = \mathbf{a}$, $\mathbf{w}(\mathbf{p}^*) = \mathbf{b}$, $\mathbf{y}(\mathbf{p}^*) = \mathbf{c}$, $\lambda^* = \lambda(\mathbf{p}^*)$, and notice $f(\cdot, \mathbf{p}^*) : A \to \mathbb{R}$. Then we have $\mathbf{y}(\mathbf{p}^*) = \mathbf{c} = \lambda^*\mathbf{a} + (1-\lambda^*)\mathbf{b} = \lambda(\mathbf{p}^*)\mathbf{v}(\mathbf{p}^*) + (1 - \lambda(\mathbf{p}^*))\mathbf{w}(\mathbf{p}^*)$. Since the choice of $\mathbf{p}^* \in P$ was arbitrary, (4.6) holds for every $\mathbf{p} \in P$. □

4.2.2 McCormick Relaxations

McCormick [85] developed a novel technique for generating convex and concave relaxations of a given function, defined as follows.

Definition 4.2.6 (Relaxations of Functions [88])**.** Given a convex set $Z \subset \mathbb{R}^n$ and a function $f : Z \to \mathbb{R}$, a convex function $f^c : Z \to \mathbb{R}$ is a convex relaxation (or convex underestimator) of f on Z if $f^c(\mathbf{z}) \leq f(\mathbf{z})$ for every $\mathbf{z} \in Z$. A concave function $f^C : Z \to \mathbb{R}$ is a concave relaxation (or concave overestimator) of f on Z if $f^C(\mathbf{z}) \geq f(\mathbf{z})$ for every $\mathbf{z} \in Z$.

The relaxations of vector-valued or matrix-valued functions are defined by applying the above inequalities componentwise.

Definition 4.2.7 (Univariate Intrinsic Function [128])**.** The function $u : B \subset \mathbb{R} \to \mathbb{R}$ is a univariate intrinsic function if, for any $A \in \mathbb{I}B$, the following are known and can be evaluated computationally:

1. an interval extension of u on A that is an inclusion function of u on A,
2. a concave relaxation of u on A,
3. a convex relaxation of u on A.

In order to construct relaxations of a function using the rules outlined by McCormick [85], the function must be *factorable*, defined as follows.

Definition 4.2.8 (Factorable Function [128])**.** A function $f : Z \subset \mathbb{R}^{n_z} \to \mathbb{R}$ is factorable if it can be expressed in terms of a finite number of factors $v_1, \ldots, v_m$ such that, given $\mathbf{z} \in Z$, $v_i = z_i$ for $i = 1, \ldots, n_z$, and for each $n_z < k \leq m$, v_k is defined as either

a) $v_k = v_i + v_j, \quad i, j < k$, or

b) $v_k = v_i v_j, \quad i, j < k$, or

c) $v_k = u_k \circ v_i, \quad i < k$, where $u_k : B_k \to \mathbb{R}$ is a univariate intrinsic function,

and $f(\mathbf{z}) = v_m(\mathbf{z})$. A vector-valued function $\mathbf{f}$ is factorable if every component f_i is factorable.

The functions f, $\mathbf{g}$, and $\mathbf{h}$ considered in this chapter are assumed to be factorable. Such an assumption is not very restrictive since this includes any function that can be represented finitely on a computer. McCormick's relaxation technique [85] computes convex and concave relaxations of factorable functions by recursively applying simple rules for relaxing binary addition, binary multiplication, and univariate composition with univariate intrinsic functions.

Definition 4.2.9 (Composite Relaxations: $\mathbf{u}_{\mathscr{G}}, \mathbf{o}_{\mathscr{G}}$)**.** Let $D \subset \mathbb{R}^{n_x}$, $Z \in \mathbb{I}D$, and $P \in \mathbb{IR}^{n_p}$. Let $\mathbf{q} : P \to Z$ and $\mathscr{G} : D \times P \to \mathbb{R}^{n_x}$. The functions $\mathbf{u}_{\mathscr{G}}, \mathbf{o}_{\mathscr{G}} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times P \to$

$\mathbb{R}^{n_x}$ are called composite relaxations of $\mathscr{G}$ on $Z \times P$ if for any $\boldsymbol{\psi}^c, \boldsymbol{\psi}^C : P \to \mathbb{R}^{n_x}$, the functions $\mathbf{u}_{\mathscr{G}}(\boldsymbol{\psi}^c(\cdot), \boldsymbol{\psi}^C(\cdot), \cdot)$ and $\mathbf{o}_{\mathscr{G}}(\boldsymbol{\psi}^c(\cdot), \boldsymbol{\psi}^C(\cdot), \cdot)$ are, respectively, convex and concave relaxations of $\mathscr{G}(\mathbf{q}(\cdot), \cdot)$ on P, provided $\boldsymbol{\psi}^c$ and $\boldsymbol{\psi}^C$ are, respectively, convex and concave relaxations of $\mathbf{q}$ on P.

Provided that $\mathscr{G}$ is factorable, functions $\mathbf{u}_{\mathscr{G}}$ and $\mathbf{o}_{\mathscr{G}}$ satisfying the previous definition can be computed using generalized McCormick relaxations as described in [128]. By the properties of generalized McCormick relaxations, the functions $\mathbf{u}_{\mathscr{G}}$ and $\mathbf{o}_{\mathscr{G}}$ are continuous on $\mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times P$.

Remark 4.2.10. Strictly speaking, by the definition of generalized McCormick relaxations and the definition of composite relaxations given in [128], the bounding information (i.e. $Z \times P$ in Def. 4.2.9) is required and should be taken as explicit arguments of $\mathbf{u}_{\mathscr{G}}$ and $\mathbf{o}_{\mathscr{G}}$. However, for notational clarity in this work, the bounding information will not be passed as arguments of the composite relaxations and instead will be stated explicitly wherever composite relaxations are used.

Remark 4.2.11. More generally, composite relaxations for any arbitrary function $\mathscr{G}(\mathbf{v}(\cdot), \mathbf{w}(\cdot), \ldots, \mathbf{z}(\cdot), \cdot)$, on P, taking arbitrarily many functions as arguments, can be constructed in an analogous manner. Also, the inner functions need not be vector valued, but can be matrix valued, by treating each column vector of the matrix-valued function as a vector-valued function and applying the above definition.

4.2.3 Subgradients

Since McCormick relaxations are potentially nondifferentiable, subgradients provide useful information to a nonsmooth optimization code or can be used to compute affine relaxations of the functions. The rules for calculating subgradients of McCormick relaxations and corresponding affine relaxations are thoroughly discussed in [88].

Definition 4.2.12 (Subgradients)**.** Let $Z \subset \mathbb{R}^n$ be a nonempty convex set, $f^c : Z \to \mathbb{R}$ be convex and $f^C : Z \to \mathbb{R}$ be concave. A vector-valued function $\mathbf{s}_f^c : Z \to \mathbb{R}^n$ is called a subgradient of f^c on Z if for every $\bar{\mathbf{z}} \in Z$, $f^c(\mathbf{z}) \geq f^c(\bar{\mathbf{z}}) + (\mathbf{s}_f^c(\bar{\mathbf{z}}))^{\mathrm{T}}(\mathbf{z} - \bar{\mathbf{z}})$,

$\forall \mathbf{z} \in Z$. Likewise, a vector-valued function $\mathbf{s}_f^C : Z \to \mathbb{R}^n$ is called a subgradient of f^C on Z if for every $\bar{\mathbf{z}} \in Z$, $f^C(\mathbf{z}) \leq f^C(\bar{\mathbf{z}}) + (\mathbf{s}_f^C(\bar{\mathbf{z}}))^{\mathrm{T}}(\mathbf{z} - \bar{\mathbf{z}})$, $\forall \mathbf{z} \in Z$.

Remark 4.2.13. Subgradients are not unique in general. The procedures in [88] compute a single element of the subdifferential, therefore the subgradient functions above are well defined. Subgradients of vector-valued functions $\mathbf{f}^c, \mathbf{f}^C : Z \to \mathbb{R}^m$, convex and concave, respectively, will be matrix-valued functions denoted $\boldsymbol{\sigma}_{\mathbf{f}}^c, \boldsymbol{\sigma}_{\mathbf{f}}^C : Z \to \mathbb{R}^{n \times m}$. Furthermore, subgradients of matrix-valued functions $\mathbf{F}^c, \mathbf{F}^C : Z \to \mathbb{R}^{m \times m}$, convex and concave, respectively, will be 3^{rd}-order tensor-valued functions denoted $\hat{\boldsymbol{\sigma}}_{\mathbf{F}}^c, \hat{\boldsymbol{\sigma}}_{\mathbf{F}}^C : Z \to \mathbb{R}^{n \times m \times m}$.

Definition 4.2.14 (Affine Relaxations)**.** Let $Z \subset \mathbb{R}^n$ be a nonempty convex set and define $\mathbf{f} : Z \to \mathbb{R}^n$. The functions $\mathbf{f}^a, \mathbf{f}^A : Z \to \mathbb{R}^n$ are called affine relaxations of $\mathbf{f}$ if $\mathbf{f}^a(\mathbf{z}) \leq \mathbf{f}(\mathbf{z}) \leq \mathbf{f}^A(\mathbf{z}) \ \ \forall \mathbf{z} \in Z$, and $\mathbf{f}^a$ and $\mathbf{f}^A$ are affine on Z.

In the same notation as the above definition, a natural choice of affine relaxations is given by

$$\mathbf{f}^a(\mathbf{z}) = \mathbf{f}^c(\bar{\mathbf{z}}) + (\boldsymbol{\sigma}_f^c(\bar{\mathbf{z}}))^{\mathrm{T}}(\mathbf{z} - \bar{\mathbf{z}}) \ \text{ and } \ \mathbf{f}^A(\mathbf{z}) = \mathbf{f}^C(\bar{\mathbf{z}}) + (\boldsymbol{\sigma}_f^C(\bar{\mathbf{z}}))^{\mathrm{T}}(\mathbf{z} - \bar{\mathbf{z}}).$$

Definition 4.2.15 (Composite Subgradients: $\mathcal{S}_{u_{\mathscr{G}}}, \mathcal{S}_{o_{\mathscr{G}}}$)**.** Let $D \subset \mathbb{R}^{n_x}$, $P \in \mathbb{IR}^{n_p}$, and $Z \in \mathbb{I}D$. Let $\mathbf{q} : P \to Z$ and $\mathscr{G} : D \times P \to \mathbb{R}^{n_x}$. Let $\mathbf{u}_{\mathscr{G}}, \mathbf{o}_{\mathscr{G}}$ be composite relaxations of $\mathscr{G}$ on $Z \times P$. The functions $\mathcal{S}_{u_{\mathscr{G}}}, \mathcal{S}_{o_{\mathscr{G}}} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p \times n_x} \times \mathbb{R}^{n_p \times n_x} \times P \to \mathbb{R}^{n_p \times n_x}$ are called composite subgradients of $\mathbf{u}_{\mathscr{G}}$ and $\mathbf{o}_{\mathscr{G}}$ on $Z \times P$, respectively, if for any $\boldsymbol{\psi}^c, \boldsymbol{\psi}^C : P \to \mathbb{R}^{n_x}$ and $\boldsymbol{\sigma}_{\psi}^c : \boldsymbol{\sigma}_{\psi}^C : P \to \mathbb{R}^{n_p \times n_x}$, the functions $\mathcal{S}_{u_{\mathscr{G}}}(\boldsymbol{\psi}^c(\cdot), \boldsymbol{\psi}^C(\cdot), \boldsymbol{\sigma}_{\psi}^c(\cdot), \boldsymbol{\sigma}_{\psi}^C(\cdot), \cdot)$, and $\mathcal{S}_{o_{\mathscr{G}}}(\boldsymbol{\psi}^c(\cdot), \boldsymbol{\psi}^C(\cdot), \boldsymbol{\sigma}_{\psi}^c(\cdot), \boldsymbol{\sigma}_{\psi}^C(\cdot), \cdot)$ are, respectively, subgradients of $\mathbf{u}_{\mathscr{G}}\left(\boldsymbol{\psi}^c(\cdot), \boldsymbol{\psi}^C(\cdot), \cdot\right)$ and $\mathbf{o}_{\mathscr{G}}\left(\boldsymbol{\psi}^c(\cdot), \boldsymbol{\psi}^C(\cdot), \cdot\right)$, provided $\boldsymbol{\psi}^c$ and $\boldsymbol{\psi}^C$ are, respectively, convex and concave relaxations of $\mathbf{q}$ on P and $\boldsymbol{\sigma}_{\psi}^c$ and $\boldsymbol{\sigma}_{\psi}^C$ are, respectively, subgradients of $\boldsymbol{\psi}^c$ and $\boldsymbol{\psi}^C$ on P.

Remark 4.2.16. Similar to composite relaxations, composite subgradients of convex and concave relaxations of any $\mathscr{G}(\mathbf{v}(\cdot), \mathbf{q}(\cdot), \ldots, \mathbf{z}(\cdot), \cdot)$ on P, taking arbitrarily many functions as arguments, can be constructed analogously to the case considered in

Definition 4.2.15. Again, the inner functions need not be vector-valued, but can be matrix-valued, by treating each column vector of the matrix-valued function as a vector-valued function and applying the above definition. As per Remark 4.2.13, subgradients of a matrix-valued function will be 3^{rd}-order tensors.

4.3 Relaxations of Implicit Functions

This section contains new developments regarding relaxations of implicit functions. Two different methods for constructing relaxations of implicit functions will be discussed. The first is to relax a fixed-point iteration for approximating the implicit function, as in [128]. This method, along with new results is discussed in detail in Section 4.3.1. This approach can be quite limited, however, and shortcomings of this method are discussed in Section 4.3.2. The second method circumvents the shortcomings of the first method by relaxing solutions of parametric algebraic systems directly, without reference to an associated fixed-point iteration, and is thus more broadly applicable. The case of parametric linear systems is discussed in Section 4.3.3. In Section 4.3.4, the case of parametric nonlinear systems is discussed.

4.3.1 Direct Relaxation of Fixed-Point Iterations

Consider the system of equations in (4.3). Let the (factorable) function $\boldsymbol{\phi} : D_x \times D_p \to \mathbb{R}^{n_x}$ be an algebraic rearrangement of $\mathbf{h}$ such that $\mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{z} - \boldsymbol{\phi}(\mathbf{z}, \mathbf{p}) \Leftrightarrow \mathbf{z} = \boldsymbol{\phi}(\mathbf{z}, \mathbf{p})$ and $D_x \times D_p \subset D_y$. For example, consider $h(z, p) = z - \sin(z+p) = 0 \Leftrightarrow z = \sin(z+p)$ or $h(z, p) = z^2 + pz + C = 0 \Leftrightarrow z = -(z^2 + C)/p$.

Assumption 4.3.1. There exists $\mathbf{x} : P \to \mathbb{R}^{n_x}$ such that $\mathbf{x}(\mathbf{p}) = \boldsymbol{\phi}(\mathbf{x}(\mathbf{p}), \mathbf{p}),\ \forall \mathbf{p} \in P$, and an interval $[\mathbf{x}^L, \mathbf{x}^U] \equiv X \in \mathbb{IR}^{n_x}$ is known such that $\mathbf{x}(P) \subset X$ and $\mathbf{x}(\mathbf{p})$ is unique in X for all $\mathbf{p} \in P$.

The parametric extension of the well-known interval-Newton method, which is discussed in [56, 101] and developed further in Chapter 3 exhibits the theoretical capabilities of finding an X satisfying this assumption. Finding such an X is really

a precursor to calculating relaxations since, for the purposes of this chapter, it is desired to relax a single implicit function.

In [128], the authors consider the computation of relaxations of $\mathbf{x}^k : P \to \mathbb{R}^{n_x}$, the approximations of $\mathbf{x}$, defined by the fixed-point iteration:

$$\mathbf{x}^{k+1}(\mathbf{p}) := \boldsymbol{\phi}(\mathbf{x}^k(\mathbf{p}), \mathbf{p}),\ \forall \mathbf{p} \in P. \tag{4.7}$$

If $\boldsymbol{\phi}(\cdot\,, \mathbf{p})$ is a contraction mapping on X for every $\mathbf{p} \in P$, then this iteration is referred to as a successive-substitution fixed-point iteration. Under this assumption, it can be shown that $\{\mathbf{x}^k\} \to \mathbf{x}$ so that this method provides relaxations of arbitrarily good approximations of $\mathbf{x}$. However, this result is rather weak in that it does not provide us with guaranteed valid relaxations of the implicit function $\mathbf{x}$ upon finite termination. In contrast, the following result provides sequences, $\{\mathbf{x}^{k,c}\}$ and $\{\mathbf{x}^{k,C}\}$, such that $\mathbf{x}^{k,c}$ and $\mathbf{x}^{k,C}$ are relaxations of $\mathbf{x}$ on P, for every $k \in \mathbb{N}$. Moreover, this result does not require contractivity of $\boldsymbol{\phi}$ on X. Thus, although approximations of the value of $\mathbf{x}$ may not even be available, valid relaxations of $\mathbf{x}$ are readily calculable.

Definition 4.3.2. Let $\mathbf{u}_\phi, \mathbf{o}_\phi$ be composite relaxations of $\boldsymbol{\phi}$ on $X \times P$. The functions $\bar{\mathbf{u}}_\phi, \bar{\mathbf{o}}_\phi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times P \to \mathbb{R}^{n_x}$ will be defined as:

$$\begin{aligned}
\bar{\mathbf{u}}_\phi(\mathbf{z}^c, \mathbf{z}^C, \mathbf{p}) &\equiv \max\{\mathbf{z}^c, \mathbf{u}_\phi(\mathbf{z}^c, \mathbf{z}^C, \mathbf{p})\}, \\
\bar{\mathbf{o}}_\phi(\mathbf{z}^c, \mathbf{z}^C, \mathbf{p}) &\equiv \min\{\mathbf{z}^C, \mathbf{o}_\phi(\mathbf{z}^c, \mathbf{z}^C, \mathbf{p})\},
\end{aligned}$$

$\forall (\mathbf{z}^c, \mathbf{z}^C, \mathbf{p}) \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times P$ with the max/min operations applied componentwise.

Definition 4.3.3. Let $\mathbf{u}_\phi, \mathbf{o}_\phi$ be composite relaxations of $\boldsymbol{\phi}$ on $X \times P$. Let $\mathcal{S}_{u_\phi}, \mathcal{S}_{o_\phi}$ be composite subgradients of $\mathbf{u}_\phi$ and $\mathbf{o}_\phi$ on $X \times P$, respectively. The functions

$\mathcal{S}_{\bar{\mathbf{u}}_\phi}, \mathcal{S}_{\bar{\mathbf{o}}_\phi} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p \times n_x} \times \mathbb{R}^{n_p \times n_x} \times P \to \mathbb{R}^{n_p \times n_x}$ will be defined as

$$\mathcal{S}_{\bar{\mathbf{u}}_\phi}(\mathbf{z}^c, \mathbf{z}^C, \boldsymbol{\sigma}_{\mathbf{z}}^c, \boldsymbol{\sigma}_{\mathbf{z}}^C, \bar{\mathbf{p}}) = \begin{cases} \boldsymbol{\sigma}_{\mathbf{z}}^c & \text{if } \bar{\mathbf{u}}_\phi(\mathbf{z}^c, \mathbf{z}^C, \bar{\mathbf{p}}) = \mathbf{z}^c \\ \mathcal{S}_{\mathbf{u}_\phi}(\mathbf{z}^c, \mathbf{z}^C, \boldsymbol{\sigma}_{\mathbf{z}}^c, \boldsymbol{\sigma}_{\mathbf{z}}^C, \bar{\mathbf{p}}) & \text{otherwise} \end{cases}$$

$$\mathcal{S}_{\bar{\mathbf{o}}_\phi}(\mathbf{z}^c, \mathbf{z}^C, \boldsymbol{\sigma}_{\mathbf{z}}^c, \boldsymbol{\sigma}_{\mathbf{z}}^C, \bar{\mathbf{p}}) = \begin{cases} \boldsymbol{\sigma}_{\mathbf{z}}^C & \text{if } \bar{\mathbf{o}}_\phi(\mathbf{z}^c, \mathbf{z}^C, \bar{\mathbf{p}}) = \mathbf{z}^C \\ \mathcal{S}_{\mathbf{o}_\phi}(\mathbf{z}^c, \mathbf{z}^C, \boldsymbol{\sigma}_{\mathbf{z}}^c, \boldsymbol{\sigma}_{\mathbf{z}}^C, \bar{\mathbf{p}}) & \text{otherwise} \end{cases}$$

$\forall (\mathbf{z}^c, \mathbf{z}^C, \boldsymbol{\sigma}_{\mathbf{z}}^c, \boldsymbol{\sigma}_{\mathbf{z}}^C, \bar{\mathbf{p}}) \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p \times n_x} \times \mathbb{R}^{n_p \times n_x} \times P$.

It should be noted that the functions $\mathcal{S}_{\bar{u}_\phi}$ and $\mathcal{S}_{\bar{o}_\phi}$ define composite subgradients of $\bar{\mathbf{u}}_\phi$ and $\bar{\mathbf{o}}_\phi$ on $X \times P$, respectively.

Theorem 4.3.4. *Let $\mathbf{x}^{0,c}, \mathbf{x}^{0,C} : P \to \mathbb{R}^{n_x}$ be defined by $\mathbf{x}^{0,c}(\mathbf{p}) = \mathbf{x}^L$ and $\mathbf{x}^{0,C}(\mathbf{p}) = \mathbf{x}^U$ for all $\mathbf{p} \in P$. Then the elements of the sequences $\{\mathbf{x}^{k,c}\}$ and $\{\mathbf{x}^{k,C}\}$ defined by $\mathbf{x}^{k+1,c}(\cdot) = \bar{\mathbf{u}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ and $\mathbf{x}^{k+1,C}(\cdot) = \bar{\mathbf{o}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ are convex and concave relaxations of $\mathbf{x}$ on P, respectively, for every $k \in \mathbb{N}$.*

Proof. $\mathbf{x}^{0,c}$ and $\mathbf{x}^{0,C}$ are trivially convex and concave relaxations of $\mathbf{x}$ on P, respectively. Suppose this is true of $\mathbf{x}^{k,c}$ and $\mathbf{x}^{k,C}$ for some $k \geq 0$. By Definition 4.2.9, $\mathbf{u}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ and $\mathbf{o}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ are also relaxations of $\mathbf{x}(\cdot) = \boldsymbol{\phi}(\mathbf{x}(\cdot), \cdot)$ on P. Since the maximum of two convex functions is convex and the minimum of two concave functions is concave, $\mathbf{x}^{k+1,c}(\cdot) = \bar{\mathbf{u}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ and $\mathbf{x}^{k+1,C}(\cdot) = \bar{\mathbf{o}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ are convex and concave relaxations of $\mathbf{x}(\cdot) = \boldsymbol{\phi}(\mathbf{x}(\cdot), \cdot)$ on P, respectively. Induction completes the proof. □

Theorem 4.3.5. *Let $\mathbf{x}^{0,c}, \mathbf{x}^{0,C} : P \to \mathbb{R}^{n_x}$ be defined by $\mathbf{x}^{0,c}(\mathbf{p}) = \mathbf{x}^L$ and $\mathbf{x}^{0,C}(\mathbf{p}) = \mathbf{x}^U$, for all $\mathbf{p} \in P$. Let $\boldsymbol{\sigma}_{\mathbf{x}}^{0,c}(\mathbf{p}) = \boldsymbol{\sigma}_{\mathbf{x}}^{0,C}(\mathbf{p}) = \mathbf{0}$, for all $\mathbf{p} \in P$. Let relaxations of $\mathbf{x}$ on P be given by $\mathbf{x}^{k+1,c}(\cdot) = \bar{\mathbf{u}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ and $\mathbf{x}^{k+1,C}(\cdot) = \bar{\mathbf{o}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$, $k \in \mathbb{N}$. Then the sequences $\{\boldsymbol{\sigma}_{\mathbf{x}}^{k,c}\}$ and $\{\boldsymbol{\sigma}_{\mathbf{x}}^{k,C}\}$ defined by*

$$\begin{aligned} \boldsymbol{\sigma}_{\mathbf{x}}^{k+1,c}(\cdot) :=& \mathcal{S}_{\bar{\mathbf{u}}_\phi}(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,c}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,C}(\cdot), \cdot), \\ \boldsymbol{\sigma}_{\mathbf{x}}^{k+1,C}(\cdot) :=& \mathcal{S}_{\bar{\mathbf{o}}_\phi}(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,c}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,C}(\cdot), \cdot) \end{aligned}$$

are, respectively, subgradients of $\mathbf{x}^{k+1,c}$ and $\mathbf{x}^{k+1,C}$ on P for $k \in \mathbb{N}$.

Proof. From the hypothesis, $\mathbf{x}^{0,c}$ and $\mathbf{x}^{0,C}$ are (constant) convex and concave relaxations of $\mathbf{x}$ on P, respectively, and $\boldsymbol{\sigma}_{\mathbf{x}}^{0,c} = \boldsymbol{\sigma}_{\mathbf{x}}^{0,C} = \mathbf{0}$ are subgradients of $\mathbf{x}^{0,c}$ and $\mathbf{x}^{0,C}$ on P, respectively. Suppose this holds for $k \in \mathbb{N}$. Then we have $\mathbf{x}^{k,c}$ and $\mathbf{x}^{k,C}$, convex and concave relaxations of $\mathbf{x}$ on P, respectively, and $\boldsymbol{\sigma}_{\mathbf{x}}^{k,c}(\cdot)$ and $\boldsymbol{\sigma}_{\mathbf{x}}^{k,C}(\cdot)$, subgradients of $\mathbf{x}^{k,c}$ and $\mathbf{x}^{k,C}$ on P, respectively. By the definition of the composite subgradient (Def. 4.2.15), subgradients of $\mathbf{u}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ and $\mathbf{o}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ on P are given by $\mathcal{S}_{\mathbf{u}_\phi}(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,c}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,C}(\cdot), \cdot)$ and $\mathcal{S}_{\mathbf{o}_\phi}(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,c}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,C}(\cdot), \cdot)$, respectively, and by Definition 4.3.3, $\mathcal{S}_{\bar{\mathbf{u}}_\phi}(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,c}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,C}(\cdot), \cdot)$ and $\mathcal{S}_{\bar{\mathbf{o}}_\phi}(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,c}(\cdot), \boldsymbol{\sigma}_{\mathbf{x}}^{k,C}(\cdot), \cdot)$ are subgradients of

$$\begin{aligned}
\mathbf{x}^{k+1,c}(\cdot) &:= \bar{\mathbf{u}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot) = \max\{\mathbf{x}^{k,c}(\cdot), \mathbf{u}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)\}, \\
\mathbf{x}^{k+1,C}(\cdot) &:= \bar{\mathbf{o}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot) = \min\{\mathbf{x}^{k,C}(\cdot), \mathbf{o}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)\}
\end{aligned}$$

on P, respectively. Induction completes the proof. □

It is of great interest to understand when this procedure for calculating relaxations works well. Although improvement on the bounds cannot be guaranteed in general, one can find cases when improvement is definitely not possible; thus providing a necessary condition for improvement.

Theorem 4.3.6. *Let $\{\mathbf{x}^k\}$ be a sequence generated by the fixed-point iteration (4.7) starting from $\mathbf{x}^0(\mathbf{p}) \in X,\ \forall \mathbf{p} \in P$. If $\mathbf{x}^k(\mathbf{p}) \notin X$ for some $\mathbf{p} \in P$, $k \in \mathbb{N}$, then the sequences $\{\mathbf{x}^{k,c}\}$ and $\{\mathbf{x}^{k,C}\}$ from Theorem 4.3.4 are such that there exists a $\mathbf{p} \in P$ such that $\mathbf{x}^{k,c}(\mathbf{p}) = \mathbf{x}^L$ or $\mathbf{x}^{k,C}(\mathbf{p}) = \mathbf{x}^U$ for every $k \in \mathbb{N}$.*

Proof. By hypothesis, $\mathbf{x}^0(\mathbf{p}) \in X$ for every $\mathbf{p} \in P$ and $\mathbf{x}^{0,c} = \mathbf{x}^L$ and $\mathbf{x}^{0,C} = \mathbf{x}^U$. Therefore $\mathbf{x}^{0,c}$ and $\mathbf{x}^{0,C}$ are convex and concave relaxations of $\mathbf{x}^0$ on P, respectively. Suppose this is true for $(K-1) \in \mathbb{N}$ where K is the iteration in which $\mathbf{x}^K(\bar{\mathbf{p}}) \notin X$ such that $\mathbf{x}^K(\bar{\mathbf{p}}) \in X,\ \forall k < K$ for some $\bar{\mathbf{p}} \in P$. Then by Definition 4.2.9 and Theorem 4.3.4, $\mathbf{u}_\phi(\mathbf{x}^{k-1,c}(\mathbf{p}), \mathbf{x}^{k-1,C}(\mathbf{p}), \mathbf{p}) \leq \boldsymbol{\phi}(\mathbf{x}^{k-1}(\mathbf{p}), \mathbf{p}) \leq \mathbf{o}_\phi(\mathbf{x}^{k-1,c}(\mathbf{p}), \mathbf{x}^{k-1,C}(\mathbf{p}), \mathbf{p})$ for every $\mathbf{p} \in P$ (noting $\mathbf{x}^{K-1}(\mathbf{p}) \in X,\ \forall \mathbf{p} \in P$). Since $\mathbf{x}^K(\mathbf{p}) = \boldsymbol{\phi}(\mathbf{x}^{K-1}(\mathbf{p}), \mathbf{p}),\ \forall \mathbf{p} \in P$, this implies $\mathbf{u}_\phi(\mathbf{x}^{K-1,c}(\mathbf{p}), \mathbf{x}^{K-1,C}(\mathbf{p}), \mathbf{p}) \leq \mathbf{x}^K(\mathbf{p}) \leq \mathbf{o}_\phi(\mathbf{x}^{K-1,c}(\mathbf{p}), \mathbf{x}^{K-1,C}(\mathbf{p}), \mathbf{p}),\ \forall \mathbf{p} \in$

P. However, since $\mathbf{x}^K(\bar{\mathbf{p}}) \notin X$, it follows that $\mathbf{u}_\phi(\mathbf{x}^{K-1,c}(\bar{\mathbf{p}}), \mathbf{x}^{K-1,C}(\bar{\mathbf{p}}), \bar{\mathbf{p}}) < \mathbf{x}^L$ or $\mathbf{o}_\phi(\mathbf{x}^{K-1,c}(\bar{\mathbf{p}}), \mathbf{x}^{K-1,C}(\bar{\mathbf{p}}), \bar{\mathbf{p}}) > \mathbf{x}^U$. Therefore $\bar{\mathbf{u}}_\phi(\mathbf{x}^{K-1,c}(\bar{\mathbf{p}}), \mathbf{x}^{K-1,C}(\bar{\mathbf{p}}), \bar{\mathbf{p}}) = \mathbf{x}^L$ or $\bar{\mathbf{o}}_\phi(\mathbf{x}^{K-1,c}(\bar{\mathbf{p}}), \mathbf{x}^{K-1,C}(\bar{\mathbf{p}}), \bar{\mathbf{p}}) = \mathbf{x}^U$. □

4.3.2 Direct Relaxation of Newton-Type Iterations

According to Theorem 4.3.6, the property that $\boldsymbol{\phi}$ maps $X \times P$ into X is desirable in order to calculate relaxations that are potential improvements on the original bounds of X. This property will be exhibited by any $\boldsymbol{\phi}$ that is a contraction mapping. Consider the system of equations (4.3) and now suppose that $\mathbf{h}$ cannot be rearranged algebraically as in the previous section, such that (4.7) is contractive. Thus, $\mathbf{h}$ will be a member of a more general class of functions. The following result guarantees that a different form of fixed-point iteration can still be constructed from any such system and under some other fixed-point results, may be guaranteed to be contractive. However, as will be shown in this section, the fact that $\boldsymbol{\phi}$ is contractive is not enough to calculate relaxations of $\mathbf{x}$ that are guaranteed to be refinements on the bounds of X using the method of Section 4.3.1. Although, this property is a necessary condition.

Proposition 4.3.7. *For any function* $\mathbf{h} : A \subset \mathbb{R}^n \to \mathbb{R}^n$*, there exists* $\boldsymbol{\phi} : A \to \mathbb{R}^n$ *such that* $\boldsymbol{\phi}(\mathbf{z}) = \mathbf{z}$ *if and only if* $\mathbf{h}(\mathbf{z}) = \mathbf{0}$*.*

Proof. Let $\boldsymbol{\phi}(\mathbf{z}) = \mathbf{z} - \mathbf{Y}\mathbf{h}(\mathbf{z})$ with $\mathbf{Y} \in \mathbb{R}^{n \times n}$ nonsingular. □

By the previous proposition, the function $\boldsymbol{\phi} : X \times P \to \mathbb{R}^{n_x}$ can be defined as

$$\boldsymbol{\phi}(\mathbf{z}, \mathbf{p}) \equiv \mathbf{z} - \mathbf{Y}(\mathbf{z}, \mathbf{p})\mathbf{h}(\mathbf{z}, \mathbf{p}) \tag{4.8}$$

with $\mathbf{Y}(\mathbf{z}, \mathbf{p}) \in \mathbb{R}^{n_x \times n_x}$ nonsingular for all $(\mathbf{z}, \mathbf{p}) \in X \times P$. Then

$$\mathbf{x}^{k+1}(\mathbf{p}) := \boldsymbol{\phi}(\mathbf{x}^k(\mathbf{p}), \mathbf{p}) \tag{4.9}$$

is a fixed-point iteration. Thus, the method of Section 4.3.1 can still, in principle, be used to construct relaxations of $\mathbf{x}$ on P. However, the following result shows that the

relaxations of $\mathbf{x}$ constructed in this way cannot be tighter than the bounds $\mathbf{x}^L$ and $\mathbf{x}^U$.

Theorem 4.3.8. *Let $\boldsymbol{\phi}$ be defined as in* (4.8) *and suppose Assumption 4.3.1 holds. Let $\mathbf{x}^{0,c}, \mathbf{x}^{0,C} : P \to \mathbb{R}^{n_x}$ be defined by $\mathbf{x}^{0,c}(\mathbf{p}) = \mathbf{x}^L$ and $\mathbf{x}^{0,C}(\mathbf{p}) = \mathbf{x}^U$ for all $\mathbf{p} \in P$. Let $\mathbf{u}_\phi$ and $\mathbf{o}_\phi$ be composite relaxations of $\boldsymbol{\phi}$ on $X \times P$ and let $\bar{\mathbf{u}}_\phi$ and $\bar{\mathbf{o}}_\phi$ be defined as in Definition 4.3.2. Let $\mathbf{x}^{k,c}, \mathbf{x}^{k,C} : P \to X$ be defined by $\mathbf{x}^{k+1,c}(\cdot) := \bar{\mathbf{u}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ and $\mathbf{x}^{k+1,C}(\cdot) := \bar{\mathbf{o}}_\phi(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$. Then $\mathbf{x}^{k,c}(\mathbf{p}) = \mathbf{x}^L$ and $\mathbf{x}^{k,C}(\mathbf{p}) = \mathbf{x}^U$ for every $\mathbf{p} \in P$ and all $k \in \mathbb{N}$.*

Proof. Let $\mathbf{f}(\mathbf{z}, \mathbf{p}) = -\mathbf{Y}(\mathbf{z}, \mathbf{p})\mathbf{h}(\mathbf{z}, \mathbf{p})$. By the rules of McCormick relaxations [128], $\mathbf{u}_\phi$ and $\mathbf{o}_\phi$ can be written as

$$\begin{aligned}
\mathbf{u}_\phi(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p}) &= \mathbf{x}^{k,c}(\mathbf{p}) + \mathbf{u}_\mathbf{f}(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p}), \\
\mathbf{o}_\phi(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p}) &= \mathbf{x}^{k,C}(\mathbf{p}) + \mathbf{o}_\mathbf{f}(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p}),
\end{aligned}$$

where, respectively, $\mathbf{u}_\mathbf{f}$ and $\mathbf{o}_\mathbf{f}$ are composite relaxations of $\mathbf{f}$ on $X \times P$. By Definition 4.2.9, $\mathbf{u}_\mathbf{f}(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ and $\mathbf{o}_\mathbf{f}(\mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot)$ are convex and concave relaxations of $\mathbf{f}(\mathbf{x}(\cdot), \cdot)$ on P, respectively for every $k \in \mathbb{N}$. By Definition, $\mathbf{f}(\mathbf{x}(\mathbf{p}), \mathbf{p}) = \mathbf{0}$, $\forall \mathbf{p} \in P$. Thus

$$\mathbf{u}_\mathbf{f}(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p}) \leq \mathbf{0} \leq \mathbf{o}_\mathbf{f}(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p})$$

hold for every $\mathbf{p} \in P$ for every $k \geq 0$. Note that $\mathbf{x}^{0,c}(\mathbf{p}) = \mathbf{x}^L$ and $\mathbf{x}^{0,C}(\mathbf{p}) = \mathbf{x}^U$. Suppose the same is true of $\mathbf{x}^{k,c}$ and $\mathbf{x}^{k,C}$, respectively. Then,

$$\begin{aligned}
\mathbf{x}^{k,c}(\mathbf{p}) + \mathbf{u}_\mathbf{f}(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p}) &\leq \mathbf{x}^{k,c}(\mathbf{p}) = \mathbf{x}^L, \\
\mathbf{x}^{k,C}(\mathbf{p}) + \mathbf{o}_\mathbf{f}(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p}) &\geq \mathbf{x}^{k,C}(\mathbf{p}) = \mathbf{x}^U.
\end{aligned}$$

By Definition 4.3.2, we have

$$\mathbf{x}^{k+1,c}(\mathbf{p}) := \max\left\{\mathbf{x}^{k,c}, \mathbf{x}^{k,c}(\mathbf{p}) + \mathbf{u_f}(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p})\right\} = \mathbf{x}^L,$$
$$\mathbf{x}^{k+1,C}(\mathbf{p}) := \min\left\{\mathbf{x}^{k,C}, \mathbf{x}^{k,C}(\mathbf{p}) + \mathbf{o_f}(\mathbf{x}^{k,c}(\mathbf{p}), \mathbf{x}^{k,C}(\mathbf{p}), \mathbf{p})\right\} = \mathbf{x}^U.$$

Induction completes the proof. □

The importance of the above theorem is that the convex and concave relaxations of the generic Newton-type form (4.8), discussed in Proposition 4.3.7, can be no tighter than the original bounds given by X, and will in fact be fixed at these bounds. This result is analogous to the reason why one cannot simply take an interval extension of the Newton iteration and expect to improve the initial bounds on a locally unique solution.[2] This result motivates the need for a different technique for calculating valid convex and concave relaxations of $\mathbf{x}$. Again, it should be noted that fixed-point iterations of different forms, such as the successive-substitution iteration discussed above and in [128], may not have the same problem, per Theorem 4.3.4, so long as $\boldsymbol{\phi}$ maps $X \times P$ into X.

The next two sections describe a different method which is capable of constructing relaxations of $\mathbf{x}$ on P that are potentially refinements of the bounds given by X, when no successive-substitution rearrangement for $\mathbf{h}$ exists that is a contraction mapping. First, the method is developed for parametric linear systems in Section 4.3.3. The extension to parametric nonlinear systems is developed in Section 4.3.4.

4.3.3 Relaxations of Solutions of Parametric Linear Systems

Consider the parametric linear system:

$$\mathbf{A}(\mathbf{p})\mathbf{z} = \mathbf{b}(\mathbf{p}), \tag{4.10}$$

with $\mathbf{A} : P \to D_a \subset \mathbb{R}^{n_x \times n_x}$ and $\mathbf{b} : P \to D_b \subset \mathbb{R}^{n_x}$ factorable, $\mathbf{z} \in \mathbb{R}^{n_x}$, and $\mathbf{p} \in P$.

[2] The reader is directed back to Chap. 3 or [101] if this is unclear.

Assumption 4.3.9.

1. There exists $\boldsymbol{\delta} : P \to \mathbb{R}^{n_x}$ such that $\mathbf{A}(\mathbf{p})\boldsymbol{\delta}(\mathbf{p}) = \mathbf{b}(\mathbf{p})$, $\forall \mathbf{p} \in P$, and an interval $[\boldsymbol{\delta}^L, \boldsymbol{\delta}^U] \equiv \Delta \in \mathbb{IR}^{n_x}$ is available such that $\boldsymbol{\delta}(P) \subset \Delta$ and $\boldsymbol{\delta}(\mathbf{p})$ is unique in Δ for every $\mathbf{p} \in P$.

2. Intervals $A \in \mathbb{I}D_a$ and $B \in \mathbb{I}D_b$ are known such that $\mathbf{A}(P) \subset A$, $\mathbf{b}(P) \subset B$, and $0 \notin A_{ii}$ for all i.

Since $\mathbf{A}$ and $\mathbf{b}$ are factorable, the intervals A and B are easily calculable using interval analysis, e.g. by calculating their natural interval extensions. The set Δ may be computed using a parametric interval linear solver such as that in [106, 107, 119]. The assumption that $0 \notin A_{ii}$, $\forall i$ implies that $a_{ii}(\mathbf{p}) \neq 0$ for all $\mathbf{p} \in P$. However, this can be relaxed by assuming that there exists a preconditioning matrix $\mathbf{Y} \in \mathbb{R}^{n_x \times n_x}$ such that the diagonal elements of $\mathbf{Y}A$ do not enclose 0 and thus the product $\mathbf{YA}(\mathbf{p})$ has nonzero diagonal elements for every $\mathbf{p} \in P$. In [101], various results on the relationship between $\mathbf{Y}$, A, and $\mathbf{A}$ are discussed. The key result of this section offers a way of calculating relaxations of solutions to parametric linear systems. To begin, the solution $\boldsymbol{\delta}$ will be characterized in semi-explicit form.

Definition 4.3.10 ($\mathbf{f}$). Define the function $\mathbf{f} : D_b \times D_a \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ such that $\mathbf{f}(\tilde{\mathbf{b}}, \tilde{\mathbf{A}}, \tilde{\boldsymbol{\delta}}) = \tilde{\boldsymbol{\delta}}^*$, where the i^{th} component of $\tilde{\boldsymbol{\delta}}^*$ is given by the loop:

$$\texttt{for} \quad i = 1, \ldots, n_x \ \texttt{do}$$
$$\tilde{\delta}_i^* := \left(\tilde{b}_i - \sum_{j<i} \tilde{a}_{ij} \tilde{\delta}_j^* - \sum_{j>i} \tilde{a}_{ij} \tilde{\delta}_j \right) / \tilde{a}_{ii} \tag{4.11}$$
$$\texttt{end}$$

where $\tilde{a}_{ij}$ is the $(i,j)^{\text{th}}$ element of $\tilde{\mathbf{A}}$, $\tilde{b}_i$ is the i^{th} component of $\tilde{\mathbf{b}}$, and $\tilde{\delta}_i$ is the i^{th} component of $\tilde{\boldsymbol{\delta}}$.

Lemma 4.3.11. *Suppose Assumption 4.3.9 holds. Then* $\boldsymbol{\delta}(\mathbf{p}) = \mathbf{f}(\mathbf{b}(\mathbf{p}), \mathbf{A}(\mathbf{p}), \boldsymbol{\delta}(\mathbf{p}))$ *for every* $\mathbf{p} \in P$*, i.e.,* $\boldsymbol{\delta}(\mathbf{p})$ *is a fixed-point of* $\mathbf{f}(\mathbf{b}(\mathbf{p}), \mathbf{A}(\mathbf{p}), \cdot)$ *for every* $\mathbf{p} \in P$*.*

Proof. By hypothesis, $\mathbf{A}(\mathbf{p})\boldsymbol{\delta}(\mathbf{p}) = \mathbf{b}(\mathbf{p})$ holds and the i^{th} equation can be expressed as

$$\sum_{j=1}^{n_x} a_{ij}(\mathbf{p})\delta_j(\mathbf{p}) = b_i(\mathbf{p}),\ \forall \mathbf{p} \in P.$$

Or, equivalently written

$$a_{ii}(\mathbf{p})\delta_i(\mathbf{p}) + \sum_{j<i} a_{ij}(\mathbf{p})\delta_j(\mathbf{p}) + \sum_{j>i} a_{ij}(\mathbf{p})\delta_j(\mathbf{p}) = b_i(\mathbf{p}),\ \forall \mathbf{p} \in P.$$

Solving for δ_i:

$$\delta_i(\mathbf{p}) = \left(b_i(\mathbf{p}) - \sum_{j<i} a_{ij}(\mathbf{p})\delta_j(\mathbf{p}) - \sum_{j>i} a_{ij}(\mathbf{p})\delta_j(\mathbf{p}) \right) / a_{ii}(\mathbf{p}),\ \forall \mathbf{p} \in P.$$

It immediately follows that

$$f_1(\mathbf{b}(\mathbf{p}), \mathbf{A}(\mathbf{p}), \boldsymbol{\delta}(\mathbf{p})) = \delta_1^*(\mathbf{p}) = \left(b_1(\mathbf{p}) - \sum_{j>1} a_{1j}(\mathbf{p})\delta_j(\mathbf{p}) \right) / a_{11}(\mathbf{p}) = \delta_1(\mathbf{p}).$$

Suppose $\delta_k = \delta_k^*$ holds for $k < n_x$. Then

$$\begin{aligned} &f_{k+1}(\mathbf{b}(\mathbf{p}), \mathbf{A}(\mathbf{p}), \boldsymbol{\delta}(\mathbf{p})) = \\ &\delta_{k+1}^*(\mathbf{p}) = \left(b_{k+1}(\mathbf{p}) - \sum_{j<k+1} a_{ij}(\mathbf{p})\delta_j^* - \sum_{j>k+1} a_{ij}\delta_{ij} \right) / a_{ii}(\mathbf{p}) \\ &= \left(b_{k+1}(\mathbf{p}) - \sum_{j<k+1} a_{ij}(\mathbf{p})\delta_j - \sum_{j>k+1} a_{ij}(\mathbf{p})\delta_{ij} \right) / a_{ii}(\mathbf{p}) = \delta_{k+1}(\mathbf{p}) \end{aligned}$$

Induction completes the proof. □

Using the characterization of $\boldsymbol{\delta}$ provided by Lemma 4.3.11, convex and concave relaxations of $\boldsymbol{\delta}$ on P can be computed by iteratively refining the bounds $\boldsymbol{\delta}^L$ and $\boldsymbol{\delta}^U$.

Theorem 4.3.12 (Relaxations of Parametric Linear Systems)**.** *Let* $\mathbf{A}^c, \mathbf{A}^C : P \to \mathbb{R}^{n_x \times n_x}$ *be convex and concave relaxations of* $\mathbf{A}$ *on* P*, respectively, and let* $\mathbf{b}^c, \mathbf{b}^C : P \to \mathbb{R}^{n_x}$ *be convex and concave relaxations of* $\mathbf{b}$ *on* P*, respectively. Let* $\mathbf{u_f}$ *and* $\mathbf{o_f}$ *be composite relaxations of* $\mathbf{f}$ *on* $B \times A \times \Delta \times P$*. Let* $\boldsymbol{\delta}^{0,c}, \boldsymbol{\delta}^{0,C} : P \to \mathbb{R}^{n_x}$ *be defined by*

$\boldsymbol{\delta}^{0,c}(\mathbf{p}) = \boldsymbol{\delta}^L$ *and* $\boldsymbol{\delta}^{0,C}(\mathbf{p}) = \boldsymbol{\delta}^U$ *for all* $\mathbf{p} \in P$. *Then the sequences* $\{\boldsymbol{\delta}^{k,c}\}$ *and* $\{\boldsymbol{\delta}^{k,C}\}$ *defined by the iteration*

$$\boldsymbol{\delta}^{k+1,c}(\cdot) := \bar{\mathbf{u}}_{\mathbf{f}}(\mathbf{b}^c(\cdot), \mathbf{b}^C(\cdot), \mathbf{A}^c(\cdot), \mathbf{A}^C(\cdot), \boldsymbol{\delta}^{k,c}(\cdot), \boldsymbol{\delta}^{k,C}(\cdot)),$$
$$\boldsymbol{\delta}^{k+1,C}(\cdot) := \bar{\mathbf{o}}_{\mathbf{f}}(\mathbf{b}^c(\cdot), \mathbf{b}^C(\cdot), \mathbf{A}^c(\cdot), \mathbf{A}^C(\cdot), \boldsymbol{\delta}^{k,c}(\cdot), \boldsymbol{\delta}^{k,C}(\cdot)),$$

are convex and concave relaxations of $\boldsymbol{\delta}$ *on* P, *respectively, for every* $k \in \mathbb{N}$, *with* $\bar{\mathbf{u}}_{\mathbf{f}}, \bar{\mathbf{o}}_{\mathbf{f}}$ *defined analogously to Def. 4.3.2.*

Proof. $\boldsymbol{\delta}^{0,c}$ and $\boldsymbol{\delta}^{0,C}$ are trivially convex and concave relaxations of $\boldsymbol{\delta}$ on P. Suppose this holds for $k \geq 0$. Then $\boldsymbol{\delta}^{k,c}$ and $\boldsymbol{\delta}^{k,C}$ are relaxations of $\boldsymbol{\delta}$ on P. By Definition 4.2.9

$$\mathbf{u}_{\mathbf{f}}(\mathbf{b}^c(\cdot), \mathbf{b}^C(\cdot), \mathbf{A}^c(\cdot), \mathbf{A}^C(\cdot), \boldsymbol{\delta}^{k,c}(\cdot), \boldsymbol{\delta}^{k,C}(\cdot)),$$
$$\mathbf{o}_{\mathbf{f}}(\mathbf{b}^c(\cdot), \mathbf{b}^C(\cdot), \mathbf{A}^c(\cdot), \mathbf{A}^C(\cdot), \boldsymbol{\delta}^{k,c}(\cdot), \boldsymbol{\delta}^{k,C}(\cdot)),$$

are convex and concave relaxations of $\mathbf{f}(\mathbf{b}(\cdot), \mathbf{A}(\cdot), \boldsymbol{\delta}(\cdot))$ on P, respectively. By Lemma 4.3.11, $\boldsymbol{\delta}(\cdot) = \mathbf{f}(\mathbf{b}(\cdot), \mathbf{A}(\cdot), \boldsymbol{\delta}(\cdot))$, and hence these are also relaxations of $\boldsymbol{\delta}$ on P. Since the maximum of two convex functions is convex and the minimum of two concave functions is concave,

$$\boldsymbol{\delta}^{k+1,c}(\cdot) := \bar{\mathbf{u}}_{\mathbf{f}}(\mathbf{b}^c(\cdot), \mathbf{b}^C(\cdot), \mathbf{A}^c(\cdot), \mathbf{A}^C(\cdot), \boldsymbol{\delta}^{k,c}(\cdot), \boldsymbol{\delta}^{k,C}(\cdot)),$$
$$\boldsymbol{\delta}^{k+1,C}(\cdot) := \bar{\mathbf{o}}_{\mathbf{f}}(\mathbf{b}^c(\cdot), \mathbf{b}^C(\cdot), \mathbf{A}^c(\cdot), \mathbf{A}^C(\cdot), \boldsymbol{\delta}^{k,c}(\cdot), \boldsymbol{\delta}^{k,C}(\cdot)),$$

are convex and concave relaxations of $\boldsymbol{\delta}$ on P, respectively. Induction completes the proof. □

Remark 4.3.13. The definition of $\mathbf{f}$ does not have explicit dependence on $\mathbf{p}$, however, this is just a special case of the general form (4.7). Therefore $\mathbf{u}_{\mathbf{f}}$ and $\mathbf{o}_{\mathbf{f}}$ are said to be composite relaxations of $\mathbf{f}$ on $B \times A \times \Delta \times P$, which is consistent with the definition of composite relaxations (Def. 4.2.9).

It should be noted that the functions $\boldsymbol{\delta}^{k,c}$ and $\boldsymbol{\delta}^{k,C}$ can be no worse than the

original bounds. Thus, Theorem 4.3.12 offers an efficient procedure for constructing relaxations of solutions to parametric linear systems that may be, potentially significant, refinements of the original bounds. It should also be mentioned that because of how $\mathbf{f}$ is defined, each component i makes use of information from the previous $j < i$ updated components. It is said that $\mathbf{f}$ is evaluated in a sequential componentwise manner. Similarly, relaxations of $\mathbf{f}$ are calculated in a sequential componentwise manner. What this amounts to is the sequential componentwise refinement of relaxations of δ_j making use of the newly calculated refinements of the previous components ($i < j$). This procedure is analogous to how the Gauss-Seidel method propagates the newly calculated ($i < j$) information forward to ($j > i$) components to get better approximations of the solution and potentially speed up convergence. Subgradients of these relaxations can also be calculated.

Theorem 4.3.14. *Let $\mathbf{A}^c, \mathbf{A}^C : P \to \mathbb{R}^{n_x \times n_x}$ be convex and concave relaxations of $\mathbf{A}$ on P, respectively, and let $\mathbf{b}^c, \mathbf{b}^C : P \to \mathbb{R}^{n_x}$ be convex and concave relaxations of $\mathbf{b}$ on P, respectively. Let $\boldsymbol{\delta}^{0,c}, \boldsymbol{\delta}^{0,C} : P \to \mathbb{R}^{n_x}$ be defined by $\boldsymbol{\delta}^{0,c}(\mathbf{p}) = \boldsymbol{\delta}^L$ and $\boldsymbol{\delta}^{0,C}(\mathbf{p}) = \boldsymbol{\delta}^U$ for all $\mathbf{p} \in P$ and $\boldsymbol{\sigma}_\delta^{0,c}(\mathbf{p}) = \boldsymbol{\sigma}_\delta^{0,C}(\mathbf{p}) = \mathbf{0}$, for all $\mathbf{p} \in P$. Let $\hat{\boldsymbol{\sigma}}_{\mathbf{A}}^c, \hat{\boldsymbol{\sigma}}_{\mathbf{A}}^C : P \to \mathbb{R}^{n_p \times n_x \times n_x}$ be subgradients of $\mathbf{A}^c, \mathbf{A}^C$ on P, respectively. Similarly, let $\boldsymbol{\sigma}_{\mathbf{b}}^c, \boldsymbol{\sigma}_{\mathbf{b}}^C : P \to \mathbb{R}^{n_p \times n_x}$ be subgradients of $\mathbf{b}^c, \mathbf{b}^C$ on P, respectively. Let relaxations of $\boldsymbol{\delta}$, $(\boldsymbol{\delta}^{k,c}, \boldsymbol{\delta}^{k,C})$, be given by Theorem 4.3.12. Let $\mathcal{S}_{\mathbf{u_f}}, \mathcal{S}_{\mathbf{o_f}}$ be composite subgradients of $\mathbf{u_f}$ and $\mathbf{o_f}$ on $B \times A \times \Delta \times P$, respectively. Then the sequences $\{\boldsymbol{\sigma}_\delta^{k+1,c}\}$ and $\{\boldsymbol{\sigma}_\delta^{k+1,C}\}$ defined by*

$$\begin{aligned}
\boldsymbol{\sigma}_\delta^{k+1,c}(\cdot) :=& \mathcal{S}_{\bar{\mathbf{u}}_\mathbf{f}}(\mathbf{b}^c(\cdot), \mathbf{b}^C(\cdot), \boldsymbol{\sigma}_{\mathbf{b}}^c(\cdot), \boldsymbol{\sigma}_{\mathbf{b}}^C(\cdot), \mathbf{A}^c(\cdot), \mathbf{A}^C(\cdot), \hat{\boldsymbol{\sigma}}_{\mathbf{A}}^c(\cdot), \hat{\boldsymbol{\sigma}}_{\mathbf{A}}^C(\cdot), \\
& \boldsymbol{\delta}^{k,c}(\cdot), \boldsymbol{\delta}^{k,C}(\cdot), \boldsymbol{\sigma}_\delta^{k,c}(\cdot), \boldsymbol{\sigma}_\delta^{k,C}(\cdot)), \\
\boldsymbol{\sigma}_\delta^{k+1,C}(\cdot) :=& \mathcal{S}_{\bar{\mathbf{o}}_\mathbf{f}}(\mathbf{b}^c(\cdot), \mathbf{b}^C(\cdot), \boldsymbol{\sigma}_{\mathbf{b}}^c(\cdot), \boldsymbol{\sigma}_{\mathbf{b}}^C(\cdot), \mathbf{A}^c(\cdot), \mathbf{A}^C(\cdot), \hat{\boldsymbol{\sigma}}_{\mathbf{A}}^c(\cdot), \hat{\boldsymbol{\sigma}}_{\mathbf{A}}^C(\cdot), \\
& \boldsymbol{\delta}^{k,c}(\cdot), \boldsymbol{\delta}^{k,C}(\cdot), \boldsymbol{\sigma}_\delta^{k,c}(\cdot), \boldsymbol{\sigma}_\delta^{k,C}(\cdot))
\end{aligned}$$

are subgradients of $\boldsymbol{\delta}^{k+1,c}$ and $\boldsymbol{\delta}^{k+1,C}$ on P, respectively, with $\mathcal{S}_{\bar{\mathbf{u}}_\mathbf{f}}$ and $\mathcal{S}_{\bar{\mathbf{o}}_\mathbf{f}}$ defined analogously to Def. 4.3.3.

Proof. The proof is analogous to that for Theorem 4.3.5. □

4.3.4 Relaxations of Solutions of Parametric Nonlinear Systems

As in Section 4.3.2, the general form of $\mathbf{h}$ will be considered such that $\mathbf{h}$ cannot be rearranged algebraically as in Section 4.3.1.

Assumption 4.3.15.

1. There exists $\mathbf{x} : P \to D_x$ such that $\mathbf{h}(\mathbf{x}(\mathbf{p}), \mathbf{p}) = \mathbf{0},\ \forall \mathbf{p} \in P$, and an interval $X \equiv [\mathbf{x}^L, \mathbf{x}^U] \subset \mathbb{I}D_x$ is available such that $\mathbf{x}(P) \subset X$ and $\mathbf{x}(\mathbf{p})$ is unique in X for all $\mathbf{p} \in P$.

2. Derivative information $\nabla_{\mathbf{x}} h_i,\ i = 1, \ldots, n_x$ is available and is factorable, say by automatic differentiation [12, 49].

3. A matrix $\mathbf{Y} \in \mathbb{R}^{n_x \times n_x}$ is known such that $M \equiv \mathbf{Y} J_{\mathbf{x}}(X, P)$ satisfies $0 \notin M_{ii}$ for all i, where $J_{\mathbf{x}}$ is an inclusion monotonic interval extension of $\mathbf{J}_{\mathbf{x}}$ on $X \times P$.

The matrix M can be calculated by taking natural interval extensions [92, 101]. Furthermore, parametric interval-Newton methods [54, 56, 101] offer a way to calculate X satisfying Assumption 4.3.15. The matrix $\mathbf{Y}$ is simply a *preconditioning matrix* and has been the topic of many articles. Specifically, [69] discusses the application to interval-Newton methods. A frequently valid choice is $\mathbf{Y} = [m(J_{\mathbf{x}}(X, P))]^{-1}$, which is popular due to its relatively efficient computation. As in Section 4.3.3, we begin by characterizing $\mathbf{x}$ in semi-explicit form.

Lemma 4.3.16. *Choose any $\mathbf{z} : P \to \mathbb{R}^{n_x}$ such that $\mathbf{z}(P) \subset X$. There exists a matrix-valued function $\mathbf{M} : P \to M$ such that*

$$-\mathbf{Y}\mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p}) = \mathbf{M}(\mathbf{p})(\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})),\ \ \forall \mathbf{p} \in P$$

with $M \equiv \mathbf{Y} J_{\mathbf{x}}(X, P)$.

Proof. From the Parametric Mean-Value Theorem 4.2.5, there exists a function $\mathbf{y}^i$:

$P \to X$ such that

$$h_i(\mathbf{x}(\mathbf{p}),\mathbf{p}) - h_i(\mathbf{z}(\mathbf{p}),\mathbf{p}) = \nabla_{\mathbf{x}} h_i(\mathbf{y}^i(\mathbf{p}),\mathbf{p})^{\mathrm{T}}(\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})), \quad \forall \mathbf{p} \in P$$

for the i^{th} component of $\mathbf{h}$. Writing the mean-value form for $i = 1, \ldots, n_x$, and noticing that $h_i(\mathbf{x}(\mathbf{p}),\mathbf{p}) = 0$ for all i, we get

$$-\mathbf{h}(\mathbf{z}(\mathbf{p}),\mathbf{p}) = \begin{bmatrix} \nabla_{\mathbf{x}} h_1(\mathbf{y}^1(\mathbf{p}),\mathbf{p})^{\mathrm{T}} \\ \nabla_{\mathbf{x}} h_2(\mathbf{y}^2(\mathbf{p}),\mathbf{p})^{\mathrm{T}} \\ \vdots \\ \nabla_{\mathbf{x}} h_{n_x}(\mathbf{y}^{n_x}(\mathbf{p}),\mathbf{p})^{\mathrm{T}} \end{bmatrix} (\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})), \quad \forall \mathbf{p} \in P.$$

Multiplying both sides by $\mathbf{Y}$, we get

$$-\mathbf{Y}\mathbf{h}(\mathbf{z}(\mathbf{p}),\mathbf{p}) = \mathbf{Y} \begin{bmatrix} \nabla_{\mathbf{x}} h_1(\mathbf{y}^1(\mathbf{p}),\mathbf{p})^{\mathrm{T}} \\ \nabla_{\mathbf{x}} h_2(\mathbf{y}^2(\mathbf{p}),\mathbf{p})^{\mathrm{T}} \\ \vdots \\ \nabla_{\mathbf{x}} h_{n_x}(\mathbf{y}^{n_x}(\mathbf{p}),\mathbf{p})^{\mathrm{T}} \end{bmatrix} (\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})), \quad \forall \mathbf{p} \in P.$$

Let $\mathbf{B} : X \times X \times \cdots \times X \times P \to \mathbb{R}^{n_x \times n_x}$ be defined so that

$$\mathbf{M}(\cdot) = \mathbf{B}(\mathbf{y}^1(\cdot), \mathbf{y}^2(\cdot), \ldots, \mathbf{y}^{n_x}(\cdot), \cdot) \equiv \mathbf{Y} \begin{bmatrix} \nabla_{\mathbf{x}} h_1(\mathbf{y}^1(\cdot),\cdot)^{\mathrm{T}} \\ \nabla_{\mathbf{x}} h_2(\mathbf{y}^2(\cdot),\cdot)^{\mathrm{T}} \\ \vdots \\ \nabla_{\mathbf{x}} h_{n_x}(\mathbf{y}^{n_x}(\cdot),\cdot)^{\mathrm{T}} \end{bmatrix}.$$

By Assumption 4.3.15-3, there exists a matrix $\mathbf{Y}$ so that $M \equiv \mathbf{Y} J_{\mathbf{x}}(X, P)$ is such that $0 \notin M_{ii}$. Since $\mathbf{y}^i(P) \subset X$ and the image $\mathbf{B}(X, X, \cdots, X, P) \subset \mathbf{Y} J_{\mathbf{x}}(X, P)$, so that $\mathbf{M}(P) \subset M$. □

It is important to notice that, for the purposes of this chapter and beyond, $\mathbf{M}$ need not be calculated explicitly. However, it is required that convex and concave relaxations of $\mathbf{M}$ on P can be calculated. This is fortuitous since it is easier to relax

$\mathbf{M}$ than calculate $\mathbf{M}$ explicitly.

Lemma 4.3.17. *Let* $\mathbf{z}$, $\mathbf{M}$, *and* $\mathbf{B}$ *be as in Lemma 4.3.16. Let* $\mathbf{u_B}, \mathbf{o_B}$ *be composite relaxations of* $\mathbf{B}$ *on* $X \times X \times \cdots \times X \times X \times P$. *Let* $\mathbf{x}^c, \mathbf{x}^C : P \to \mathbb{R}^{n_x}$ *be convex and concave relaxations of* $\mathbf{x}$ *on* P, *respectively, such that* $\mathbf{x}^c(\mathbf{p}) \leq \mathbf{z}(\mathbf{p}) \leq \mathbf{x}^C(\mathbf{p})$, $\forall \mathbf{p} \in P$. *Then the functions*

$$\mathbf{M}^c(\cdot) \equiv \mathbf{u_B}(\mathbf{x}^c(\cdot), \mathbf{x}^C(\cdot), \ldots, \mathbf{x}^c(\cdot), \mathbf{x}^C(\cdot), \cdot)$$
$$\mathbf{M}^C(\cdot) \equiv \mathbf{o_B}(\mathbf{x}^c(\cdot), \mathbf{x}^C(\cdot), \ldots, \mathbf{x}^c(\cdot), \mathbf{x}^C(\cdot), \cdot),$$

are convex and concave relaxations of $\mathbf{M}$ *on* P, *respectively.*

Proof. By Assumption 4.3.15-2, $\nabla_{\mathbf{x}} h_i$, $i = 1, \ldots, n_x$, is available and factorable. We know that for each $\mathbf{p} \in P$ and all $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_x$, either $x_j(\mathbf{p}) \leq y_j^i(\mathbf{p}) \leq z_j(\mathbf{p})$ or $z_j(\mathbf{p}) \leq y_j^i(\mathbf{p}) \leq x_j(\mathbf{p})$. Also, we have valid relaxations such that $\mathbf{x}^c(\mathbf{p}) \leq \mathbf{z}(\mathbf{p}) \leq \mathbf{x}^C(\mathbf{p})$, $\forall \mathbf{p} \in P$. Thus, it is clear $\mathbf{x}^c(\mathbf{p}) \leq \mathbf{y}^i(\mathbf{p}) \leq \mathbf{x}^C(\mathbf{p})$, $\forall \mathbf{p} \in P$ and $i = 1, \ldots, n_x$. Since $\mathbf{x}^c$ and $\mathbf{x}^C$ are convex and concave relaxations of $\mathbf{y}^i$ on P for $i = 1, \ldots, n_x$ by Definition 4.2.9, and $\mathbf{u_B}$ and $\mathbf{o_B}$ are composite relaxations of $\mathbf{B}$ on $X \times X \times \ldots \times X \times X \times P$, it follows directly that $\mathbf{M}^c$ and $\mathbf{M}^C$ are valid convex and concave relaxations of $\mathbf{M}$ on P. □

Two different techniques for constructing relaxations of solutions of parametric nonlinear systems, that rely on the above results, will now be presented along with very general composite relaxation results. The complete results and procedures regarding constructing relaxations of solutions of parametric nonlinear systems will then be presented.

Definition 4.3.18 ($\boldsymbol{\psi}$)**.** Let $\mathbf{b} : X \times P \to \mathbb{R}^{n_x}$ such that $\mathbf{b} \equiv \mathbf{Yh}$. Define the function $\boldsymbol{\psi} : X \times M \times X \times P \to \mathbb{R}^{n_x}$ such that $\forall (\tilde{\mathbf{z}}, \tilde{\mathbf{M}}, \tilde{\mathbf{x}}, \mathbf{p}) \in X \times M \times X \times P$,

$\boldsymbol{\psi}(\tilde{\mathbf{z}}, \tilde{\mathbf{M}}, \tilde{\mathbf{x}}, \mathbf{p}) = \tilde{\mathbf{x}}^*$, where the i^{th} component of $\tilde{\mathbf{x}}^*$ is given by the loop:

$$\texttt{for} \quad i = 1, \ldots, n_x \ \texttt{do}$$

$$\tilde{x}_i^* := \tilde{z}_i - \left(b_i(\tilde{\mathbf{z}}, \mathbf{p}) + \sum_{j<i} \tilde{m}_{ij}(\tilde{x}_j^* - \tilde{z}_j) + \sum_{j>i} \tilde{m}_{ij}(\tilde{x}_j - \tilde{z}_j) \right) / \tilde{m}_{ii}, \tag{4.12}$$

$$\texttt{end.}$$

This is simply a formal definition of a single iteration of the parametric version of the Gauss-Seidel method and is very closely related to the function $\mathbf{f}$ from the linear systems section above. The following result shows that if relaxations of $\mathbf{x}$ are known, they can be refined. Later, the full method, that is practical computationally, for refining relaxations of $\mathbf{x}$ iteratively is presented which relies on this result.

Theorem 4.3.19. *Let $\mathbf{z}$ and $\mathbf{M}$ be as in Lemma 4.3.16. Let $\mathbf{M}^c, \mathbf{M}^C : P \to \mathbb{R}^{n_x \times n_x}$ be relaxations of $\mathbf{M}$ on P, let $\mathbf{x}^{k,c}, \mathbf{x}^{k,C} : P \to \mathbb{R}^{n_x}$ be relaxations of $\mathbf{x}$ on P, and let $\mathbf{z}^c, \mathbf{z}^C : P \to \mathbb{R}^{n_x}$ be relaxations of $\mathbf{z}$ on P. Let $\mathbf{u}_\psi$ and $\mathbf{o}_\psi$ be composite relaxations of $\boldsymbol{\psi}$ on $X \times M \times X \times P$. Then convex and concave relaxations of $\mathbf{x}$ on P are given by*

$$\begin{aligned}
\mathbf{x}^{k+1,c}(\cdot) :=& \bar{\mathbf{u}}_\psi \left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot \right), \\
\mathbf{x}^{k+1,C}(\cdot) :=& \bar{\mathbf{o}}_\psi \left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot \right),
\end{aligned}$$

respectively, with $\bar{\mathbf{u}}_\psi$ and $\bar{\mathbf{o}}_\psi$ defined analogously to Def. 4.3.2.

Proof. Similar to the linear systems result above, we will show that $\mathbf{x}$ is a fixed-point of $\boldsymbol{\psi}$. By Lemma 4.3.16

$$\mathbf{M}(\mathbf{p})(\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})) = -\mathbf{Y}\mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p}), \ \forall \mathbf{p} \in P,$$

and $0 \notin M_{ii} \supset m_{ii}(P),\ \forall i$. Now, it is clear that, for $i = 1, \ldots, n_x$, we can write

$$x_i(\mathbf{p}) = z_i(\mathbf{p}) - \Bigg(b_i(\mathbf{z}(\mathbf{p}), \mathbf{p}) + \sum_{j<i} m_{ij}(\mathbf{p})(x_j(\mathbf{p}) - z_j(\mathbf{p})) + \sum_{j>i} m_{ij}(\mathbf{p})(x_j(\mathbf{p}) - z_j(\mathbf{p})) \Bigg) / m_{ii}(\mathbf{p})$$

with $\mathbf{b} = \mathbf{Yh}$. It immediately follows that

$$\begin{aligned} &\psi_1(\mathbf{z}(\mathbf{p}), \mathbf{M}(\mathbf{p}), \mathbf{x}(\mathbf{p}), \mathbf{p}) = x_1^*(\mathbf{p}) \\ &= z_1(\mathbf{p}) - \left(b_1(\mathbf{z}(\mathbf{p}), \mathbf{p}) + \sum_{j>1} m_{1j}(\mathbf{p})(x_j(\mathbf{p}) - z_j(\mathbf{p})) \right) / m_{11}(\mathbf{p}) = x_1(\mathbf{p}). \end{aligned}$$

Similar to the proof of Lemma 4.3.11, using induction, $x_i(\mathbf{p}) = \psi_i(\mathbf{z}(\mathbf{p}), \mathbf{M}(\mathbf{p}), \mathbf{x}(\mathbf{p}), \mathbf{p}) = x_i^*,\ \forall i$. Therefore $\mathbf{x}$ is a fixed-point of $\boldsymbol{\psi}$ for every $\mathbf{p} \in P$. From the hypothesis and Definition 4.2.9, it follows that

$$\begin{aligned} &\mathbf{u}_\psi \left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot \right), \\ &\mathbf{o}_\psi \left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot \right) \end{aligned}$$

are relaxations of $\boldsymbol{\psi}(\mathbf{z}(\cdot), \mathbf{M}(\cdot), \mathbf{x}(\cdot), \cdot)$ on P that are also relaxations of $\mathbf{x}(\cdot) = \boldsymbol{\psi}(\mathbf{z}(\cdot), \mathbf{M}(\cdot), \mathbf{x}(\cdot), \cdot)$ on P. It immediately follows that

$$\begin{aligned} \mathbf{x}^{k+1,c}(\cdot) :=& \bar{\mathbf{u}}_\psi \left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot \right), \\ \mathbf{x}^{k+1,C}(\cdot) :=& \bar{\mathbf{o}}_\psi \left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot \right), \end{aligned}$$

are convex and concave relaxations of $\mathbf{x}$ on P, respectively. □

As in Section 4.3.3, the sequential componentwise refinement of relaxations of $\mathbf{x}$ enable the calculations of subsequent components ($j > i$) to make use of the newly calculated refinements of the previous components ($j < i$).

Theorem 4.3.20. *Let* $\mathbf{z}$ *and* $\mathbf{M}$ *be as in Lemma 4.3.16. Let* $\mathbf{M}^c, \mathbf{M}^C : P \to \mathbb{R}^{n_x \times n_x}$ *be relaxations of* $\mathbf{M}$ *on* P*, let* $\mathbf{x}^{k,c}, \mathbf{x}^{k,C} : P \to \mathbb{R}^{n_x}$ *be relaxations of* $\mathbf{x}$ *on* P*, and let*

$\mathbf{z}^c, \mathbf{z}^C : P \to \mathbb{R}^{n_x}$ *be relaxations of* $\mathbf{z}$ *on* P*. Let* $\hat{\boldsymbol{\sigma}}^c_{\mathbf{M}}, \hat{\boldsymbol{\sigma}}^C_{\mathbf{M}} : P \to \mathbb{R}^{n_p \times n_x \times n_x}$ *be subgradients of* $\mathbf{M}^c$ *and* $\mathbf{M}^C$ *on* P*, respectively, let* $\boldsymbol{\sigma}^{k,c}_{\mathbf{x}}, \boldsymbol{\sigma}^{k,C}_{\mathbf{x}} : P \to \mathbb{R}^{n_p \times n_x}$ *be subgradients of* $\mathbf{x}^{k,c}$ *and* $\mathbf{x}^{k,C}$ *on* P*, respectively, and let* $\boldsymbol{\sigma}^c_{\mathbf{z}}, \boldsymbol{\sigma}^C_{\mathbf{z}} : P \to \mathbb{R}^{n_p \times n_x}$ *be subgradients of* $\mathbf{z}^c$ *and* $\mathbf{z}^C$ *on* P*, respectively. Let* $\mathcal{S}_{\mathbf{u}_\psi}$ *and* $\mathcal{S}_{\mathbf{o}_\psi}$ *be composite subgradients of* $\mathbf{u}_\psi$ *and* $\mathbf{o}_\psi$ *on* $X \times M \times X \times P$*, respectively. Then we have*

$$\begin{aligned}
\boldsymbol{\sigma}^{k+1,c}_{\mathbf{x}}(\cdot) :=& \mathcal{S}_{\bar{\mathbf{u}}_\psi}(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \boldsymbol{\sigma}^c_{\mathbf{z}}(\cdot), \boldsymbol{\sigma}^C_{\mathbf{z}}(\cdot), \\
& \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \hat{\boldsymbol{\sigma}}^c_{\mathbf{M}}(\cdot), \hat{\boldsymbol{\sigma}}^C_{\mathbf{M}}(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}^{k,c}_{\mathbf{x}}(\cdot), \boldsymbol{\sigma}^{k,C}_{\mathbf{x}}(\cdot), \cdot), \\
\boldsymbol{\sigma}^{k,C}_{\mathbf{x}}(\cdot) :=& \mathcal{S}_{\bar{\mathbf{o}}_\psi}(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \boldsymbol{\sigma}^c_{\mathbf{z}}(\cdot), \boldsymbol{\sigma}^C_{\mathbf{z}}(\cdot), \\
& \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \hat{\boldsymbol{\sigma}}^c_{\mathbf{M}}(\cdot), \hat{\boldsymbol{\sigma}}^C_{\mathbf{M}}(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}^{k,c}_{\mathbf{x}}(\cdot), \boldsymbol{\sigma}^{k,C}_{\mathbf{x}}(\cdot), \cdot)
\end{aligned}$$

are subgradients of

$$\begin{aligned}
\mathbf{x}^{k+1,c}(\cdot) &:= \bar{\mathbf{u}}_\psi\left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot\right) \\
\mathbf{x}^{k+1,C}(\cdot) &:= \bar{\mathbf{o}}_\psi\left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot\right)
\end{aligned}$$

on P*, with* $\mathcal{S}_{\bar{\mathbf{u}}_\mathbf{f}}$ *and* $\mathcal{S}_{\bar{\mathbf{o}}_\mathbf{f}}$ *defined analogously to Def. 4.3.3.*

Proof. The proof is analogous to that for Theorem 4.3.5. □

A second technique for constructing relaxations of solutions of parametric nonlinear systems will now be presented.

Definition 4.3.21 ($\boldsymbol{\chi}$)**.** The function $\boldsymbol{\chi} : X \times M \times X \times P \to \mathbb{R}^{n_x}$ will be defined as

$$\boldsymbol{\chi}(\tilde{\mathbf{z}}, \tilde{\mathbf{M}}, \tilde{\mathbf{x}}, \mathbf{p}) \equiv \tilde{\mathbf{z}} - \mathbf{Y}\mathbf{h}(\tilde{\mathbf{z}}, \mathbf{p}) + (\mathbf{I} - \tilde{\mathbf{M}})(\tilde{\mathbf{x}} - \tilde{\mathbf{z}}), \tag{4.13}$$

$\forall(\tilde{\mathbf{z}}, \tilde{\mathbf{M}}, \tilde{\mathbf{x}}, \mathbf{p}) \in X \times M \times X \times P$.

Theorem 4.3.22. *Let* $\mathbf{z}$ *and* $\mathbf{M}$ *be as in Lemma 4.3.16. Let* $\mathbf{M}^c, \mathbf{M}^C : P \to \mathbb{R}^{n_x \times n_x}$ *be relaxations of* $\mathbf{M}$ *on* P*, let* $\mathbf{x}^{k,c}, \mathbf{x}^{k,C} : P \to \mathbb{R}^{n_x}$ *be relaxations of* $\mathbf{x}$ *on* P*, and let* $\mathbf{z}^c, \mathbf{z}^C : P \to \mathbb{R}^{n_x}$ *be relaxations of* $\mathbf{z}$ *on* P*. Let* $\mathbf{u}_\chi$ *and* $\mathbf{o}_\chi$ *be composite relaxations of*

$\boldsymbol{\chi}$ *on* $X \times M \times X \times P$*. Then convex and concave relaxations of* $\mathbf{x}$ *on* P *are given by*

$$\begin{aligned}\mathbf{x}^{k+1,c}(\cdot) :=& \bar{\mathbf{u}}_\chi\left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot\right),\\ \mathbf{x}^{k+1,C}(\cdot) :=& \bar{\mathbf{o}}_\chi\left(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot\right),\end{aligned}$$

respectively, with $\bar{\mathbf{u}}_\chi$ *and* $\bar{\mathbf{o}}_\chi$ *defined analogously to Def. 4.3.2.*

Proof. First, we will show that $\mathbf{x}$ is a fixed-point of $\boldsymbol{\chi}$. By Proposition 4.3.7, we can write $\boldsymbol{\phi}(\mathbf{w}, \mathbf{p}) = \mathbf{w} - \mathbf{Y}\mathbf{h}(\mathbf{w}, \mathbf{p})$ so that $\boldsymbol{\phi}(\mathbf{w}, \mathbf{p}) = \mathbf{w} \Leftrightarrow \mathbf{h}(\mathbf{w}, \mathbf{p}) = \mathbf{0}$. Now,

$$\begin{aligned}\boldsymbol{\phi}(\mathbf{x}(\mathbf{p}), \mathbf{p}) &= \boldsymbol{\phi}(\mathbf{x}(\mathbf{p}), \mathbf{p}) + \boldsymbol{\phi}(\mathbf{z}(\mathbf{p}), \mathbf{p}) - \boldsymbol{\phi}(\mathbf{z}(\mathbf{p}), \mathbf{p}),\\ &= \mathbf{x}(\mathbf{p}) - \mathbf{Y}\mathbf{h}(\mathbf{x}(\mathbf{p}), \mathbf{p}) + \mathbf{z}(\mathbf{p}) - \mathbf{Y}\mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p}) - \mathbf{z}(\mathbf{p}) + \mathbf{Y}\mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p}),\\ &= \mathbf{z}(\mathbf{p}) - \mathbf{Y}\mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p}) + (\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})) - \mathbf{Y}(\mathbf{h}(\mathbf{x}(\mathbf{p}), \mathbf{p}) - \mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p})),\end{aligned}$$

for all $\mathbf{p} \in P$. From the definition of $\mathbf{M}$ and $\mathbf{z}$, $\mathbf{Y}(\mathbf{h}(\mathbf{x}(\mathbf{p}), \mathbf{p}) - \mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p})) = \mathbf{M}(\mathbf{p})(\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p}))$ holds. Substituting in we get

$$\begin{aligned}\boldsymbol{\phi}(\mathbf{x}(\mathbf{p}), \mathbf{p}) &= \mathbf{z}(\mathbf{p}) - \mathbf{Y}\mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p}) + (\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})) - \mathbf{M}(\mathbf{p})(\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})),\\ &= \mathbf{z}(\mathbf{p}) - \mathbf{Y}\mathbf{h}(\mathbf{z}(\mathbf{p}), \mathbf{p}) + (\mathbf{I} - \mathbf{M}(\mathbf{p}))(\mathbf{x}(\mathbf{p}) - \mathbf{z}(\mathbf{p})),\\ &= \boldsymbol{\chi}\left(\mathbf{z}(\mathbf{p}), \mathbf{M}(\mathbf{p}), \mathbf{x}(\mathbf{p})\right).\end{aligned}$$

Thus, $\mathbf{x}(\mathbf{p}) = \boldsymbol{\phi}(\mathbf{x}(\mathbf{p}), \mathbf{p}) = \boldsymbol{\chi}(\mathbf{z}(\mathbf{p}), \mathbf{M}(\mathbf{p}), \mathbf{x}(\mathbf{p}))$. From the hypothesis and by Definition 4.2.9, it follows that

$$\begin{aligned}&\mathbf{u}_\chi(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot),\\ &\mathbf{o}_\chi(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot),\end{aligned}$$

are relaxations of $\boldsymbol{\chi}(\mathbf{z}(\cdot), \mathbf{M}(\cdot), \mathbf{x}(\cdot), \cdot)$ on P that are also relaxations of

$\mathbf{x}(\cdot) = \boldsymbol{\chi}(\mathbf{z}(\cdot), \mathbf{M}(\cdot), \mathbf{x}(\cdot), \cdot)$ on P. It immediately follows that

$$\begin{aligned}\mathbf{x}^{k+1,c}(\cdot) :=& \bar{\mathbf{u}}_\chi(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot),\\ \mathbf{x}^{k+1,C}(\cdot) :=& \bar{\mathbf{o}}_\chi(\mathbf{z}^c(\cdot), \mathbf{z}^C(\cdot), \mathbf{M}^c(\cdot), \mathbf{M}^C(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot),\end{aligned}$$

are convex and concave relaxations of $\mathbf{x}$ on P, respectively. □

Remark 4.3.23. Similar to how $\boldsymbol{\psi}$, from above, and $\mathbf{f}$ from Section 4.3.3 were defined, it is easy to rearrange $\boldsymbol{\chi}$ to be calculated in a sequential componentwise fashion.

Remark 4.3.24. The subgradient result for $\boldsymbol{\psi}$, Theorem 4.3.20, trivially holds with $\boldsymbol{\psi}$ replaced by $\boldsymbol{\chi}$.

One hypothesis that the above results rely upon is the existence of an appropriate function $\mathbf{z} : P \to X$, for which relaxations are readily available. Without such a function, convex and concave relaxations of $\mathbf{x}$ that are potential improvements on the initial bounds cannot be calculated. This issue is addressed next.

Definition 4.3.25. Let $\mathbf{x}^a, \mathbf{x}^A : P \to \mathbb{R}^{n_x}$ be any affine relaxations of $\mathbf{x}$ on P, respectively. For some $\lambda \in [0,1]$ define the function $\mathbf{z} : P \to \mathbb{R}^{n_x}$ with the following procedure:

```
for  i = 1, ..., n_x  do
     ξ_i(·) := λ x_i^a(·) + (1 − λ) x_i^A(·)
     Ξ_i := [ξ_i^L, ξ_i^U] = [ min_{p∈P} ξ_i(p), max_{p∈P} ξ_i(p) ]
     if  ξ_i^L < x_i^L  then
          x̂_i^a(·) := x_i^L,  else  x̂_i^a(·) := x_i^a(·)
     if  ξ_i^U > x_i^U  then
          x̂_i^A(·) := x_i^U,  else  x̂_i^A(·) := x_i^A(·)
     z_i(·) := λ x̂_i^a(·) + (1 − λ) x̂_i^A(·)
end
```

It should be noted that the interval $\Xi_i = \left[\min_{\mathbf{p}\in P} \xi(\mathbf{p}), \max_{\mathbf{p}\in P} \xi(\mathbf{p})\right]$ can be calculated easily and efficiently for each i using interval analysis. Also, defining $\mathbf{z}$ to be affine is important because affine functions are trivially convex *and* concave, so that the calculation of valid relaxations is trivial.

Lemma 4.3.26. *Suppose $\mathbf{x}^a, \mathbf{x}^A : P \to \mathbb{R}^{n_x}$ are any affine relaxations of $\mathbf{x}$ on P. Then the function $\mathbf{z} : P \to \mathbb{R}^{n_x}$, defined in Definition 4.3.25, is affine and maps P into X.*

Proof. Consider a single i and set $\Xi_i := [\xi_i^L, \xi_i^U]$ as in Definition 4.3.25. It should be noted that the cases where $x_i^A(\mathbf{p}) \leq x_i^L$ and/or $x_i^a(\mathbf{p}) \geq x_i^U$ for any $\mathbf{p} \in P$ cannot occur since, by definition $x_i^a(\mathbf{p}) \leq x_i(\mathbf{p}) \leq \mathbf{x}_i^A(\mathbf{p}),\ \forall \mathbf{p} \in P$, implying $x_i(\mathbf{p}) \leq x_i^L$ and/or $x_i(\mathbf{p}) \geq x_i^U$, violating Assumption 4.3.15-1. First, consider the case that $x_i^L \leq \xi_i^L$ and $\xi_i^U \leq x_i^U$. Trivially, $z_i(\cdot) := \lambda x_i^a(\cdot) + (1-\lambda)x_i^A(\cdot)$ satisfies $x_i^L \leq z_i(\cdot) \leq x_i^U$, $\forall \mathbf{p} \in P$, and thus z_i maps P into X_i and since it is a convex combination of affine functions, it is affine. Next, consider the case that $\xi_i^L < x_i^L$ and $x_i^U < \xi_i^U$. Then $z_i(\cdot) := \lambda x_i^L + (1-\lambda)x_i^U$ maps P into X_i, trivially, and since it is a convex combination of two affine (constant) functions, it is affine. Consider the case that only one bound is violated, say $\xi_i^L < x_i^L$ and $\xi_i^U \leq x_i^U$. Then $z_i(\cdot) := \lambda x_i^L + (1-\lambda)x_i^A(\cdot)$ and since x_i^L is affine (constant) and x_i^A is affine, z_i is affine and $x_i^L \leq \lambda x_i^L + (1-\lambda)x_i^A(\cdot)$. A similar argument can be made for the case in which the upper bound is violated: $\xi_i^U > x_i^U$ and $x_i^L \leq \xi_i^L$. Therefore $\mathbf{z}$ is affine and maps P into X. □

The `if` statements in Definition 4.3.25 check, for a particular choice of λ, whether or not the hyperplanes defined by $\lambda\mathbf{x}^a(\cdot) + (1-\lambda)\mathbf{x}^A(\cdot)$ will violate the bounds on X for some i^{th} component. If that is the case, the hyperplane is calculated so as to not violate the bounds on X. A convenient choice for the i^{th} hyperplane is simply the plane that lies in the middle corresponding to $\lambda = 0.5$. Other choices for $\mathbf{z}$ exist. For instance, in one dimension, the function z can be taken to be the secant connecting the endpoints $x(p^L)$ and $x(p^U)$. The above result together with the definition of the composite subgradient, (Def. 4.2.15), offers an automatic way to calculate $\mathbf{z}$ that is

valid for *all* systems in general. In order to simplify the notation for later results, the following procedure will be defined.

Subroutine 4.3.27 (`Aff`)**.**

$$
\begin{aligned}
&\texttt{Aff}(\mathbf{c}, \mathbf{C}, \boldsymbol{\sigma}_{\mathbf{c}}, \boldsymbol{\sigma}_{\mathbf{C}}, \lambda, X, P, \bar{\mathbf{p}})\{\\
&\texttt{for} \quad i = 1, \ldots, n_x \ \texttt{do}\\
&\qquad X_i^a := c_i + \sum_{j=1}^{n_p} (\boldsymbol{\sigma}_{\mathbf{c}}{}^{\mathrm{T}})_{ij} (P_j - \bar{p}_j)\\
&\qquad X_i^A := C_i + \sum_{j=1}^{n_p} (\boldsymbol{\sigma}_{\mathbf{C}}{}^{\mathrm{T}})_{ij} (P_j - \bar{p}_j)\\
&\qquad \Xi_i := \lambda X_i^a + (1-\lambda) X_i^A\\
&\qquad \texttt{if} \ \ \xi_i^L < x_i^L \ \ \texttt{then}\\
&\qquad\qquad (\boldsymbol{\sigma}_{\mathbf{c}})_{ji} := 0, \ \forall j = 1, \ldots, n_p\\
&\qquad\qquad\qquad c_i := x_i^L \ \ \texttt{endif}\\
&\qquad \texttt{if} \ \ \xi_i^U > x_i^U \ \ \texttt{then}\\
&\qquad\qquad (\boldsymbol{\sigma}_{\mathbf{C}})_{ji} := 0, \ \forall j = 1, \ldots, n_p\\
&\qquad\qquad\qquad C_i := x_i^U \ \ \texttt{endif}\\
&\texttt{end}\\
&\texttt{return} \ \{\mathbf{c}, \mathbf{C}, \boldsymbol{\sigma}_c, \boldsymbol{\sigma}_C\}\\
&\}
\end{aligned}
$$

Remark 4.3.28. Note that the first three computations in Subroutine 4.3.27 are interval computations performed using interval analysis.

Theorem 4.3.29. *Let* $\mathbf{x}^{0,c}, \mathbf{x}^{0,C} : P \to \mathbb{R}^{n_x}$ *be defined as* $\mathbf{x}^{0,c}(\mathbf{p}) = \mathbf{x}^L$ *and* $\mathbf{x}^{0,C}(\mathbf{p}) = \mathbf{x}^U$ *for every* $\mathbf{p} \in P$*. Let* $\boldsymbol{\sigma}_{\mathbf{x}}^{0,c}, \boldsymbol{\sigma}_{\mathbf{x}}^{0,C} : P \to \mathbb{R}^{n_p \times n_x}$ *be defined as* $\boldsymbol{\sigma}_{\mathbf{x}}^{0,c}(\mathbf{p}), \boldsymbol{\sigma}_{\mathbf{x}}^{0,C}(\mathbf{p}) = \mathbf{0}$ *for every* $\mathbf{p} \in P$*. Let* $\mathbf{u_B}, \mathbf{o_B}$ *be composite relaxations of* $\mathbf{B}$ *on* $X \times \ldots \times X \times P$ *and* $\bar{\mathbf{u}}_\psi, \bar{\mathbf{o}}_\psi$ *be composite relaxations of* $\boldsymbol{\psi}$ *on* $X \times M \times X \times P$*. Let* $\mathcal{S}_{\mathbf{u}_B}, \mathcal{S}_{\mathbf{o}_B}$ *be composite subgradients of* $\mathbf{u_B}, \mathbf{o_B}$*, respectively. Then, for any choice of*

$\{\bar{\mathbf{p}}^k\}$, and $\{\lambda^k\}$ with $\bar{\mathbf{p}}^k \in P$ and $\lambda^k \in [0,1]$ for $k \in \mathbb{N}$, the elements of the sequences $\{\mathbf{x}^{k,c}\}$ and $\{\mathbf{x}^{k,C}\}$ defined by the iteration:

$$
\begin{aligned}
(\mathbf{c}, \mathbf{C}, \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}) &:= \mathtt{Aff}(\mathbf{x}^{k,c}(\bar{\mathbf{p}}^k), \mathbf{x}^{k,C}(\bar{\mathbf{p}}^k), \boldsymbol{\sigma}_\mathbf{x}^{k,c}(\bar{\mathbf{p}}^k), \boldsymbol{\sigma}_\mathbf{x}^{k,C}(\bar{\mathbf{p}}^k), \lambda^k, X, P, \bar{\mathbf{p}}^k) \\
\mathbf{x}^{k,a}(\mathbf{p}) &:= \mathbf{c} + (\boldsymbol{\sigma}_\mathbf{c})^\mathrm{T}(\mathbf{p} - \bar{\mathbf{p}}^k), \quad \forall \mathbf{p} \in P \\
\mathbf{x}^{k,A}(\mathbf{p}) &:= \mathbf{C} + (\boldsymbol{\sigma}_\mathbf{C})^\mathrm{T}(\mathbf{p} - \bar{\mathbf{p}}^k), \quad \forall \mathbf{p} \in P \\
\mathbf{z}^k(\cdot) &:= \lambda^k \mathbf{x}^{k,a}(\cdot) + (1 - \lambda^k)\mathbf{x}^{k,A}(\cdot) \\
\boldsymbol{\sigma}_\mathbf{z}^k &:= \lambda^k \boldsymbol{\sigma}_\mathbf{c} + (1 - \lambda^k)\boldsymbol{\sigma}_\mathbf{C} \\
\mathbf{M}^{k,c}(\cdot) &:= \mathbf{u}_\mathbf{B}(\mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \ldots, \mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \cdot) \\
\mathbf{M}^{k,C}(\cdot) &:= \mathbf{o}_\mathbf{B}(\mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \ldots, \mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \cdot) \\
\hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,c}(\cdot) &:= \mathcal{S}_{u_B}\left(\mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}, \ldots, \mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}, \cdot\right) \\
\hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,C}(\cdot) &:= \mathcal{S}_{o_B}\left(\mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}, \ldots, \mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}, \cdot\right) \\
\mathbf{x}^{k+1,c}(\cdot) &:= \bar{\mathbf{u}}_\psi\left(\mathbf{z}^k(\cdot), \mathbf{z}^k(\cdot), \mathbf{M}^{k,c}(\cdot), \mathbf{M}^{k,C}(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot\right) \\
\mathbf{x}^{k+1,C}(\cdot) &:= \bar{\mathbf{o}}_\psi\left(\mathbf{z}^k(\cdot), \mathbf{z}^k(\cdot), \mathbf{M}^{k,c}(\cdot), \mathbf{M}^{k,C}(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot\right) \\
\boldsymbol{\sigma}_\mathbf{x}^{k+1,c}(\cdot) &:= \mathcal{S}_{\bar{\mathbf{u}}_\psi}\Big(\mathbf{z}^k(\cdot), \mathbf{z}^k(\cdot), \boldsymbol{\sigma}_\mathbf{z}^k, \boldsymbol{\sigma}_\mathbf{z}^k, \mathbf{M}^{k,c}(\cdot), \mathbf{M}^{k,C}(\cdot), \hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,c}(\cdot), \hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,C}(\cdot), \\
&\qquad \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_\mathbf{x}^{k,c}(\cdot), \boldsymbol{\sigma}_\mathbf{x}^{k,C}(\cdot), \cdot\Big) \\
\boldsymbol{\sigma}_\mathbf{x}^{k+1,C}(\cdot) &:= \mathcal{S}_{\bar{\mathbf{o}}_\psi}\Big(\mathbf{z}^k(\cdot), \mathbf{z}^k(\cdot), \boldsymbol{\sigma}_\mathbf{z}^k, \boldsymbol{\sigma}_\mathbf{z}^k, \mathbf{M}^{k,c}(\cdot), \mathbf{M}^{k,C}(\cdot), \hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,c}(\cdot), \hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,C}(\cdot), \\
&\qquad \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_\mathbf{x}^{k,c}(\cdot), \boldsymbol{\sigma}_\mathbf{x}^{k,C}(\cdot), \cdot\Big)
\end{aligned}
$$

are convex and concave relaxations of $\mathbf{x}$ on P, respectively.

Proof. By definition, $\mathbf{x}^{0,c}$ and $\mathbf{x}^{0,C}$ are, respectively, convex and concave relaxations of $\mathbf{x}$ on P. Similarly, $\boldsymbol{\sigma}_\mathbf{x}^{0,c}$ and $\boldsymbol{\sigma}_\mathbf{x}^{0,C}$ are subgradients of $\mathbf{x}^{0,c}$ and $\mathbf{x}^{0,C}$ on P, respectively. Suppose this holds for arbitrary $k \in \mathbb{N}$. Then $\mathbf{x}^{k,c}$ and $\mathbf{x}^{k,C}$ are, respectively, convex and concave relaxations of $\mathbf{x}$ on P and $\boldsymbol{\sigma}_\mathbf{x}^{k,c}$ and $\boldsymbol{\sigma}_\mathbf{x}^{k,C}$ are subgradients on P. Then it follows from the definition of Subroutine 4.3.27 that $\mathbf{x}^{k,a}$ and $\mathbf{x}^{k,A}$ are affine relaxations of $\mathbf{x}$ on P. Furthermore, $\mathbf{z}^k$ is affine and maps into X by Lemma 4.3.26. From the definition of $\mathbf{z}^k$, it is clear that $\mathbf{x}^{k,a}(\mathbf{p}) \leq \mathbf{z}^k(\mathbf{p}) \leq \mathbf{x}^{k,A}(\mathbf{p}),\ \forall \mathbf{p} \in P$, which implies that

$\mathbf{M}^{k,c}$ and $\mathbf{M}^{k,C}$ are relaxations of $\mathbf{M}$ on P by Lemma 4.3.17. Moreover, $\boldsymbol{\sigma}_{\mathbf{c}}$ and $\boldsymbol{\sigma}_{\mathbf{C}}$ are subgradients of $\mathbf{x}^{k,a}$ and $\mathbf{x}^{k,A}$, respectively, so that $\hat{\boldsymbol{\sigma}}^{k,c}_{\mathbf{M}}$ and $\hat{\boldsymbol{\sigma}}^{k,C}_{\mathbf{M}}$ are subgradients of $\mathbf{M}^{k,c}$ and $\mathbf{M}^{k,C}$ on P, respectively, by Definition 4.2.15. By Theorem 4.3.19,

$$\begin{aligned}
\mathbf{x}^{k+1,c}(\cdot) &:= \bar{\mathbf{u}}_{\psi}\left(\mathbf{z}^{k}(\cdot),\mathbf{z}^{k}(\cdot),\mathbf{M}^{k,c}(\cdot),\mathbf{M}^{k,C}(\cdot),\mathbf{x}^{k,c}(\cdot),\mathbf{x}^{k,C}(\cdot),\cdot\right)\\
\mathbf{x}^{k+1,C}(\cdot) &:= \bar{\mathbf{o}}_{\psi}\left(\mathbf{z}^{k}(\cdot),\mathbf{z}^{k}(\cdot),\mathbf{M}^{k,c}(\cdot),\mathbf{M}^{k,C}(\cdot),\mathbf{x}^{k,c}(\cdot),\mathbf{x}^{k,C}(\cdot),\cdot\right)
\end{aligned}$$

are relaxations of $\mathbf{x}$ on P and by Definition 4.3.3 and Theorem 4.3.20

$$\begin{aligned}
\boldsymbol{\sigma}^{k+1,c}_{\mathbf{x}}(\cdot) := \mathcal{S}_{\bar{\mathbf{u}}_{\psi}}\Big(&\mathbf{z}^{k}(\cdot),\mathbf{z}^{k}(\cdot),\boldsymbol{\sigma}^{k}_{\mathbf{z}},\boldsymbol{\sigma}^{k}_{\mathbf{z}},\mathbf{M}^{k,c}(\cdot),\mathbf{M}^{k,C}(\cdot),\hat{\boldsymbol{\sigma}}^{k,c}_{\mathbf{M}}(\cdot),\hat{\boldsymbol{\sigma}}^{k,C}_{\mathbf{M}}(\cdot),\\
&\mathbf{x}^{k,c}(\cdot),\mathbf{x}^{k,C}(\cdot),\boldsymbol{\sigma}^{k,c}_{\mathbf{x}}(\cdot),\boldsymbol{\sigma}^{k,C}_{\mathbf{x}}(\cdot),\cdot\Big)\\
\boldsymbol{\sigma}^{k+1,C}_{\mathbf{x}}(\cdot) := \mathcal{S}_{\bar{\mathbf{o}}_{\psi}}\Big(&\mathbf{z}^{k}(\cdot),\mathbf{z}^{k}(\cdot),\boldsymbol{\sigma}^{k}_{\mathbf{z}},\boldsymbol{\sigma}^{k}_{\mathbf{z}},\mathbf{M}^{k,c}(\cdot),\mathbf{M}^{k,C}(\cdot),\hat{\boldsymbol{\sigma}}^{k,c}_{\mathbf{M}}(\cdot),\hat{\boldsymbol{\sigma}}^{k,C}_{\mathbf{M}}(\cdot),\\
&\mathbf{x}^{k,c}(\cdot),\mathbf{x}^{k,C}(\cdot),\boldsymbol{\sigma}^{k,c}_{\mathbf{x}}(\cdot),\boldsymbol{\sigma}^{k,C}_{\mathbf{x}}(\cdot),\cdot\Big)
\end{aligned}$$

are subgradients of $\mathbf{x}^{k+1,c}$ and $\mathbf{x}^{k+1,C}$, respectively. Induction completes the proof. □

Therefore, the iterations outlined in the above theorem can be regarded as methods for potentially refining the calculated bounds on $\mathbf{x}$ or any other initial convex and concave bounds on $\mathbf{x}$. However, the above theorem does not guarantee that the calculated convex and concave relaxations will in fact always be improvements on the initial bounds. Nevertheless, the theorem is important because it does offer a way to calculate relaxations that are no worse than the original bounds and potentially tighter, unlike the situation discussed in Theorem 4.3.8. To illustrate relaxations constructed using this result, the following simple example is given.

Example 4.3.30. Consider the system $h(z,p) = z^2 + pz + 4$ with $p \in P = [6,9]$. The two real roots are given by the quadratic formula. Using the parametric interval-Newton method [56, 101], two conservative intervals, $X^1 = [-0.78, -0.4]$ and $X^2 = [-10.0, -5.0]$, were calculated that are guaranteed to each contain a unique solution $x(p)$ such that $h(x(p),p) = 0,\ \forall p \in P$. Three different z functions were used, each corresponding to a different $\lambda^k = \lambda$ value, and convex and concave relaxations of $x(p)$

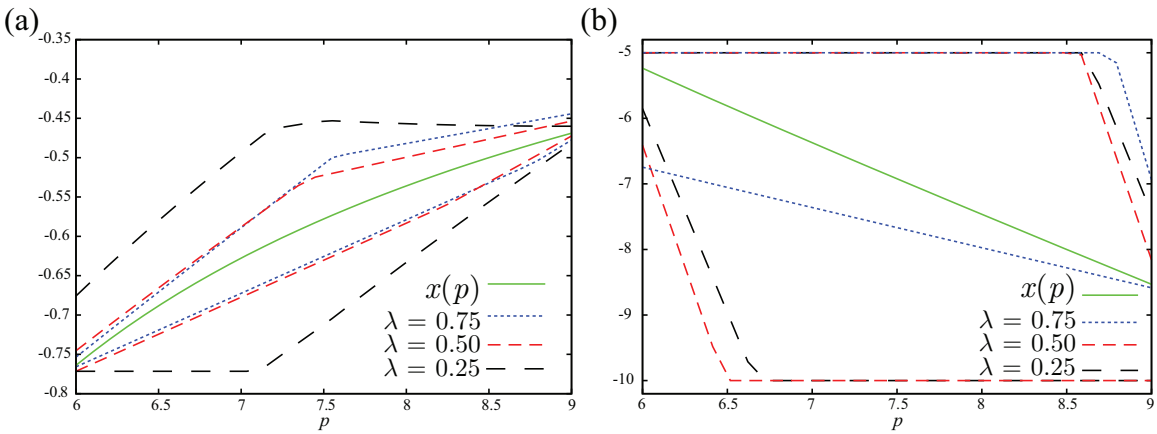

Figure 4-1: Relaxations of the solution in (a) X^1 and (b) X^2 for the relaxations of implicit functions simple example.

were constructed. For each λ, $\bar{p}^k = \bar{p}$ was chosen to be the midpoint of P. Figure 4-1 shows the relaxations for the two solutions corresponding to each λ value after applying two iterations of the procedure, after which, no significant refinements could be made.

Another method for refining the bounds of an implicit function through McCormick relaxations can be derived from Theorem 4.3.22. The following method is the analog to the Krawczyk interval method for bounding solutions of nonlinear systems.

Theorem 4.3.31. *Let* $\mathbf{x}^{0,c}, \mathbf{x}^{0,C} : P \to \mathbb{R}^{n_x}$ *be defined as* $\mathbf{x}^{0,c}(\mathbf{p}) = \mathbf{x}^L$ *and* $\mathbf{x}^{0,C}(\mathbf{p}) = \mathbf{x}^U$ *for every* $\mathbf{p} \in P$. *Let* $\boldsymbol{\sigma}^{0,c}_{\mathbf{x}}, \boldsymbol{\sigma}^{0,C}_{\mathbf{x}} : P \to \mathbb{R}^{n_p \times n_x}$ *be defined as* $\boldsymbol{\sigma}^{0,c}_{\mathbf{x}}(\mathbf{p}) = \boldsymbol{\sigma}^{0,C}_{\mathbf{x}}(\mathbf{p}) = \mathbf{0}$ *for every* $\mathbf{p} \in P$. *Let* $\mathbf{u_B}, \mathbf{o_B}$ *be composite relaxations of* $\mathbf{B}$ *on* $X \times \ldots \times X \times P$ *and* $\bar{\mathbf{u}}_\chi, \bar{\mathbf{o}}_\chi$ *be composite relaxations of* $\boldsymbol{\chi}$ *on* $X \times M \times X \times P$. *Let* $\mathcal{S}_{\mathbf{u}_B}, \mathcal{S}_{\mathbf{o}_B}$ *be composite subgradients of* $\mathbf{u_B}, \mathbf{o_B}$, *respectively. Then, for any choice of* $\{\bar{\mathbf{p}}^k\}$, *and* $\{\lambda^k\}$ *with* $\bar{\mathbf{p}}^k \in P$ *and* $\lambda^k \in [0,1]$ *for* $k \in \mathbb{N}$, *the elements of the sequences* $\{\mathbf{x}^{k,c}\}$ *and* $\{\mathbf{x}^{k,C}\}$

defined by the iteration:

$$
\begin{aligned}
(\mathbf{c}, \mathbf{C}, \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}) &:= \texttt{Aff}(\mathbf{x}^{k,c}(\bar{\mathbf{p}}^k), \mathbf{x}^{k,C}(\bar{\mathbf{p}}^k), \boldsymbol{\sigma}_\mathbf{x}^{k,c}(\bar{\mathbf{p}}^k), \boldsymbol{\sigma}_\mathbf{x}^{k,C}(\bar{\mathbf{p}}^k), \lambda^k, X, P, \bar{\mathbf{p}}^k) \\
\mathbf{x}^{k,a}(\mathbf{p}) &:= \mathbf{c} + (\boldsymbol{\sigma}_\mathbf{c})^\mathrm{T}(\mathbf{p} - \bar{\mathbf{p}}^k), \quad \forall \mathbf{p} \in P \\
\mathbf{x}^{k,A}(\mathbf{p}) &:= \mathbf{C} + (\boldsymbol{\sigma}_\mathbf{C})^\mathrm{T}(\mathbf{p} - \bar{\mathbf{p}}^k), \quad \forall \mathbf{p} \in P \\
\mathbf{z}^k(\cdot) &:= \lambda^k \mathbf{x}^{k,a}(\cdot) + (1 - \lambda^k)\mathbf{x}^{k,A}(\cdot) \\
\boldsymbol{\sigma}_\mathbf{z}^k &:= \lambda^k \boldsymbol{\sigma}_\mathbf{c} + (1 - \lambda^k)\boldsymbol{\sigma}_\mathbf{C} \\
\mathbf{M}^{k,c}(\cdot) &:= \mathbf{u}_\mathbf{B}(\mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \ldots, \mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \cdot) \\
\mathbf{M}^{k,C}(\cdot) &:= \mathbf{o}_\mathbf{B}(\mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \ldots, \mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \cdot) \\
\hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,c}(\cdot) &:= \mathcal{S}_{u_B}\left(\mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}, \ldots, \mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}, \cdot\right) \\
\hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,C}(\cdot) &:= \mathcal{S}_{o_B}\left(\mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}, \ldots, \mathbf{x}^{k,a}(\cdot), \mathbf{x}^{k,A}(\cdot), \boldsymbol{\sigma}_\mathbf{c}, \boldsymbol{\sigma}_\mathbf{C}, \cdot\right) \\
\mathbf{x}^{k+1,c}(\cdot) &:= \bar{\mathbf{u}}_\chi\left(\mathbf{z}^k(\cdot), \mathbf{z}^k(\cdot), \mathbf{M}^{k,c}(\cdot), \mathbf{M}^{k,C}(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot\right) \\
\mathbf{x}^{k+1,C}(\cdot) &:= \bar{\mathbf{o}}_\chi\left(\mathbf{z}^k(\cdot), \mathbf{z}^k(\cdot), \mathbf{M}^{k,c}(\cdot), \mathbf{M}^{k,C}(\cdot), \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \cdot\right) \\
\boldsymbol{\sigma}_\mathbf{x}^{k+1,c}(\cdot) &:= \mathcal{S}_{\bar{\mathbf{u}}_\chi}\Big(\mathbf{z}^k(\cdot), \mathbf{z}^k(\cdot), \boldsymbol{\sigma}_\mathbf{z}^k, \boldsymbol{\sigma}_\mathbf{z}^k, \mathbf{M}^{k,c}(\cdot), \mathbf{M}^{k,C}(\cdot), \hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,c}(\cdot), \hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,C}(\cdot), \\
&\qquad \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_\mathbf{x}^{k,c}(\cdot), \boldsymbol{\sigma}_\mathbf{x}^{k,C}(\cdot), \cdot\Big) \\
\boldsymbol{\sigma}_\mathbf{x}^{k+1,C}(\cdot) &:= \mathcal{S}_{\bar{\mathbf{o}}_\chi}\Big(\mathbf{z}^k(\cdot), \mathbf{z}^k(\cdot), \boldsymbol{\sigma}_\mathbf{z}^k, \boldsymbol{\sigma}_\mathbf{z}^k, \mathbf{M}^{k,c}(\cdot), \mathbf{M}^{k,C}(\cdot), \hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,c}(\cdot), \hat{\boldsymbol{\sigma}}_\mathbf{M}^{k,C}(\cdot), \\
&\qquad \mathbf{x}^{k,c}(\cdot), \mathbf{x}^{k,C}(\cdot), \boldsymbol{\sigma}_\mathbf{x}^{k,c}(\cdot), \boldsymbol{\sigma}_\mathbf{x}^{k,C}(\cdot), \cdot\Big)
\end{aligned}
$$

are convex and concave relaxations of $\mathbf{x}$ *on* P*, respectively, for* $k \in \mathbb{N}$*.*

Proof. The proof is analogous to the proof of Theorem 4.3.29. □

Remark 4.3.32. There are many alternative implementations of the iterations in Theorems 4.3.29 and 4.3.31. Computationally, evaluating the relaxations constructed using the iterations in Theorems 4.3.29 and 4.3.31 can only be done at a single $\mathbf{p}$. In order to accomplish this, relaxations at $\bar{\mathbf{p}}^k$ must first be computed. Therefore, one such alternative implementation is to choose a single $\bar{\mathbf{p}}^k = \bar{\mathbf{p}} \in P$ and apply one of the iterations to get affine relaxation information, and subsequently, use this information to define the $\mathbf{z}^k$ function. With this information calculated up front, the first 9 instructions are no longer dependent on the iteration k.

4.4 Global Optimization of Implicit Functions

The continuous *Branch-and-Bound* (B&B) framework is a popular deterministic algorithm for solving globally nonconvex NLPs as in (4.1). It is discussed in [59, 63] thoroughly. The B&B algorithm relies on refining bounds on the global optima while rigorously ruling out potentially large regions of the search space where global optima are guaranteed not to lie, termed fathoming. The algorithm is guaranteed to terminate in finitely many iterations when ϵ-tolerance has been reached. B&B will be employed here to solve programs with embedded implicit functions, as in (4.4), in a similar fashion. In fact, the B&B algorithm will be applied to (4.4) without modifying any of its underlying features or procedures. Therefore, the only difference between the B&B algorithm presented here and the B&B algorithm for standard form global optimization problems, is simply how the functions involved are evaluated and how their relaxations are calculated. Before presenting the full algorithm, the NLP subproblems, on which it relies, will be discussed.

4.4.1 Upper-Bounding Problem

Given a subinterval, P^l, of the decision space P, define the upper-bounding problem:

$$\begin{aligned} \min_{\mathbf{z}\in X, \mathbf{p}\in P^l} \; & f(\mathbf{z},\mathbf{p}) \\ \text{s.t.} \; & \mathbf{g}(\mathbf{z},\mathbf{p}) \leq \mathbf{0}. \\ & \mathbf{h}(\mathbf{z},\mathbf{p}) = \mathbf{0} \end{aligned} \tag{4.14}$$

This problem is solved locally to obtain a local solution $(\hat{\mathbf{z}}^l, \hat{\mathbf{p}}^l)$, if one exists. Lastly, a valid upper bound on the optimal solution value will be defined as $f_l^{UBD} \equiv f(\hat{\mathbf{z}}^l, \hat{\mathbf{p}}^l)$.

4.4.2 Lower-Bounding Problem

Given a subinterval, P^l, of the decision space P, define the lower-bounding problem:

$$f_l^{LBD} = \min_{\mathbf{p}\in P^l} f^c(\mathbf{p}) = \mathrm{u}_f(\mathbf{x}^c(\mathbf{p}), \mathbf{x}^C(\mathbf{p}), \mathbf{p}) \tag{4.15}$$
$$\text{s.t. } \mathbf{g}^c(\mathbf{p}) = \mathbf{u}_\mathbf{g}(\mathbf{x}^c(\mathbf{p}), \mathbf{x}^C(\mathbf{p}), \mathbf{p}) \leq \mathbf{0},$$

where the composite relaxations u_f and $\mathbf{u}_\mathbf{g}$ will be constructed by first using the procedures outlined in Section 3 for constructing convex and concave relaxations of the implicit function $\mathbf{x}$ on P^l and then applying the rules of generalized McCormick relaxations for composition. The lower-bounding problem (4.15) is convex by construction and is solved to global optimality. Denote the solution found by $\check{\mathbf{p}}$, if it exists, and let $f_l^{LBD} \equiv \mathrm{u}_f(\mathbf{x}^c(\check{\mathbf{p}}), \mathbf{x}^C(\check{\mathbf{p}}), \check{\mathbf{p}})$.

4.4.3 Global Optimization Algorithm

Algorithm 4.1 (Global Optimization of Implicit Functions)**.**

1. **(Initialization)**
   (a) *Set* $\Sigma = \{P\}$.
   (b) *Set* $k := 0$, $\epsilon_{\text{tol}} > 0$, $\alpha_0 = +\infty$, $\beta_0 = -\infty$.
2. **(Termination)**
   (a) *Check if* $\Sigma = \emptyset$. *If true, terminate, the instance is infeasible*
   (b) *Check if* $\alpha_k - \beta_k \leq \epsilon_{tol}$. *If true, terminate,* $f^* := \alpha_k$ *is an* ϵ_{tol}*-optimal estimate for the optimal objective function value and* $\mathbf{p}^*$ *is a feasible point at which* f^* *is attained.*
   (c) *Delete from* Σ *all nodes* P^l *with* $f_l^{LBD} \geq \alpha_k$ *and set* $\beta_k := \min_{P^l\in\Sigma} f_l^{LBD}$.
3. **(Node Selection)**
   (a) *Pop and delete a node* P^l *from stack* Σ *such that* $\beta_k = f_l^{LBD}$.

4. **(Lower-Bounding Procedure)**

(a) Solve convex lower-bounding problem (4.15) *globally on* P^l.

(b) If no feasible solution exists, set $f_l^{LBD} := +\infty$, *otherwise set* $f_l^{LBD} := \mathrm{u}_f(\mathbf{x}^c(\check{\mathbf{p}}), \mathbf{x}^C(\check{\mathbf{p}}), \check{\mathbf{p}})$. *If a feasible solution is found that is feasible in* (4.4) *and* $f(\mathbf{x}(\check{\mathbf{p}}), \check{\mathbf{p}}) < \alpha_k$, *set* $\alpha_k := f(\mathbf{x}(\check{\mathbf{p}}), \check{\mathbf{p}})$, *and* $\mathbf{p}^* := \check{\mathbf{p}}$.

5. **(Upper-Bounding Procedure** (optional)**)**

(a) Solve the NLP subproblem (4.14) *locally on* P^l.

(b) If a feasible solution is found and $f_l^{UBD} < \alpha_k$, *set* $\alpha_k := f_l^{UBD}$, $\mathbf{p}^* := \hat{\mathbf{p}}$.

6. **(Fathoming)**

(a) Check if $f_l^{LBD} = +\infty$ *or* $f_l^{LBD} \geq \alpha_k$. *If true, go to 2.*

7. **(Branching)**

(a) Find $j \in \arg\max_{i=1,\ldots,n_p} w(P_i^l)$ *and create two new nodes* $P^{l'}$ *and* $P^{l''}$ *by bisecting* P_j^l.

(b) Set $f_{l'}^{LBD}, f_{l''}^{LBD} := f_l^{LBD}$ *and push the new nodes onto top of stack* Σ.

(c) Set $k := k + 1$, *go to 2.*

4.4.4 Finite Convergence

Guaranteed finite ϵ_{tol}-optimal convergence of Algorithm 4.1 is established in this section.

Definition 4.4.1 (X)**.** Let $X : \mathbb{I}P \to \mathbb{IR}^{n_x}$ be a continuous, interval-valued function which is both an interval extension and inclusion function of $\mathbf{x}$ on P such that for each $\mathbf{p} \in P$, $\mathbf{x}(\mathbf{p})$ is the unique solution of $\mathbf{h}(\mathbf{x}(\mathbf{p}), \mathbf{p}) = \mathbf{0}$ in $X(P)$.

It is assumed that such a function X is readily available by some procedure, such as the parametric extension of interval-Newton methods discussed in [56, 101] or the parameterized generalized bisection procedure discussed in Chapter 3.

Assumption 4.4.2. For $Z \equiv X(P)$, there exist continuous functions $F : \mathbb{I}Z \times \mathbb{I}P \to \mathbb{IR}$ and $G : \mathbb{I}Z \times \mathbb{I}P \to \mathbb{IR}^{n_g}$ such that F is both an interval extension and an inclusion function of f on $Z \times P$ and G is both an interval extension and an inclusion function of $\mathbf{g}$ on $Z \times P$.

For f and $\mathbf{g}$ factorable and continuous on open sets containing $Z \times P$, F and G are calculable by taking natural interval extensions [92, 101].

Lemma 4.4.3. *Consider a nested sequence of intervals $\{P^q\}$ (i.e. $P^m \subset P^q$, $\forall m > q$), $P^q \subset P$, $q \in \mathbb{N}$, such that $\{P^q\} \to [\bar{\mathbf{p}}, \bar{\mathbf{p}}]$ for some $\bar{\mathbf{p}} \in P$. Let $\mathbf{x}_q^c, \mathbf{x}_q^C$ be relaxations of $\mathbf{x}$ on P^q. Let $f_q^c(\cdot) = \mathrm{u}_f^q(\mathbf{x}_q^c(\cdot), \mathbf{x}_q^C(\cdot), \cdot)$ be a convex relaxation of the objective function f on P^q. Let $\hat{f}_q^c = \min_{\mathbf{p} \in P^q} f_q^c(\mathbf{p})$. Then $\lim_{q\to\infty} \hat{f}_q^c = f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$.*

Proof. From continuity of X on $\mathbb{I}P$, it is clear that $\lim_{q\to\infty} X(P^q) = X([\bar{\mathbf{p}}, \bar{\mathbf{p}}])$ and since X is an interval extension of $\mathbf{x}$, $X([\bar{\mathbf{p}}, \bar{\mathbf{p}}]) = [\mathbf{x}(\bar{\mathbf{p}}), \mathbf{x}(\bar{\mathbf{p}})]$. Let F^q be an interval function satisfying Assumption 4.4.2 on $\mathbb{I}X(P^q) \times \mathbb{I}P^q$. Then, by continuity of F, we have $\lim_{q\to\infty} F^q(X(P^q), P^q) = F([\mathbf{x}(\bar{\mathbf{p}}), \mathbf{x}(\bar{\mathbf{p}})], [\bar{\mathbf{p}}, \bar{\mathbf{p}}]) = [f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}}), f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})] = f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$. By construction, $\hat{f}_q^c(\mathbf{p}) \in F^q(X(P^q), P^q)$, $\forall \mathbf{p} \in P^q$ for every q, and therefore it follows $\lim_{q\to\infty} \hat{f}_q^c = f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$. □

Lemma 4.4.4. *Suppose Algorithm 4.1 generates an infinite sequence of nested nodes $\{P^q\}$, then $\lim_{q\to\infty} P^q = [\bar{\mathbf{p}}, \bar{\mathbf{p}}]$.*

Proof. Each node P^q is a subinterval partition of P that is an n_p-dimensional rectangle. The branching rule is a bisection along one of the longest edges of the currently selected node P^q. This result follows analogously from Proposition IV.2 in [63]. □

Lemma 4.4.5. *Suppose Algorithm 4.1 generates an infinite sequence of nested nodes $\{P^q\}$, then $\{P^q\} \to [\bar{\mathbf{p}}, \bar{\mathbf{p}}]$ and $\bar{\mathbf{p}}$ is feasible in* (4.4).

Proof. By Lemma 4.4.4, if Algorithm 4.1 generates an infinite sequence of nested nodes $\{P^q\}$, then $\{P^q\} \to [\bar{\mathbf{p}}, \bar{\mathbf{p}}]$. Suppose $\bar{\mathbf{p}}$ is infeasible in the original problem, i.e. $g_i(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}}) > \mathbf{0}$ for some $i = 1, 2, \ldots, n_g$. Let $\mathbf{g}^c(\cdot) = \mathbf{u}_{\mathbf{g}}(\mathbf{x}^c(\cdot), \mathbf{x}^C(\cdot), \cdot)$. By continuity of $\mathbf{g}$, there exists an open ball, of radius $\delta > 0$, around $\bar{\mathbf{p}}$, labeled $B_\delta(\bar{\mathbf{p}})$,

such that $\hat{\mathbf{p}} \in B_\delta(\bar{\mathbf{p}}) \Rightarrow g_i(\mathbf{x}(\hat{\mathbf{p}}), \hat{\mathbf{p}}) > 0$ for some $i = 1, 2, \ldots, n_g$. This implies that for some finite q', $P^{q'} \subset B_\delta(\bar{\mathbf{p}})$. Therefore, there exists a $q'' > q'$ such that for some $i = 1, 2, \ldots, n_g$, we have $g_i^c(\mathbf{p}) > 0,\ \forall \mathbf{p} \in P^{q''}$, where continuity of g_i^c (and $\mathbf{x}^c, \mathbf{x}^C$) on P follows from the definition of composite relaxations (Def. 4.2.9) and the properties of generalized McCormick relaxations [128]. Thus, the convex lower bounding problem (4.15) is infeasible for all $q > q''$. Finally, the node containing $\bar{\mathbf{p}}$ would be fathomed no later than at node $q'' + 1$. Therefore, Algorithm 4.1 cannot generate an infinite sequence of nested nodes that converge to an infeasible point. □

Lemma 4.4.6. *Suppose an infinite sequence of nested nodes, $\{P^q\}$, is generated by Algorithm 4.1. Let $f_q^c(\cdot) = \mathrm{u}_f^q(\mathbf{x}_q^c(\cdot), \mathbf{x}_q^C(\cdot), \cdot)$ and $\mathbf{g}_q^c(\cdot) = \mathbf{u}_\mathbf{g}{}^q(\mathbf{x}_q^c(\cdot), \mathbf{x}_q^C(\cdot), \cdot)$ be convex relaxations of f and $\mathbf{g}$ on P^q, respectively. Let $f_q^{*,c} = \min_{\mathbf{p} \in P^q} f_q^c(\mathbf{p}) : \mathbf{g}_q^c(\mathbf{p}) \leq \mathbf{0}$. Then $\{P^q\} \to [\bar{\mathbf{p}}, \bar{\mathbf{p}}]$ and $\lim_{q \to \infty} f_q^{*,c} = f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$.*

Proof. By Lemma 4.4.5, $\{P^q\} \to [\bar{\mathbf{p}}, \bar{\mathbf{p}}]$, with $\bar{\mathbf{p}} \in P$ feasible. Let $\hat{f}_q^c = \min_{\mathbf{p} \in P^q} f_q^c(\mathbf{p})$. Since $\hat{f}_q^c$ is the solution of the convex unconstrained problem, it is clear that $\hat{f}_q^c \leq f_q^{*,c}$. Since $f_q^{*,c}$ is a rigorous lower bound of $f(\mathbf{x}(\cdot), \cdot)$ on P^q, we have $\hat{f}_q^c \leq f_q^{*,c} \leq f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$. Since $\lim_{q \to \infty} \hat{f}_q^c = f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$ from Lemma 4.4.3, it is clear that $\lim_{q \to \infty} f_q^{*,c} = f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$. □

Lemma 4.4.7. *Let f^* denote the globally optimal objective function value for (4.4). The sequence of lower bounds generated by Algorithm 4.1 is either finite or satisfies $\lim_{k \to \infty} \beta_k = f^*$.*

Proof. This result follows from Theorem 2.1 in [60] where the hypotheses are guaranteed by Lemmas 4.4.4-4.4.6 above. □

Lemma 4.4.8. *Suppose that an infinite sequence of nested nodes, $\{P^q\}$, is generated by Algorithm 4.1. Also, suppose that the upper-bounding problem (4.14) can locate a feasible point for every $q \geq q'$ for some finite q', and thus a valid upper bound can be located in every subsequent node. Then, the upper-bounding operation converges to the global solution of (4.4), i.e. $\lim_{k \to \infty} \alpha_k = f^*$.*

Proof. From Lemma 4.4.5, if Algorithm 4.1 generates an infinite sequence of nested nodes, $\{P^q\}$, then $\{P^q\} \to [\bar{\mathbf{p}}, \bar{\mathbf{p}}]$ and $\bar{\mathbf{p}}$ is feasible. From Lemma 4.4.6, we know that $\lim_{q\to\infty} f_q^{*,c}(\mathbf{p}) = f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$. Suppose that $\bar{\mathbf{p}}$ is not a global minimizer. Then $f^* < f(\mathbf{x}(\bar{\mathbf{p}}), \bar{\mathbf{p}})$ implying that for some q'' we have $f_{q''}^{*,c} > f^*$. However, using the bound-improving node selection property of Algorithm 4.1, this node would have never been selected again for branching. Therefore $\bar{\mathbf{p}}$ must be a global minimizer $\mathbf{p}^* = \bar{\mathbf{p}}$.

From continuity of f, for some $\epsilon > 0$, there exists an open ball of radius $\delta > 0$ around $\mathbf{p}^*$, $B_\delta(\mathbf{p}^*)$, such that $\mathbf{p} \in B_\delta(\mathbf{p}^*) \Rightarrow |f(\mathbf{x}(\mathbf{p}), \mathbf{p}) - f(\mathbf{x}(\mathbf{p}^*), \mathbf{p}^*)| < \epsilon$, where continuity of $\mathbf{x}$ on P follows from continuous differentiability of $\mathbf{h}$ and the implicit function theorem.

By hypothesis, after some finite q', a feasible point $\hat{\mathbf{p}} \in P^q$ can be found that provides a valid upper bound f_q^{UBD}. By the bound-improving property, if f_q^{UBD} is lower than the current upper bound α_k, then $\alpha_k := f_q^{UBD}$. For q large enough, a feasible point $\mathbf{p}$ will be located such that $\mathbf{p} \in B_\delta(\mathbf{p}^*)$. By continuity of f, we have $|f(\mathbf{x}(\mathbf{p}), \mathbf{p}) - f(\mathbf{x}(\mathbf{p}^*), \mathbf{p}^*)| < \epsilon \Rightarrow |f_q^{UBD} - f(\mathbf{x}(\mathbf{p}^*), \mathbf{p}^*)| < \epsilon \Rightarrow f_q^{UBD} < f(\mathbf{x}(\mathbf{p}^*), \mathbf{p}^*) + \epsilon$. Since $f(\mathbf{x}(\mathbf{p}^*), \mathbf{p}^*) \leq \alpha_k \leq f_q^{UBD}$, we have $f(\mathbf{x}(\mathbf{p}^*), \mathbf{p}^*) \leq \alpha_k < f(\mathbf{x}(\mathbf{p}^*), \mathbf{p}^*) + \epsilon$. Thus $\lim_{k\to\infty} \alpha_k = f(\mathbf{x}(\mathbf{p}^*), \mathbf{p}^*) = f^*$. □

Theorem 4.4.9 (Finite Convergence)**.** *After finitely many iterations, Algorithm 4.1 terminates with either ϵ-optimal global solutions, such that $\alpha_k - \beta_k \leq \epsilon_{\text{tol}}$, or a guarantee that the problem is infeasible.*

Proof. Follows immediately from Lemma 4.4.7 and Lemma 4.4.8 and the deletion by infeasibility rule. □

4.5 Illustrative Examples

For the following illustrative examples, Algorithm 4.1 was implemented in `C++`. The hierarchy of the information flow for the implementation is shown in Figure 4-2. The convex lower-bounding problems were solved using PBUN, a nonsmooth optimization

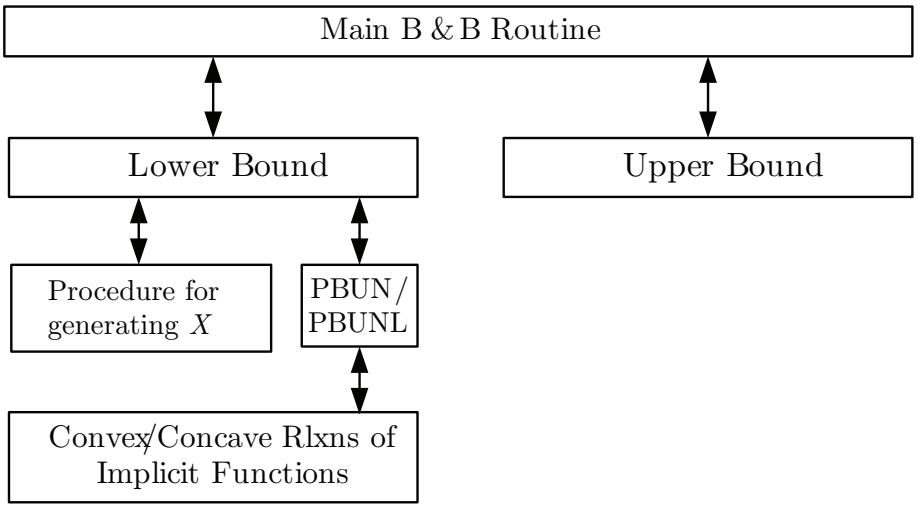

Figure 4-2: The hierarchy of the flow of information for the implementation of the global optimization of implicit functions algorithm (Alg. 4.1).

algorithm developed in [83]. If a lower-bounding problem returned a feasible point $\mathbf{p}$, the model equations were solved at this point using Newton's method with Gauss-Seidel and the objective function was evaluated for an upper bound on the solution, instead of solving (4.14) locally. The methods used for calculating valid X intervals are discussed briefly for each example.

Example 4.5.1. Let $Z \in \mathbb{IR}^3$ and $P \in \mathbb{IR}^3$. Consider the objective function $f : Z \times P \to \mathbb{R}$ defined as

$$f(\mathbf{z}, \mathbf{p}) =$$
$$\sum_{j=1}^{3} \left([a_j(p_j - c_j)]^2 + \sum_{i \neq j} a_i(p_i - c_i) - 5 \left((j-1)(j-2)(z_2 - z_1) + \sum_{i=1}^{3} (-1)^{i+1} z_i \right) \right)^2 \tag{4.16}$$

with a_i, c_i constants for $i = 1, 2, 3$, given in Table 4.5.1.

Example 1 Constants	
a_1	37.3692
c_1	0.602
a_2	18.5805
c_2	1.211
a_3	6.25
c_3	3.60

Table 4.1: Constants for the objective function for the global optimization of implicit functions example.

Consider the equality constraints

$$\mathbf{h}(\mathbf{z},\mathbf{p}) = \begin{pmatrix} 1.00\times 10^{-9}(\exp[38z_1]-1) + p_1 z_1 - 1.6722 z_2 + 0.6689 z_3 - 8.0267 \\ 1.98\times 10^{-9}(\exp[38z_2]-1) + 0.6622 z_1 + p_2 z_2 + 0.6622 z_3 + 4.0535 \\ 1.00\times 10^{-9}(\exp[38z_3]-1) + z_1 - z_2 + p_3 z_3 - 6.0 \end{pmatrix} \tag{4.17}$$

$$= \mathbf{0}.$$

The full-space optimization formulation is

$$\begin{aligned} &\min_{(\mathbf{z},\mathbf{p})\in Z\times P} f(\mathbf{z},\mathbf{p}) \\ &\text{s.t. } \mathbf{h}(\mathbf{z},\mathbf{p}) = \mathbf{0} \\ &Z = [-5,5]^3 \\ &P = [0.6020, 0.7358]\times[1.2110, 1.4801]\times[3.6, 4.4]. \end{aligned} \tag{4.18}$$

The reduced-space, box-constrained, formulation becomes

$$\begin{aligned} &\min_{\mathbf{p}\in P} f(\mathbf{x}(\mathbf{p}),\mathbf{p}) \\ &P = [0.6020, 0.7358]\times[1.2110, 1.4801]\times[3.6, 4.4] \end{aligned} \tag{4.19}$$

Using the parametric interval-Newton method with interval Gauss-Seidel, an interval,

X, that conservatively bounds the implicit function $\mathbf{x}$ on all of P, can be calculated:

$$X = [0.5180, 0.5847] \times [-3.9748, -3.0464] \times [0.3296, 0.5827].$$

It is apparent that X is significantly tighter than Z. For each branch of P popped off the stack, the interval-Newton method is applied to further refine the interval X such that Definition 4.4.1 holds. This problem has a suboptimal local minimum at $\mathbf{p} = (0.602, 1.46851, 3.6563)$ with a value of 731.197 and a global minimum at $\mathbf{p}^* = (0.703918, 1.43648, 3.61133)$ with a value of 626.565. This problem was solved in 0.4 seconds with Algorithm 4.1 taking 43 iterations with tolerances for convergence as 10^{-3} for relative error and absolute error.

For comparison, this problem was modeled in GAMS version 23.9 [116] using the BARON solver [138] with preprocessing turned off. For a fair comparison, the local search procedure for obtaining an upper bound was also turned off. Starting with the variable interval Z, BARON failed to solve the problem noting "No feasible solution was found." Using the interval X calculated above, BARON solved the problem and returned the global solution in 1 second after 810 iterations. Plots of the implicit objective function $f(\mathbf{x}(\mathbf{p}), \mathbf{p})$ are shown below in Figure 4-3 for three different values of p_3. Similarly, the implicit objective function and corresponding relaxations are shown in Figure 4-4 for the same three values of p_3.

Example 4.5.2. Consider the reactor-separator-recycle process system shown in Figure 4-5. The plant is designed to produce 50kmol/h of monochlorobenzene with an undesired side-reaction producing dichlorobenzene. The reactor is a continuous-stirred tank reactor (CSTR) and each separator column is designed to perform sharp splits. As in [25], the optimization variables are the reactor volume and the operating parameters. The hydrochloric acid (HCl) produced by each reaction is assumed to be eliminated by a stripping operation whose costs are not taken into account, as in [73]. For the purposes of this chapter, the operating parameters of interest will be the reaction rate constants, assuming they can be manipulated by way of reactor temperature or catalysts etc. Therefore, for this formulation, $n_x = 11$ and $n_p = 3$.

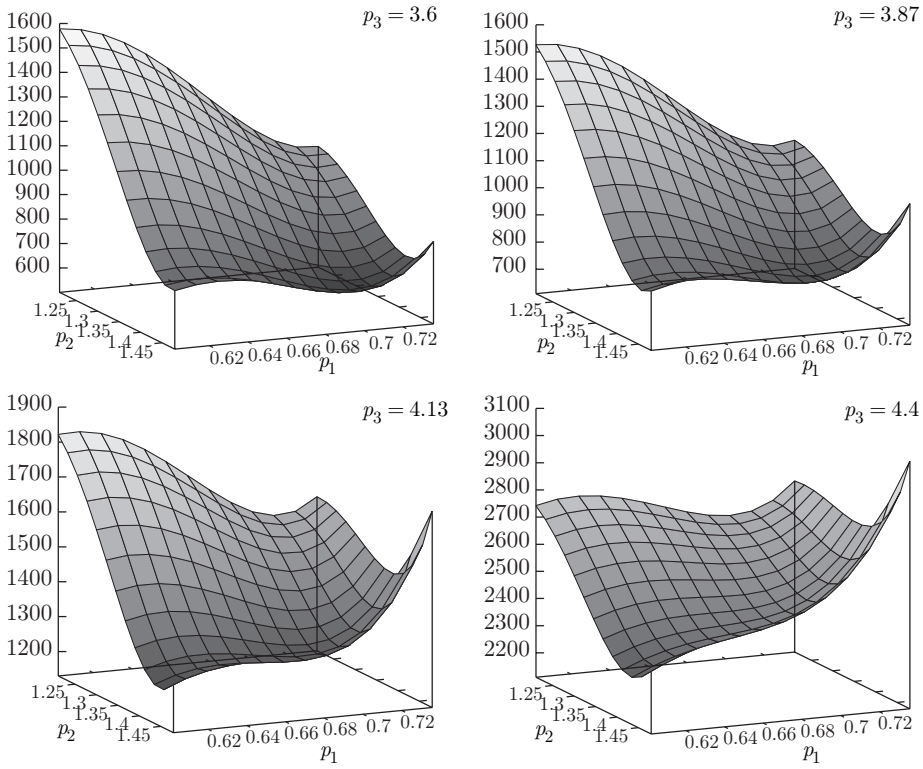

Figure 4-3: The objective function of the global optimization of implicit functions example on $P_1 \times P_2$ at three different p_3 values.

The state vector $\mathbf{z}$ was taken as

$$\mathbf{z} = (F_1, F_2, F_3, y_{3,A}, y_{3,B}, y_{3,C}, F_4, y_{4,B}, y_{4,C}, F_6, F_7)$$

and the initial intervals are given in Table 4.2.

For this problem, it was not enough to use the parametric interval-Newton method with interval Gauss-Seidel from Chapter 3 alone. In order to produce the required interval X that satisfies Definition 4.4.1, at each partition of P popped off of the stack, forward-backward constraint propagation[3] [66, 125] is first applied. Second, the linear-programming contractor method [6, 24, 66] was applied. Finally, parametric interval-Newton with interval Gauss-Seidel was applied in an attempt to further refine the interval.

[3]Forward-backward constraint propagation will be discussed in more detail in Chapter 8.

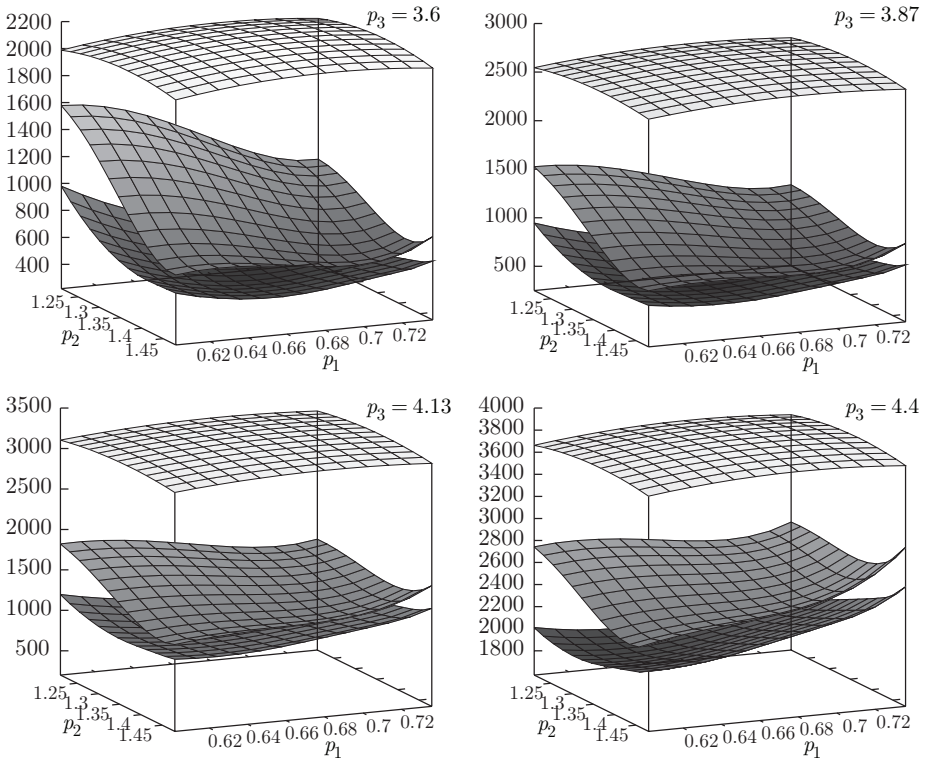

Figure 4-4: The objective function of the global optimization of implicit functions example on $P_1 \times P_2$ at three different p_3 values and corresponding convex and concave relaxations.

The model equations are as follows. The mixer model equation is

$$0 = F_1 + F_7 - F_2 \tag{4.20}$$

where F_j represent the total molar flowrate (in kmol/h) of stream j. The reactor model equations are given by

$$0 = F_2 - y_{3,A}F_3 - vr_1 \ (A \text{ balance on reactor}) \tag{4.21}$$

$$0 = v(r_1 - r_2) - F_5 \ (B \text{ balance on reactor}) \tag{4.22}$$

$$0 = vr_2 - y_{3,C}F_3 \ (C \text{ balance on reactor}) \tag{4.23}$$

$$0 = 1 - y_{3,A} - y_{3,B} - y_{3,C} \tag{4.24}$$

with v as the reactor volume in m^3 and the reaction rates r_1 and r_2 (in $\mathrm{kmol}/(\mathrm{m}^3\mathrm{h})$)

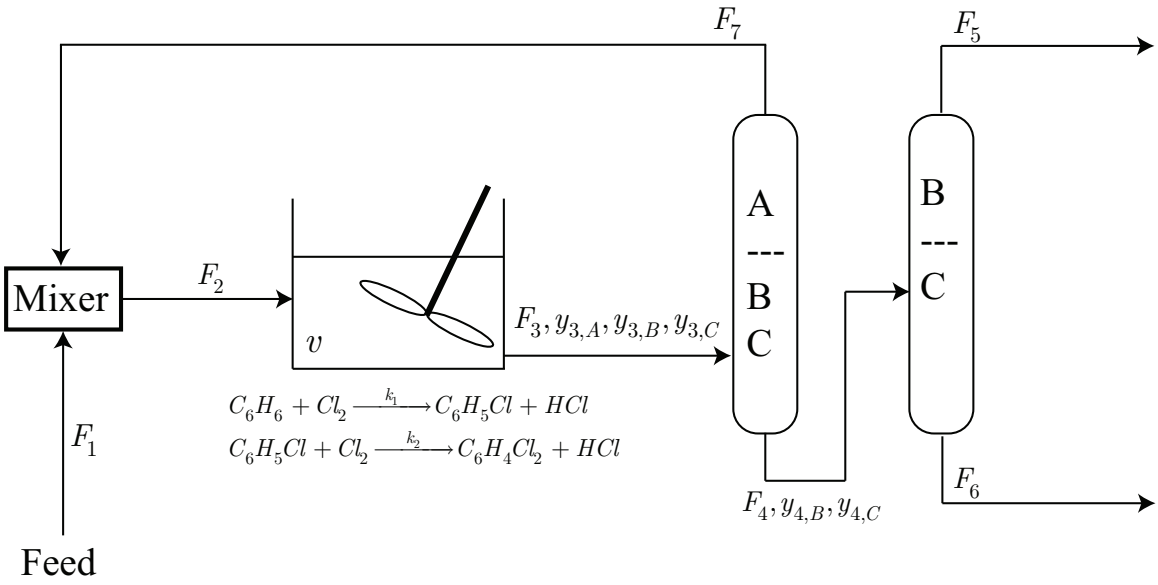

Figure 4-5: The process flow diagram for the reactor-separator-recycle process system for producing monochlorobenzene. Note that HCl is eliminated by a stripping operation not shown. A=benzene, B=monochlorobenzene, C=dichlorobenzene.

$F_1 \in$	$[50, 150]$	$F_4 \in$	$[50, 60]$
$F_2 \in$	$[100, 300]$	$y_{4,B} \in$	$[0.142857, 1]$
$F_3 \in$	$[100, 300]$	$y_{4,C} \in$	$[0.142857 \times 10^{-3}, 1]$
$y_{3,A} \in$	$[0.5, 0.9]$	$F_6 \in$	$[0, 10]$
$y_{3,B} \in$	$[0.1, 0.5]$	$F_7 \in$	$[40, 250]$
$y_{3,C} \in$	$0.001, 0.2$		

Table 4.2: The initial intervals for the reactor-separator-recycle example.

are given by the following expressions

$$r_1 = k_1 C_A = \frac{k_1 y_{3,A}}{y_{3,A}\hat{v}_A + y_{3,B}\hat{v}_B + y_{3,C}\hat{v}_C} \tag{4.25}$$

$$r_2 = k_2 C_B = \frac{k_2 y_{3,B}}{y_{3,A}\hat{v}_A + y_{3,B}\hat{v}_B + y_{3,C}\hat{v}_C} \tag{4.26}$$

where $\hat{v}_i$ is the molar volume of species $i \in \{A, B, C\}$ in $\mathrm{m}^3/\mathrm{kmol}$. The symbols $y_{j,i}$ represent the mole fraction of species i in stream j. The model equations for the first

separator are given by

$$0 = F_3 - F_4 - F_7 \text{ (overall balance on first separator)} \tag{4.27}$$

$$0 = y_{3,A}F_3 - F_7 \text{ (A balance on first separator)} \tag{4.28}$$

$$0 = y_{3,B}F_3 - y_{4,B}F_4 \text{ (B balance on first separator).} \tag{4.29}$$

Finally, the model equations for the second separator are given by

$$0 = F_4 - F_5 - F_6 \text{ (overall balance on second separator)} \tag{4.30}$$

$$0 = y_{4,B}F_4 - F_5 \text{ (B balance on second separator)} \tag{4.31}$$

$$0 = y_{4,C}F_4 - F_6 \text{ (C balance on second separator).} \tag{4.32}$$

The capital costs of the first and second separators, respectively, are given by

$$C_1^{cap} = 132718 + F_3(369y_{3,A} - 1113.9y_{3,B}) \tag{4.33}$$

$$C_2^{cap} = 25000 + F_4(6984.5y_{4,B} - 3869.53y_{4,C}^2). \tag{4.34}$$

The capital cost of the reactor is given by

$$C_{CSTR}^{cap} = 25764 + 8178v. \tag{4.35}$$

The operating cost of the reactor is considered to be negligible and the operating costs of the first and second separators, respectively, are given by

$$C_1^{op} = F_3(3 + 36.11y_{4,A} + 7.71y_{4,B})(C_{Steam} + C_{Cool}) \tag{4.36}$$

$$C_2^{op} = F_4(26.21 + 29.45y_{4,B})(C_{Steam} + C_{Cool}). \tag{4.37}$$

The economic objective is to minimize the annualized venture cost, given by

$$C^{ann} = \frac{1}{\alpha}\left(C_1^{cap} + C_2^{cap} + C_{CSTR}^{cap}\right) + \beta\left(C_1^{op} + C_2^{op}\right), \tag{4.38}$$

Example 4.5.2 Constants	
$\hat{v}_A$	$8.937 \times 10^{-2}\text{m}^3\text{kmol}^{-1}$
$\hat{v}_B$	$1.018 \times 10^{-1}\text{m}^3\text{kmol}^{-1}$
$\hat{v}_C$	$1.130 \times 10^{-1}\text{m}^3\text{kmol}^{-1}$
α	2.5 yr
β	0.52
C_{Steam}	$\$21.67 \times 10^{-3}\text{kJ}^{-1}\text{yr}^{-1}$
C_{Cool}	$\$4.65 \times 10^{-3}\text{kJ}^{-1}\text{yr}^{-1}$

Table 4.3: The constants for the reactor-separator-recycle model (Ex. 4.5.2).

where α is the payout time and β is the tax rate. The constants for this example are given in Table 4.3.

The reactor volume was considered as $v \in [14, 24]\text{m}^3$ and the reaction rate constants were considered as $k_1 \in [0.405, 0.415]\text{h}^{-1}$ and $k_2 \in [0.0545, 0.0555]\text{h}^{-1}$. Algorithm 4.1 solved the problem with an optimal objective of $\$289,780\text{yr}^{-1}$ with $v = 21.68\text{m}^3, k_1 = 0.415\text{h}^{-1}, k_2 = 0.0545\text{h}^{-1}$ taking 148 seconds and 1209 iterations with convergence tolerances of 10^{-2} and 10^{-3} for absolute and relative error, respectively.

Again, for comparison, this example was modeled in GAMS version 23.9 [116] using the BARON solver [138] with preprocessing turned off. Similar to the previous example, the local search procedure for obtaining an upper bounds was also turned off. Exploiting all of the other features of the program, the model was solved in an impressive time of 0.4 seconds. It was identified that the algorithm relied heavily on certain bounds-tightening strategies for convergence. For instance, if either the nonlinear-feasibility-based range reduction option or the linear-feasibility-based range reduction option were turned off, the algorithm failed to converge. This indicates that these range-reduction strategies play a pivotal role in the convergence of the BARON solver. Similarly, if the LP-contractor strategy, employed here for generating a suitable X box satisfying Definition 4.4.1, was switched off, Algorithm 4.1 fails to converge since no such X satisfying Definition 4.4.1 can be calculated.

Example 4.5.3. Consider the parameter estimation example presented in [88] which was adapted from [130]. This problem attempts to determine whether or not a pro-

posed kinetic mechanism sufficiently predicts the behavior of a reacting system for which experimental data is available. The following kinetic mechanism is proposed:

$$
\begin{gathered}
Z + Y \xrightarrow{k_1} A, \quad A + O_2 \underset{k_{2f/K_2}}{\overset{k_{2f}}{\longleftrightarrow}} D \\
A + O_2 \underset{k_{3f/K_2}}{\overset{k_{3f}}{\longleftrightarrow}} B, \qquad B \xrightarrow{k_4} M + N \\
2A \xrightarrow{k_5} P
\end{gathered}
$$

which is modeled as a system of nonlinear ODEs:

$$
\begin{gathered}
\frac{dc_A}{dt} = k_1 c_Z c_Y - c_{O_2}(k_{2f} + k_{3f}) c_A + \frac{k_{2f}}{K_2} c_D + \frac{k_{3f}}{K_3} c_B - k_5 c_A^2 \\
\frac{dc_B}{dt} = k_{3f} c_{O_2} c_A - \left(\frac{k_{3f}}{K_3} + k_4 \right) c_B, \quad \frac{dc_D}{dt} = k_{2f} c_A c_{O_2} - \frac{k_{2f}}{K_2} c_D \\
\frac{dc_Y}{dt} = -k_{1s} c_Y c_Z, \qquad \frac{dc_Z}{dt} = -k_1 c_Y c_Z \\
c_A(t=0) = 0,\ c_B(t=0) = 0,\ c_D(t=0) = 0,\ c_Y(t=0) = 0.4,\ c_Z(t=0) = 140
\end{gathered}
\tag{4.39}
$$

where c_j is the concentration (in appropriate units) of species j, $T = 273$, $K_2 = 46 \exp[6500/T - 18]$, $K_3 = 2K_2$, $k_1 = 53$, $k_{1s} = k_1 \times 10^{-6}$, $k_5 = 1.2 \times 10^{-3}$, and $c_{O_2} = 2 \times 10^{-2}$. The uncertain model parameters are $\mathbf{p} = (k_{2f}, k_{3f}, k_4)$ with $k_{2f} \in [10, 1200]$, $k_{3f} \in [10, 1200]$, and $k_4 \in [0.001, 40]$. Each experimental measurement is given in the form of $I_d = c_A + \frac{2}{21} c_B + \frac{2}{21} c_D$. The same data used in [88] is used here and can be found in Appendix C or downloaded from `http://yoric.mit.edu/libMC/libmckinexdata.txt`.

Using the implicit-Euler discretization scheme, the time domain is discretized into $n = 200$ evenly-spaced nodes and the solution of the ODE system (4.39) can be approximated, with reasonable accuracy, as the solution of a corresponding nonlinear algebraic system with $5n$ state variables and 3 parameters. The method of [88] was not applicable to this implicit scheme and so in [88] the method was demonstrated using the explicit-Euler discretization scheme. As an aside, approximating the solution of an ODE system using the explicit Euler numerical integration method may suffer

from numerical instabilities when the problem is *stiff* (i.e., when the solution exhibits fast transient behavior) whereas the implicit technique, albeit more computationally expensive per time step, is unconditionally stable and can therefore handle much larger time steps than the explicit approach. The ODE (4.39) is considered to be moderately stiff and so either approach may work well. For $i = 1, \ldots, n$, the resulting nonlinear algebraic system is

$$\begin{aligned}
0 &= c_A^{i-1} - c_A^i + \Delta t \left(k_1 c_Y^i c_Z^i - c_{O_2}(k_{2f} + k_{3f}) c_A^i + \frac{k_{2f}}{K_2} c_D^i + \frac{k_{3f}}{K_3} c_B^i - k_5 {c_A^i}^2 \right) \\
0 &= c_B^{i-1} - c_B^i + \Delta t \left(k_{3f} c_{O_2} c_A^i - \left(\frac{k_{3f}}{K_3} + k_4 \right) c_B^i \right) \\
0 &= c_D^{i-1} - c_D^i + \Delta t \left(k_{2f} c_A^i c_{O_2} - \frac{k_{2f}}{K_2} c_D^i \right) \\
0 &= c_Y^{i-1} - c_Y^i + \Delta t \left(-k_{1s} c_Y^i c_Z^i \right) \\
0 &= c_Z^{i-1} - c_Z^i + \Delta t \left(-k_1 c_Y^i c_Z^i \right)
\end{aligned} \tag{4.40}$$

where, for $n = 200$, $\Delta t = 0.01$. The resulting explicit NLP formulation therefore has $5n + 3$ variables with

$$\mathbf{z} = (c_A^1, c_B^1, c_D^1, c_Y^1, c_Z^1, \ldots, \ldots, \ldots, \ldots, \ldots, c_A^{200}, c_B^{200}, c_D^{200}, c_Y^{200}, c_Z^{200}).$$

By solving the system for the state variables as implicit functions of the parameters, the resulting implicit NLP formulation has just 3 independent variables. This can be done using two different techniques. The first, which is not recommended, is to treat the nonlinear system of equations as fully coupled and essentially solve for the state variables simultaneously. Thus, in order to construct relaxations of implicit functions, using this technique would require relaxing 1000 implicit functions simultaneously. The second technique, which is how numerical integration is typically performed, exploits the block structure of the problem.

Taking a look at the sparsity pattern of the system, a portion of which is shown in Figure 4-6, it is easy to notice that each equation at node i is only dependent on the variables at node i and the variables at node $i - 1$. Therefore, if the variables

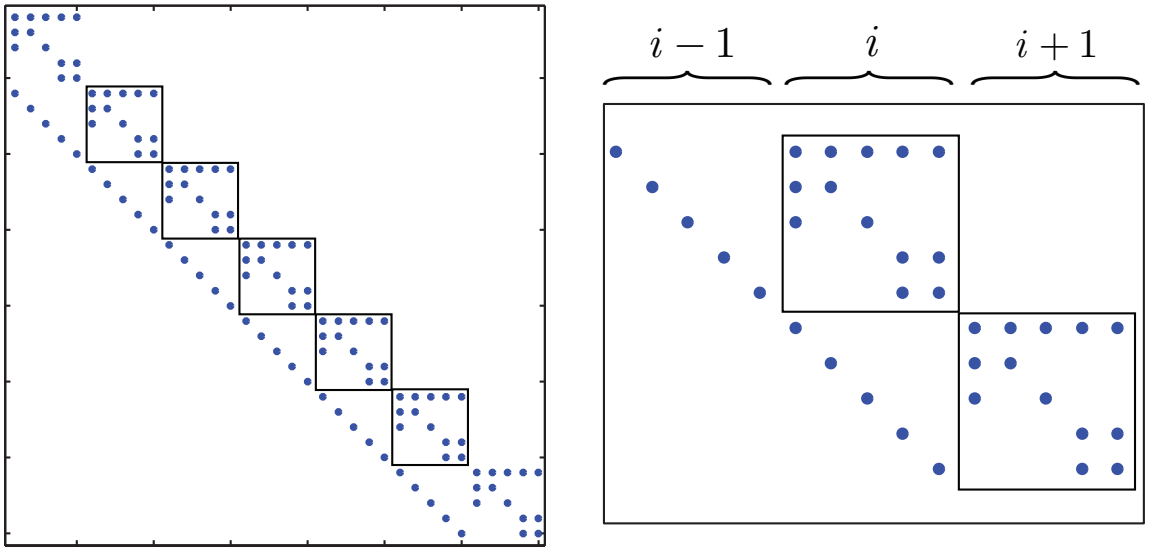

Figure 4-6: (Left) The sparsity pattern of the system with $n = 7$ discretization. Each 5×5 block is highlighted to show how the system can be solved in a sequential block-by-block fashion. (Right) An expanded view of three time steps showing how information from the previous node is used to solve the 5×5 system associated with the current node.

at node $i-1$ are known, node i can be solved as a system of 5 nonlinear equations. Since node 0 is specified by the initial conditions, this technique can be applied sequentially from node 1 to node 200. Again, this is how the implicit Euler numerical integration method is applied. Constructing relaxations is then done in an analogous fashion. Relaxations are constructed for each system of 5 equations using the method of Section 4.3.4 and subsequently used in the construction of relaxations of each system associated with the next node with the relaxations of node 0 taken to be exactly the initial conditions for all $\mathbf{p} \in P$ (since they are constant on P). The initial intervals are taken as $c_j^i \in [0, 140],\ j \neq Y, \forall i$ and $c_Y^i \in [0, 0.4],\ \forall i$. This approach is recommended over the simultaneous approach as it is not only significantly less computationally expensive, but it also produces much tighter relaxations.

The objective function for this problem is stated as

$$f(\mathbf{z}, \mathbf{p}) = \sum_{i=1}^{n} \left(I^i - I_d^i\right)^2$$

where $I^i = c_A^i + \frac{2}{21} c_B^i + \frac{2}{21} c_D^i,\ i = 1, \ldots, n$, with $c_A^i, c_B^i, c_D^i,\ i = 1, \ldots, n$, given by

k_{2f}	k_{3f}	k_4	$f^* \times 10^{-4}$
235.04	1048.8	0.33151	1.7066726
350.72	931.25	0.38279	1.7056881
678.53	596.96	0.82748	1.7024373
765.26	450.21	12.414	1.6807190
355.02	926.55	11.766	1.7056560
740.18	523.81	13.717	1.6993238
735.88	528.60	13.993	1.6995289
627.16	552.87	12.187	1.7051711
775.44	437.23	17.576	1.6802801

Table 4.4: Suboptimal local minima of Example 4.5.3 (using the sequential block solve technique) found using the multi-start SQP approach.

the solution of the nonlinear system (4.40) and $I_d^i,\ i = 1, \ldots, n$, are the experimental data mentioned previously, found in Appendix C.

In an effort to survey the topological features of the objective function for this problem, multistart optimization techniques were employed. The explicit (full space) NLP formulation (i.e., with 1003 variables and 1000 equality constraints) was solved by multi-starting the MINOS solver [97] in GAMS version 23.9 [116]. Only one optimum was found and it happened to correspond with the global solution. Alternatively, the implicit NLP formulation, where the implicit functions are evaluated using the second technique described above (i.e., sequential block solution), was then solved by multi-starting the MATLAB SQP solver. In this case, eight suboptimal local minima were found along with the global minimum. The suboptimal local minima that were found are reported in Table 4.4. This is a rather interesting result because it means that, for this problem, the reduced-space formulation has many suboptimal local minima, whereas the full-space formulation may not have any. This is consistent with what was found in the Methanol-to-Hydrocarbons Example in [39] and is not a result that holds in general.

The reduced-space NLP was solved using Algorithm 4.1 taking 2×10^5 seconds (55.6h) and 69981 iterations with convergence tolerances of 10^{-2} and 10^{-3} for absolute and relative error, respectively. The optimal parameter values were found, $\mathbf{p}^* = (797.145, 423.545, 13.6096)$ with $f^* = 16796$, and the "best fit" corresponding to the

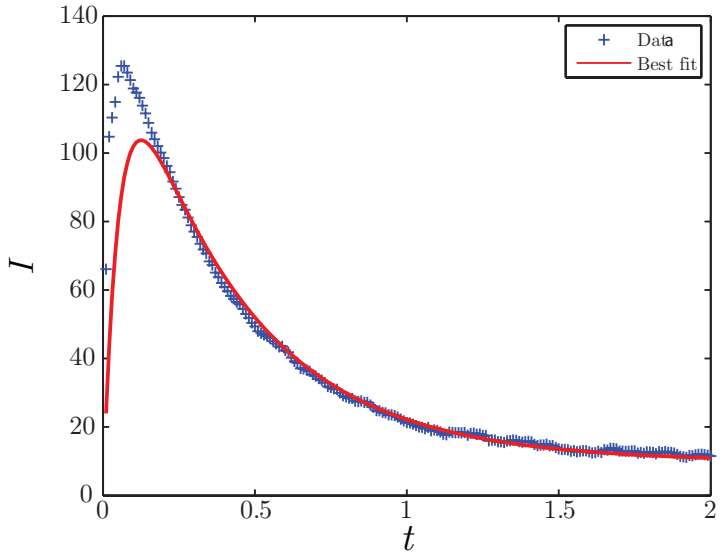

Figure 4-7: The optimal "best fit" of the model plotted against the experimental data.

optimal solution $\mathbf{p}^*$ was plotted against the experimental data in Figure 4-7. As was concluded in [88], the model with the best fit parameters does not agree with experimental data at early times. Since a certificate of global optimality was obtained, one can conclude that the model cannot represent the physical system at early times.

For comparison, the full-space NLP was modeled in GAMS 23.9 [116] using BARON [138] with preprocessing turned off. Both a selective-branching strategy and the standard strategy of branching on all variables were studied. Using MINOS as the local-search algorithm for solving the upper-bounding problem, BARON converged to the solution found using the multi-start approach discussed above within just a few seconds, for each branching strategy. However, using SNOPT [48] as the local-search algorithm for solving the upper-bounding problem, BARON converged to a suboptimal solution in just a few seconds for each branching strategy. It should be noted that in each case, BARON terminates normally claiming that it found a solution with a guarantee of global optimality. The behavior of BARON here is not fully understood and so it is considered to be ineffective at solving this problem. Alternatively, each strategy was solved without using local-search algorithms for the

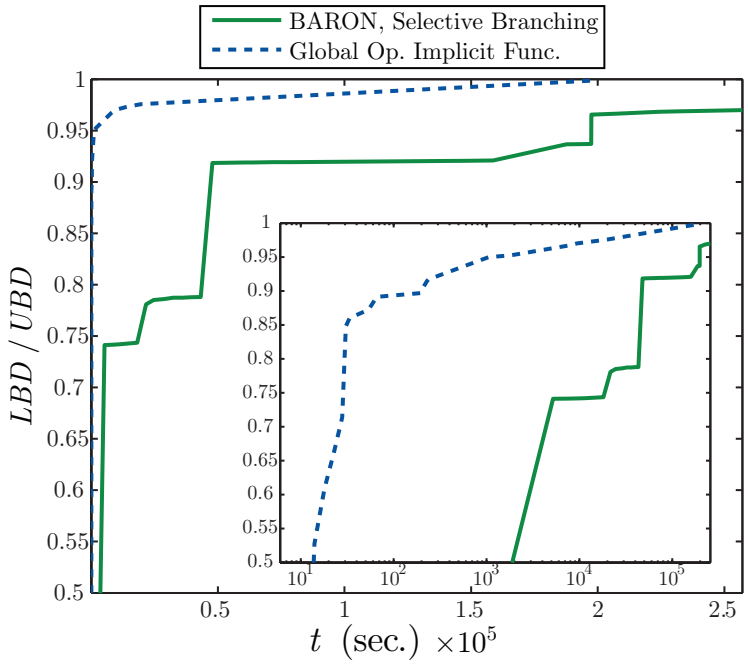

Figure 4-8: The performance of three methods on the kinetic mechanism example in terms of convergence.

upper-bounding problems. When considering the strategy of branching on all of the variables, BARON fails to solve the problem. For this case, the algorithm terminates after about 460 seconds with the result that no feasible solution could be found. Again, this is a very strange result since the problem is indeed feasible. Figure 4-8 is a plot showing the performance, in terms of the ratio of the lower and upper bounds versus CPU time in seconds, of Algorithm 4.1 versus BARON with selective branching and without a local-solve upper-bounding procedure. After about 30 seconds, Algorithm 4.1 improves on the bounds quite effectively, even without a local-search upper-bounding procedure. It takes BARON about 50000 seconds to achieve the same level of convergence as Algorithm 4.1 achieves after the 30 second mark. After about 1000 seconds, Algorithm 4.1 begins to exhibit slow but consistent improvement on the bounds until it converges. This is likely due to the node clustering problem and is the topic of future research. It takes 2×10^5 seconds for Algorithm 4.1 to converge to the global solution whereas BARON is about 96.5% converged at this time. The BARON selective branching strategy fails to converge even after more than 70 hours when the maximum number of iterations of 100000 is reached. At this time BARON

is only 97% converged. It is clear that for this problem, Algorithm 4.1 performs more favorably than BARON.

4.6 Concluding Remarks

In this chapter, a reformulation of the standard NLP with equality constraints has been proposed that is an equivalent formulation offering a potentially large reduction in dimensionality. This approach is similar to that taken in Chapter 2 for eliminating the equality constraints and formulating the SIP (2.5) with implicit functions embedded. By solving the quality constraints for the dependent variables as implicit functions of the independent variables, they are eliminated from the program and the implicit functions are embedded within the objective function and inequality constraint(s). In order to solve the reduced-space problem, new results for relaxing implicit functions were developed. In particular, the calculations of convex and concave relaxations, and subgradients, of implicit functions were developed.

One new result was presented that guarantees that relaxations of a successive-substitution iteration are also valid relaxations of the implicit function. Another key result pertaining to solutions of parametric linear systems was presented. This result states that relaxations of the solution of a parametric linear system can be calculated iteratively in a fashion analogous to the Gauss-Seidel method. It was demonstrated that relaxations of the generic Newton-type iteration cannot be refinements of the original bounds on the implicit function. This proves that direct relaxations of Newton-type iterations are not useful, but relaxations of convergent successive-substitution iterations may be useful. Because of this, new methods, analogous to interval Newton-type methods, were developed that essentially relax the implicit functions by relaxing the mean-value theorem. These novel developments offer ways to calculate relaxations of an implicit function that is a parametric solution of a general nonlinear system of equations that cannot be approximated via a successive-substitution iteration. Furthermore, subgradients of such relaxations can be calculated, which are useful in the solution of the resulting nonsmooth convex

program.

Utilizing these new results, a reduced-space global optimization algorithm was proposed for solving nonconvex NLPs with embedded implicit functions. The algorithm was shown to converge in finitely many iterations to an ϵ-optimal solution. The algorithm was applied to three instructive numerical examples which demonstrate its applicability. Together with the developments in Chapter 3, these developments successfully accomplish objectives (1) and (2) listed at the end of Chapter 2. In the next chapter, these developments will be applied to global optimization problems constrained by large sparse (parametric linear) systems. In Chapter 7, these developments will be applied to solve implicit SIPs.

Chapter 5

Global Optimization of Large Sparse Systems

Similar to the previous chapter, this chapter will focus on solving global optimization problems of the form:

$$\begin{aligned} &\min_{\mathbf{z}\in X, \mathbf{p}\in P} f(\mathbf{z}, \mathbf{p}) \\ \text{s.t. } &\mathbf{g}(\mathbf{z}, \mathbf{p}) \leq \mathbf{0} \\ &\mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{0}. \end{aligned} \tag{5.1}$$

However, now the model $\mathbf{h}$ is considered to be specifically of the form of a parametric linear system:

$$\mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{A}(\mathbf{p})\mathbf{z} - \mathbf{b}(\mathbf{p}) = \mathbf{0} \Rightarrow \mathbf{A}(\mathbf{p})\mathbf{z} = \mathbf{b}(\mathbf{p}),$$

where $\mathbf{A} : P \to D_a$ and $\mathbf{b} : P \to D_b$, where $\mathbf{A}$ is large and sparse[1] and the sets D_a and D_b are defined as in Chapter 4. For consistency with that chapter, the solution of the parametric linear system will be denoted as the implicit function $\boldsymbol{\delta} : P \to \mathbb{R}^{n_x}$.

Parametric linear systems arise in a number of modeling problems. In particular, they commonly arise from the discretization of a time-independent partial differential

[1]Here, *large* and *sparse* matrices will be considered to have $> 10^2$ elements which are mostly zeros.

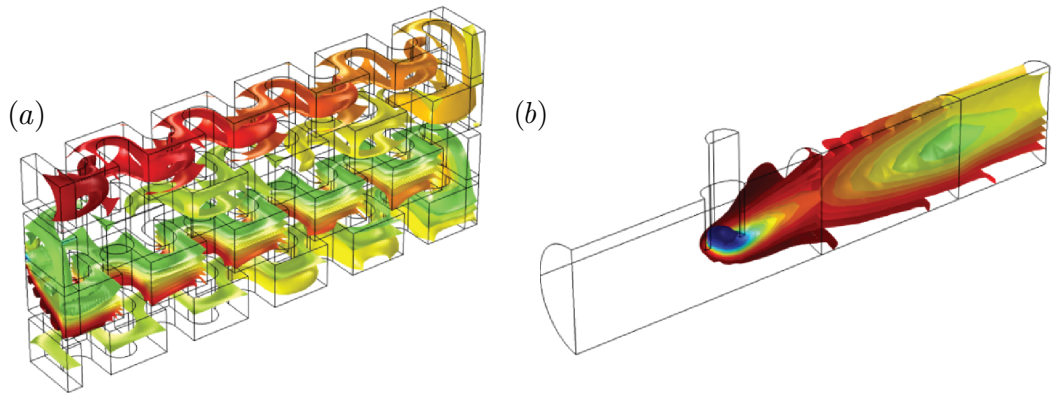

Figure 5-1: (a) A plate reactor for the production of fine chemicals. (b) A plug-flow reactor with injection needle. (Photo credit: Comsol)

equation (PDE)—or a system of PDEs—which is a popular method for approximating their solutions.[2] PDEs are encountered frequently when modeling complex systems from first principles and theory.[3] An example of such a problem is the optimal design of a chemical reactor whose model involves the steady-state species conservation equations (under appropriate assumptions):

$$\mathbf{v}\cdot\nabla C_i - D_i\nabla^2 C_i - R_{Vi} = 0, \quad i = 1, 2, \ldots, n, \tag{5.2}$$

where $\mathbf{v}$ is the velocity of the medium, C_i is the concentration of species i in the medium, D_i is the diffusivity of species i in the medium, and R_{Vi} is the molar rate of formation of species i per unit volume. The solution of two discretized models of chemical reactors are shown in Figure 5-1, both of which exhibit complex flow and heat transfer characteristics. There are many other applications of these types of problems that range from oncology [1, 124] to computational fluid dynamics (CFD) [65, 123, 144] to finance [20, 35, 36], among others.

In essence, this chapter will demonstrate the application of the algorithm developed in Chapter 4 using the relaxation technique of Section 4.3.3. As noted previously, the results in [88] laid the foundations for global optimization of implicit functions

[2] In other words, approximate solutions of PDEs can be obtained by applying discretization and finite differencing to formulate a parametric linear system and solving it.

[3] As opposed to phenomenological models.

which can be evaluated by an algorithm with a fixed number of iterations known *a priori*. The main contribution of this chapter is the discussion behind the implementation of the algorithm and some numerical analysis and comparison with the method of [88].

5.1 Computational Efficiency

One point that was emphasized in the previous chapter was that the solution time of deterministic global optimization algorithms, in the worst case, scales exponentially with the number of optimization variables. Therefore, the computational efficiency of all internal routines and algorithms—such as the upper- and lower-bounding procedures—is of great importance in order to minimize computational effort. In the case of global optimization of implicit functions, this means that the procedures for bounding and evaluating implicit functions must be highly optimized.[4]

5.1.1 Matrix Storage

Since the matrix $\mathbf{A}$ has $n_x \times n_x$ elements, in order to store it on a computer, an array with size n_x^2 must be allocated, which may be prohibitively large. Similarly, simply accessing the data in this array may be a computational bottleneck since it is an $\mathcal{O}(n_x^2)$ operation.[5] However, if $\mathbf{A}$ is sparse, most of its elements are zeros. Since $\mathbf{A}$ is also large, this amounts to unnecessarily storing a large number of zeros in the array. By storing only the necessary information of $\mathbf{A}$, a significant reduction in the number of operations and memory usage can be taken advantage of.

There are a few different methods for storing sparse matrices on a computer. The main idea is to store every nonzero entry and its position. The simplest method for doing this—and the one employed later in this chapter—is called the *coordinate* format [121], or oftentimes referred to as the standard triple format. The idea here is

[4]Referring to memory management and minimizing the overall computational cost of the associated procedures.

[5]The standard convention for computational complexity is used here meaning that this operation is proportional to n_x squared floating-point operations or FLOPs.

to calculate the number (or maximum number) of nonzero entries, denoted nz, and allocate three arrays of length nz. Two of the arrays will store the row coordinate and the column coordinate of each nonzero entry, respectively, and the third array will store each nonzero value at the respective coordinates. For example:

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 7 \\ 0 & 0 & 8 & 0 \end{pmatrix} \Leftrightarrow \begin{cases} \text{row index} & (1 \quad 3 \quad 4 \quad 3) \\ \text{column index} & (1 \quad 2 \quad 3 \quad 4) \\ \text{matrix element} & (5 \quad 9 \quad 8 \quad 7) \end{cases}$$

This example is not terribly convincing of the benefits of using this sparse storage format since storing the original matrix requires storing 16 double-precision numbers while the coordinate storage format requires storing 4 double-precision numbers and 8 integers. However, the benefits of sparse matrix storage are magnified when the size of the sparse matrix is very large. There are other more clever storage procedures that reduce the total storage requirement marginally over this method, such as *compressed sparse row*, *compressed sparse column*, *linked lists*, etc. [34, 121]. It is common that nz is known precisely, as well as the positions of all the nonzero elements, and so creating/storing the $\mathbf{A}$ matrix is an $\mathcal{O}(nz)$ operation that requires two arrays of nonnegative integers and one of double precision numbers. Similarly, accessing data from a sparse matrix stored in this format is an $\mathcal{O}(nz)$ operation.

5.1.2 Matrix Structure

The types of models that are most often encountered in engineering applications—which are the focus here—give rise to very large sparse matrices with special structure. In particular, the matrices that typically arise have a *banded* structure. Figure 5-2 depicts the concept of a banded matrix and its *bandwidth*. Exploiting this special structure can prove to be very advantageous in terms of computational efficiency since every implicit function evaluation requires the solution of a parametric linear system. For instance, solving a parametric linear system for only a single parameter value using

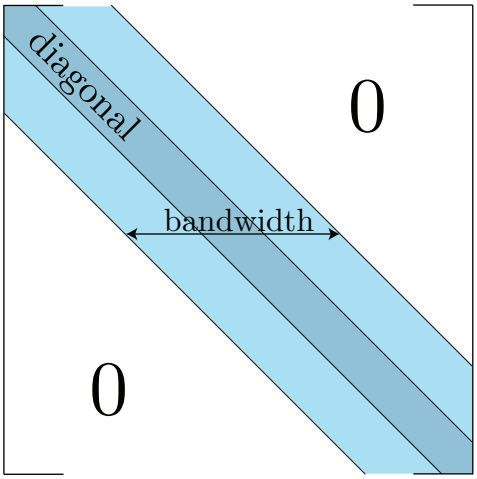

Figure 5-2: A banded matrix is depicted and its bandwidth is defined.

naive Gauss elimination is an $\mathcal{O}(n_x^3)$ operation, resulting in very expensive function evaluations. However, in [7] the author explains that "Gauss elimination for tightly banded matrices is particularly efficient, because for each row there are at most [the half-bandwidth] elements below the diagonal that must be eliminated". Labeling the half-bandwidth as m, the author then explains that Gaussian elimination is reduced to an $\mathcal{O}(m^2 n_x)$ operation for banded matrices, where $m << n_x$ [7]. Therefore, for small m, Gauss elimination is actually quite efficient. This is just one example but it motivates the importance of exploiting the structure of $\mathbf{A}$ in terms of computational efficiency.

Using graph theory to represent the structure of a sparse matrix, methods have been developed for permuting sparse matrices to have special structure, which, in turn, can be exploited for computational efficiency. For instance, the bandwidth (or half-bandwidth) is one important feature of banded matrices that may drastically impact computational efficiency. This fact has sparked a large amount of interest to research what are called *reordering schemes* for permuting sparse matrices into matrices with special structure, such as a banded matrix with the minimum bandwidth. There are many other reordering schemes that focus on other features of sparse matrices to increase computational efficiency. For instance, reordering schemes play an important role in parallel implementations of solution methods [121] and such implementations may require special structure other than minimum bandwidth.

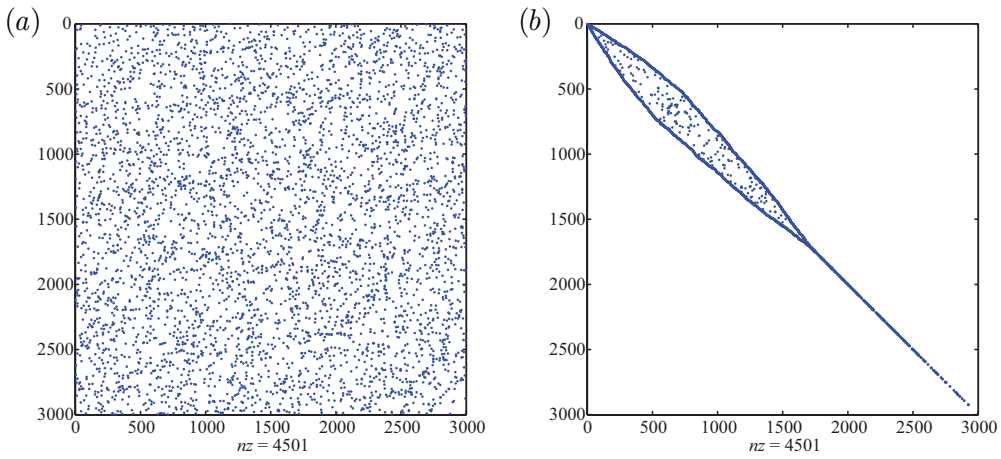

Figure 5-3: (a) The sparsity pattern of a randomly-generated sparse symmetric matrix. (b) The sparsity pattern of the same matrix as (a) after applying the reverse Cuthill-McKee reordering scheme.

Perhaps the most commonly used reordering scheme for minimizing bandwidth is the reverse Cuthill-McKee algorithm [34], which is simply the reversed ordering of what is achieved using the Cuthill-McKee algorithm [32]. In [34], the authors state that although there have been several other reordering schemes proposed, "they do not offer any significant advantages". Figure 5-3 shows just how dramatic an effect the reverse Cuthill-McKee algorithm may have on the bandwidth of a sparse matrix.

Another important feature of sparse matrices is *diagonal dominance*.

Definition 5.1.1 (Diagonally Dominant)**.** A matrix $\mathbf{A}$ is *diagonally dominant* if

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|, \; i = 1, 2, \ldots, n_x,$$

where a_{ij} is the $(i,j)^{\text{th}}$ element of $\mathbf{A}$. If the inequality holds strictly, $\mathbf{A}$ is said to be *strictly diagonally dominant*.

For parametric systems $\mathbf{A}(\mathbf{p})$, the above definition must hold for every $\mathbf{p} \in P$. If $\mathbf{A}$ is diagonally dominant, a host of theoretical and numerical implications result. For instance, results regarding the applicability of certain algorithms, stability, convergence properties, etc. [28, 121]. Luckily enough, diagonally dominant matrices are

commonly encountered when modeling engineering applications.

5.1.3 Numerical Solution Methods: Direct vs. Iterative

Direct methods refer to a class of numerical methods that are used in an attempt to solve problems by applying a finite sequence of operations. Not only are direct methods very robust, but they have the ability to calculate the exact solution of the linear system if numerical roundoff error can be avoided. An example of a direct method for solving linear systems is Gauss elimination, mentioned previously. The calculation of interval bounds and convex/concave relaxations of solutions of parametric linear systems is straightforward since the rules for their calculation/construction are simply propagated through the finite sequence of operations in the standard manner.

Alternatively, *iterative methods* attempt to solve problems by calculating sequences of approximations of the solution and terminate after meeting some error tolerance criterion. One advantage of iterative methods is that, in comparison to direct methods, their computational cost is far lower. Due to computational limitations, iterative methods are often the only technique applicable to solving large sparse systems. An example of an iterative method for solving linear systems is Gauss-Seidel. Iterative methods for bounding solutions of parametric linear systems are discussed in [106, 107, 119]. Similarly, an iterative method for constructing convex/concave relaxations of solutions of parametric linear systems was developed in Section 4.3.3.

In Chapter 4, it was discussed that the work in [88] was the first attempt at global optimization of implicit functions. However, the method of [88] was restricted to problems in which the implicit functions could be evaluated using direct methods [88]. Specifically, they present a parameter estimation problem involving the one-dimensional heat transport model with an affine heat-source term formulated as a parametric linear system whose $\mathbf{A}$ matrix is a tridiagonal banded matrix[6] [88]. This model may be an ideal candidate for direct methods because of its structure.

However, as the model complexity increases and/or includes more than one spatial

[6]A tridiagonal matrix is a banded matrix with $m = 1$. Solving tridiagonal systems using Gauss elimination is an $\mathcal{O}(n_x)$ operation.

dimension, direct methods may not be favored due to their higher storage requirement and computational cost. This is primarily due to a characteristic of direct methods known as *fill-in*. Fill-in is simply when some matrix operation generates and stores a nonzero element which was formerly a zero and in turn increases the number of nonzero elements that must be accounted for and operated on. In other words, nonzero elements *fill in* the sparse matrix. Reducing fill-in is a primary goal of some reordering schemes [121].

Fill-in may have a negative impact on more than just storage and computational efficiency as well. Consider the requirement to bound solutions of parametric linear systems. If fill-in is significant, this means that there are many more interval operations required, and therefore there is a large potential for considerable overestimation. Subsequently, this translates to the construction of convex/concave relaxations; producing relaxations that are less tight than if fill-in could be avoided. Alternatively, fill-in can be avoided entirely by applying an iterative method.

In [31], it is stated that "as the problem size grows, there is a point at which direct methods become too expensive in terms of computational time and storage." It is for this reason, and the others listed above, that iterative methods are favored for global optimization of large sparse systems.

5.1.4 Preconditioning

As was seen in Chapters 3 and 4, preconditioning matrices[7] may play a pivotal role in the ability to solve parametric systems as well as calculate bounds and construct relaxations of solutions of parametric systems. As was shown in previous sections, for a preconditioner $\mathbf{Y} \in \mathbb{R}^{n_x \times n_x}$, a preconditioned system takes the form

$$\mathbf{YA}(\mathbf{p})\mathbf{z} = \mathbf{Yb}(\mathbf{p}),$$

and its solution is the same as the original system. Preconditioning large sparse systems will be the focus here. For the concerns of this chapter, preconditioning

[7] Also referred to as preconditioners.

offers a way to manipulate and scale large sparse systems to have special structure and theoretical properties that are ideal for iterative solvers. Consequently, preconditioning matrices potentially accelerate iterative methods and guarantee convergence properties of iterative methods for solving linear systems.

In Chapter 3, the preconditioner used is equivalent to the inverse of $\mathbf{A}(\hat{\mathbf{p}})$ with $\hat{\mathbf{p}} = mid(P)$, for a given P. The problem with using this preconditioner for large sparse systems is that it is computationally expensive in multiple ways. For instance, in general, the inverse of a sparse matrix (if it exists) is not sparse. Furthermore, the product of a dense matrix and a sparse matrix is not typically sparse. Therefore, after spending an extraordinarily large amount of computational effort to calculate the preconditioner, the resulting preconditioned system is large and dense. Therefore, in this case, the computational cost associated with preconditioning the system is likely to outweigh the gain in computational efficiency of the iterative method used to solve the preconditioned system. In [121], it is stated that "finding a good preconditioner [to solve a specific problem] is often viewed as a combination of art and science." This has sparked much interest in finding *optimal* preconditioners.

Numerous preconditioners—each with their own advantages and disadvantages—are discussed in [121]. The simplest, and possibly the least expensive, preconditioner that preserves sparsity and may be quite effective for diagonally dominant systems is the *diagonal* preconditioner, or sometimes called the Jacobi preconditioner [121]. This preconditioner is simply the inverse of the diagonal elements of $\mathbf{A}$:

$$\mathbf{Y}(\mathbf{p}) = \begin{pmatrix} 1/a_{11}(\mathbf{p}) & 0 & \cdots & \cdots & 0 \\ 0 & 1/a_{22}(\mathbf{p}) & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1/a_{n_x n_x}(\mathbf{p}) \end{pmatrix}.$$

The elements of the resulting preconditioned matrix $\mathbf{G}(\mathbf{p}) = \mathbf{YA}(\mathbf{p})$ are then given by $g_{ij}(\mathbf{p}) = a_{ij}(\mathbf{p})/a_{ii}(\mathbf{p})$. Of course, this requires that $a_{ii}(\mathbf{p}) \neq 0,\ \forall \mathbf{p} \in P$. When applied to a system that will be solved using Gauss-Seidel, it eliminates the need for the division by the diagonal elements at each iteration. For systems requiring many

iterations to converge, this may offer a significant performance benefit. It should be noted that for parametric linear systems, which are the focus here, it is most beneficial if the preconditioner is constant, and not a function of $\mathbf{p}$. That way, for every point $\mathbf{p}$ at which the implicit function needs to be calculated, the preconditioner need not be updated. This is especially important if the preconditioner is expensive to calculate.

5.2 Methods

A few different methods for constructing relaxations of solutions of parametric linear systems were implemented and studied. As mentioned above, each of these methods can be categorized as either a direct or iterative approach. Each method is discussed in this section. For analysis and benchmarking purposes, a general code was written that, for a chosen method, constructs relaxations at a predetermined number of parameter values and clocks the overall computation.

5.2.1 Direct Approach

For comparison with the iterative techniques being considered below, a direct approach was implemented. This approach is the same as that in [88] implemented to exploit the banded structure (i.e., it only considers matrix operations within the bands). In order for this approach to offer a fair comparison with the iterative methods, the reverse Cuthill-McKee reordering scheme was implemented to permute the system prior to the application of the method.

5.2.2 Iterative Approach

For comparison purposes, multiple iterative methods for constructing relaxations of solutions of parametric linear systems were implemented. In particular, the method discussed in Section 4.3.3 was implemented in multiple ways: one that exploits sparsity fully (for non-preconditioned and diagonally preconditioned systems), a non-preconditioned sparse *multiple inclusion* implementation, and a dense inverse mid-

point preconditioned method. For each of the implementations that exploit sparsity, the coordinate storage format was used, and all `for` loops were done over nonzero elements only.

Non-Preconditioned and Diagonally Preconditioned Implementation

This implementation exploits sparsity fully by only looping over the nonzero elements of $\mathbf{A}$. Essentially, the method described in Section 4.3.3 constructs relaxations of the i^{th} component of the implicit function $\boldsymbol{\delta}$ by using the information of the i^{th} equation of the linear system. For instance, relaxing δ_i amounts to relaxing the expression from (4.11) which includes two summations over off-diagonal elements of $\mathbf{A}$. Since the row and column index vectors contain the information for keeping track of the coordinates of the nonzero elements of $\mathbf{A}$, rather than the summations being over a total of $n_x - 1$ elements, they are only over the half-bandwidth m number of elements each, in the worst case that the matrix $\mathbf{A}$ is dense within its bands. Therefore, constructing relaxations of $\boldsymbol{\delta}$ is an $\mathcal{O}(nz)$ operation. Of course, these relaxations can be iteratively refined and so constructing the tightest possible relaxations of $\boldsymbol{\delta}$ is an $\mathcal{O}(knz)$ operation, where k is the number of iterations it takes to refine the relaxations as much as possible (until they converge). This implementation works best for the original non-preconditioned system and preconditioned systems where sparsity is preserved, such as diagonally preconditioned systems.

Multiple Inclusion Implementation

This implementation is similar to the previous one in that it only loops over nonzero elements. Whereas the previous method considers only constructing relaxations of δ_i using information from row i, this method construct relaxations of δ_j, $\forall j = i - m, \ldots, i, \ldots i + m$ using the information from row i corresponding to $a_{ij} \neq 0$. For instance, for some row i, relaxations of δ_j will be calculated by relaxing the right-hand

side of

$$\delta_j(\mathbf{p}) = \left(b_i(\mathbf{p}) - \sum_{l<i} a_{il}(\mathbf{p})\delta_l(\mathbf{p}) - \sum_{l>i} a_{il}(\mathbf{p})\delta_l(\mathbf{p}) \right) / a_{ij}(\mathbf{p}),\ a_{ij}(\mathbf{p}) \neq 0,\ \forall \mathbf{p} \in P,$$

for each j. After looping over all rows, the method then intersects all the relaxations δ_j for each j in an attempt to make use of all available information for constructing the tightest possible relaxations. The benefit of this method is that it is unnecessary to guarantee that the diagonal elements are nonzero (or do not enclose zero). Therefore it is unnecessary to precondition on the basis of manipulating $\mathbf{A}$ to have nonzero diagonal elements. The downside of this method is that each iteration of refining the relaxations is much more expensive than the previous implementation. Also, no new information is obtained by using this method, and therefore it yields the same results as the previous implementation as long as $\mathbf{A}$ has an appropriate structure (e.g. nonzero diagonal elements).

Dense Preconditioned Implementation

This implementation can be regarded as a naive approach to calculating relaxations of $\boldsymbol{\delta}$. This is because it amounts to taking a large sparse system, calculating an expensive preconditioner, and in turn, calculating relaxations of the solution of a large dense preconditioned system. Although it is quite effective for small systems, it is extremely expensive since simply constructing $\mathbf{Y}$ is an $\mathcal{O}(n_x^3)$ operation using LU-decomposition.[8] Although this computation can be done up front (and only once), calculating $\mathbf{G}(\mathbf{p}) = \mathbf{YA}(\mathbf{p})$ is an $\mathcal{O}(n_x^3)$ operation and constructing relaxations of $\boldsymbol{\delta}$ is $\mathcal{O}(kn_x^2)$, where again k is the number of iterative refinements taken. It is clear that for large systems, the $\mathcal{O}(n_x^3)$ procedure will dominate and so the overall procedure for constructing relaxations of $\boldsymbol{\delta}$ is $\mathcal{O}(n_x^3)$.

[8] Using Gauss elimination alone for matrix inversion amounts to an $\mathcal{O}(n_x^4)$ computation.

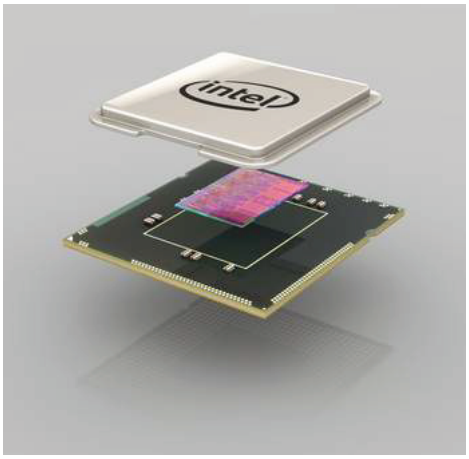

Figure 5-4: An exploded view of a packaged CPU. The packaged device consists of a substrate (black bottom), the processor die (multicolor center), and the heat spreader (metallic top). (Photo credit: Intel®)

5.3 Case Study

To demonstrate global optimization of large sparse systems, a parameter-estimation problem is considered which arises from modeling steady-state heat transport in a packaged semiconductor. Figure 5-4 shows the construction of a packaged semiconductor, specifically, that of a microprocessor or central processing unit (CPU).

5.3.1 Model and Objective

An illustration of the model system is shown in Figure 5-5. It is assumed that the heat spreader is a solid material that covers the substrate and the die with no void space. Furthermore, conduction in the substrate is negligible, and convection on every surface except the top is negligible. The physical design parameters of the system are contained in Table 5.1. To model heat transport for this system, the 3-dimensional energy conservation equation with an affine heat-source term (H_V) was used:

$$\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) + \frac{H_V}{k_c} = 0, \tag{5.3}$$

where T is the temperature, k_c is the (constant) thermal conductivity of the material, and the spatial coordinates will be denoted in the standard way by x, y, and z. Heat

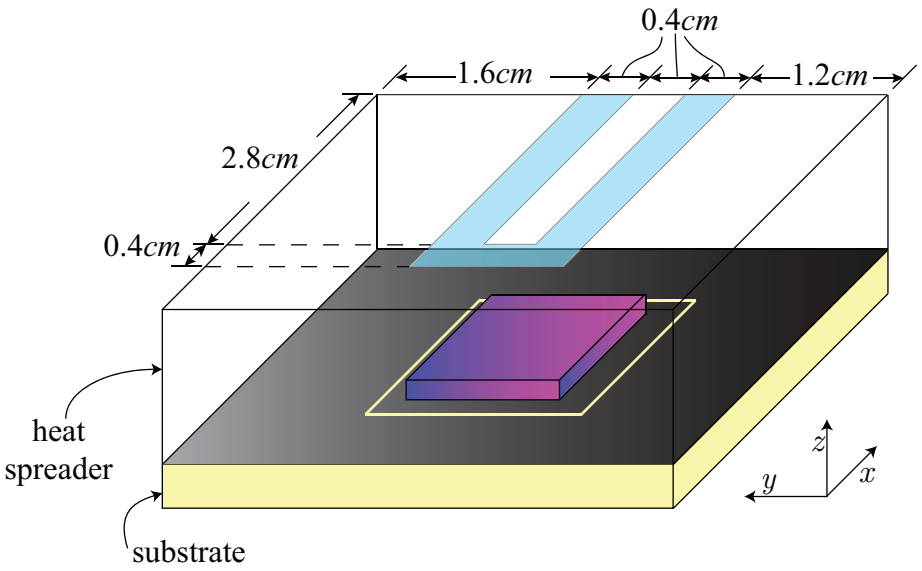

Figure 5-5: An illustration of the packaged CPU model for the parameter estimation problem. The dimensions and layout differ from the actual CPU shown in Figure 5-4 to introduce asymmetry for an irregular temperature profile. For clarity, the heat spreader is depicted as being completely transparent.

is generated in the model by the die, which is centered on the xy-plane, through the application of electric current (i.e. Joule heating). Resistance is expected to be affine in temperature, and so the source term takes the form:

$$H_V = q_0 \left(1 + \alpha(T - T_0)\right) \tag{5.4}$$

where q_0 and α are constants pertaining to physical properties of the semiconductor, and T_0 is the reference temperature, taken to be equal to the ambient (or bulk) temperature T_b. It is assumed that there is negligible heat flux on all of the boundaries except for the top surface where there is convection. The blue region on the top surface in Figure 5-5 is a region of convection where a liquid-cooling coil contacts the heat spreader. Convection occurs over the rest of the top surface of the heat spreader—albeit to a lesser degree—due to a nearby fan. The boundary conditions

are:

$$\begin{aligned} &\mathbf{n}\cdot\nabla T = 0 && \text{at every surface except } z = 1cm \\ &-\mathbf{n}\cdot\nabla T = h_1(T_{surf} - T_b)/k_c && z = 1cm \text{ blue region} \\ &-\mathbf{n}\cdot\nabla T = h_2(T_{surf} - T_b)/k_c && z = 1cm \text{ not blue region} \end{aligned} \tag{5.5}$$

where $\mathbf{n}$ is the unit normal directed away from the heat spreader, T_{surf} is the temperature of the fluid at the interface (assumed to be the same as the surface temperature of the solid), h_1 is the heat transfer coefficient of the fluid in blue region and h_2 is the heat transfer coefficient of the fluid elsewhere on the top surface. For this problem, the model parameters for fitting will be h_1, the heat transfer coefficient for the blue area, and T_b, the ambient or bulk temperature. Therefore

$$\mathbf{p} = (T_b \;\; h_1)^{\mathrm{T}}.$$

In order to construct the parametric linear system—whose solution will be an approximate solution of the original 3-dimensional PDE—a standard finite-differencing approach is used. First, a uniform discretization is applied (i.e. a discretization with an equal number of grid points in each spatial dimension $N_x = N_y = N_z$). A standard node-numbering scheme is used:

$$i_\delta = (k-1)N_xN_y + (i-1)N_y + j,$$

where i, j, k are the indices corresponding to the grid point (x_i, y_j, z_k). Thus, the grid points get numbered sequentially from 1 to $n_x = N_xN_yN_z$. In order to obtain a sufficient approximation of the solution of (5.3), a fine discretization of the spatial dimensions is required (i.e. N_x, N_y, N_z are sufficiently large). Since the problem size, n_x, is equal to $N_xN_yN_z$, it is easy to see why the problem size can be substantially large, even for a relatively coarse discretization. After applying the discretization, finite differencing is used to approximate the partial derivatives. For the interior nodes (i.e. nodes not on the boundary of the system), the approximate partial derivatives

CPU Design Specifications	
Packaged dimensions $(x \times y \times z)$	$4cm \times 4cm \times 1cm$
Die dimensions $(x \times y \times z)$	$1.6cm \times 1.2cm \times 0.1cm$
Substrate thickness	$0.2cm$
k_c	$4.0W/mK$
α	$-2.0 \times 10^{-6}K^{-1}$
q_0	$4500.0W/m^3$
h_1	$\in [1.5, 10]W/m^2K$
h_2	$0.0862W/m^2K$
T_b	$\in [250, 400]K$

Table 5.1: The physical design specifications for the CPU packaging problem.

are given by

$$\frac{T_{i_\delta - N_x N_y} - 2T_{i_\delta} + T_{i_\delta + N_x N_y}}{(\Delta z)^2} + \frac{T_{i_\delta - N_y} - 2T_{i_\delta} + T_{i_\delta + N_y}}{(\Delta x)^2} + \frac{T_{i_\delta - 1} - 2T_{i_\delta} + T_{i_\delta + 1}}{(\Delta z)^y} = -\frac{q_0}{k_c}(1 + \alpha(T_{i_\delta} - T_b)),$$

where $q_0 = 0$ everywhere except in the die. The boundary conditions are as follows:

$$\begin{aligned}
&\tfrac{T_{i_\delta+1} - T_{i_\delta}}{\Delta y} = 0 && \text{at } y_j = 0cm \\
&\tfrac{T_{i_\delta} - T_{i_\delta-1}}{\Delta y} = 0 && \text{at } y_j = 4cm \\
&\tfrac{T_{i_\delta+N_y} - T_{i_\delta}}{\Delta x} = 0 && \text{at } x_i = 0cm \\
&\tfrac{T_{i_\delta} - T_{i_\delta-N_y}}{\Delta x} = 0 && \text{at } x_i = 4cm \\
&\tfrac{T_{i_\delta+N_y N_x} - T_{i_\delta}}{\Delta z} = 0 && \text{at } z_k = 0cm \\
&\tfrac{T_{i_\delta} - T_{i_\delta-N_x N_y}}{\Delta z} = h_1(T_{i_\delta} - T_b)/k_c && \text{at } z_k = 1cm \text{ blue region} \\
&\tfrac{T_{i_\delta} - T_{i_\delta-N_x N_y}}{\Delta z} = h_2(T_{i_\delta} - T_b)/k_c && \text{at } z_k = 1cm \text{ not blue region}
\end{aligned} \tag{5.6}$$

After applying finite differencing, with some simple algebraic manipulation, a parametric linear system can be formed:

$$\mathbf{A}(\mathbf{p})\mathbf{z} = \mathbf{b}(\mathbf{p}),\ z_{i_\delta} = T_{i_\delta},\ \forall i_\delta = 1, \ldots, N_x N_y N_z.$$

Since this is a parameter estimation problem, the optimization formulation has a

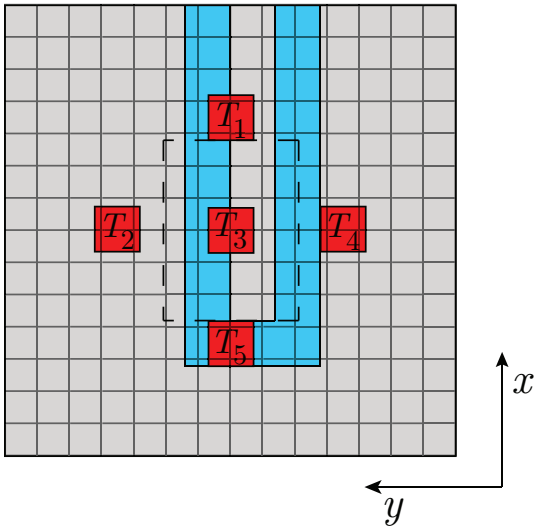

Figure 5-6: An overhead view of the packaged CPU model. Temperature sensors corresponding to the fictitious experiment are depicted in red. The grid lines correspond to a 15×15 mesh. The dotted rectangle corresponds to the location of the die projected onto the surface.

least-squares objective function:

$$f(\boldsymbol{\delta}(\mathbf{p}), \mathbf{p}) = \sum_{n=1}^{N_{\text{sensors}}} \left(T_n - \bar{\delta}_n(\mathbf{p})\right)^2, \tag{5.7}$$

where N_{sensors} is the number of fictitious experimental temperature sensors, T_n is the temperature reading taken by the n^{th} sensor averaged over the area of the sensor, and $\bar{\delta}_n$ will be the model-predicted temperature as the average temperature over the number of mesh points (nodes) that lie within the n^{th} sensor region. Figure 5-6 depicts an overhead view of the packaged CPU model with the sensors corresponding to the fictitious experiment shown in red. The gridding corresponds to a 15×15 mesh, and each δ_i corresponds to the temperature—as predicted by the model—at each of the intersections of the grid lines. Note that the sensors $n = 1, 2, 3, 4$ each correspond to regions enclosing two mesh points (nodes). The fictitious experimental data is given in Table 5.2. For the iterative method from Section 4.3.3, the variable interval $Z_{i_\delta} = [250, 1000]$, $\forall i_\delta = 1, \ldots, N_x N_y N_z$, was chosen from physical intuition. The interval X satisfying Definition 4.4.1 was calculated using parametric interval Gauss-Seidel [101].

Experimental Temperature Data	
T_1	$411.0K$
T_2	$448.85K$
T_3	$422.125K$
T_4	$492.05K$
T_5	$394.7K$

Table 5.2: The experimental temperature data for CPU design problem.

5.3.2 Comparison of Methods

Numerical experiments were conducted to demonstrate the performance of each of the methods discussed above in terms of computational complexity. The experiments timed how long it took to evaluate a convex relaxation of one of the states δ_i at 36 $\mathbf{p}$ values while varying the number of discretization points (size of the system) used to approximate the solution of the PDE. This simulates the computational complexity associated with evaluating the objective function of the convex lower-bounding problem as a function of problem size. The results of the numerical experiments are shown in Figure 5-7. The results for the multiple-inclusion implementation were omitted for clarity.

It was observed that sufficiently tight relaxations of $\boldsymbol{\delta}$ were obtained after $N_x N_y N_z$ iterations of the relaxation technique discussed in Section 4.3.3. The unfortunate consequence of this result is the increased computational cost of calculating these relaxations. As discussed in the methods section above, the non-preconditioned and diagonally preconditioned implementations exhibit $\mathcal{O}(knz)$ complexity. For this example,

$$nz = 4N_yN_z + 4(N_x - 2)N_z + 4(N_x - 2)(N_y - 2) + 7(N_x - 2)(N_y - 2)(N_z - 2),$$

which is on the order of n_x. Therefore, since $k = n_x$, the complexity of calculating relaxations of $\boldsymbol{\delta}$ was hypothesized to be $\mathcal{O}(n_x^2)$. As can be seen from Figure 5-7, the (numerical) experimental data supports this hypothesis.

As discussed in the Methods section above, the Gauss elimination implementations and the dense preconditioned implementation were hypothesized to have $\mathcal{O}(n_x^3)$

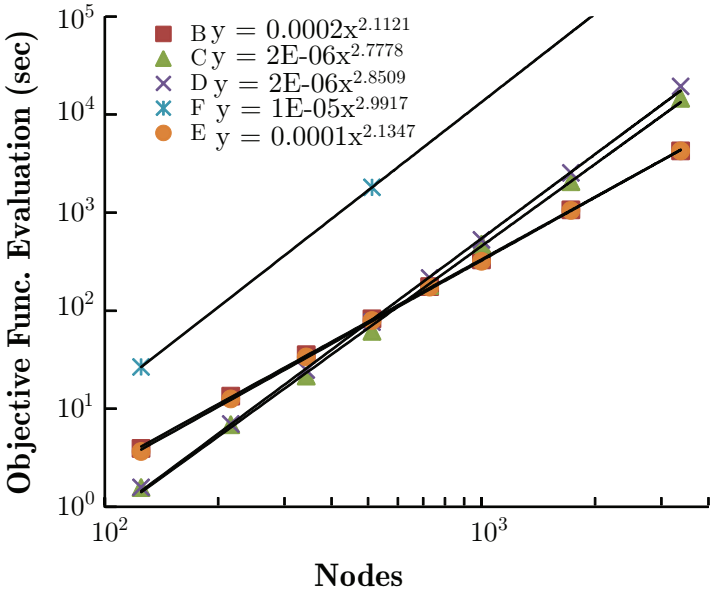

Figure 5-7: The scaling of the evaluation of convex relaxations for each method as a function of the problem size. Method B: Non-preconditioned impl. Method C: Direct method w/RCMK. Method D: Direct method. Method E: Diagonally preconditioned imp. Method F: Inverse-midpoint (dense) preconditioned imp.

complexity with the reordered implementation being slightly less expensive (due to minimizing fill-in). Figure 5-7 supports this hypothesis as well. Interestingly, the Gauss elimination implementations required less CPU seconds to calculate relaxations for $n_x < 729$ (corresponding to $N_x = N_y = N_z = 9$). This is likely due to the overhead of calculating interval bounds using the (parametric) interval linear solver, which is equivalent to the interval Gauss-Seidel iteration [101]. Since the direct methods scale worse with system size than the iterative method, even with this overhead, the iterative methods perform better than the direct methods for $n_\delta \geq 729$ ($N_x = N_y = N_z \geq 9$). In order to adequately model the system, $N_x = N_y = N_z \geq 15$ is required, and so for this model, the iterative methods perform more favorably.

5.4 Concluding Remarks

In this chapter, the results developed in Section 4.3.3 were explored further and applied to large sparse systems which commonly arise in engineering applications. A brief background on large sparse systems was given including various approaches for

solving them numerically as well as computational effort and storage requirements. However, the main contribution is a parameter-estimation case study motivated by an engineering design problem. The computational effort of the iterative relaxation method of Section 4.3.3 was compared against the direct relaxation method of [88]. The method of Section 4.3.3 performed favorably as it scaled with the square of the dimension of the problem as opposed to the cubic scaling of the direct method of [88]. These initial results demonstrate that the iterative approach of Section 4.3.3 for constructing relaxations may be quite effective for large sparse systems and with future research and the proper computer implementation, global optimization of large sparse systems can be solved quickly and efficiently.

Chapter 6

Relaxations of Implicit Functions Revisited

In this chapter, the theoretical arguments behind the construction of convex and concave relaxations of implicit functions are revisited. In particular, in Chapter 4, the assumptions regarding uniqueness of an implicit function $\mathbf{x}$ on P may be too restrictive in some cases. The effects of relaxing the uniqueness assumptions are explored in this chapter.

6.1 Direct Relaxation of Fixed-Point Iterations

In Section 4.3.1, Assumption 4.3.1 was applied, which stated that for $P \in \mathbb{IR}^{n_p}$, there exists an implicit function $\mathbf{x} : P \to \mathbb{R}^{n_x}$ such that it is a fixed-point of the function $\boldsymbol{\phi}$ (as defined in that section). Furthermore, it was assumed that an interval X was known such that $\mathbf{x}(P) \subset X$ and $\mathbf{x}(\mathbf{p})$ is unique in X for all $\mathbf{p} \in P$.

However, these assumptions are not entirely necessary for the main results of that section (Theorems 4.3.4, 4.3.5, and 4.3.6) to hold. The results of that section are generalized as follows.

Theorem 6.1.1. *Let $\boldsymbol{\phi}$ be defined as in Section 4.3.1. Suppose $\boldsymbol{\phi}$ has n fixed-points $\mathbf{x}_i(\mathbf{p})$, $i = 1, \ldots, n$, for every $\mathbf{p} \in P$ such that $\mathbf{x}_i(P) \subset X$ for some $X \in \mathbb{IR}^{n_x}$ for $i = 1, \ldots, n$. Then, Theorem 4.3.4 holds.*

Proof. As long as $\mathbf{x}_i(P) \subset X,\ i = 1, \ldots, n$, the proof of Theorem 4.3.4 holds under the relaxed hypotheses since $\mathbf{x}^{k,c}(\cdot\,)$ and $\mathbf{x}^{k,C}(\cdot\,)$ are relaxations of $\mathbf{x}_i(\cdot\,) = \boldsymbol{\phi}(\mathbf{x}_i(\cdot\,), \cdot\,)$ for $i = 1, \ldots, n$ on P. □

Theorem 6.1.2. *Suppose the hypotheses of Theorem 6.1.1 hold. Then, Theorem 4.3.5 holds.*

Proof. The proof of Theorem 4.3.5 holds here with $\mathbf{x}$ replaced by $\mathbf{x}_i,\ i = 1, \ldots, n$. □

Theorem 6.1.3. *Suppose the hypotheses of Theorem 6.1.1 hold. Then, Theorem 4.3.6 holds.*

Proof. The proof of Theorem 4.3.6 holds here without modification. □

If more than one fixed point of $\boldsymbol{\phi}$ exists in X, it is likely that the hypotheses of Theorem 4.3.6 will hold and therefore the relaxations calculated by applying Theorem 6.1.1 will not be improvements on some of the interval bounds.

6.2 Relaxations of Solutions of Parametric Linear Systems

In Section 4.3.3, Assumption 4.3.9 was applied. Assumption 4.3.9-1 is essentially the same as Assumption 4.3.1 but stated with respect to parametric linear systems. In that section, it was already discussed how Assumption 4.3.9-2 can be relaxed by introducing a preconditioning matrix $\mathbf{Y} \in \mathbb{R}^{n_x \times n_x}$. The results of Section 4.3.3 are generalized as follows.

Theorem 6.2.1. *Let* $\mathbf{A}$ *and* $\mathbf{b}$ *be as in Section 4.3.3 and suppose either Assumption 4.3.9-2 holds or that we have a proper preconditioning matrix* $\mathbf{Y}$*. Suppose that there exist* n *implicit functions* $\boldsymbol{\delta}_i,\ i = 1, \ldots, n$*, such that* $\mathbf{A}(\mathbf{p})\boldsymbol{\delta}_i(\mathbf{p}) = \mathbf{b}(\mathbf{p}),\ i = 1, \ldots, n$*, for all* $\mathbf{p} \in P$ *and* $\boldsymbol{\delta}_i(P) \subset \Delta,\ i = 1, \ldots, n$ *for* $\Delta \in \mathbb{IR}^{n_x}$*. Then, Theorem 4.3.12 holds.*

Proof. As long as $\boldsymbol{\delta}_i(P) \subset \Delta,\ i = 1, \ldots, n$ holds, the proof of Theorem 4.3.12 holds under the relaxed hypotheses since $\boldsymbol{\delta}^{k,c}$ and $\boldsymbol{\delta}^{k,C}$ are relaxations of $\boldsymbol{\delta}_i,\ i = 1, \ldots, n_x$ on P. □

Theorem 6.2.2. *Suppose the hypotheses of Theorem 6.2.1 hold. Then, Theorem 4.3.14 holds.*

Proof. The proof of Theorem 4.3.14 holds here with $\boldsymbol{\delta}$ replaced with $\boldsymbol{\delta},\ i = 1, \ldots, n$. □

6.3 Relaxations of Solutions of Parametric Nonlinear Systems

Similar to the previous two sections, the uniqueness assumption is not required as long as proper interval bounds are known or are calculable that enclose all relevant implicit functions. Each result of Section 4.3.4 will not be generalized explicitly here. However, it should be known that the results still hold without the requirement that X encloses a unique implicit function as long as X encloses *all* relevant implicit functions.

6.4 Global Optimization of Implicit Functions

In the previous sections of this chapter, the uniqueness assumption imposed on Chapter 4 was relaxed. Under the relaxed assumption, the results of Section 4.3 are still valid. However, when considering using these relaxations within the branch-and-bound algorithm for global optimization (Alg. 4.1), convergence issues arise. In Section 4.4, finite convergence of Algorithm 4.1 was established under the relatively strict definition of X (Def. 4.4.1), which requires that there exists a unique implicit function enclosed by X. If X encloses multiple implicit functions, Definition 4.4.1 does not hold and the algorithm will fail to converge.

In the next chapter, the concept of semi-infinite optimization with embedded implicit functions is revisited. With the developments of Chapter 4, the solution of implicit SIPs, including the robust simulation SIP (2.5), is formalized.

Chapter 7

Semi-Infinite Optimization with Implicit Functions

In this chapter, the solution of implicit semi-infinite programs is discussed. Using the developments of Chapter 3 for bounding implicit functions as well as the algorithm for global optimization of implicit functions developed in Chapter 4, an algorithm for solving semi-infinite programs with implicit functions embedded is presented. The algorithm is guaranteed to converge to global ϵ-optimality in finitely many iterations given the existence of a Slater point arbitrarily close to a minimizer. Besides the Slater point assumption, it is assumed that the functions are continuous and factorable, and that the model equations are once continuously differentiable. The algorithm applies to implicit SIPs in general, and is therefore not restricted to only the robust feasibility SIP (2.5). As a consequence, a much more general optimization approach to process design problems will be discussed along with a more general implicit SIP formulation.

7.1 Introduction

Many engineering design problems give rise to optimization problems whose feasible sets are parameterized. As motivated in Chapter 1, this is because it is often of great interest to study performance and/or safety of engineering systems under parametric uncertainty. Particularly, it is important to study the performance/safety in the face

of the worst case, which gives rise to equality-constrained bilevel programs of the form:

$$\begin{aligned} f^* = \quad & \min_{\mathbf{y}} f(\mathbf{y}) \\ & \text{s.t. } 0 \geq \max_{\mathbf{p},\mathbf{z}} g(\mathbf{z},\mathbf{y},\mathbf{p}) \\ & \qquad \text{s.t. } \mathbf{h}(\mathbf{z},\mathbf{y},\mathbf{p}) = \mathbf{0} \\ & \mathbf{y} \in Y = \{\mathbf{y} \in \mathbb{R}^{n_y} : \mathbf{y}^L \leq \mathbf{y} \leq \mathbf{y}^U\} \\ & \mathbf{p} \in P = \{\mathbf{p} \in \mathbb{R}^{n_p} : \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\} \\ & \mathbf{z} \in D_x \subset \mathbb{R}^{n_x} \end{aligned} \tag{7.1}$$

which is a more general form of the constrained max-min program (2.3), introduced in Chapter 2. For the purposes of maintaining consistency with the standard form of SIPs and the SIP literature, in this chapter, the variable $\mathbf{p}$ will be the standard parameterization variables which may represent parametric uncertainty and/or various other model parameters such as the controls. The variables $\mathbf{y}$ will be introduced as the decision variables of the outer program, which might correspond to various design variables etc. In standard SIP form, the decision variables are typically taken to be $\mathbf{x}$. For consistency with the previous chapters, the variables $\mathbf{x}$ will be reserved for the state variables as implicit functions, whereas the variables $\mathbf{z}$ will still represent the internal state variables.

In [89], an algorithm for solving general nonconvex bilevel programs to ϵ global optimality was developed. However, since (7.1) contains equality constraints, Assumption 3 in [89] cannot be satisfied and therefore is not applicable to (7.1). Similarly, in [139] an algorithm for global optimization of bilevel programs was presented. The authors rely on the convergence result of [18], which is only guaranteed to terminate in finitely many iterations provided the functions are convex.

The objective function $f : D_y \to \mathbb{R}$ and the inequality constraint function $g : D_x \times D_y \times D_p \to \mathbb{R}$ are continuous and are factorable in the sense that they are composed from elementary arithmetic operations and transcendental intrinsic functions. The

equality constraints are considered as the system of equations representing a steady-state model of the system of interest:

$$\mathbf{h}(\mathbf{z}, \mathbf{y}, \mathbf{p}) = \mathbf{0} \tag{7.2}$$

with $\mathbf{h} : D_x \times D_y \times D_p \to \mathbb{R}^{n_x}$ factorable and continuously differentiable on its domain with $D_x \subset \mathbb{R}^{n_x}$,$D_y \subset \mathbb{R}^{n_y}$,$D_p \subset \mathbb{R}^{n_p}$ open. Due to the complexity of many process systems models, the bilevel formulation (7.1) is intractable, or even impossible to solve.

In a similar scheme to the previous chapters, the equality constraints can be used to eliminate $\mathbf{z}$ from (7.1) and in an analogous fashion to Chapter 2, the bilevel program (7.1) can be reformulated as an equivalent SIP without equality constraints. Again, if such $\mathbf{z}$ exist that satisfy (7.2) for each $(\mathbf{y}, \mathbf{p}) \in Y \times P \subset D_y \times D_p$, then it defines an implicit function of $(\mathbf{y}, \mathbf{p})$, expressed as $\mathbf{x}(\mathbf{y}, \mathbf{p})$. It will again be assumed that at least one implicit function $\mathbf{x} : Y \times P \to X$ exists such that $\mathbf{h}(\mathbf{x}(\mathbf{y}, \mathbf{p}), \mathbf{y}, \mathbf{p}) = \mathbf{0},\ \forall (\mathbf{y}, \mathbf{p}) \in Y \times P$ with $X \subset D_x$. Conditions guaranteeing uniqueness of $\mathbf{x} \in X$ were discussed in Chapter 3, as well as a method for calculating a relevant X. Given the existence of an implicit function $\mathbf{x}$ (and its uniqueness in X), the equality constraints can be eliminated and (7.1) can be expressed as:

$$\begin{aligned} f^* = \quad & \min_{\mathbf{y}} f(\mathbf{y}) \\ & \text{s.t. } 0 \geq \max_{\mathbf{p}} g(\mathbf{x}(\mathbf{y}, \mathbf{p}), \mathbf{y}, \mathbf{p}) \\ & \mathbf{y} \in Y = \{\mathbf{y} \in \mathbb{R}^{n_y} : \mathbf{y}^L \leq \mathbf{y} \leq \mathbf{y}^U\} \\ & \mathbf{p} \in P = \{\mathbf{p} \in \mathbb{R}^{n_p} : \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\}. \end{aligned} \tag{7.3}$$

Furthermore, using an identity similar to (2.4), the inner maximization program can be expressed as:

$$\max_{\mathbf{p} \in P} g(\mathbf{x}(\mathbf{y}, \mathbf{p}), \mathbf{y}, \mathbf{p}) \leq 0 \Leftrightarrow g(\mathbf{x}(\mathbf{y}, \mathbf{p}), \mathbf{y}, \mathbf{p}) \leq 0,\ \forall \mathbf{p} \in P, \tag{7.4}$$

where the latter constraint is referred to as the (implicit) semi-infinite constraint. The following implicit SIP, which is equivalent to the original bilevel program (7.1), can then be formulated:

$$\begin{aligned} f^* = \quad & \min_{\mathbf{y}} f(\mathbf{y}) \\ & \text{s.t. } g(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}) \leq 0,\ \forall \mathbf{p} \in P \\ & \mathbf{y} \in Y = \{\mathbf{y} \in \mathbb{R}^{n_y} : \mathbf{y}^L \leq \mathbf{y} \leq \mathbf{y}^U\} \\ & P = \{\mathbf{p} \in \mathbb{R}^{n_p} : \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\}. \end{aligned} \tag{7.5}$$

For a chemical engineering application, $\mathbf{z}$ may represent internal state variables, such as composition, determined by an equation of state or some other physics, the variables $\mathbf{y}$ may represent design variables such as chemical reactor dimensions or pipe lengths, and $\mathbf{p}$ may represent uncertain model parameters such as reaction rate constants. In this case, f may represent some economic objective related to sizing and g may represent a critical performance and/or safety constraint such as a constraint on selectivity or temperature. The global solution (if one exists) will correspond to the worst-case realization of uncertainty and address the question of optimal reactor design under uncertainty. Alternatively, $\mathbf{y}$ may represent uncertainty in the system and $\mathbf{p}$ may represent the controls. The function f may then represent some metric of uncertainty and g may again represent a performance and/or safety constraint. In that case, the global solution (if one exists) will correspond to the worst-case realization of uncertainty for which there exists a control setting such that the system meets specification. This formulation addresses the question of feasibility of the design as well as the determination of the maximum allowable uncertainty realization such the design remains feasible.

Solving SIPs which have only explicit functions, referred to as explicit SIPs herein, has been an active area of research for years. An overview of the previous application of explicit SIPs to real-world problems with theoretical results and available methods can be found in [58, 82, 111]. The contributions that have specific importance and implications for this thesis are summarized below.

Blankenship and Falk [18] presented an efficient cutting-plane algorithm for approximating solutions of explicit SIPs which amounts to solving two nonlinear programs (NLPs), to global optimality in the general case, at each iteration. Their algorithm generates a sequence of (infeasible) points that converge to the solution of the SIP in the limit [18]. Given appropriate convexity assumptions, their algorithm converges finitely to a feasible solution [18]. Their method is applicable to SIPs in general and they make specific mention of the application to the max-min problem. The max-min problem is further explored by Falk and Hoffman in [41] for general nonconvex functions. The cutting-plane algorithm relies on the techniques of discretization and what is called local reduction, which is a technique for theoretically describing (locally) the SIP feasible region with finitely many constraints [110]. Most SIP algorithms employ these techniques in various ways.

Zuhe et al. [145] presented a method based on interval analysis for solving explicit min-max problems, again, which are special instances of explicit SIPs. Their method is applicable to min-max problems with twice continuously differentiable explicit functions. Interval analysis was used to dynamically exclude regions of the search space guaranteed not to contain solutions [145]. It was suggested that, using the properties of interval analysis and generalized bisection, their method converges in finitely many iterations [145]. Bhattacharjee et al. [13] applied interval analysis to the general case of explicit SIPs in order to construct what is called the inclusion-constrained reformulation, which is a valid restriction of the original explicit SIP. This idea was used further in the first algorithm for generating SIP-feasible points finitely, that relies on the inclusion-constrained reformulation [14]. A lower-bounding procedure that relies on McCormick's convex and concave relaxations [85] and discretization was introduced [14]. Together with the inclusion-constrained reformulation and the branch-and-bound (B&B) framework, Bhattacharjee et al. was able to solve SIPs to global optimality with guaranteed finite ϵ-optimal convergence [14]. As previously mentioned, this algorithm was employed in [134] to solve implicit max-min problems cast as implicit SIPs. Due to the overestimation of inclusion functions and the fact that the size of the upper- and lower-bounding problems grow rapidly with depth in

the branch-and-bound tree [14], this algorithm can be ineffective at solving implicit SIPs modeling more complex processes. Bianco and Piazzi [15] developed a hybrid genetic-interval algorithm for solving SIPs. The hybrid algorithm approach attempts to circumvent the computational complexity of purely deterministic approaches while avoiding the problem of generating bounds on the extremum that aren't necessarily rigorous and a final solution that may be infeasible, inherent to purely stochastic (genetic) approaches. Although the authors state that the deterministic part of their algorithm can guarantee feasibility of the final solution, they also state that it cannot determine guaranteed (rigorous) bounds on the optimal solution.

Stein and Still [132] solved explicit SIPs, with g convex, as a Stackelberg game using an interior-point method. By convexity of g, they were able to exploit the first-order optimality conditions to characterize the solution set of the inner program and solve equivalent finite nonlinear programs [132]. Floudas and Stein [44] used a similar idea and constructed concave relaxations of g on P using αBB [2]. They then replaced the inner program with its KKT optimality conditions and solved the resulting finite nonlinear program with complementarity constraints [44]. By doing so, the resulting program is a restriction of the original explicit SIP and therefore, upon solution, generates SIP-feasible points [44]. This idea was concurrently discussed by Mitsos et al. [90], where they also considered a technique closely related to the inclusion-constrained reformulation [13, 14] but instead used interval analysis to further construct McCormick-based concave relaxations [85] of g on P to restrict the inner program and generate SIP-feasible points finitely.

More recently, Mitsos [87] developed an algorithm based on the ideas of Blankenship and Falk [18] that relies on a new relaxation technique for the upper-bounding procedure that requires the right-hand side of the semi-infinite constraint to be perturbed from zero. This formulation results in solving at least three NLP subproblems to global optimality, in the general case, and the computational results reported are quite promising [87]. The key contribution is the novel upper-bounding procedure that is guaranteed to generate SIP-feasible points after finitely many iterations. It is stated explicitly that the algorithm only requires continuity of f and g and the exis-

tence of a Slater point arbitrarily close to a SIP minimizer, "provided the functions can be handled by the NLP solver" [87]. Therefore, this algorithm *could* be applied to solve (7.1) while handling the equality constraints directly, without requiring the introduction of the implicit function, by reformulating each nonconvex subproblem as an equality-constrained global optimization problem. However, this strategy is not advisable since the algorithm would then force the number of variables in the upper- and lower-bounding subproblems to increase with each iteration.[1] Thus, these subproblems become increasingly more expensive to solve with each iteration. However, this algorithm is a promising candidate for the global solution of SIPs with implicit functions embedded.

With the exception of [134], all of the aforementioned methods were developed to solve explicit SIPs (or explicit min-max programs). The major complication with formulating the bilevel program in (7.1) as the SIP in (7.5), is that an implicit function $\mathbf{x}$, which may not have a known closed algebraic form, becomes embedded within the semi-infinite constraint g. Therefore, $\mathbf{x}$ (and g) may not be evaluated directly, but must be approximated using a numerical method, such as Newton's method or some other fixed-point iteration. In order to modify previously developed methods that rely on relaxations of the inner program, it must be possible to construct relaxations of $g(\mathbf{x}(\cdot\,,\mathbf{p}),\cdot\,,\mathbf{p})$, $\forall\mathbf{p}\in P$, on Y. However, in order to relax $g(\mathbf{x}(\cdot\,,\mathbf{p}),\cdot\,,\mathbf{p})$, $\forall\mathbf{p}\in P$, on Y, convex and concave relaxations of the implicit function $\mathbf{x}(\cdot\,,\mathbf{p})$ on Y, must be calculable. As previously mentioned, this has been achieved for problems in which the implicit function $\mathbf{x}$ could be approximated using the successive-substitution fixed-point iteration [134]. The theoretical details of these relaxations were presented in [128]. This chapter is an improvement on the previous results discussed in [134] and consider solving SIPs with more general implicit functions embedded that can be evaluated using *any* available method, such as Newton's method, instead of being restricted to the successive-substitution case. This work will make use of a modified version of the algorithm developed by Mitsos [87], where the solution of each of the (implicit) subproblems will be performed using the novel relaxation techniques and

[1] This result is illustrated in Appendix B.

the global optimization algorithm developed in Chapter 4.

Just to recap, in Chapter 4 theoretical developments were made to construct convex and concave relaxations of more general implicit functions. The construction of these relaxations are analogous in many ways to how interval bounds were calculated for implicit functions in Chapter 3 using parametric interval-Newton methods. By applying parametric interval-Newton methods to a function $\mathbf{h}$, under certain conditions discussed in Chapter 3, an interval can be calculated that bounds a unique root, $\mathbf{x}$, of $\mathbf{h}$ over the set $Y \times P$. Taking these bounds as *initial* relaxations of $\mathbf{x}$, they can be iteratively refined using the methods in Chapter 4 to produce convex and concave relaxations of $\mathbf{x}$ on $Y \times P$. As a result, global optimization of implicit functions was developed.

In the next section, the global optimization algorithm for SIPs with embedded implicit functions is discussed. The application to min-max and max-min problems is made explicit, immediately following the statement of the algorithm. Finally, three numerical examples are given that illustrate the solution of implicit SIPs to global optimality.

7.2 Global Solution of SIPs with Implicit Functions Embedded

The global optimization algorithm for solving implicit SIPs is based entirely on the cutting-plane algorithm presented by Mitsos [87] which itself is based on the algorithm developed by Blankenship and Falk [18] but with a novel upper-bounding procedure. The algorithm, as applied to explicit SIPs, is guaranteed to produce SIP-feasible points after finitely many iterations under the assumption that there exists a Slater point arbitrarily close to a minimizer [87]. As previously mentioned, the algorithm relies on the ability to solve three nonconvex NLP subproblems to global optimality. The three subproblems are discussed below.

7.2.1 Lower-Bounding Problem

The lower-bounding procedure comes from a simple relaxation technique based on an adaptive discretization procedure originally described in [18]. In this case, the SIP is reduced to an implicit NLP by considering only a finite number of constraints corresponding to realizations of $\mathbf{p} \in P^{LBD}$ with $P^{LBD} \subset P$, as a finite set. The lower-bounding problem is formulated as

$$\begin{aligned} f^{LBD} = \quad & \min_{\mathbf{y}} f(\mathbf{y}) \\ & \text{s.t. } g(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}) \leq 0, \ \ \forall \mathbf{p} \in P^{LBD} \\ & \mathbf{y} \in Y = \{\mathbf{y} \in \mathbb{R}^{n_y} : \mathbf{y}^L \leq \mathbf{y} \leq \mathbf{y}^U\}. \end{aligned} \tag{7.6}$$

In order to guarantee $f^{LBD} \leq f^*$, the lower-bounding problem must be solved to global optimality.

7.2.2 Inner Program

The inner program, stated explicitly in (7.3) and (7.4), which is equivalent to the semi-infinite constraint, defines the SIP feasible region. Thus, given a candidate $\bar{\mathbf{y}} \in Y$, feasibility can be determined by solving the inner (in general nonconvex) program:

$$\bar{g}(\bar{\mathbf{y}}) = \max_{\mathbf{p} \in P} g(\mathbf{x}(\bar{\mathbf{y}},\mathbf{p}),\bar{\mathbf{y}},\mathbf{p}). \tag{7.7}$$

The point $\bar{\mathbf{y}}$ is feasible in the SIP (7.5) if $\bar{g}(\bar{\mathbf{y}}) \leq 0$. Therefore, in order to determine feasibility of a candidate $\bar{\mathbf{y}}$, the inner program (7.7) must be solved to global optimality for the general case.

7.2.3 Upper-Bounding Problem

The upper-bounding problem comes from bounding the semi-infinite constraint away from zero by introducing a parameter $\epsilon^{g,k}$, referred to as the *restriction parameter* [87], and reducing the SIP to an implicit NLP by only considering a finite number of

constraints corresponding to realizations of $\mathbf{p} \in P^{UBD}$, where $P^{UBD} \subset P$ is a finite set. The upper-bounding problem is formulated as

$$\begin{aligned} f^{UBD} = \quad & \min_{\mathbf{y}} f(\mathbf{y}) \\ & \text{s.t. } g(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}) \leq -\epsilon^{g,k}, \ \ \forall \mathbf{p} \in P^{UBD} \\ & \mathbf{y} \in Y = \{\mathbf{y} \in \mathbb{R}^{n_y} : \mathbf{y}^L \leq \mathbf{y} \leq \mathbf{y}^U\}. \end{aligned} \tag{7.8}$$

As mentioned in [87], the upper-bounding problem (7.8) must be solved to global optimality in order for the algorithm to solve the original SIP (7.5) to global optimality. However, any valid upper bound, $f^{UBD} \geq f^*$, can be obtained by solving (7.8) locally for $\bar{\mathbf{y}}$ and verifying that it is feasible in the original SIP (7.5).

7.2.4 Algorithm

In this section, the algorithm used for solving globally SIPs with implicit functions embedded to guaranteed ϵ-optimality is given. Again, as presented, this algorithm is an adaptation of the algorithm given by Mitsos in [87] to SIPs with implicit functions embedded. Finite convergence of the algorithm for explicit SIPs was previously proven [87]. The results proven by Mitsos in [87] extend directly to the implicit SIP algorithm provided finite convergence of each implicit NLP subproblem can be guaranteed. The latter result was proven in Chapter 4. The assumptions on which the algorithm relies are stated explicitly in the following.

Assumption 7.2.1.

1. The functions $f : D_y \to \mathbb{R}$, $g : D_x \times D_y \times D_p \to \mathbb{R}$, and $\mathbf{h} : D_x \times D_y \times D_p \to \mathbb{R}^{n_x}$ are factorable and continuous on their domains.

2. Derivative information $\nabla_{\mathbf{y}} h_i, \ i = 1, \ldots, n_y$ is available and is factorable, say by automatic differentiation [12, 49].

3. There exists $\mathbf{x} : Y \times P \to D_x$ such that $\mathbf{h}(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}) = \mathbf{0}, \ \ \forall (\mathbf{y},\mathbf{p}) \in Y \times P$, and an interval $X \subset \mathbb{I}D_x$ is available such that $\mathbf{x}(Y,P) \subset X$ and $\mathbf{x}(\mathbf{y},\mathbf{p})$ is

unique for every $(\mathbf{y}, \mathbf{p}) \in Y \times P$.

4. A matrix $\mathbf{\Psi} \in \mathbb{R}^{n_y \times n_y}$ is known such that $A \equiv \mathbf{\Psi} J_{\mathbf{y}}(X, Y, P)$ satisfies $0 \notin A_{ii}$ for all i, where $J_{\mathbf{y}}$ is an inclusion monotonic interval extension of the Jacobian matrix of $\mathbf{h}$.

5. There exists a point $\mathbf{y}^S \in Y$ with $g(\mathbf{x}(\mathbf{y}^S, \mathbf{p}), \mathbf{y}^S, \mathbf{p}) < 0, \ \forall \mathbf{p} \in P$ such that $f(\mathbf{y}^S) - f^* < \epsilon_{\text{tol}}$.

Assumptions 7.2.1(1)-(4) are essentially required for constructing convex and concave relaxations for global optimization of implicit functions. Assumption 7.2.1(3) can be satisfied by applying parametric interval-Newton methods and their theoretical results discussed in Chapter 3. The matrix $\mathbf{\Psi}$ is a preconditioning matrix and has been the focus of many research articles. The application to interval-Newton methods is discussed in [69], among others. The interval-valued matrix A can be calculated efficiently by taking natural interval extensions (see Chap. 3) and thus satisfy Assumption 7.2.1(4). Assumption 7.2.1(5) is the ϵ_{tol}-optimal SIP-Slater point condition. Altogether, satisfying Assumption 7.2.1 guarantees that the following algorithm terminates in finitely many iterations with a certificate of optimality and a rigorous ϵ_{tol}-optimal feasible point (see Chap. 4 and [87]). The algorithm for semi-infinite optimization with implicit functions embedded is presented in the following.

Algorithm 7.1 (Global Solution of Implicit SIPs)**.**

1. **(Initialization)**

   (a) Set $LBD = -\infty$, $UBD = +\infty$, $\epsilon_{\text{tol}} > 0$, $k := 0$.

   (b) Set initial parameter sets $P^{LBD} = P^{LBD,0}$, $P^{UBD} = P^{UBD,0}$.

   (c) Set initial restriction parameter $\epsilon^{g,0} > 0$ *and* $r > 1$.

2. **(Termination)** *Check* $UBD - LBD \leq \epsilon_{\text{tol}}$.

   (a) If true, terminate.

   (b) Else $k := k + 1$, *continue.*

3. **(Lower-Bounding Problem)** *Solve the lower-bounding problem (7.6) to global optimality.*

   (a) *Set* $LBD := f^{LBD}$, *set* $\bar{\mathbf{y}}$ *equal to the estimate of an optimal solution found, continue.*

4. **(Inner Program)** *Solve the inner program (7.7) to global optimality.*

   (a) *If* $g(\mathbf{x}(\bar{\mathbf{y}}, \bar{\mathbf{p}}), \bar{\mathbf{y}}, \bar{\mathbf{p}}) = \bar{g}(\bar{\mathbf{y}}) \leq 0$, *set* $\mathbf{y}^* := \bar{\mathbf{y}}$, $UBD := f(\bar{\mathbf{y}})$, *terminate algorithm.*

   (b) *Else, add* $\bar{\mathbf{p}}$ *to* P^{LBD}, *continue.*

5. **(Upper-Bounding Problem)** *Solve the upper-bounding problem (7.8) to global optimality.*

   (a) *If feasible:*

      i. *Set* $\bar{\mathbf{y}}$ *equal to the optimal solution found and solve the lower-level program (7.7) to global optimality.*

      ii. *If* $\bar{g}(\bar{\mathbf{y}}) < 0$*:*

         A. *If* $f(\bar{\mathbf{y}}) \leq UBD$, *set* $UBD := f(\bar{\mathbf{y}})$, $\mathbf{y}^* := \bar{\mathbf{y}}$, *continue.*

         B. *Set* $\epsilon^{g,k+1} := \epsilon^{g,k}/r$, *go to 2.*

      iii. *Else* ($\bar{g}(\bar{\mathbf{y}}) \geq 0$), *add* $\bar{\mathbf{p}}$ *to* P^{UBD}, *go to 2.*

   (b) *Else (infeasible), set* $\epsilon^{g,k+1} := \epsilon^{g,k}/r$, *go to 2.*

It should be noted that the subproblems can only be solved finitely to within some chosen tolerances. In order to guarantee that the SIP algorithm is rigorous, the convergence tolerances for the subproblems must be set such that they are smaller than ϵ_{tol}.

7.3 Application to Max-Min and Min-Max Problems

Constrained min-max problems:

$$\min_{\mathbf{y}\in Y} \max_{\mathbf{p}\in P,\mathbf{z}\in X} \{G(\mathbf{z},\mathbf{y},\mathbf{p}) : \mathbf{h}(\mathbf{z},\mathbf{y},\mathbf{p}) = \mathbf{0}\} \tag{7.9}$$

and constrained max-min problems:

$$\max_{\mathbf{y}\in Y} \min_{\mathbf{p}\in P,\mathbf{z}\in X} \{G(\mathbf{z},\mathbf{y},\mathbf{p}) : \mathbf{h}(\mathbf{z},\mathbf{y},\mathbf{p}) = \mathbf{0}\}, \tag{7.10}$$

with $G : D_x \times D_y \times D_p \to \mathbb{R}$, can also be solved using Algorithm 7.1. The min-max case results in solving the implicit program:

$$G^* = \min_{\mathbf{y}\in Y} \max_{\mathbf{p}\in P} G(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}) \tag{7.11}$$

which can be formulated as an implicit SIP using the same technique in Chapter 2, by introducing a variable $\eta \in H \subset \mathbb{R}$ and writing:

$$\begin{aligned} \eta^* = \min_{\mathbf{y}\in Y,\eta\in H} \eta \\ \text{s.t.} \quad \max_{\mathbf{p}\in P} G(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}) \le \eta. \end{aligned} \tag{7.12}$$

Using the relationship (7.4) and setting $g(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p},\eta) = G(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}) - \eta$, the following SIP can be written:

$$\begin{aligned} \eta^* = \min_{\mathbf{y}\in Y,\eta\in H} \eta \\ \text{s.t.} \quad g(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p},\eta) \le 0, \ \forall \mathbf{p} \in P, \end{aligned} \tag{7.13}$$

which is equivalent to the implicit SIP in (7.5). The implicit SIP algorithm can be applied directly to this problem without any modification by setting $n_y := n_y + 1$ and treating η as the $n_y + 1$ component of $\mathbf{y}$. As mentioned in Chapter 2, an optimal

solution value of $\eta^* \leq 0$ implies $G^* \leq 0$, and alternatively, $\eta^* > 0$ implies $G^* > 0$.

The constrained max-min problem reformulation is slightly different. This case amounts to solving

$$G^* = \max_{\mathbf{y}\in Y} \min_{\mathbf{p}\in P} G(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}). \tag{7.14}$$

Again the variable $\eta \in H \subset \mathbb{R}$ is introduced and (7.14) is written as

$$\eta^* = \max_{\mathbf{y}\in Y, \eta\in H} \eta \tag{7.15}$$

$$\text{s.t. } \eta \leq G(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p}),\ \forall \mathbf{p} \in P \tag{7.16}$$

(which is equivalent to the robust SIP formulation (2.5)) by using the relationship (7.4) and equivalently as

$$-\eta^* = \min_{\mathbf{y}\in Y, \eta\in H} -\eta \tag{7.17}$$

$$\text{s.t. } g(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p},\eta) \leq 0,\ \forall \mathbf{p} \in P \tag{7.18}$$

by using the identity $g(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p},\eta) = \eta - G(\mathbf{x}(\mathbf{y},\mathbf{p}),\mathbf{y},\mathbf{p})$. Now, the implicit SIP algorithm can be applied without any modification by again setting $n_y := n_y + 1$ and treating η as the $n_y + 1$ component of $\mathbf{y}$. Now, analogous to the min-max case, and optimal solution value of $\eta^* \leq 0$ implies that $G^* \leq 0$ and $\eta^* > 0$ implies $G^* > 0$.

7.4 Examples

Example 7.4.1. Consider the following illustrative example with $n_x = n_p = n_y = 1$:

$$\begin{aligned} f(y) =& (y-3.5)^4 - 5(y-3.5)^3 - (y-3.5)^2 + 30(y-3.5) \\ h(z,y,p) =& z - (y - y^3/6 + y^5/120)/\sqrt{z} - p = 0 \\ g(z,y,p) =& z + \cos(y - p/90) - p \leq 0,\ \ \forall p \in P \\ & y \in Y = [0.5, 8.0] \\ & p \in P = [80, 120]. \end{aligned}$$

The objective function and implicit semi-infinite constraint are shown in Figure 7-1. An interval $X = [68.8, 149.9]$, guaranteed to contain a unique implicit function $x : Y \times P \to X$ was obtained using the parametric interval-Newton method discussed in Chapter 3.

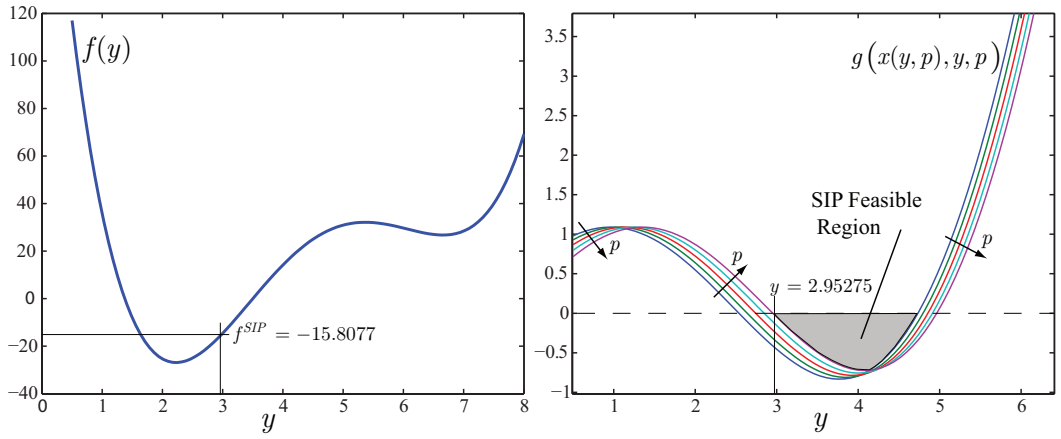

Figure 7-1: The objective function and implicit semi-infinite constraint for Example 7.4.1.

Example 7.4.2. Consider the robust design of an isothermal flash separator under uncertainty. We wish to verify robust operation of a proposed design in the face of the worst-case realization of uncertainty. The flash separator is designed to separate a ternary mixture of n-butane, n-pentane, and n-hexane, with molar fractions of 0.5, 0.4, and 0.1, respectively. The separator is designed to create a vapor product stream with no more than 0.05 mole-fraction of n-hexane. To do so, it is designed to operate at 85°C and a pressure no greater than 5100torr (6.80bar). It is expected that during operation, the vessel temperature, or simply the thermocouple reading, may vary by as much as ±5°C. For this system there are six unknowns: the compositions of the vapor and liquid streams. Three species balance equations and three phase-behavior equations can be written, resulting in a dimensionality $n_x = 6$. However, an alternative, and equivalent, model formulation with $n_x = 1$ can be formulated by writing the stream composition model equations in terms of the cut fraction $\hat{\alpha}$:

$$h(\hat{\alpha}, \tau, p) = \sum_i \frac{\gamma_i(K_i(\tau, p) - 1)}{(K_i(\tau, p) - 1)\hat{\alpha} + 1} = 0,$$

where τ will be the temperature (uncertain) variable, the cut fraction, $\hat{\alpha}$, is defined as the fraction of the feed that leaves in the vapor stream (internal state variable), p is the vessel pressure which can be controlled in order to mitigate fluctuations in τ, K_i is the vapor-liquid equilibrium coefficient for the i^{th} chemical species, and γ_i is the mole-fraction of chemical species i in the feed. Solving $h(\hat{\alpha}, \tau, p) = 0$ for $\hat{\alpha}$ defines the cut fraction as an implicit function of temperature and pressure, $\alpha : T \times P \to X$. Any value $\alpha \notin [0, 1]$ is nonphysical so the interval $X = [0, 1]$ was considered. For this system, the vapor-liquid equilibrium coefficient can be calculated as

$$K_i(\tau, p) = \frac{p_i^{sat}(\tau)}{p}$$

for each chemical component i with

$$\log_{10} p_i^{sat}(\tau) = A_i - \frac{B_i}{C_i + \tau},$$

with τ in °C and p_i^{sat} in torr. The Antoine coefficients A_i, B_i, C_i are available in Table 7.1. For robust design problems, one must consider the worst-case realization of

Ex. 2 Antoine Coefficients				
i	A_i	B_i	C_i	Temp. Range
1: n-butane	7.00961	1022.48	248.145	$-138.29 - 152.03$°C
2: n-pentane	7.00877	1134.15	238.678	$-129.73 - 196.5$°C
3: n-hexane	6.9895	1216.92	227.451	$-95.31 - 234.28$°C

Table 7.1: Antoine coefficients for the ternary mixture in Example 7.4.2 [143].

uncertainty and examine if there exists a control setting that allows the design to still meet the performance and/or safety specification. This problem can be formulated mathematically as a max-min problem:

$$\max_{\tau \in T} \min_{p \in P} G(\alpha(\tau, p), \tau, p)$$
$$T = [80, 90]$$
$$P = [4400, 5100],$$

which, if $G(\alpha(\tau^*, p^*), \tau^*, p^*) \leq 0$, the design is robustly feasible, or simply, for the worst-case realization of uncertainty, there exists a control setting such that the system meets specification. The lower bound on the control variable comes from a

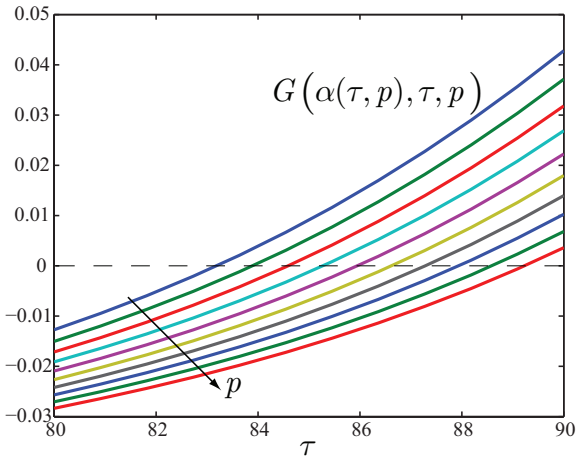

Figure 7-2: The design constraint function for Example 7.4.2.

requirement that there are two phases present in the separator at all times (i.e., any lower pressure will flash all of the liquid into the vapor phase). According to the previous discussion, this problem can be reformulated as an implicit SIP:

$$\begin{aligned} &\min_{\tau\in T,\eta\in H} \; -\eta \\ &\text{s.t.} \;\; \eta - G(\alpha(\tau,p),\tau,p) \leq 0, \;\; \forall p \in P. \end{aligned}$$

The performance specification can be written as

$$G(\alpha(\tau,p),\tau,p) = \frac{\gamma_3 K_3(\tau,p)}{(K_3(\tau,p)-1)\alpha(\tau,p)+1} - 0.05 \leq 0,$$

which comes from material balances on the system. Figure 7-2 shows G plotted against τ.

Example 7.4.3. Consider the engineering problem of optimal design of a continuous-stirred tank reactor (CSTR) for the chlorination of benzene, shown in Figure 7-3. The

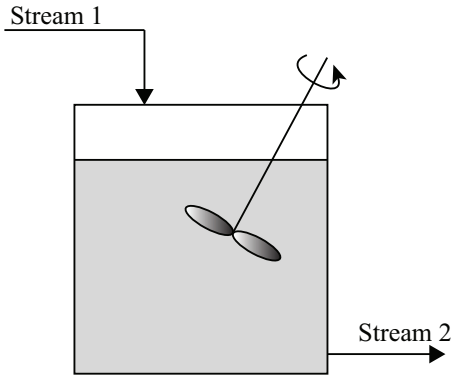

Figure 7-3: The continuous-stirred tank reactor for Example 7.4.3.

reactions taking place are

$$C_6H_6 + Cl_2 \xrightarrow{k_1} C_6H_5Cl + HCl$$
$$C_6H_5Cl + Cl_2 \xrightarrow{k_2} C_6H_4Cl_2 + HCl$$

where the rate constants k_1 and k_2 (hr^{-1}), as well as the feed flowrate F_1 (kmol/h), will be considered as uncertainty parameters, $\mathbf{p} = (k_1 \;\; k_2 \;\; F_1)^{\mathrm{T}}$. The design variable will be the reactor volume (m^3), $y = v$. The reaction kinetics can be considered to be first-order with respect to benzene and chlorobenzene and the reactions are irreversible [73]. For simplicity, A will denote C_6H_6, B will denote C_6H_5Cl, and C will denote $C_6H_4Cl_2$. Therefore, there are a total of four unknowns: the composition of the product stream and the product stream flowrate in terms of A, B, and C, $\mathbf{z} = (z_A \;\; z_B \;\; z_C \;\; F_2)^{\mathrm{T}}$. In this formulation, $n_x = 1$, $n_y = 4$, and $n_p = 3$. Note that F_1 and F_2 are the flowrates (kmol/h) in terms of the chemical components A, B, and C only. The model equations are then:

$$\mathbf{h}(\mathbf{z}, \mathbf{y}, \mathbf{p}) = \begin{pmatrix} z_{A,1}p_3 - z_1 z_4 - y r_1 \\ z_{B,1}p_3 - z_2 z_4 + y(r_1 - r_2) \\ z_{C,1}p_3 - z_3 z_4 + y r_2 \\ 1 - z_1 - z_2 - z_3 \end{pmatrix} = \mathbf{0} \tag{7.19}$$

with $z_{i,1}$ as the mole-fraction of chemical component i in the feed stream, and the

reaction rates r_1 and r_2 are given by:

$$r_1 = p_1 z_1/(z_1 V_A + z_2 V_B + z_3 V_C),$$
$$r_2 = p_2 z_2/(z_1 V_A + z_2 V_B + z_3 V_C).$$

with V_i as the molar volumes of chemical component i: $V_A = 8.937 \times 10^{-2}\text{m}^3/\text{kmol}$, $V_B = 1.018 \times 10^{-1}\text{m}^3/\text{kmol}$, $V_C = 1.13 \times 10^{-1}\text{m}^3/\text{kmol}$. The feed was taken to be pure benzene.

For this particular system, the design objective is to minimize the reactor volume while satisfying the performance constraint that at least 22kmol C_6H_5Cl/h is produced:

$$\min_{y \in Y} y$$
$$\text{s.t. } 22 - x_2(y, \mathbf{p}) x_4(y, \mathbf{p}) \leq 0, \ \forall \mathbf{p} \in P.$$

The uncertainty interval will be $P = [0.38, 0.42] \times [0.053, 0.058] \times [60, 70]$, the design interval will be $Y = [10, 20]$. From the parametric interval-Newton method, an interval $X = [0.15, 0.85] \times [0.3, 0.65] \times [0.0, 0.12] \times [60, 70]$ was calculated that encloses an implicit function $\mathbf{x} : Y \times P \to X$ such that $\mathbf{h}(\mathbf{x}(y, \mathbf{p}), y, \mathbf{p}) = \mathbf{0}, \ \forall (y, \mathbf{p}) \in Y \times P$.

7.5 Experimental Conditions and Results

Algorithm 7.1 was implemented in `C++`. Each NLP subproblem was solved using the algorithm for global optimization of implicit functions developed in Chapter 4, which was also implemented in `C++` and utilizes the library `MC++`[26]. The algorithm for global optimization of implicit functions relies on the ability to solve convex nonsmooth subproblems. For this task, the nonsmooth bundle solvers PBUN and PBUNL [83] were utilized with default settings for the NLP lower-bounding problems and the objective function was evaluated at NLP feasible points to obtain valid upper bounds on the NLP. Since the constrained bundle solver (PBUNL) can only handle affine con-

straints, affine relaxations of the convex constraints with respect to reference points must be calculated. Two sets of experiments were conducted:

Case 1: A single reference point—taken as the midpoint of Y—was used to construct affine relaxations of constraints.

Case 2: Three reference points—the lower bound, the midpoint, and the upper bound of Y—were used to construct affine relaxations of the constraints and used simultaneously.

The numerical experiments were performed using a PC with an Intel Core2 Quad 2.66GHz CPU operating Linux. For each example, absolute and relative convergence tolerances of 10^{-7} and 10^{-5}, respectively, were used for the NLP subproblems.

7.5.1 Example 7.4.1

For the SIP algorithm, each constraint set was initialized as empty, $\epsilon^{g,0} = 0.9$, $r = 2.0$, and $\epsilon_{\text{tol}} = 10^{-3}$. For each set of experiments, the implicit SIP algorithm was applied and the global optimal solution with an objective function value of $f^* = -15.8077$ at $y^* = 2.95275$. Convergence was observed in 2 iterations. For this example, the algorithm terminates after the lower-bounding problem furnishes a SIP-feasible point (Step 4a of the algorithm). Therefore, for this example, the parameter r does not affect the performance of the implicit SIP algorithm. Interestingly, Case 1 converged after 0.097 seconds while Case 2 converged after 0.085 seconds. This may indicate that the added computational cost of Case 2 is outweighed by the benefit of having more precise approximations of the constraints.

7.5.2 Example 7.4.2

For the implicit SIP algorithm, each constraint set was initialized as empty, $\epsilon^{g,0} = 0.9$, and $\epsilon_{\text{tol}} = 10^{-4}$. For both Case 1 and Case 2, the implicit SIP algorithm was applied and the global optimal solution was obtained with $\eta^* = 3.56 \times 10^{-3}$, $\tau^* = 90°\text{C}$, $p^* = 5100\text{torr}$. For Case 2, the algorithm terminates in 3 iterations taking 0.25

seconds after the lower-bounding problem furnishes an SIP-feasible point. Thus, as previously mentioned, the parameter r has no effect on the performance of the algorithm.

Case 1 was more interesting in that the algorithm doesn't terminate with the lower-bounding problem furnishing an SIP-feasible point but it terminates at Step 2 of the algorithm. For a modest value of $r = 32$, the algorithm converges in only 6 iterations, taking 0.557sec. The performance of the algorithm for Case 1 can be found in Figure 7-4. Similar to the results discussed in [87], a relatively small value for r

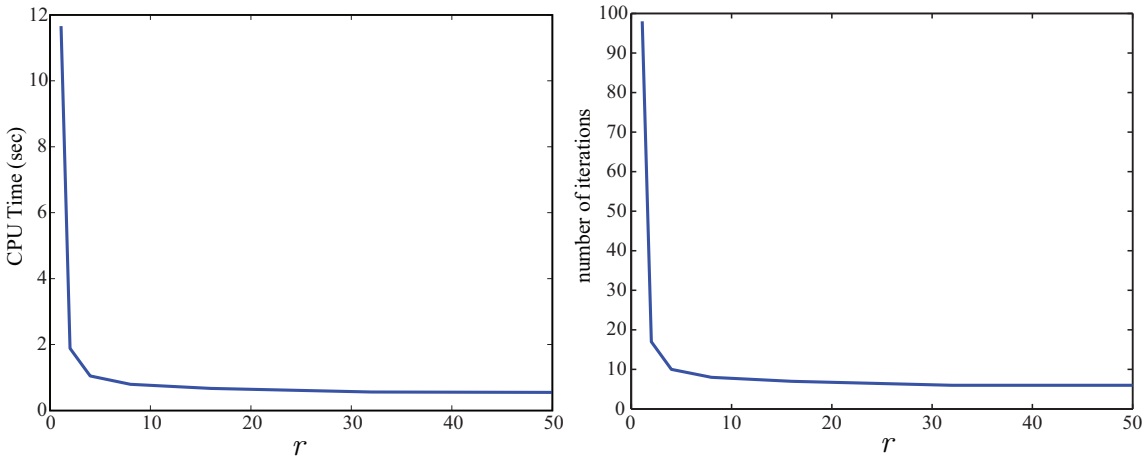

Figure 7-4: The computational effort in terms of the number of iterations the algorithm takes to solve Case 1 of Example 7.4.2 versus the reduction parameter r.

resulted in the implicit SIP algorithm taking many iterations to converge. As r was increased, the number of iterations taken to converge, as well as the total solution time, plateaued. For this example, by using a relatively large reduction parameter value ($r = 32$), the total number of iterations of the implicit SIP algorithm could be reduced by an order of magnitude and the solution time by almost two orders, over using $r = 1.1$.

Returning to the idea of robust design, since $\eta^* > 0$, the flash separator design is not robust. However, as can be seen from Figure 7-2, if the design can be improved such that the temperature (or thermocouple reading) may only vary by $\pm 4°C$, the design appears to be robust. This result was verified by the implicit SIP algorithm converging after 6 iterations to the optimal solution with $\eta^* = -1.04 \times 10^{-3}$.

7.5.3 Example 7.4.3

For the SIP algorithm, each constraint set was initialized as empty, $\epsilon^{g,0} = 0.9$, and $\epsilon_{\text{tol}} = 10^{-4}$.

For Case 2, the implicit SIP algorithm was applied and the global optimal solution was obtained with $f^* = y^* = 10.1794\text{m}^3$, $\mathbf{p}^* = (0.38 \;\; 0.058 \;\; 60)^{\text{T}}$. Therefore, in order to produce at least 22kmol/h of chlorobenzene, taking into account uncertainty in the input flowrate and the reaction rate constants, the reactor volume must be 10.1794m^3. Note that the worst-case realization of uncertainty is exactly what is to be expected; in order to have the least amount of chlorobenzene in the product stream, k_1 should be the smallest value it can take, k_2 should be the largest it can take, and the least amount of benzene should be fed to the reactor. For a value of $r = 18$, the algorithm converges in 7 iterations and 11.71 seconds. The performance of the algorithm for Case 2 can be found in Figure 7-5.

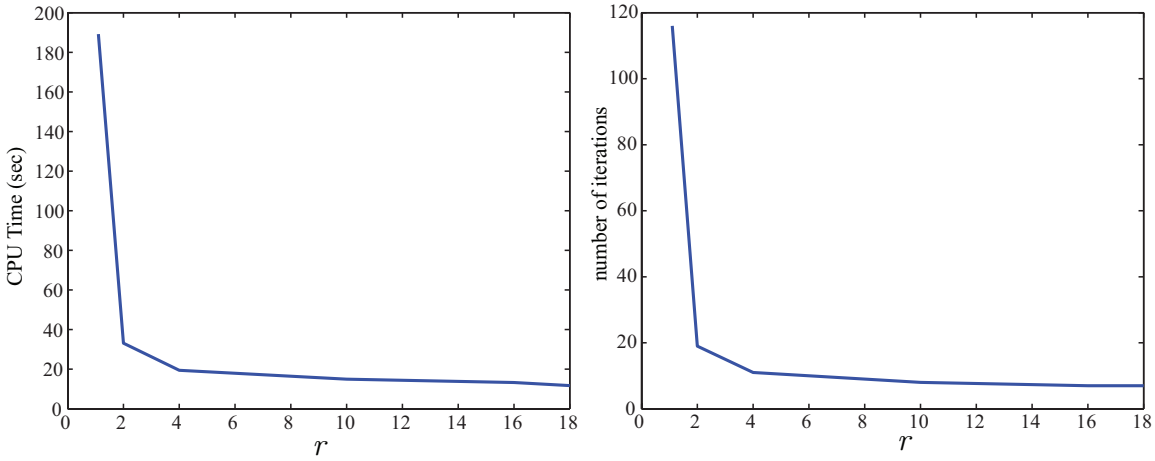

Figure 7-5: The computational effort in terms of the number of iterations the algorithm takes to solve Case 2 of Example 7.4.3 versus the reduction parameter r.

Similar to Case 1 of Example 7.4.2, a small value for r resulted in the implicit SIP algorithm taking many iterations to converge. As r was increased, the number of iterations required to converge, as well as the total solution time dropped drastically and plateaued. A parameter value of $r = 18$ reduced the solution time by two orders of magnitude over $r = 1.1$. For each example, the implicit SIP algorithm performed

very favorably converging after only a few iterations of the algorithm.

For this example, Case 1 failed to converge within 200 iterations of the algorithm. This result is simply a consequence of using PBUNL which only accepts affine constraints. In this case, since the affine constraints are being constructed with reference to the midpoint of X, the solver apparently fails to ever return a point that is feasible in the original SIP.

7.6 Concluding Remarks

In this chapter, a class of bilevel programs that commonly arise in engineering design problems was reformulated as a semi-infinite program with implicit functions embedded. An algorithm for solving SIPs with implicit functions embedded was presented which is an adaptation of a recently developed algorithm for solving standard SIPs.

As a proof-of-concept, three numerical examples were presented that illustrate the global solution of implicit SIPs using this algorithm. The first example illustrated the solution of a simple numerical system that fits the implicit SIP form given in (7.5). The second example was an engineering problem of robust design under uncertainty, originally cast as a constrained max-min problem as in (2.3). It was then reformulated as an implicit SIP of the form in (7.5) and solved using the implicit SIP algorithm. The third example was an engineering problem of optimal design of a chemical reactor considering uncertainty in the kinetic parameters and formulated as an SIP.

A method was presented for reformulating equality-constrained bilevel programs as SIPs with embedded implicit functions, requiring that:

1. all functions involved are continuous,
2. all functions involved are factorable,
3. derivative information for the equality constraint functions is available and is factorable,
4. there exists at least one solution $\mathbf{x}$ to the system of equations in (7.2) for every $(\mathbf{y}, \mathbf{p}) \in Y \times P$, and

5. there exists a Slater point arbitrarily close to a SIP minimizer.

To solve the resulting implicit SIP, the global optimization algorithm developed by Mitsos [87] has been adapted. The algorithm relies on the ability to solve three non-convex implicit NLP subproblems to global optimality. This is performed utilizing the relaxation methods and the deterministic algorithm for global optimization of implicit functions which were developed in Chapter 4. Algorithm 4.1 relies on the ability to solve nonsmooth lower- and/or upper-bounding problems at each iteration. This can be done using any available nonsmooth optimization algorithm or using the calculated subgradient information to construct affine relaxations and transform the problem into a linear program and solved using any efficient LP optimization algorithm. For this chapter, the nonsmooth bundle solvers PBUN and PBUNL [83], were utilized. Note that the requirements (2) and (3) are only due to current limitations of the algorithm for global optimization of implicit functions. The requirements (4) and (5) imply that the SIP is feasible and (1) and (5) are required for guaranteed ϵ-optimal convergence of the original explicit SIP algorithm [87] after finitely many iterations. Altogether, these requirements guarantee ϵ-optimal convergence of Algorithm 7.1.

Due to the limitations of the PBUNL solver, only affine constraints could be used. Since the implicit semi-infinite constraint is almost surely nonlinear, affine relaxations must be constructed. For the numerical examples, two sets of experiments were conducted: one using a single reference point for constructing affine relaxations of the constraints and another using three reference points for constructing affine relaxations of the constraints and using them all simultaneously. The first method was hypothesized to be advantageous since it required less computational effort to calculate the constraints. Alternatively, the second method was hypothesized to be advantageous since using multiple reference points results in better approximations of the constraints, which in turn may speed up convergence of the overall algorithm. For each experiment, it was observed that Case 2 was superior to Case 1 in terms of total CPU time. For Example 7.4.3, Case 1 even failed to converge after 200 iterations. This was likely due to the affine relaxations of the semi-infinite constraint not being very tight, resulting in PBUNL failing to find a solution that is feasible in the original

SIP.

In the next chapter, worst-case design of subsea production facilities is addressed. A slightly modified version of Algorithm 7.1 is applied and the feasibility problem is solved for various cases.

Chapter 8

Robust Simulation and Design of Subsea Production Facilities

In this chapter, the problem of designing subsea production facilities for the worst case is revisited. In particular, a model of a subsea separator process is presented that can act as a framework for modeling systems involving more complex unit operations models in future case studies. A complete implementation of the algorithm for robust simulation and design is presented which utilizes the developments of the previous chapters and elsewhere. Finally, the robust simulation SIP (2.5) is solved for this model using the robust simulation algorithm implementation.

8.1 Background

In solving SIPs with implicit functions embedded, meaningful global bounding information for the semi-infinite constraint function g is required. Calculating this information is often a limiting step in the overall performance of the algorithm because all that is known about the state variables initially are their natural bounds, which in turn lead to a prohibitively large initial bound on g from which no meaningful information can be deduced. Since interval-Newton-type methods, discussed in Chapter 3, often prove to be ineffective in refining sufficiently large initial intervals, this poses a serious problem for the algorithm. Although interval-Newton methods

are quite effective on smaller intervals, a method that can obtain meaningful bounds on the function g starting with large initial intervals on the state variables efficiently is necessary for the overall success and performance of the algorithm.

Interval analysis has been widely applied to many simulation and optimization applications in chemical engineering, e.g., [6, 80]. In [80] strategies for bounding the solution of interval linear systems were presented, which were solved in the context of the interval-Newton method. The authors reviewed several preconditioning techniques for the above mentioned method and proposed a new bounding approach based on the use of linear programming (LP) strategies. They demonstrated the performance of the proposed technique on global optimization problems such as parameter estimation and molecular modeling. In [6], interval-based global optimization of modular process models is addressed. In their work, the authors explored the use of five different interval contraction methods to improve the performance of the interval optimization algorithm of [24]. The contraction methods used were: consistency techniques, constraint propagation, LP contractors, interval Gaussian elimination, and the interval-Newton contractor. Using a set of mathematical problems and chemical engineering flowsheet design problems such as the Haverly pooling problem, reactor flowsheet problem, and a reactor network problem, they compared the impact of various contraction methods on the overall performance of the interval optimization algorithm. Their computational experiments showed that the LP contractors performed the best while the constraint propagation and interval Gaussian elimination methods were ineffective.

In the context of interval contraction, there exist several methods developed by researchers outside of the process engineering community. For a detailed review of such methods, see [66]. In [125] the fundamentals of interval analysis on directed acyclic graphs (DAGs) for global optimization and constraint propagation were presented. The proposed framework overcomes the limitation of propagating each constraint individually by taking into account the effects of any common subexpressions that are shared by many constraints. Later, the above framework was extended to perform adaptively forward evaluations and backward projections on only some select nodes

of a DAG [141]. The computational study showed that the adaptive framework performs at least one to two orders of magnitude faster than the other state-of-the-art interval constraint propagation techniques.

More recently, the adaptive DAG framework of [141] was used in a branch-and-prune algorithm to find multiple steady states in homogeneous azeotropic and ideal two-product distillation problems [5]. Their computational experiments showed some promising results from the application of constraint propagation techniques of [141].

In this work, a forward-backward constraint propagation technique, similar to the DAG framework of [125], will be discussed and exploited. The technique is used to obtain meaningful bounds on the implicit functions of (2.5). Thus, the goal is to expedite the above bounding procedure over a given large initial box using the constraint propagation technique, and subsequently obtain rigorous, tight, and convergent bounds on implicit functions using the interval-Newton method. Combining the strengths of forward-backward constraint propagation and interval-Newton methods seems to be a promising approach to obtaining useful bounding information required for solving (2.5), and this will be the focus of the proposed solution framework.

8.2 Robust Simulation Algorithm Implementation

In Chapter 7, a cutting-plane algorithm (Alg. 7.1) for solving SIPs with implicit functions embedded was presented. Furthermore, it was shown how constrained max-min and min-max problems can be reformulated as implicit SIPs and solved using Algorithm 7.1. In this section, the problem of worst-case design is addressed, for which Algorithm 7.1 was demonstrated to be effective at solving. However, since robust simulation and design is effectively a worst-case feasibility problem (see Chap. 2), it may be unnecessary to solve the SIP formulation (2.5) to global optimality since a guarantee of feasibility or infeasibility solves the problem. This detail was identified in [134] where two termination criteria were added to the SIP algorithm of [14]. The new criteria terminate the algorithm if a rigorous lower bound on the solution is obtained such that $\eta^{LBD} > 0$, in which case the design is infeasible, or if a rigorous upper bound

is obtained such that $\eta^{UBD} \leq 0$, in which case, the design meets robust feasibility. Similarly, these termination criteria can be added to the implicit SIP algorithm (Alg. 7.1) and will be utilized in this section to solve the feasibility problem. With the addition of the new termination criteria, the algorithm is expected to be drastically more efficient since solving the SIP to global optimality is quite expensive. However, in order to guarantee that the algorithm terminates after finitely many iterations, the standard ϵ-optimality termination criterion must remain present. If, however, the algorithm terminates with ϵ-optimality, further investigation is required to determine robust feasibility of the design rigorously. These termination criteria are identical to those of the algorithm in [134], labeled 3(d) and 4(d), respectively, which was written with reference to solving the maximization problem (2.5).

Since Algorithm 7.1 is to be applied here, it is again required that the global optimization subproblems, with implicit functions embedded, can be solved. Consequently, efficiently calculating rigorous, tight, and convergent bounds on implicit functions is required, as previously discussed in this chapter and in Chapter 4. As a reminder, in order to calculate these bounds in this chapter, a combination of the forward-backward constraint propagation technique and the parametric interval-Newton method will be employed.

8.2.1 Forward-Backward Propagation of Intervals

Different interval arithmetic implementations have been developed in the past, e.g. for `C++` [22, 72]. These provide a new data type and use operator and function overloading to calculate interval extensions of arithmetic expressions. They can be easily used for the forward interval evaluation of an explicit factorable function. However, in order to evaluate the backward propagation of intervals, it is necessary to keep information about intermediate factors in memory. A similar requirement is found in the reverse mode of automatic differentiation [49]. There, a record of each operation is kept on a so-called *tape* during the forward function evaluation. During the reverse pass, the tape is read to reconstruct the operation and calculate the derivative. The stored information includes the type of operation and the address of the operands in the

tape.

Here, for the implementation of a backward interval propagation, it is proposed to proceed in a slightly different fashion. First, the factorable function is parsed using operator and function overloading to construct its computational graph. All other operations will be performed on this graph object. In contrast to typical AD implementations, the computational graph, which can be thought of as a kind of tape, is persistent in memory and can be reused after it is constructed once. Basically, the graph is stored in an array where each element, or factor, contains information about the type of operation and the address of the operands. In addition, for each factor, an interval is also stored. These intervals can be accessed for the independent and dependent variables, i.e., variables and function values, respectively. Also, it is possible to provide bounds on some specific intermediate factors, if desired.

Forward interval evaluation Prior to the forward interval evaluation, an interval is specified for each independent variable. Also, intervals can be specified for intermediate factors. Then, during the forward evaluation, the graph can be traversed element-by-element and an inclusion interval can be constructed for each factor according to its operation type since each factor depends only on factors that have been evaluated already and for which this inclusion information is already available. If an element is an intermediate factor for which bounds have been provided, then these bounds are intersected with the newly calculated interval so as to provide potentially tighter bounds. If the intersection is empty, then this bound can not be satisfied for all possible realizations of the independent variables. Once all factors have been calculated, the inclusion intervals of the dependent variables, i.e., the function values, are exported from the graph object.

Backward interval propagation After a forward interval evaluation has provided valid bounds on each factor, the intervals for the dependent variables can be updated by intersecting these with additional information such as constraints that must be satisfied. Then, the computational graph will be traversed in reverse order. For

example, suppose that the current factor is $v_k = v_i + v_j$. Then, it also must be true that $v_i = v_k - v_j$ and $v_j = v_k - v_i$. Analogous rules can be constructed for other operations too, where more discussion can be found in [125]. This provides additional bounds on the operands v_i and v_j, which can be intersected with their current bounds resulting in potentially tighter bounds. Again, factor after factor is re-visited until the first factor, that is not an independent variable, is reached. If an intersection resulted in an empty interval, then one can conclude that no possible realizations of the independent variables on the original box can satisfy the constraints. Otherwise, potentially tighter intervals have been computed for the independent variables.

It is possible to perform multiple forward evaluations and backward propagation steps consecutively as these do not necessarily converge in a single iteration. The following illustrative example illustrates the forward evaluation and backward propagation steps.

Example 8.2.1. Consider

$$f(z,p) = z^2 + zp + 4,\ X^0 = [-0.8, -0.3],\ P = [6, 9].$$

A factorable representation is given by

$$v_1 = z^2$$

$$v_2 = zp$$

$$v_3 = v_1 + v_2$$

$$v_4 = v_3 + 4$$

Forward interval evaluation results in

$$V_1 = [-0.8, -0.3]^2 = [0.09, 0.64]$$
$$V_2 = [-0.8, -0.3] \cdot [6, 9] = [-7.2, -1.8]$$
$$V_3 = [0.09, 0.64] + [-7.2, -1.8] = [-7.11, -1.16]$$
$$V_4 = [-7.11, -1.16] + [4, 4] = [-3.11, 2.84].$$

Prior to the backward pass, we set $V_4 = [0, 0]$. First, we update V_3 according to $V_3 := V_3 \cap V_4 - 4 = [-4, -4]$. Next, we reverse the assignment $V_3 = V_1 + V_2$ to update V_1 and V_2: $V_1 := V_1 \cap V_3 - V_2 = [0.09, 0.64]$, $V_2 := V_1 \cap V_3 - V_1 = [-4.64, -4.09]$. Then, $V_2 = X \cdot P$ is reversed: $X := X \cap V_2/P = [-0.7734, -0.4544]$, $P := P \cap V_2/X = [6, 9]$. Lastly, $V_1 = X^2$ is reversed: $X := \text{hull}\{X \cap -\sqrt{V_1}, X \cap \sqrt{V_1}\} = [-0.7734, -0.4544]$, which concludes the backward interval propagation. As a result, $X = [-0.7734, -0.4544]$ is a refinement of the original interval X^0 with the guarantee that any $x \in [-0.8, -0.3]$ for which there exists a $p \in P$ with $f(z, p) = 0$ is also contained in X.

In some cases, it is possible that a univariate function operating on an intermediate factor is only defined for a subset of $\mathbb{R}$, e.g., $\phi(z) = \cos^{-1}[z]$ is only defined for $z \in [-1, 1]$. In the model presented in the next section, a domain violation of the $\cos^{-1}$ function corresponds to the physical phenomenon of flooding of the gas-liquid separator unit. In this case, the model is invalid and evaluating it returns no meaningful solution. In order to prevent numerical artifacts from impacting the forward interval evaluation, the following convention will used. Consider the univariate function $\phi : D \subset \mathbb{R} \to \mathbb{R}$. If $z \notin D$ then $\phi(z) \equiv \emptyset$. Let Φ be an interval extension of ϕ and suppose X is an interval that is not fully contained in D. In this case it is safe to evaluate $\Phi(X \cap D)$ to obtain conservative bounds on the image of X under ϕ. It may be possible that $X \cap D = \emptyset$ which means that all points in X cause domain violations and, hence, the separator floods. Otherwise, at least one operating condition exist that does not cause flooding and the model can be evaluated safely.

8.2.2 SIP Algorithm

The algorithm for solving the robust simulation SIP was also implemented using `C++` in a manner analogous to Chapter 7 with two important additions. First, the additional stopping criteria discussed previously in this chapter were added to the algorithm, and second, the forward-backward constraint propagation technique was added. The flowchart for the algorithm is shown in Figure 8-1. All other details

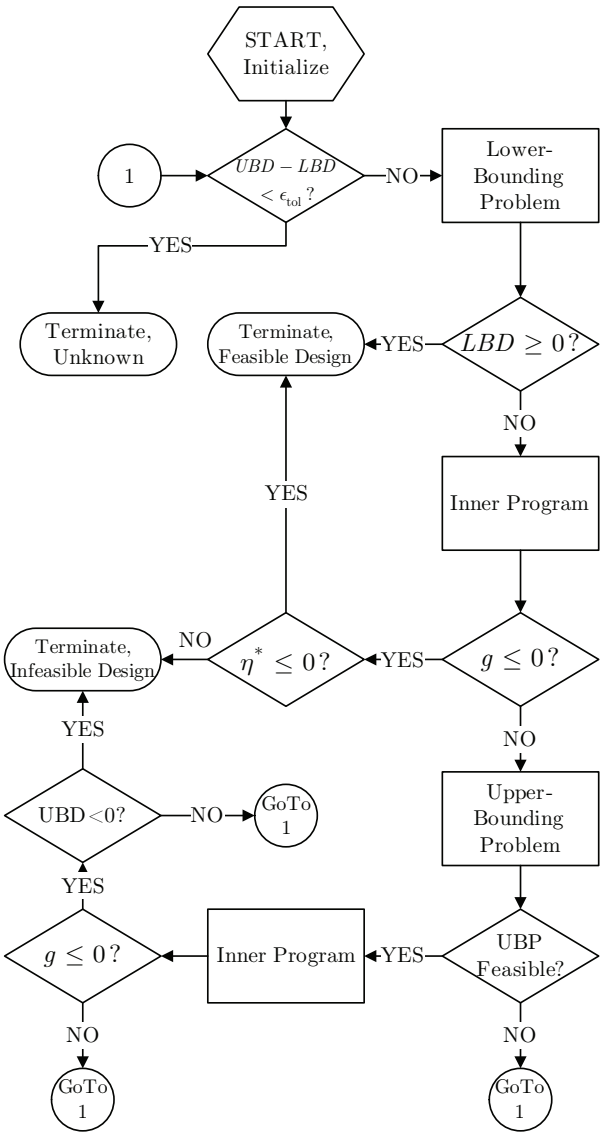

Figure 8-1: The simplified flowchart for the main SIP algorithm (Alg. 7.1) adapted as the robust simulation algorithm.

of the algorithm are identical to Chapter 7. In particular, the global optimization

subproblems are solved using the global optimization of implicit functions algorithm developed in Chapter 4 (Alg. 4.1) which relies on rigorous and convergent interval bounds on implicit functions and the ability to solve nonsmooth convex programs. PBUNS and PBUNL [83] are again employed to solve nonsmooth convex problems. Similar to Chapter 7, in order to circumvent the limitations imposed by PBUNL (i.e., only accepting affine inequality constraints), affine relaxations of convex nonlinear inequality constraints with respect to multiple reference points are used. The hierarchy for the information flow of the algorithm for global optimization of implicit functions is shown in Figure 8-2.

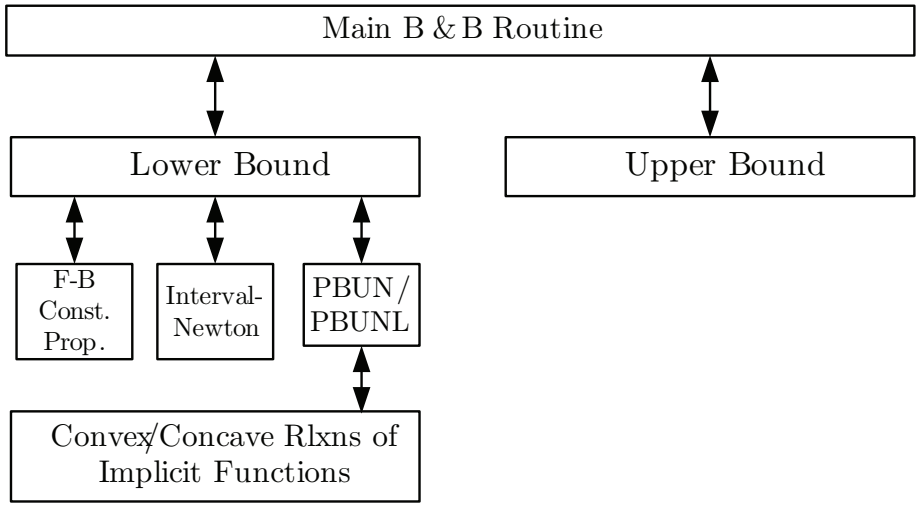

Figure 8-2: The hierarchy of the flow of information for global optimization of implicit functions.

The required interval bounding information is obtained using a combination of the forward-backward propagation technique discussed above and the parametric interval-Newton method discussed in Chapter 3. When the algorithm is first initialized, the computational graph corresponding to the system model is constructed. This graph is then made available to the global optimization of implicit functions algorithm. Interval bounds are constructed by passing initial bounds on the state variables—which are nothing more than natural bounds on the variables—and pertinent bounds on the controls and uncertainty parameters to the forward-backward constraint propagation implementation using the previously constructed graph. Forward-backward propagation is iterated until the interval is sufficiently refined, converges, or it is guaranteed

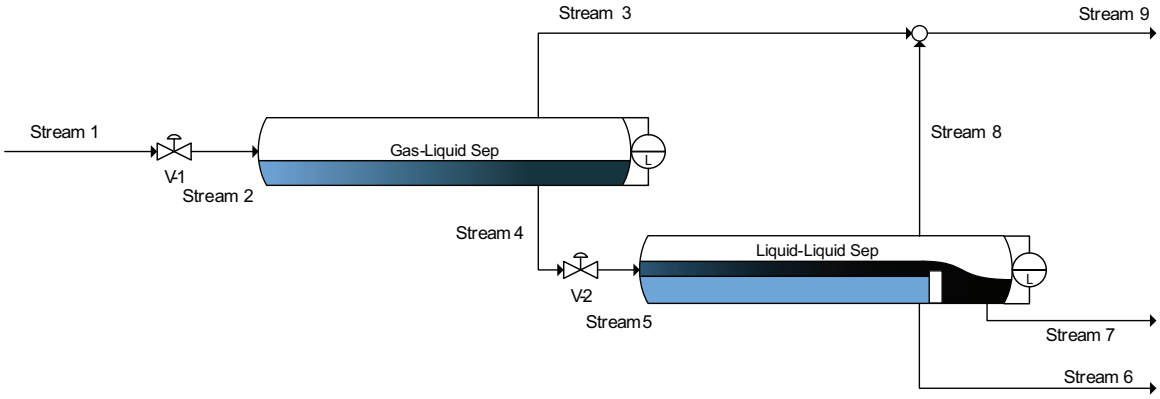

Figure 8-3: The ultra deepwater subsea separation process. (Photo credit: the author)

that no implicit function exists within the original bounds signaling an infeasible subproblem, in which case the interval is discarded. The resulting (nonempty) interval is then passed to the parametric interval-Newton method (implemented componentwise) to be potentially refined further. The resulting interval is then passed to the bundle solver to be used in the construction of the relaxations of implicit functions.

8.3 Model

The subsea separator is considered to be at the heart of subsea production facilities since it is the key process system for performing upstream material separation as it is being produced from the wellhead. In this model, it is considered that a three-phase mixture of oil/water/gas is being sufficiently separated to allow for re-injection of the water back into the environment and the production of separate oil and gas streams. It is assumed that sand has been separated from this stream prior to being fed to this separator model. Figure 8-3 shows the process flow diagram of the subsea separator with some modeling details.

The model consists of two control valves, a gas mixer, a gas-liquid separator (GLS), and a liquid-liquid separator (LLS). There is a control valve (V-1) on the inlet to the GLS as well as a control valve (V-2) on the inlet to the LLS. The gas outlet streams from each of the separators are combined in the gas mixer to form the gas product stream. Uncertainty in the inputs to the process will be in the form of

the feed gas/water/oil composition. The full details of the model including the model equations are discussed in detail in the following sections.

8.3.1 Model Assumptions

Since the model is meant to be a simple initial approach, there is a list of assumptions that may not necessarily hold true for a more detailed model or for the physical system. However, since various levels of complexity may be added to this model, certain assumptions can be eliminated in the future. For the purposes of this chapter, the following assumptions are made.

1. Ideal homogeneous mixtures in multiphase streams and the GLS.
2. No liquid entrainment in the gas phase.
3. Perfect oil-water phase separation in the LLS.
4. Unrestricted flow from the LLS implying constant phase volumes.
5. Horizontal separator vessels are horizontal cylinders with flat end-caps.
6. Oil and water phases remain in a homogeneous emulsion with only the gas phase separating in the GLS.

8.3.2 Input Parameters

The various physical properties of the system can be specified by the user as input parameters. The following tables contain the parameter values used in this study.

Fluid Properties		
API	35	American Petroleum Institute gravity
SG_G	0.6	specific gravity of gas
SG_W	1.0	specific Gravity of water
ρ_W°	1000	density of water (kg/m^3) at standard conditions

Table 8.1: The fluid properties used in the subsea separator model.

Physical Design Specifications		
C_{v1}	1.0	V-1 sizing coefficient (kg/Pa$^{1/2}$min)
C_{v2}	10.05	V-2 sizing coefficient (kg/Pa$^{1/2}$min)
L_{GLS}	$\in \{4.0, 5.0\}$	length (m) of GLS
R_{GLS}	$\in \{0.4, 0.6\}$	radius (m) of GLS
P_{GLS}	39.5	operating pressure (atm) of GLS
L_{LLS}	5.0	length (m) of LLS
R_{LLS}	0.8	radius (m) of LLS
P_{LLS}	39.5	operating pressure (atm) of LLS
H_{LLS}	0.6	liquid level in the LLS (m)

Table 8.2: The physical design specifications of the subsea separator model.

Input Conditions		
P_{well}	54.5	wellhead pressure (atm)
ξ_{G1}	$\in [0, 1]$	mass fraction of gas, uncertain
ξ_{W1}	$\in [0, 1]$	mass fraction of water, uncertain
ξ_{O1}	$\in [0, 1]$	mass fraction of oil, uncertain

Table 8.3: The input conditions for the subsea separator model.

The following calculation for the specific-gravity of oil is used:

$$SG_O = \frac{141.5}{131.5 + API}.$$

The specific gravity of the mixture at the wellhead is

$$SG_{mix} = (\xi_{G1}/SG_G + \xi_{W1}/SG_W + \xi_{O1}/SG_O)^{-1}.$$

Of course, the following constraint on the mixture at the wellhead must hold:

$$\xi_{G1} + \xi_{W1} + \xi_{O1} = 1.$$

Control Settings		
u_1	$\in [0, 1]$	valve V-1 opening
u_2	$\in [0, 1]$	valve V-2 opening

Table 8.4: The control settings for the subsea separator model.

8.3.3 Control Valve V-1

The variables associated with V-1 are:

$$\dot{m}_1, \dot{m}_2, P_1, P_2, \xi_{G1}, \xi_{O1}, \xi_{W1}, \xi_{G2}, \xi_{O2}, \xi_{W2}$$

where $\dot{m}_i$ is the mass flowrate of Stream i in kg/min, P_i is the pressure of Stream i in Pa, and ξ_{ji} is the mass fraction of component j in Stream i. Similarly, the vector of mass fractions can be expressed as $\boldsymbol{\xi}_i = (\xi_{Gi}, \xi_{Wi}, \xi_{Oi})$.

The model equations are:

$$P_1 = P_{well}\frac{101325\text{Pa}}{1\text{atm}} \text{ (specified from wellhead)}$$

$$P_2 = P_{GLS}\frac{101325\text{Pa}}{1\text{atm}} \text{ (GLS specified design pressure)}$$

$$\boldsymbol{\xi}_1 = \boldsymbol{\xi}_2 \text{ (specified, source of disturbance uncertainty)}$$

The mass flow rate through the valve is given by

$$\dot{m}_1 = \dot{m}_2 = u_1 C_{v1}\sqrt{\frac{P_1 - P_2}{SG_{mix}}} \tag{8.1}$$

where u_1 is the control setting.

8.3.4 Gas-Liquid Separator

The variables associated with the GLS are:

$$\dot{m}_2, \dot{m}_3, \dot{m}_4, P_2, P_3, P_4, H_{GLS}, \rho_4, V_{GLS}, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3, \boldsymbol{\xi}_4,$$

where H_{GLS} is the liquid level in the GLS in m, ρ_4 is the density of the mixture in Stream 4 in kg/m^3, and V_{GLS} is the liquid volume in the GLS in m^3. The associated model equations are given below.

The pressure relationships are given by:

$$\begin{aligned} P_3 &= P_2 \ \text{(specified by design)} \\ P_4 &= P_3 + \rho_4 g_a H_{GLS} \end{aligned} \tag{8.2}$$

The liquid volume in the GLS is given by:

$$V_{GLS} = L_{GLS}\left((H_{GLS} - R_{GLS})\sqrt{2R_{GLS}H_{GLS} - H_{GLS}^2} + R_{GLS}^2 \cos^{-1}\left[1 - \frac{H_{GLS}}{R_{GLS}}\right]\right), \tag{8.3}$$

where g_a is the acceleration of gravity in m/s^2. The mass balances are:

$$\begin{aligned} \dot{m}_2 &= \text{specified by wellhead} \\ \dot{m}_2 &= \dot{m}_3 + \dot{m}_4 \\ 1 &= \xi_{G4} + \xi_{W4} + \xi_{O4} \\ \xi_{G2}\dot{m}_2 &= \xi_{G3}\dot{m}_3 + \xi_{G4}\dot{m}_4 \\ \xi_{W2}\dot{m}_2 &= \xi_{W4}\dot{m}_4 \ \text{(no water in the tops)} \\ \boldsymbol{\xi}_2 &= \text{specified by wellhead composition} \\ \boldsymbol{\xi}_3 &= (1,0,0) \ \text{(only gas in Stream 3)} \end{aligned}$$

Where the density of Stream 4 is given by:

$$\rho_4 = \rho_W^{\circ}(\xi_{G4}/SG_G + \xi_{W4}/SG_W + \xi_{O4}/SG_O)^{-1} \tag{8.4}$$

Lastly, the model describing the gas-liquid separation is a simple exponential decay

$$\xi_{G4} = \xi_{G2} \exp\left[-0.5\frac{V_{GLS}}{\dot{m}_4/\rho_4}\right] \tag{8.5}$$

where the constant 0.5 is a separator performance factor and the quantity $V_{GLS}/(\dot{m}_4/\rho_4)$ is a residence time of the liquid solution in the GLS.

8.3.5 Control valve V-2

The control valve V-2 is almost identical to V-1. The associated variables are:

$$P_4, P_5, \dot{m}_4, \dot{m}_5, \boldsymbol{\xi}_4, \boldsymbol{\xi}_5, \rho_4.$$

The mass-balance equations are

$$\boldsymbol{\xi}_4 = \boldsymbol{\xi}_5$$

$$\dot{m}_4 = \dot{m}_5 = u_2 C_{v2} \sqrt{\frac{P_4 - P_5}{\rho_4 / \rho_W^\circ}}$$

The outlet pressure is given by

$$P_5 = P_{LLS} \frac{101325 Pa}{1 atm}.$$

8.3.6 Liquid-Liquid Separator

The LLS is very similar to the GLS. One key difference is the liquid level in the LLS is specified assuming no restrictions on exit stream flowrates. The associated variables are

$$\dot{m}_5, \dot{m}_6, \dot{m}_7, \dot{m}_8, \boldsymbol{\xi}_5, \boldsymbol{\xi}_6, \boldsymbol{\xi}_7, \boldsymbol{\xi}_8, P_8, V_{LLS}, V_{oil}, H_{LLS}, \rho_7,$$

where ρ_7 is the density of the solution in Stream 7 in kg/m^3, V_{LLS} is the total liquid volume in the LLS in m^3, H_{LLS} is the total liquid level in the LLS in m, and V_{oil} is the volume of just the oil/gas mixture in the LLS.

The liquid volume in the LLS is given by

$$V_{LLS} = L_{LLS} \left((H_{LLS} - R_{LLS}) \sqrt{2 R_{LLS} H_{LLS} - H_{LLS}^2} + R_{LLS}^2 \cos^{-1} \left[1 - \frac{H_{LLS}}{R_{LLS}} \right] \right) \tag{8.6}$$

where the liquid height H_{LLS} is specified. The volume of the oil/gas mixture phase in the separator is given by the following relationship assuming an ideal mixture

$$V_{oil} = V_{LLS} \left(\frac{\dot{m}_7}{\rho_7} \frac{\rho_5}{\dot{m}_5} \right). \tag{8.7}$$

The quantity $\dot{m}_7\rho_5/\rho_7\dot{m}_5$ is the volume fraction of the combined oil and gas exiting in Stream 7 with respect to the total solution incoming in Stream 5. The product with V_{LLS} is therefore the volume of the oil/gas solution for which further gas-liquid separation is taking place.

The mass-balance equations are:

$$\begin{aligned}
\boldsymbol{\xi}_8 &= (1,0,0) \text{ (only gas in Stream 8)} \\
\boldsymbol{\xi}_6 &= (0,1,0) \text{ (only water in Stream 6)} \\
1 &= \xi_{G7} + \xi_{O7} \text{ (only gas and oil in Stream 7)} \\
\dot{m}_5 &= \dot{m}_6 + \dot{m}_7 + \dot{m}_8 \\
\xi_{G5}\dot{m}_5 &= \xi_{G8}\dot{m}_8 + \xi_{G7}\dot{m}_7 \\
\xi_{W5}\dot{m}_5 &= \xi_{W6}\dot{m}_6 \\
\xi_{W7} &= 0 \text{ (no water in Stream 7).}
\end{aligned}$$

Further gas-liquid separation is again being modeled as a simple exponential decay

$$\xi_{G7} = \xi_{G5} \exp\left[-0.01 \frac{V_{oil}}{\dot{m}_7/\rho_7}\right] \tag{8.8}$$

where the constant 0.01 is a separator performance factor and the quantity $V_{oil}/\dot{m}_7\rho_7$ is a residence time of the gas/oil mixture in the separator.

The density of the oil/gas mixture stream is given by

$$\rho_7 = \rho_W^\circ(\xi_{G7}/SG_G + \xi_{O7}/SG_O)^{-1}.$$

Since the liquid outlet streams have no restrictions to flow, their flow-pressure relationships can be ignored. The pressure in the gas stream 8 is specified:

$$P_8 = P_{LLS}\frac{101325\text{Pa}}{1\text{atm}}.$$

8.3.7 Gas Mixer

The gas mixer is simply a junction of two pure gas streams. The associated variables are

$$\boldsymbol{\xi}_8, \boldsymbol{\xi}_3, \boldsymbol{\xi}_9, \dot{m}_3, \dot{m}_8, \dot{m}_9, P_3, P_8, P_9.$$

The associated equations are

$$\boldsymbol{\xi}_3 = \boldsymbol{\xi}_9$$

$$\dot{m}_9 = \dot{m}_3 + \dot{m}_8$$

$$P_9 = \min\{P_3, P_8\}$$

8.3.8 Model Structure

The total size of the system ends up as 11 state variables, 1 uncertain parameter, and 2 control variables. The computational graph for the subsea separator model is shown in Figure 8-4. It depicts how all the state variables, uncertainty parameters, control variables, and intermediate variables are coupled through the model equations. The corresponding occurrence matrix is shown in Figure 8-5 which depicts only how the model equations depend on the state variables, which is most important for equation solving.

8.4 Case Study

8.4.1 Pointwise Numerical Simulation

Pointwise numerical simulation was performed to study the behavior of the model over a range of uncertainty parameter and control values using the JACOBIAN® process simulator [112]. In effect, this resulted in a very coarse-grain view of the system under varying input conditions and control actions. It should be noted that the gas mixer model equations contain the min operator, which is nonsmooth. This does not pose any problems for the JACOBIAN® solver, which can handle nonsmoothness, and

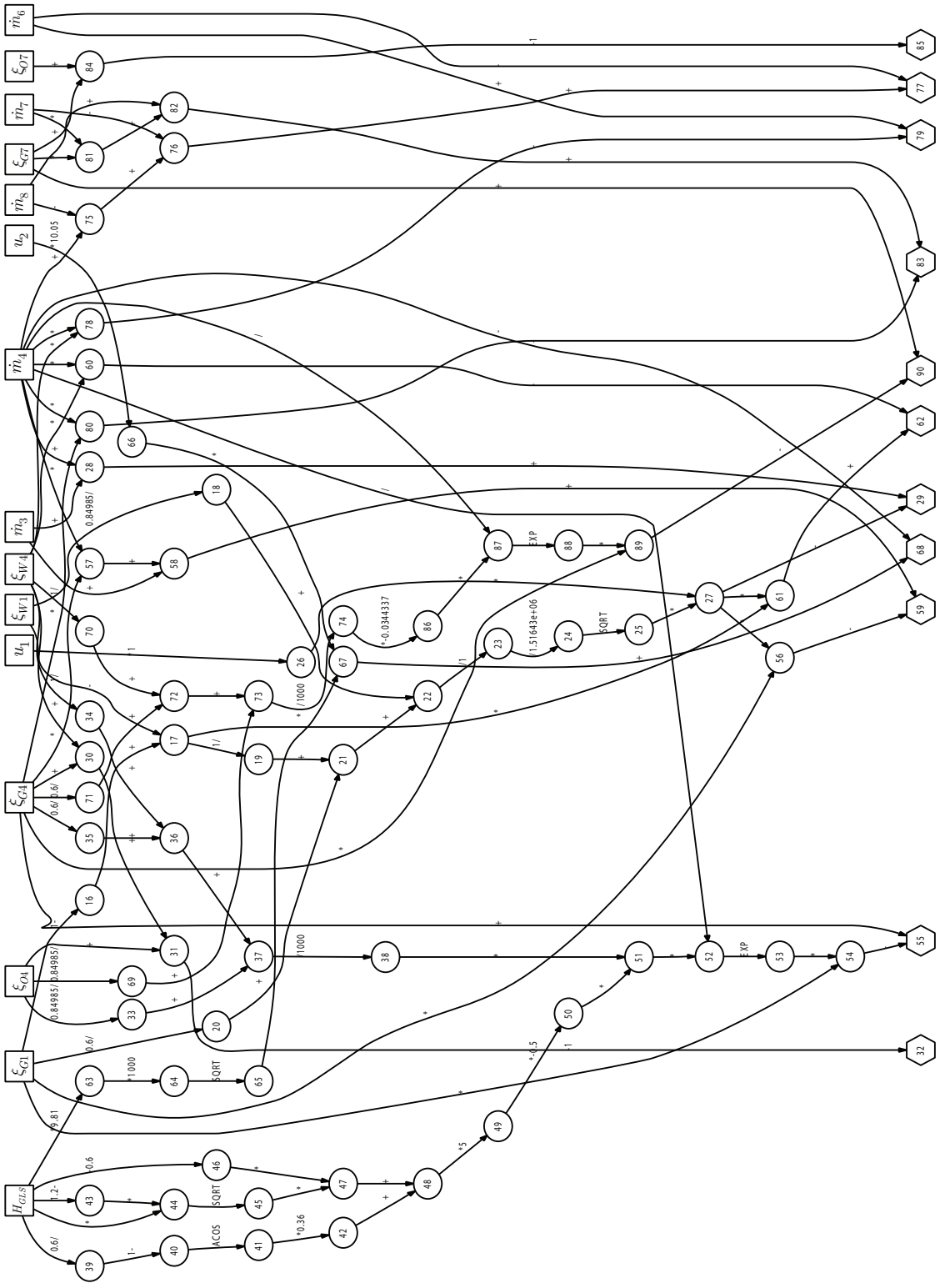

Figure 8-4: The computational graph associated with the subsea separator model. The square nodes represent all of the variables (including intermediates), the circle nodes represent intermediate expressions, and the hexagonal nodes represent the model outputs.

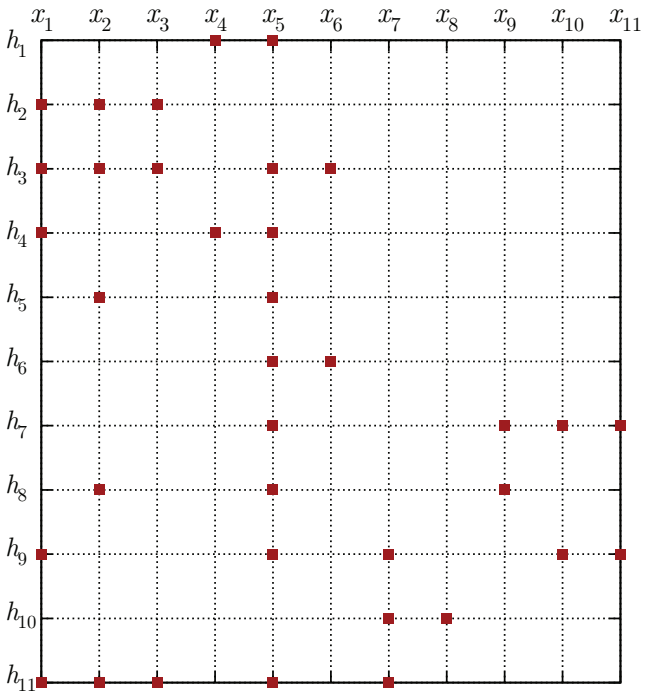

Figure 8-5: The occurrence matrix associated with the subsea separator model.

since that particular equation is not required to solve the model, it does not introduce any problems for the SIP algorithm either.

For this study, the mass fractions of the gas (ξ_{G1}) and water (ξ_{W1}) in the input stream were varied holding the oil fraction (ξ_{O1}) constant. The physical interpretation of this may be a gas bubble being produced from the wellhead. The gas fractions of 0.35, 0.4, 0.45, and 0.5 were studied, and the oil fraction was set to 0.4. The dimensions of the GLS were set to $R_{GLS} = 0.6$m and $L_{GLS} = 5$m. The control actions were varied between 30% open and fully-opened positions in 5% increments. A constraint was imposed that the gas carry-under (GCU), which is the fraction of gas in the oil product stream, had to be less than (or equal to) 5%. The inherent physical constraint that the liquid volume in the GLS could not be greater than the total volume of the GLS vessel was imposed by the design. A coarse-grain view of the operating envelope for each finite realization of gas fraction of the incoming stream as a function of the control actions are shown in Figure 8-6. Each contour plot is the result of running 196 steady-state simulations, taking a total time of approximately

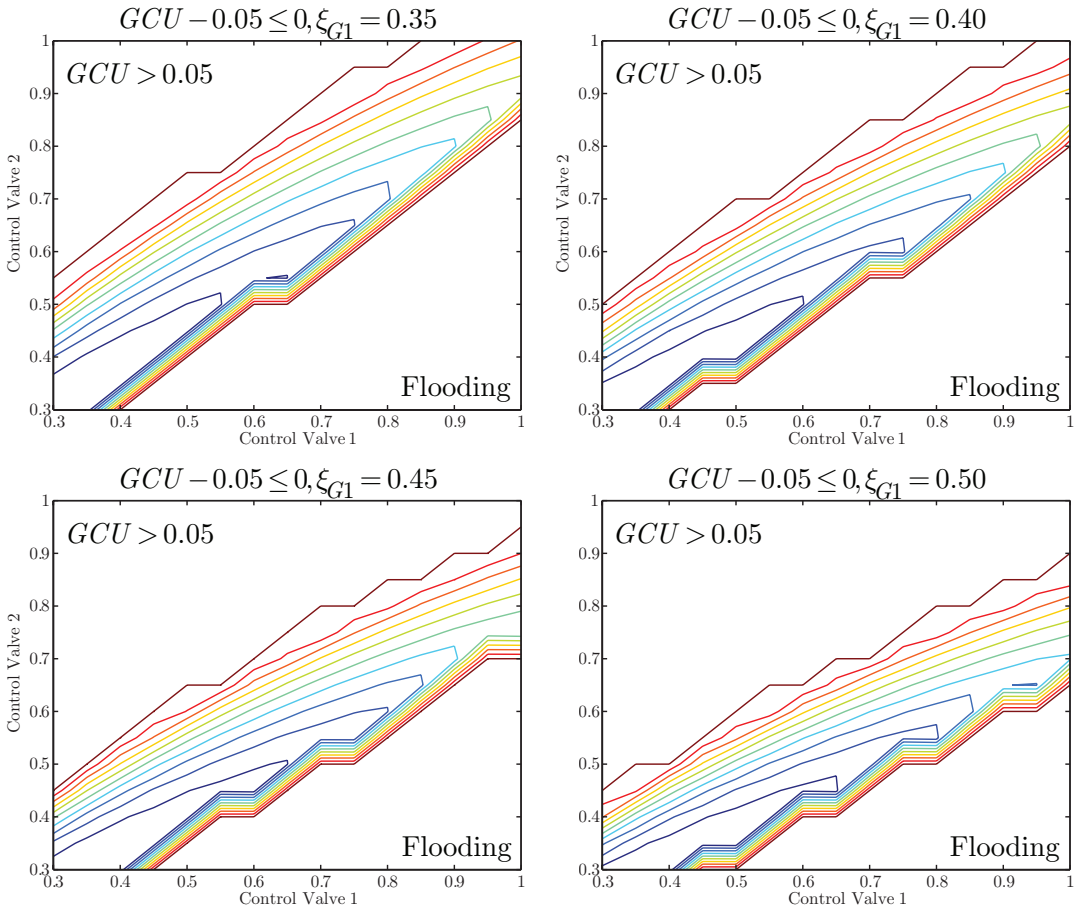

Figure 8-6: The coarse-grain approximations of the operating envelope of the subsea separator process model as a function of control actions for varying input compositions (uncertainty).

45 seconds for each gas fraction.

It is apparent that for even relatively simple models, the process systems can often exhibit complex nonconvex behavior. It is interesting to note that the region of feasible operation of the subsea separator system is actively constrained by the region where the GLS vessel floods. This suggests that the GLS performs optimally when it is nearly flooded. This property has important physical, as well as numerical, implications that will be discussed later.

In Chapter 2, it was stated that explicit enumeration (pointwise simulation) was an inadequate procedure for guaranteeing robust feasibility of the process. It is not only because pointwise numerical simulations are computationally expensive, but because in order to certify robust feasibility, every realization of uncertainty (and

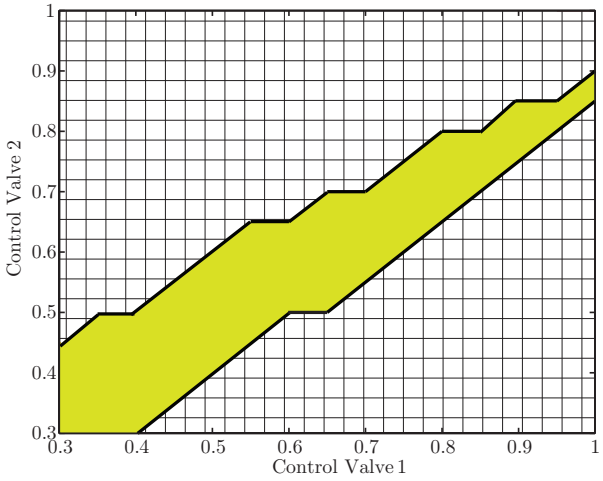

Figure 8-7: The feasible operating envelope for four realizations of uncertainty corresponding to the subsea separator model is shown as the solid band.

controls) must be simulated. However, in order to guarantee the process does not satisfy robust feasibility, it is sufficient to guarantee that at least one of the operating envelopes shown in Figure 8-6, explicitly enumerating every (infinite) control realization, has an empty intersection with the control interval. This implies that for some realization of uncertainty, there does not exist a feasible control setting such that the performance/safety specification is satisfied and therefore the process does not satisfy robust feasibility. The intersection of the feasible regions of the pointwise simulations are shown in Figure 8-7. This region corresponds to the control settings that are feasible for the four realizations of uncertainty simulated. In other words, a control interval that intersects this region is guaranteed to have a nonempty intersection with the four operating envelopes considered in the pointwise simulation. In fact this region corresponds to the control settings for which the design is feasible (with respect to the four uncertainty realizations) even in the event that the control valves cannot be adjusted. Although no conclusive information can be obtained from this result, with respect to the four realizations of uncertainty, the system appears to meet robust feasibility. In the next section, the robust simulation SIP (2.5) is solved for this model and a mathematically rigorous and conclusive result regarding robust feasibility is obtained.

8.4.2 Robust Simulation

Similar to the previous case study, the mass fraction of the gas in the feed stream from the wellhead was considered to be uncertain. Likewise, the oil composition was held constant. The controls simply correspond to the control valve positions. Stated formally:

$$\begin{aligned}\mathbf{p} &= (\xi_{G1}),\\ \mathbf{u} &= (u_1, u_2),\\ \mathbf{z} &= (\xi_{G4}, \xi_{W4}, \xi_{O4}, \dot{m}_3, \dot{m}_4, H_{GLS}, \xi_{G7}, \xi_{O7}, \dot{m}_6, \dot{m}_7, \dot{m}_8).\end{aligned}$$

Again, in order to avoid pump damage, the performance specification for the model requires that the mass fraction of gas in the oil product stream (GCU) was less than or equal to some specified amount, $G^{\max}$, which will be varied for the purpose of demonstrating the algorithm. Stated formally, the performance specification is $g(\mathbf{z}, \mathbf{u}, \mathbf{p}) = \xi_{G7} - G^{\max} \leq 0$, where $\mathbf{z}$ is the vector of all the internal state variables, such as flow rates and compositions of each stream. Since the simulation algorithm solves the model for the state variables as implicit functions of the controls and uncertainty parameters (represented as $\mathbf{x} : U \times P \to X$), the performance specification can be stated as the following nonlinear implicit function:

$$g(\mathbf{x}(\mathbf{u}, \mathbf{p}), \mathbf{u}, \mathbf{p}) = x_{G7}(\mathbf{u}, \mathbf{p}) - G^{\max} \leq 0,$$

where x_{G7} is the relevant component of $\mathbf{x}$ representing the mass fraction of gas in the oil product stream. Therefore, the robust simulation SIP (2.5) can be written for this

problem as:

$$\begin{aligned} \eta^* = & \max_{\mathbf{p}\in P, \eta\in\mathbb{R}} \eta \\ & \text{s.t. } \eta \leq x_{G7}(\mathbf{u},\mathbf{p}) - G^{\max}, \ \forall \mathbf{u} \in U \\ & P = [0.35, 0.50] \\ & U = [0,1]\times[0,1], \end{aligned} \tag{8.9}$$

or equivalently as:

$$\begin{aligned} -\eta^* = & \min_{\mathbf{p}\in P, \eta\in\mathbb{R}} -\eta \\ & \text{s.t. } \eta - x_{G7}(\mathbf{u},\mathbf{p}) + G^{\max} \leq 0, \ \forall \mathbf{u} \in U \\ & P = [0.35, 0.50] \\ & U = [0,1]\times[0,1]. \end{aligned} \tag{8.10}$$

It is clear from Figure 8-6 that certain control valve settings lead to flooding of the GLS. In other words, if the control valve V-2 was allowed to close too much and the control valve V-1 was allowed to be open too much, the GLS would flood. This phenomena has not only physical implications, but more importantly, numerical ones, which were discussed in Section 8.2.1. The physical issue with this scenario is that the GLS effectively becomes useless as no gas-liquid separation can occur. Numerically, since the GLS was modeled as a horizontal cylinder, the model equations contain the term $\cos^{-1}[1 - H/R]$, which is only defined on the set $\{H \in \mathbb{R} : 0 \leq H \leq 2R\}$. Thus, if the control valves are allowed to take values from fully-opened to fully-closed, it is easy to produce scenarios with $H > 2R$, and numerically, there is a domain violation of the $\cos^{-1}$ term and the model has no solution. When this situation was encountered in the study from the previous section, with results depicted in Figure 8-6, the process simulator simply fails, as expected. Such domain violations are the topic of future research.

For this study, the valid interval from which H_{GLS} may take values is $[0, 2R_{GLS}]$.

However, the model solution is an implicit function of the controls $\mathbf{u}$ and the uncertain variable $\mathbf{p}$, and therefore, the liquid level in the GLS is dependent on both $\mathbf{u}$ and $\mathbf{p}$. Since the level in the GLS is limited to the interval $[0, 2R_{GLS}]$, it is clear that the implicit function $\mathbf{x}$ does not exist for every $(\mathbf{u}, \mathbf{p}) \in U \times P$. Because of this, Algorithm 4.1 may encounter three situations. The first situation is one where a partition $U^l \times P^l$ is popped off the stack in which $\mathbf{x}$ exists for every point $(\mathbf{u}, \mathbf{p}) \in U^l \times P^l$, after applying forward-backward constraint propagation. This situation is of course no different than if $\mathbf{x}$ exists on all of $U \times P$. The second situation that may be encountered is one where $\mathbf{x}$ doesn't exist for any $(\mathbf{u}, \mathbf{p}) \in U^l \times P^l$ after applying forward-backward constraint propagation. In this case, the subproblem is simply labeled as infeasible and the partition $U^l \times P^l$ is discarded. The third situation which may be encountered is one where $\mathbf{x}$ only exists for some $(\mathbf{u}, \mathbf{p}) \in U^l \times P^l$ after applying forward-backward constraint propagation. The question arises of how Algorithm 4.1 may address this situation. For the purposes of this thesis, when this situation is encountered, relaxations of $\mathbf{x}$ are still constructed but since $\mathbf{x}$ does not exist on all of $U^l \times P^l$, the relaxations that are constructed are relaxations of $\mathbf{x}$ on $U^{l'} \times P^{l'} \subset U^l \times P^l$, where $\mathbf{x}$ exists for every $(\mathbf{u}, \mathbf{p}) \in U^{l'} \times P^{l'}$ and does not exist for any $(\mathbf{u}, \mathbf{p}) \in (U^{l'} \times P^{l'}) \cap (U^l \times P^l)$. In this case, relaxations of $\mathbf{x}$ do not exist for $(\mathbf{u}, \mathbf{p}) \in (U^{l'} \times P^{l'}) \cap (U^l \times P^l)$. The following example illustrates this idea.

Example 8.4.1. Consider $x(p) = \cos^{-1}[1 - p/2]$ on $P = [-2, 6]$. A factorable representation is given by

$$v_1 = p/2$$
$$v_2 = 1 - v_1$$
$$v_3 = \cos^{-1} v_2$$

Calculating the forward interval evaluation of x on P according to Section 8.2.1 results

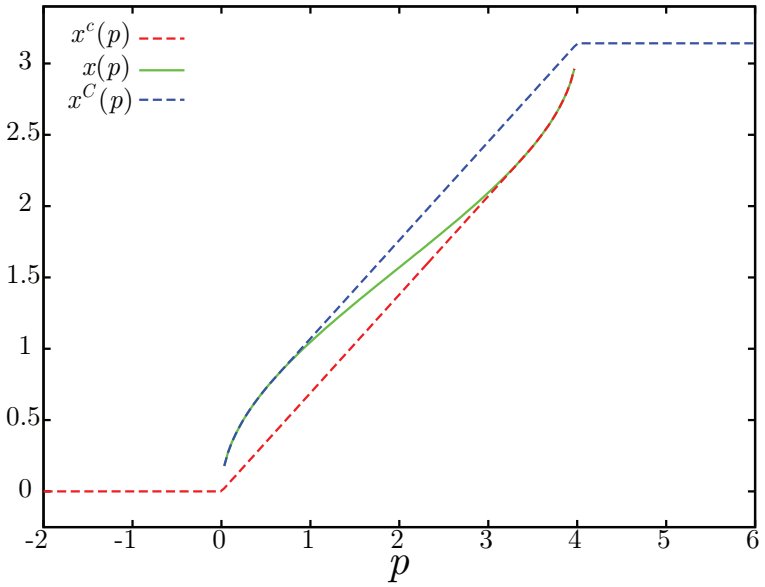

Figure 8-8: Convex and concave relaxations of $x(p) = \cos^{-1}[1 - p/2]$ on $P = [-2, 6]$ which exist only for $p \in [0, 4]$.

in:

$$V_1 = [-2, 6]/2 = [-1, 3]$$

$$V_2 = 1 - [-1, -3] = [-2, 2]$$

$$V_3 = \cos^{-1}([-2, 2] \cap [-1, 1]) = \cos^{-1}([-1, 1]) = [0, \pi].$$

Furthermore, the results of Chapter 4 can be applied and relaxations of x on P can be calculated. Of course, these relaxations are only guaranteed to exist for $\mathbf{p} \in [0, 4]$, as Figure 8-8 illustrates.

If the bundle solver requires the evaluation of a function at a point in which it doesn't exist, the bundle solver simply fails. In the event that the bundle solver fails, the lower bound from the interval evaluation is used as the lower bound on the node and it is either fathomed or branched in the normal way. In essence, the problematic regions of the search space are systematically discarded by the algorithm.

In order to get a sense of the effectiveness of each of the interval methods on this problem, forward-backward constraint propagation technique was applied to the

	X^0	X^{FB}	X^N
ξ_{G4}	$[9.46305 \times 10^{-3}, 0.36]$	$[0.080707, 0.154937]$	$[0.11869, 0.11869]$
ξ_{W4}	$[0.24, 0.375]$	$[0.316898, 0.344735]$	$[0.330492, 0.330492]$
ξ_{O4}	$[0.4, 0.625]$	$[0.500327, 0.602395]$	$[0.550819, 0.550819]$
$\dot{m}_3$	$[0, 304.517]$	$[205.261, 256.99]$	$[231.61, 231.61]$
$\dot{m}_4$	$[541.364, 845.881]$	$[588.891, 640.62]$	$[614.271, 614.271]$
H_{GLS}	$[0.462165, 0.7992]$	$[0.546875, 0.647172]$	$[0.59503, 0.59503]$
ξ_{G7}	$[8.6697 \times 10^{-3}, 0.348991]$	$[0.0762948, 0.148717]$	$[0.113166, 0.113166]$
ξ_{O7}	$[0.651009, 0.99133]$	$[0.851283, 0.923705]$	$[0.886834, 0.886834]$
$\dot{m}_6$	$[203.011, 203.011]$	$[203.11, 203.11]$	$[203.11, 203.11]$
$\dot{m}_7$	$[338.352, 642.869]$	$[338.352, 437.609]$	$[381.528, 381.528]$
$\dot{m}_8$	$[0, 304.517]$	$[0, 73.4416]$	$[29.7311, 29.7311]$

Table 8.5: Forward-backward constraint propagation was applied with 100 iterations to X^0 resulting in the refined interval X^{FB}. Interval-Newton with interval Gauss-Seidel was then applied to X^{FB} resulting in the refined interval X^N after 5 iterations.

model[1] with 100 iteration at the point $\mathbf{p} = (0.36)$ and $\mathbf{u} = (0.6, 0.8)$. The initial interval X^0, which is calculated automatically from physical information about the problem, is shown in the second column of Table 8.5. Forward-backward constraint propagation took 0.006 seconds to complete and the results are reported in the third column of Table 8.5. Finally, (parametric) interval-Newton with interval Gauss-Seidel was applied iteratively to the refined interval from the forward-backward constraint propagation technique. Recall from Chapter 3 that this iteration will terminate in finitely many iterations when using rounded interval arithmetic. Although this is true, it is often unnecessary to continue iterating until this point since the resulting interval may be unnecessarily precise. For this study, the intervals are said to be converged when an absolute tolerance on the bounds of 10^{-9} is met. For this demonstration, interval-Newton converged after 5 iterations, taking 0.008 seconds. The result of applying interval-Newton is shown in the last column of Table 8.5. It can be seen that forward-backward constraint propagation was very effective in refining the initial interval, however it failed to converge to degeneracy in 100 iterations. In fact, forward-backward constraint propagation is nearly converged after 100 iterations and no further refinement is seen after 200 iterations. However, interval-Newton is very

[1]The compact dimensions were used, $L_{GLS} = 4.0$m and $R_{GLS} = 0.4$m.

effective in further refining the interval returned from forward-backward constraint propagation. It should be noted that interval-Newton applied to X^0 is not effective at all and simply returns X^0.

For this model, considerable overestimation was encountered using interval analysis. In order to circumvent the issues that arise due to excessive overestimation, the uncertainty interval P was subdivided and the robust simulation SIP was solved for each subdivision of P. The uncertainty intervals considered were $P^1 = [0.35, 0.3875]$, $P^2 = [0.3875, 0.425]$, $P^3 = [0.425, 0.4625]$, and $P^4 = [0.4625, 0.50]$. In order to demonstrate the effectiveness and applicability of the algorithm, four different robust simulation case studies were performed varying the size of the GLS and the performance specification.

Case 1

The control interval $U = [0.35, 0.8]^2$ was considered. The GLS dimensions were such that $R_{GLS} = 0.6$m, and $L_{GLS} = 5$m. The performance specification was such that $G^{\max} = 0.05$.

The robust simulation algorithm solved the problem in a total time of 3.35sec, with a rigorous upper bound on η of -0.0104, implying $\eta^* \leq 0$. That is, for all realizations of wellhead compositions within the intervals considered, there exists a control setting from the interval considered such that the design meets the product purity specification that no more than 5% of the oil product stream can be gas, in the worst case.

Case 2

The same control interval and GLS dimensions as the previous case are considered. The performance constraint was made much more strict with $G^{\max} = 0.015$.

The robust simulation algorithm solved the problem in a total time of 14.7 seconds, with a rigorous upper bound on η of -1.25×10^{-3}, implying $\eta^* \leq 0$. The solution time was about twice as long for this case as compared to Case 1, with the more relaxed performance specification. This is likely due to the fact that the feasible region for

this problem is significantly smaller than that of Case 1.

Case 3

The same control interval as the previous cases is used and the GLS dimensions are such that $R_{GLS} = 0.4$m and $L_{GLS} = 4$m. These new dimensions result in the GLS having roughly 36% of the volume of that in the previous two studies. The performance specification is such that $G^{\max} = 0.05$. Intuitively, the smaller GLS dimensions will result in the new design having reduced performance as compared to the larger design. Therefore, it is expected that if this design is feasible, the feasible region of the SIP will be much smaller than that of Case 1.

The robust simulation algorithm solved the problem in a total time of 598.3 seconds with a rigorous upper bound on η of -5.77×10^{-3}. Again, this means that $\eta^* \leq 0$, implying that the design is feasible. The solution time of the algorithm suggests that verifying robust feasibility of the more compact design is much more difficult than for the original larger design. Again, this is likely due to the significantly smaller feasible region of the SIP.

Case 4

The purpose of this final study is to demonstrate the behavior of the algorithm when the design is not nearly as robust as that in the previous studies. The dimensions of the GLS are the same as in Case 3. For this study, the control valve V-2 is stuck at 50% and the control valve V-1 will have limited movement between 30%-35%. The performance specification will be such that $G^{\max} = 0.05$.

The robust simulation algorithm solved the problem in a total time of 25.8 seconds with a rigorous lower bound on η of 1.32×10^{-2}, implying $\eta^* > 0$. As expected, this more compact design with extremely limited control actuation does not satisfy robust feasibility. With regards to the solution time, the SIP algorithm performed favorably for this study. This seems rather counterintuitive since guaranteeing infeasibility of the design requires locating an SIP-feasible point that has a corresponding objective function value that is greater than zero (such a point provides a rigorous lower bound

on η). However, for this problem, in order to guarantee infeasibility of the design, it simply needs to verify infeasibility of the design for one of the four uncertainty intervals. Therefore, although generating a rigorous lower bound on η that is greater than zero may be computationally expensive (especially for a problem with such a small SIP-feasible region), once it is done, the algorithm can terminate without needing to solve the remaining SIPs for the other uncertainty intervals.

8.4.3 Algorithm Performance

Case	n	t (sec.)	$LBPs$	n_{LBP}	$UBPs$	n_{UBP}	IPs	n_{IP}
1	2	1.675	2	19	1	1	2	6
2	3	3.684	3	24	2	1	3	8
3	2	149.58	2	16	1	1	2	2726
4	7	25.794	7	15	7	36	10	45

Table 8.6: The performance of the algorithm depicted in Figure 8-1 for each case.

The performance of the robust simulation algorithm varies wildly between cases. Table 8.6 shows how the algorithm performed for each case in terms of the average number of iterations taken by the robust simulation algorithm (n), the average solution time (t), the average number of lower-bounding problems solved ($LBPs$), the average number of branch-and-bound iterations taken to solve each lower-bounding-problem (n_{LBP}), the average number of upper-bounding problems solved ($UBPs$), the average number of branch-and-bound iterations taken to solve each upper-bounding problem (n_{UBP}), the average number of inner programs solved (IPs), and the average number of branch-and-bound iterations taken to solve each inner program (n_{IP}). Besides n and t, the values in Table 8.6 are rounded up to the nearest integer.

For Case 1 and Case 2, the algorithm is quite effective and performs as expected. The algorithm takes roughly twice as long to solve Case 2 as it does to solve Case 1. Again, this is likely because the feasible region is significantly smaller than that of Case 1. You can see from the data in Table 8.6 that both the lower-bounding problem and the inner program require slightly more effort to solve for Case 2 than for Case 1 but also that the robust simulation algorithm requires an extra iteration for Case

2 than for Case 1.

It is clear from Table 8.6 that Case 3 poses the most difficulty for the algorithm. As can be seen from the performance numbers, the inner programs are taking many more iterations as compared to the other cases. This could be due to many issues but it is most likely due to the ineffectiveness of the interval methods to refine the bounds on the state variables efficiently for this specific case. As a result, Algorithm 4.1 must take many iterations before significant refinement of the variable bounds can occur. That is, the bounds converge very slowly for this case, resulting in very expensive inner programs. Although Case 4 is also considering the same separator dimensions as Case 3, the algorithm performs very differently. Two likely reasons for this have been identified. First, the numerical behavior of the model seems to be much more favorable for the restricted controls. The second reason is that Case 4 is not robustly feasible and the algorithm can verify this on P^1 and subsequently terminate.

8.5 Concluding Remarks

In this chapter, the problem of rigorous worst-case design of subsea production facilities was addressed and solved using the methods developed throughout this thesis. Since this problem was cast as a feasibility problem, it was identified that solving the nonconvex implicit SIP to global optimality may not be necessary. In response, two additional criteria were implemented in order to terminate the implicit SIP algorithm if a rigorous guarantee on feasibility/infeasibility can be deduced.

Due to the complex behavior of process systems models, it was identified early on that the overestimation encountered using interval analysis with these models may be detrimental to calculating useful bounding information for implicit functions. To combat this, an automatic forward-backward constraint propagation technique was implemented to help refine interval bounding information required by the algorithm.

In order to demonstrate the effectiveness and performance of the algorithm on the worst-case design of a subsea separator, a model was developed and four case studies were performed. Overall, the algorithm performed favorably obtaining rigor-

ous guarantees on robust feasibility/infeasibility with relatively little effort for each study.

Chapter 9

Conclusions, Future Work and Opportunities

9.1 Interval Methods

In Chapter 3 the idea of parametric interval methods was discussed and parameterized generalized bisection was introduced. The key contribution was a method for bounding rigorously all solution branches of parameterized systems. In order to verify existence and uniqueness of solution branches within an interval, a sharper existence and uniqueness test was developed since the classical *inclusion* test was difficult and often impossible to satisfy for parameterized systems. However, even with the development of this more effective existence and uniqueness test, there are still numerous examples where existence and uniqueness cannot be verified (the test can never be satisfied) and parameterized generalized bisection is ineffective (e.g. the algorithm returns many interval boxes for which it was unable to determine any information). In addition, these systems in which existence and uniqueness tests fail often exhibit convergence problems when interval Newton-type methods are applied (i.e. as P is partitioned to degeneracy, X does not converge to a degenerate interval).

Future work in this area should focus on developing even more effective existence and uniqueness tests, if they exist, as well as developing a method to identify a priori systems with convergence problems (i.e. interval iterations do not produce degenerate

intervals even as the parameter interval is partitioned to degeneracy). In both Chapter 4 and Chapter 8, parametric interval-Newton was combined with other interval methods to overcome convergence problems that were encountered when parametric interval-Newton was applied on its own. In Chapter 4, both forward-backward constraint propagation as well as the linear-programming contractor method were applied. In Chapter 8, forward-backward constraint propagation was employed alongside parametric interval-Newton. Although forward-backward constraint propagation is extremely inexpensive, the linear-programming contractor method is rather expensive as it requires the solution of $2n_x$ LPs at each iteration of the B&B algorithm. By identifying when convergence problems will be encountered, it may be possible to employ effective interval contraction methods selectively. For instance, the reactor-separator-recycle system in Chapter 4 has regions of the parameter space where parametric interval-Newton is effective on its own but completely ineffective in other regions on its own.

9.2 Relaxations of Implicit Functions

In Chapter 4, methods for constructing convex and concave relaxations of implicit functions were developed. The methods are iterative in nature and rely on the ability to calculate appropriate interval bounds. Similarly, in order to guarantee convergence of the B&B algorithm, the interval bounds must converge to degeneracy as the parameter interval is partitioned to degeneracy. Therefore, relaxations of implicit functions suffer from the same pitfalls identified above.

Interestingly, the number of iterations performed for constructing relaxations has a drastic impact on their tightness for some problems whereas there is no improvement after a single iteration for other problems. It may be worthwhile to explore this behavior further and develop a method to identify a priori (possibly per node) how many iterations will be required to construct tight relaxations. This would ensure that an unnecessary number of iterations aren't being taken or that too few iterations aren't being taken; both scenarios would be very inefficient and negatively affect the

performance of the B&B algorithm.

Analysis of the convergence order and node clustering effect when used within the B&B algorithm [19] may be also worth exploring. It is expected that the convergence order for relaxations of implicit functions is between first- and second-order and that in general, problems solved using the approach in Chapter 4 will suffer from clustering.

Lastly, in Chapter 6, the uniqueness assumptions of Chapter 4 were relaxed. It was shown that the theoretical results surrounding the construction of convex and concave relaxations of implicit functions still hold. Furthermore, in Chapter 8, the idea of an implicit function only existing on part of the parameter space was mentioned. These results are clearly still in their infancy and it would be interesting to further explore these ideas which may be more applicable, in general, to process systems examples.

9.3 Global Optimization of Large Sparse Systems

The purpose of Chapter 5 was primarily to demonstrate the application of Section 4.3.3 on large sparse systems as a "jumping-off point". Using a proper computer implementation of the algorithm, streamlined specifically for large sparse systems, it is expected that the relaxations of Section 4.3.3 will be ideal for solving problems of global optimization of large sparse systems in the general case.

In [101], the author states that for certain problem structures, interval Gauss elimination will yield the hull and will be even more effective than interval Gauss-Seidel in these cases. It is expected that these results translate directly to the construction of relaxations and that the direct approach of [88] will be more effective for constructing tight relaxations than the method in Section 4.3.3 for some problem structures. Since it was shown that the direct approach scales poorly with the dimension of the system, identifying the proper problem structure a priori may allow the user to switch between relaxation techniques when necessary. Furthermore, this may enable the user to determine situations where the added cost of the direct method may be outweighed by the tightness of the relaxations.

9.4 Robust Simulation and Design

In Chapter 8, the robust feasibility problem was solved for the subsea separator model for various scenarios. That is, the question "Given the worst-case realization of uncertainty (from the interval of uncertainty), does there exist a control (from the interval of control values) such that the performance/safety constraint will be satisfied?" Throughout this thesis, only a single performance/safety constraint was considered. In [87], Mitsos briefly comments on how the SIP algorithm could possibly be extended to systems with more than one semi-infinite constraint. It would be of interest to explore this idea and develop a modification to Algorithm 7.1 to efficiently handle multiple implicit semi-infinite constraints. Thus enabling us to address worst-case design and feasibility problems with more than one performance/safety constraint.

As mentioned in the footnotes of Chapter 2, an idea complementary to the idea of the operating envelope is the concept of flexibility. It may also be of interest to address the flexibility of the process and explore what the largest interval of uncertainty can be such that a feasible control still exists. This is what is known as the "inverse problem" and can be formulated as the following program:

$$\begin{aligned} & \max_{\boldsymbol{\delta}\in\mathbb{R}^n} f(\boldsymbol{\delta}) \\ \text{s.t.} \quad & \max_{\mathbf{p}\in P(\boldsymbol{\delta})} g(\mathbf{x}(\mathbf{u},\mathbf{p}),\mathbf{u},\mathbf{p}) \leq 0, \quad \forall \mathbf{u}\in U \end{aligned} \tag{9.1}$$

where f is simply some objective defining how to maximize the effect of the uncertainty growth parameter $\boldsymbol{\delta}$ and the uncertainty interval is now taken as the interval-valued mapping $P : \mathbb{R}^n \to \mathbb{IR}^{n_p}$. This problem is known as a generalized semi-infinite program (GSIP) with implicit functions embedded. Algorithms for solving GSIPs with explicit functions have been discussed previously [79, 133, 140]. The next step is to make the extension from previous works, such as [79], to GSIPs with implicit functions embedded to solve (9.1).

Appendix A

A Note on Selective Branching

The purpose of this appendix is to present the argument that in order for the selective branching algorithm of [38] to incorporate equality constraints, they must be in the form of a parametric linear system. Consider the equality constraints

$$\mathbf{h}(\mathbf{z}, \mathbf{p}) = \mathbf{0},\ (\mathbf{z}, \mathbf{p}) \in X \times P,$$

which can be expressed as the following inequality constraints

$$\begin{aligned} h_i(\mathbf{z}, \mathbf{p}) &\leq 0,\ i = 1, \ldots, n_x \\ -h_i(\mathbf{z}, \mathbf{p}) &\leq 0,\ i = 1, \ldots, n_x. \end{aligned}$$

However, according to [38], each inequality constraint (expressed generically as w) must be able to be expressed as

$$w(\mathbf{z}, \mathbf{p}) = w^A(\mathbf{z}) + \sum_{j \in Q} w_j^B(\mathbf{z}) w_j^C(\mathbf{p}) + w^D(\mathbf{p}) \tag{A.1}$$

where Q is an index set indicating the bilinear interactions between functions of $\mathbf{z}$ and functions of $\mathbf{p}$. The functions w^A and $w_j^B,\ j \in Q$, must be convex on X and the functions $w_j^C,\ j \in Q$, and w^D need to be continuous on P. For the sake of argument, suppose that functions of both $\mathbf{z}$ and $\mathbf{p}$ exist only as bilinear interactions

in the expressions for $\mathbf{h}$. If this is not the case—say if $\mathbf{z}$ and $\mathbf{p}$ are arguments of an intrinsic transcendental function—then the method of [38] may not applicable to this system unless it can be reformulated so its functions fit the form of (A.1). Without loss of generality, choose some $i = 1, \ldots, n_x$. Then the following must hold

$$h_i(\mathbf{z}, \mathbf{p}) = w^A(\mathbf{z}) + \sum_{j \in Q} w_j^B(\mathbf{z}) w_j^C(\mathbf{p}) + w^D(\mathbf{p}) \leq 0$$
$$-h_i(\mathbf{z}, \mathbf{p}) = -w^A(\mathbf{z}) - \sum_{j \in Q} w_j^B(\mathbf{z}) w_j^C(\mathbf{p}) - w^D(\mathbf{p}) \leq 0.$$

By the requirements laid out in [38], both w^A and $-w^A$ must be convex. Therefore w^A must be affine on X. Now consider the middle "bilinear interactions" term. In order to satisfy Conditions W 6 in [38], for each $j \in Q$, either w_j^B is affine or $w_j^C \geq 0$. Suppose we have w_j^B affine. Then $-w_j^B$ is also affine. Now, suppose w_j^B is convex nonaffine, then $w_j^C \geq 0$ on P. Since w_j^B is nonaffine it must be true that $w_j^C \leq 0$ on P since $-w_j^B$ is concave nonaffine and would otherwise violate Conditions W 1 in [38]. Since we have both $w_j^C \geq 0$ and $w_j^C \leq 0$ on P, then $w_j^C = 0$ on P, implying that there can be no bilinear interaction. Therefore, if Q is the index set indicating bilinear interactions between functions of $\mathbf{z}$ and functions of $\mathbf{p}$, then $w_j^B,\ j \in Q$, is affine. Since both $w_j^B,\ j \in Q$, and w^A are affine on X, then the system is in the form of a parametric linear system $\mathbf{A}(\mathbf{p})\mathbf{z} = \mathbf{b}(\mathbf{p})$. This completes the argument.

Appendix B

A Note on Solving Explicit SIPs

In Chapter 7, it was mentioned that the SIP algorithm of [87] could potentially be applied to solve (7.1) without reformulating as an implicit SIP. In that section, it is stated that "this strategy is not advisable since the algorithm would then force the number of variables in the upper- and lower-bounding subproblems to increase with each iteration." Here, we demonstrate this result. Again, the lower-bounding problem is the result of reducing the SIP to an NLP by considering only a finite number of constraints corresponding to specific realizations of $\mathbf{p}$. Here, each of these realizations will be denoted as $\mathbf{p}_i$ with $i \in I^{LBD}$, where I^{LBD} is a finite index set. This is analogous to the notation in Chapter 7 where P^{LBD} was a finite set whose elements correspond to specific realizations of $\mathbf{p}$. Then, for each $\mathbf{p}_i,\ i \in I^{LBD}$, there is a corresponding $\mathbf{z}_i$ that is dependent on $\mathbf{y}$. Thus, the lower-bounding problem can be stated as:

$$\begin{aligned} f^{LBD} = \quad & \min_{\mathbf{y},\mathbf{z}_i,\ i\in I^{LBD}} f(\mathbf{y}) \\ & \text{s.t. } g(\mathbf{z}_i, \mathbf{y}, \mathbf{p}_i) \leq 0,\ \forall i \in I^{LBD} \\ & \quad\ \mathbf{h}(\mathbf{z}_i, \mathbf{y}, \mathbf{p}_i) = \mathbf{0},\ \forall i \in I^{LBD}. \end{aligned}$$

From this formulation, it is clear to see how as I^{LBD} grows with each iteration, so does the number of variables that the optimizer sees. Although the inner-program is

not plagued with this issue, the upper-bounding problem is as well. Thus, at each iteration of the SIP algorithm, three global optimization subproblems must be solved; two of which become increasingly larger and therefore prohibitively more expensive to solve.

Appendix C

Kinetic Mechanism Experimental Data

t^i	I_d^i	t^i	I_d^i	t^i	I_d^i	t^i	I_d^i	t^i	I_d^i
0.01	66.0952	0.41	59.619	0.81	28.4429	1.21	18.1524	1.61	12.4238
0.02	104.762	0.42	58.2857	0.82	28.3476	1.22	18.1952	1.62	12.5143
0.03	110.333	0.43	57.4762	0.83	27.5429	1.23	17.8476	1.63	12.9143
0.04	114.905	0.44	56.4762	0.84	27.4333	1.24	17.9095	1.64	12.5714
0.05	122.238	0.45	55.8095	0.85	27.6048	1.25	17.5048	1.65	13.3667
0.06	125.429	0.46	54.5238	0.86	27.1762	1.26	17.500	1.66	13.2286
0.07	125.429	0.47	53.000	0.87	27.200	1.27	15.9619	1.67	13.7905
0.08	123.476	0.48	51.8571	0.88	26.4333	1.28	16.2095	1.68	13.7571
0.09	121.286	0.49	50.4286	0.89	25.7619	1.29	16.181	1.69	13.5905
0.10	118.857	0.50	49.381	0.90	24.8095	1.30	15.6952	1.70	12.9667
0.11	117.667	0.51	47.9524	0.91	24.7429	1.31	15.7095	1.71	12.981
0.12	116.143	0.52	47.3714	0.92	24.2857	1.32	15.4619	1.72	12.8857
0.13	113.857	0.53	46.8952	0.93	24.1714	1.33	15.9476	1.73	12.919
0.14	111.571	0.54	46.4857	0.94	23.5667	1.34	16.000	1.74	13.0143
0.15	108.81	0.55	45.9048	0.95	23.5476	1.35	16.1952	1.75	13.0095
0.16	105.952	0.56	45.0762	0.96	23.3952	1.36	16.1143	1.76	12.3857
0.17	104.048	0.57	44.3238	0.97	22.919	1.37	15.7429	1.77	12.5571
0.18	102.048	0.58	43.4143	0.98	22.3095	1.38	15.5762	1.78	12.3429
0.19	100.143	0.59	43.5429	0.99	21.8048	1.39	15.7048	1.79	12.7571
0.20	98.5238	0.60	42.3619	1.00	21.2857	1.40	15.8095	1.80	12.681
0.21	96.2381	0.61	41.8381	1.01	21.2048	1.41	15.6667	1.81	12.5429
0.22	94.381	0.62	40.2381	1.02	20.8429	1.42	14.9048	1.82	12.1857
0.23	91.6667	0.63	39.1286	1.03	20.4429	1.43	14.5857	1.83	12.7905
0.24	89.5714	0.64	38.7857	1.04	20.0048	1.44	14.7524	1.84	12.5571
0.25	87.1429	0.65	37.081	1.05	19.9381	1.45	14.7571	1.85	12.8429
0.26	84.8571	0.66	36.9524	1.06	19.500	1.46	14.9762	1.86	12.5476
0.27	83.4286	0.67	36.581	1.07	19.8667	1.47	14.5333	1.87	12.5714
0.28	81.1905	0.68	36.281	1.08	18.9333	1.48	14.5524	1.88	12.3762
0.29	78.9048	0.69	35.3476	1.09	19.1381	1.49	14.0143	1.89	11.9952
0.30	77.0476	0.70	34.8905	1.10	18.9619	1.50	13.6286	1.90	11.4571
0.31	75.4762	0.71	34.1667	1.11	18.5476	1.51	13.4429	1.91	11.300
0.32	73.4762	0.72	33.6714	1.12	17.9048	1.52	13.4667	1.92	11.1524
0.33	71.8095	0.73	32.9667	1.13	17.7571	1.53	13.319	1.93	11.681
0.34	70.6667	0.74	31.8429	1.14	18.5333	1.54	12.9333	1.94	11.619
0.35	68.381	0.75	31.5429	1.15	18.3762	1.55	13.1238	1.95	11.9048
0.36	67.3333	0.76	31.1476	1.16	18.3571	1.56	12.7476	1.96	12.000
0.37	65.0952	0.77	30.9905	1.17	18.3286	1.57	12.9333	1.97	12.0762
0.38	63.7143	0.78	29.9571	1.18	18.2762	1.58	13.0714	1.98	11.9143
0.39	62.0476	0.79	29.1333	1.19	18.3952	1.59	13.0714	1.99	11.7619
0.40	60.8571	0.80	28.7857	1.20	17.5952	1.60	12.7619	2.00	11.5333

Table C.1: Experimental data for the kinetic mechanism example.

Bibliography

[1] G. S. Abdoulaev, K. Ren, and A. H. Hielscher. Optical tomography as a PDE-constrained optimization problem. *Inverse Problems*, 21:1507, 2005.

[2] C. S. Adjiman and C. A. Floudas. Rigorous convex underestimators for general twice-differentiable problems. *Journal of Global Optimization*, 9:23–40, 1996.

[3] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, Inc., New York, 1983.

[4] J. C. Alexander and J. A. Yorke. The homotopy continuation method: numerically implementable topological procedures. *Transactions of the American Mathematical Society*, 242:271–284, 1978.

[5] A. Baharev, L. Kolev, and E. Rév. Computing multiple steady states in homogeneous azeotropic and ideal two-product distillation. *AIChE Journal*, 57(6):1485–1495, June 2011.

[6] S. Balendra and I. D. L. Bogle. Modular global optimisation in chemical engineering. *Journal of Global Optimization*, 45(1):169–185, 2009.

[7] K. J. Beers. *Numerical Methods for Chemical Engineering*. Cambridge University Press, Cambridge, 2007.

[8] R. E. Bellman. *Dynamic Programming*. Princeton University Press, New Jersey, 1957.

[9] R. E. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[10] R. E. Bellman and L. A. Zadeh. Decision-making in a fuzzy environment. *Management Science*, 17:141–161, 1970.

[11] A. Ben-Tal and A. Nemirovski. Robust optimization - methodology and applications. *Math. Program*, Ser. B(92):453–480, Feb 2002.

[12] C. Bendtsen and O.Stauning. FADBAD, a flexible C++ package for automatic differentiation. Technical Report 1996-x5-94, Technical University of Denmark, 1996.

[13] B. Bhattacharjee, W. H. Green Jr., and P. I. Barton. Interval methods for semi-infinite programs. *Comp. Opt. and App.*, 30:63–93, 2005.

[14] B. Bhattacharjee, P. Lemonidis, W. H. Green Jr., and P. I. Barton. Global solution of semi-infinite programs. *Math. Program*, 103:283–307, 2005.

[15] C.G. Bianco and A. Piazzi. A hybrid algorithm for infinitely constrained optimization. *International Journal of Systems Science*, 32(1):91–102, 2001.

[16] D. Biello. One year after BP oil spill, at least 1.1 million barrels still missing. *Scientific American*, April 25 2011.

[17] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, Heidelberg, second edition, 2011.

[18] J. W. Blankenship and J. E. Falk. Infinitely constrained optimization problems. *J. Optim. Theory and App.*, 19(2):261–281, June 1976.

[19] A. Bompadre and A. Mitsos. Convergence rate of McCormick relaxations. *Journal of Global Optimization*, 52(1):1–28, 2012.

[20] I. Bouchouev and V. Isakov. Uniqueness, stability and numerical methods for the inverse problem that arises in financial markets. *Inverse Problems*, 15:95–116, 1999.

[21] BP oil spill seriously harmed deep-sea corals, scientists warn. *The Guardian*, March 26 2012.

[22] H. Bronnimann, G. Melquiond, and S. Pion. The design of the Boost interval arithmetic library. *Theoretical Computer Science*, 351(1):111–118, 2006.

[23] James V. Burke and Michael L. Overton. Variational analysis of non-Lipschitz spectral functions. *Mathematical Programming*, 90:317–351, 2001.

[24] R. P. Byrne and I. D. L. Bogle. Global optimization of constrained non-convex programs using reformulation and interval analysis. *Computers & Chemical Engineering*, 23:1341–1350, 1999.

[25] R. P. Byrne and I. D. L. Bogle. Global optimization of modular process flowsheets. *Ind. Eng. Chem. Res.*, 39:4296–4301, 2000.

[26] B. Chachuat. MC++: A versatile library for McCormick relaxations and Taylor models. http://www3.imperial.ac.uk/people/b.chachuat/research.

[27] B. Chandran and H. Balakrishnan. A dynamic programming algorithm for robust runway scheduling. *Proceedings of the 2007 American Control Conference*, pages 1161–1166, July 2007.

[28] S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers*. McGraw-Hill, New York, fifth edition, 2006.

[29] C. Chatfield. Model uncertainty, data mining and statistical inference. *J. Royal Statistical Society*, 158(3):419–466, 1995.

[30] L. Cheng, E. Subrahmanian, and A. W. Westerberg. Design and planning under uncertainty: Issues on problem formulation and solution. *Comp. and Chem. Eng.*, 27:781–801, 2003.

[31] H. N. Cofer and M. A. Stadtherr. Reliability of iterative linear equation solvers in chemical process simulation. *Comp. and Chem. Eng.*, 20(9):1123–1132, 1996.

[32] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings 24th National Conference of the Association for Computing Machinery*, New Jersey, 1969. Brandon Press.

[33] D. Draper. Assessment and propagation of model uncertainty. *J. Royal Statistical Society*, 57(1):45–97, 1995.

[34] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford Scientific Publications, Oxford, 1992.

[35] B. Dupire. Pricing with a smile. *Risk*, 7:18–20, 1994.

[36] H. Egger and H. W. Engl. Tikhonov regularization applied to the inverse problem of option pricing: convergence analysis and rates. *Inverse Problems*, 21:1027, 2005.

[37] L. El Ghaoui and Hervé Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, 1997.

[38] T. G. Epperly and E. N. Pistikopoulos. A reduced space branch and bound algorithm for global optimization. *Journal of Global Optimization*, 11:287–311, 1997.

[39] W. R. Esposito and C. A. Floudas. Global optimization for the parameter estimation of differential-algebraic systems. *Industrial and Engineering Chemical Research*, 39:1291–1310, 2000.

[40] F. Facchinei and J. S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems*. Springer, New York, 2003.

[41] J. E. Falk and K. Hoffman. A nonconvex max-min problem. *Naval Research Logistics Quarterly*, 24(3):441–450, 1977.

[42] J. E. Falk and R. M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15(9):550–569, 1969.

[43] C. A. Floudas, Z. H. Gumus, and M. G. Ierapetritou. Global optimization in design under uncertainty: Feasibility test and flexibility index problems. *Ind. Eng. Chem. Res.*, 40(20):4267–4282, 2001.

[44] C. A. Floudas and O. Stein. The adaptive convexification algorithm: A feasible point method for semi-infinite programming. *SIAM J. Optim.*, 18(4):1187–1208, 2007.

[45] Fukushima faced 14-metre tsunami. *World Nuclear News*, March 23 2011.

[46] D. M. Gay. Perturbation bounds for nonlinear equations. *SIAM Journal on Numerical Analysis*, 18(4):654–663, 1981.

[47] D. M. Gay. Computing perturbation bounds for nonlinear algebraic equations. *SIAM Journal on Numerical Analysis*, 20(3):638–651, 1983.

[48] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[49] A. Griewank and A. Walther. *Evaluating Derivatives*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, January 2008.

[50] Andreas Griewank and George F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, Philadelphia, 1992.

[51] I. E. Grossmann and R. W. H. Sargent. Optimum design of chemical plants with uncertain parameters. *AIChE Journal*, 24(6):1021–1028, November 1978.

[52] K. P. Halemane and I. E. Grossmann. Optimal process design under uncertainty. *AIChE Journal*, 29(3):425–433, May 1983.

[53] E. R. Hansen. A globally convergent interval method for computing and bounding real roots. *BIT*, 18:415–424, 1978.

[54] E. R. Hansen and R. I. Greenberg. Interval Newton methods. *Applied Mathematics and Computation*, 12:89–98, 1983.

[55] E. R. Hansen and S. Sengupta. Bounding solutions of systems of equations using interval analysis. *BIT*, 21:203–211, 1981.

[56] E. R. Hansen and G. W. Walster. *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, second edition, 2004.

[57] J. Herron. No closure yet on BP oil spill costs. *The Wall Street Journal*, November 2 2010.

[58] R. Hettich and K. O. Kortanek. Semi-infinite programming: Theory, methods, and applications. *SIAM Review*, 35(3):380–429, September 1993.

[59] R. Horst. A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization. *Journal of Optimization Theory and Applications*, 51(2):271–291, November 1986.

[60] R. Horst. Deterministic global optimization with partition sets whose feasibility is not known: Application to concave minimization, reverse convex constraints, DC-programming, and Lipschitzian optimization. *Journal of Optimization Theory and Applications*, 58(1):11–37, July 1988.

[61] R. Horst and N. V. Thoai. Conical algorithm for the global minimization of linearly constrained decomposable concave minimization problems. *Journal of Optimization Theory and Applications*, 74:469–486, 1992.

[62] R. Horst and N. V. Thoai. Constraint decomposition algorithms in global optimization. *Journal of Global Optimization*, 5:333–348, 1994.

[63] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer, Berlin, third edition, 1996.

[64] M. G. Ierapetritou, J. Acevedo, and E. N. Pistikopoulos. An optimization approach for process engineering problems under uncertainty. *Comp. and Chem. Eng.*, 20(6-7):703–709, 1996.

[65] G. C. Itle, A. G. Salinger, R. P. Pawlowski, J. N. Shadid, and L. T. Biegler. A tailored optimization strategy for PDE-based design: application to a CVD reactor. *Computers and Chemical Engineering*, 28(3):291–302, 2004.

[66] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer-Verlag, London, 2001.

[67] P. Kall and S. W. Wallace. *Stochastic Programming*. John Wiley & Sons, Chichester, second edition, 1994.

[68] R. B. Kearfott. Abstract generalized bisection and a cost bound. *Mathematics of Computation*, 49(179):187–202, 1987.

[69] R. B. Kearfott. Preconditioners for the interval Gauss-Seidel method. *SIAM Journal on Numerical Analysis*, 27(3):804–822, 1990.

[70] R. B. Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, Boston, 1996.

[71] D. S. Kirby and O. Fiksen. A dynamic optimisation model for the behaviour of tunas at ocean fronts. *Fisheries Oceanography*, 9:328–342, 2000.

[72] O. Knüppel. PROFIL/BIAS—a fast interval library. *Computing*, 53(3-4):277–287, September 1994.

[73] A. C. Kokossis and C. A. Floudas. Synthesis of isothermal reactor-separator-recycle systems. *Chemical Engineering Science*, 46:1361–1383, 1991.

[74] L. V. Kolev. Automatic computation of a linear interval enclosure. *Reliable Computing*, 7:17–28, 2001.

[75] L. V. Kolev and I. P. Nenov. Cheap and tight bounds on the solution set of perturbed systems of nonlinear equations. *Reliable Computing*, 7:399–408, 2001.

[76] R. Krawczyk. Newton-algorithmen zur bestimmung con nullstellen mit fehlerschranken. *Computing*, 4:187–201, 1969.

[77] R. Krawczyk. Interval iterations for including a set of solutions. *Computing*, 32:13–31, 1984.

[78] B. M. Kwak and Jr. E. J. Haug. Optimum design in the presence of parametric uncertainty. *Journal of Optimization Theory and Applications*, 19(4):527–546, 1976.

[79] P. Lemonidis. *Global Optimization Algorithms for Semi-Infinite and Generalized Semi-Infinite Programs*. PhD thesis, MIT, 2008.

[80] Y. Lin and Mark A. Stadtherr. Advances in interval methods for deterministic global optimization in chemical engineering. *Journal of Global Optimization*, 29(3):281–296, July 2004.

[81] A. G. Little. Pumped up: Chevron drills down 30,000 feet to tap oil-rich Gulf of Mexico. *Wired Magazine*, 15(9), October 2007.

[82] M. Lopez and G. Still. Semi-infinite programming. *European Journal of Op. Res.*, 180:491–518, 2007.

[83] L. Luksan and J. Vlcek. Algorithms for non-differentiable optimization. *ACM Transactions on Mathematical Software*, 27(2):193–213, June 2001.

[84] R. K. Malik and R. R. Hughes. Optimal design of flexible chemical processes. *Computers and Chemical Engineering*, 3:473–485, 1979.

[85] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I-convex underestimating problems. *Math. Program*, 10:147–175, 1976.

[86] C. Miranda. Un'osservatione su un teorema di brower. *Boll. Un. Mat. Ital.*, 3:5–7, 1940.

[87] A. Mitsos. Global optimization of semi-infinite programs via restriction of the right-hand side. *Optimization*, 60(10-11):1291–1308, 2011.

[88] A. Mitsos, B. Chachuat, and P. I. Barton. McCormick-based relaxations of algorithms. *SIAM J. Optim.*, 20(2):573–601, December 2009.

[89] A. Mitsos, P. Lemonidis, and P. I. Barton. Global solution of bilevel programs with a nonconvex inner program. *Journal of Global Optimization*, 42(4):475–513, 2008.

[90] A. Mitsos, P. Lemonidis, C. K. Lee, and P. I. Barton. Relaxation-based bounds for semi-infinite programs. *SIAM J. Optim*, 19(1):77–113, February 2008.

[91] R. E. Moore. A test for existence of solutions to nonlinear systems. *SIAM Journal on Numerical Analysis*, 14(4):611–615, 1977.

[92] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.

[93] R. E. Moore and J. B. Kioustelidis. A simple test for accuracy of approximate solutions to nonlinear (or linear) systems. *SIAM Journal on Numerical Analysis*, 17(4):521–529, 1980.

[94] A. P. Morgan. A method for computing all solutions to systems of polynomials equations. *ACM Transactions on Mathematical Software*, 9(1):1–17, 1983.

[95] J. M. Mulvey, R. J. Vanderbei, and S. A. Zenios. Robust optimization of large-scale systems. *Operations Research*, 43(2):264–281, 1995.

[96] J. R. Munkres. *Analysis On Manifolds*. Westview Press, Boulder, CO, 1991.

[97] B. A. Murtagh and M. A. Saunders. MINOS 5.5 user's guide. Technical report, Stanford University, 1983.

[98] L. D. Muu and W. Oettli. Combined branch-and-bound and cutting plane methods for solving a class of nonlinear programming problems. *Journal of Global Optimization*, 3:377–391, 1993.

[99] A. Neumaier. *The enclosure of solutions of parameter dependent systems*, pages 269–286. Reliability in Computing. Academic Press, Inc., San Diego, CA, 1988.

[100] A. Neumaier. Rigorous sensitivity analysis for parameter-dependent systems of equations. *Journal of Mathematical Analysis and Applications*, 144:16–25, 1989.

[101] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.

[102] N. Nishida, A. Ichikawa, and E. Tazaki. Synthesis of optimal process systems with uncertainty. *Ind. Eng. Chem., Process Des. Develop.*, 13(3):209–214, 1974.

[103] Deepwater Gulf of Mexico 2009: Interim report of 2009 highlights. 2009. OCS Report MS2009-016, US Dept. of the Interior, Minerals Management Service.

[104] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Computer Science and Applied Mathematics. Academic Press, Inc., Boston, 1970.

[105] A. T. Phillips and J. B. Rosen. A parallel algorithm for constrained concave quadratic global optimization. *Mathematical Programming*, 42:421–448, 1988.

[106] E. D. Popova. *On the solution of parametrised linear systems*, pages 127–138. Scientific Computing Validated Numerics Interval Methods. Kluwer Academic Publishers, Boston, 2001.

[107] E. D. Popova. Parametric interval linear solver. *Numerical Algorithms*, 37:345–356, 2004.

[108] E. D. Popova and W. Kraämer. Parametric fixed-point iteration implemented in C-XSC. Technical report, Wissenschaftliches Rechnen/Softwaretechnologie, 2003.

[109] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Mathematics and its Applications. Ellis Horwood Limited, 1984.

[110] R. Reemtsen and S. Görner. *Numerical Methods for Semi-Infinite Programming: A Survey*. Nonconvex Optimization and Its Applications: Semi-Infinite Programming. Kluwer Academic Publishers, Boston, 1998.

[111] Rembert Reemtsen and Jan-J. Rückmann, editors. *Nonconvex Optimization and Its Applications: Semi-Infinite Programming*. Kluwer Academic Publishers, Boston, 1998.

[112] RES Group. *JACOBIAN Process Simulator*. www.openbio.resgroupsoftware.com/jacobian.html.

[113] W. C. Rheinboldt. Computation of critical boundaries on equilibrium manifolds. *SIAM Journal on Numerical Analysis*, 19(3):653–669, 1982.

[114] W. C. Rheinboldt. On the computation of multi-dimensional solution manifolds of parametrized equations. *Numerische Mathematik*, 53:165–181, 1988.

[115] W. C. Rheinboldt and J. V. Burkardt. A locally parameterized continuation process. *ACM Transactions on Mathematical Software*, 9(2):215–235, 1983.

[116] R. E. Rosenthal. *GAMS–A user's manual*. GAMS Development Corp., Washington DC, January 2012.

[117] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, New York, third edition, 1976.

[118] S.M. Rump. Rigorous sensitivity analysis for systems of linear and nonlinear equations. *Mathematics of Computation*, 54(190):721–736, 1990.

[119] S.M. Rump. Verification methods for dense and sparse systems of equations. In *Topics in Validated Compuations*, Topics in Validated Compuations, pages 63–136. Elsevier, 1994.

[120] H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, March 1996.

[121] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2nd edition, 2003.

[122] N. V. Sahinidis. Optimization under uncertainty: state-of-the-art and opportunities. *Comp. and Chem. Eng.*, 28(6-7):971–983, 2004.

[123] A. G. Salinger, R. P. Pawlowski, J. N. Shadid, B. van Bloemen Waanders, R. Bartlett, G. C. Itle, and L. T. Biegler. Optimization in large-scale reacting flows using MPSalsa and sequential quadratic programming. In *Large-scale PDE-constrained optimization*, Lecture Notes in Computational Science and Engineering, Heidelberg, 2003. Springer-Verlag.

[124] M. Salloum, R. Ma, and L. Zhu. Enhancement in treatment planning for magnetic nanoparticle hyperthermia: Optimization of the heat absorption pattern. *Int. J. Hyperthermia*, 25(4):309–321, 2009.

[125] H. Schichl and A. Neumaier. Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization*, 33(4):541–562, December 2005.

[126] S. Scholtes. Introduction to piecewise differentiable equations. Technical report, University of Karlsruhe, 1994. Habilitation Thesis.

[127] J. K. Scott. *Reachability Analysis and Deterministic Global Optimization of Differential-Algebraic Systems*. PhD thesis, MIT, 2012.

[128] J. K. Scott, M. D. Stuber, and P. I. Barton. Generalized McCormick relaxations. *Journal of Global Optimization*, 51(4):569–606, 2011.

[129] R. Seydel. *Practical Bifurcation and Stability Analysis*. Interdisciplinary Applied Mathematics. Springer-Verlag, New York, second edition, 1994.

[130] A. B. Singer. *Global Dynamic Optimization*. PhD thesis, Massachusetts Institute of Technology, 2004.

[131] B. Srinivasan, D. Bonvin, E. Visser, and A. Palanki. Dynamic optimization of batch processes II. Role of measurements in handling uncertainty. *Comp. and Chem. Eng.*, 27:27–44, 2002.

[132] O. Stein and G. Still. Solving semi-infinite optimization problems with interior point techniques. *SIAM J. Control Optim.*, 42(3):769–788, 2003.

[133] G. Still. Generalized semi-infinite programming: Theory and methods. *European Journal of Op. Res.*, 119(2):301–313, December 1999.

[134] M. D. Stuber and P. I. Barton. Robust simulation and design using semi-infinite programs with implicit functions. *Int. J. of Reliability and Safety*, 5(3/4):378–397, 2011.

[135] U. Rashid Sumaila, Andres M. Cisneros-Montemayor, Andrew Dyck, Ling Huang, William Cheung, Jennifer Jacquet, Kristin Kleisner, Vicky Lam, Ashley McCrea-Strub, Wilf Swartz, Reg Watson, Dirk Zeller, and Daniel Pauly. Impact of the Deepwater Horizon well blowout on the economics of US Gulf fisheries. *Can. J. Fish. Aquat. Sci.*, 69:499–510, 2012.

[136] R. E. Swaney and I. E. Grossmann. An index for operational flexibility in chemical process design. Part I: Formulation and theory. *AIChE Journal*, 31(4):621–630, April 1985.

[137] R. E. Swaney and I. E. Grossmann. An index for operational flexibility in chemical process design. Part II: Computational algorithms. *AIChE Journal*, 31(4):631–641, April 1985.

[138] M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005.

[139] A. Tsoukalas, B. Rustem, and E. N. Pistikopoulos. A global optimization algorithm for generalized semi-infinite, continuous minimax with couples constraints and bilevel programs. *Journal of Global Optimization*, 44:235–250, 2009.

[140] F. Guerra Vazquez, J.-J. Rückmann, O. Stein, and G. Still. Generalized semi-infinite programming: A tutorial. *Journal of Computational and Applied Mathematics*, 217:394–419, 2008.

[141] X. H. Vu, H. Schichl, and D. Sam-Haroud. Interval propagation and search on directed acyclic graphs for numerical constraint solving. *Journal of Global Optimization*, 45(4):499–531, December 2009.

[142] C. Yalcin and A. W. Stott. Dynamic programming to investigate financial impacts of mastitis control decisions in milk production systems. *Journal of Dairy Research*, 67:515–528, 2000.

[143] C. L. Yaws, P. K. Narasimhan, and C. Gabbula. *Yaws' Handbook of Antoine Coefficients for Vapor Pressure*. Knovel, second electronic edition, 2009.

[144] D.P. Young, W.P. Huffman, R.G. Melvin, C.L Hilmes, and F.T. Johnson. Nonlinear elimination in aerodynamic analysis and design optimization. In *Large-scale PDE-constrained optimization*, Lecture Notes in Computational Science and Engineering, Heidelberg, 2003. Springer-Verlag.

[145] S. Zuhe, A. Neumaier, and M. C. Eiermann. Solving minimax problems by interval methods. *BIT Numerical Mathematics*, 30:742–751, 1990.