CARNEGIE MELLON UNIVERSITY

An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering

A Dissertation

Submitted to the Graduate School

in Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

in

Chemical Engineering

by

Andreas Wächter

Pittsburgh, Pennsylvania

January 29, 2002

Abstract

Nonlinear programming (NLP) has become an essential tool in process engineering, leading to profit gains through improved plant designs and better control strategies. The rapid advance in computer technology enables engineers to consider increasingly complex systems, where existing optimization codes reach their practical limits. The objective of this dissertation is the design, analysis, implementation, and evaluation of a new NLP algorithm that is able to overcome the current bottlenecks, particularly in the area of process engineering.

The proposed algorithm follows an interior point approach, thereby avoiding the combinatorial complexity of identifying the active constraints. Emphasis is laid on flexibility in the computation of search directions, which allows the tailoring of the method to individual applications and is mandatory for the solution of very large problems. In a full-space version the method can be used as general purpose NLP solver, for example in modeling environments such as AMPL. The reduced space version, based on coordinate decomposition, makes it possible to tailor linear algebra work to particular problem structures, such as those arising in dynamic optimization, or even to re-use existing simulation software. If second derivatives are available, they can be exploited explicitly, or otherwise approximated by quasi-Newton methods. In addition, as a compromise between those two options, a conjugate gradient method is implemented for the computation of the tangential step, for which two preconditioners are proposed. This makes the reduced space approach attractive even for problems with many degrees of freedom. Global convergence is enforced by a novel filter line search procedure, which aims to improve efficiency and robustness over traditional merit function approaches.

The discussion of the theoretical aspects includes a detailed analysis of a new type of global convergence failure inherent to many current interior point NLP solvers.

Global convergence of the filter line search method, introduced to overcome this convergence problem, is proven under mild assumptions. It is further shown that the Maratos effect can be avoided in this framework by second order correction steps, which enables fast local convergence.

The practical performance of the proposed method as a general purpose NLP solver is tested on a large variety of NLP test problems. Among the implemented line search methods, the new filter approach seems superior to those based on merit functions. The new optimization code also compares favorably with two other recently developed interior point NLP solvers.

The potential of the new method in the area of process engineering is demonstrated on two dynamic optimization examples with different characteristics, which are solved using orthogonal collocation in a new implementation of the elemental decomposition. Here, the effectiveness of the individual strategies to handle second derivative information is compared. The results confirm that the best choice depends on the particular application. The largest instance addressed is a model with 70 differential and 356 algebraic equations, discretized over $1,500$ elements. The resulting NLP with more than 2 million variables and $4,500$ degrees of freedom is solved in less than 7 hours on a Linux workstation.

Gewidmet meinen Eltern Christa und Theo Wächter,

für ihre Liebe und rückhaltlose Unterstützung.

Acknowledgments

I am extremely grateful to have had the privilege of working with my advisor Larry Biegler, who was a true *Doktorvater* to me. Larry is an excellent and knowledgeable teacher, an inexhaustible fountain of ideas, and an inspiring and open mentor, who gave me the optimal balance of guidance and freedom. I thank him for his immense patience, his caring and generous support, and for always having an open ear for my questions and concerns. I admire his seemingly endless energy, friendliness, and humor, which made the work with him a joyful experience.

My appreciation is also expressed to the remaining members of my committee, Ignacio Grossmann, David Sholl, and Reha Tütüncü for their valuable comments on this dissertation. I would like to thank Reha also for his time in interesting math discussions and for taking the effort of teaching me the basics of interior point methods in a reading course.

I am very indebted to Jorge Nocedal, who became a very encouraging mentor to me, and who introduced me into the math programming crowd. I thank him for his friendship, his warm hospitality during my visits at Northwestern, as well as his valuable advice and support in many situations. Through numerous intriguing discussions, his contagious optimism together with his amazing ability to always find new important questions from unexpected viewpoints had a significant and inspiring influence on this work.

I would like to thank my research group mates and the other department members for keeping the atmosphere "at work" very pleasant and friendly. Particularly, I

would like to express my gratitude to those with whom I worked closely, Roscoe Bartlett, Arturo Cervantes, Yi-Dong Lang, and Arvind Raghunathan, for the fruitful discussions and pleasant conversations, and for the joint projects that contributed to this dissertation.

Further I would like to acknowledge the following people who in various ways have contributed to the success of this dissertation: Andrea Walther and Olaf Vogel for their extensive and fast support with ADOL-C; Richard Waltz for providing KNITRO and discussing aspects of its implementation; Hande Benson and Robert Vanderbei for their help regarding their AMPL models and LOQO; Jose Luis Morales for sharing with me his PERL scripts for a convenient analysis of the test runs; Jorge Moré for discussing subtle issues regarding TRON and for providing its latest version; Richard Byrd and Marcelo Marazzi for inspiring conversations regarding convergence failures and filter methods; Anders Forsgren and Göran Sporre for interesting discussions about counter examples during a memorable visit to Stockholm; Caroline Sainvitu and Philippe Toint for their valuable feedback on the convergence proofs of the filter method; and Robert Grosch for his help on the air separation model.

Of course, my life during the past four and a half years would have never been this exciting and joyful without all the wonderful people that I met in Pittsburgh, Chicago, and other places (You know who you are!!!), and without my dear German friends back home, who despite my annoyingly slow email responses did not forget me (Ihr wißt auch, wer Ihr seid!!!). My deepest gratitude for your friendship!

A very special Thank You to John, Melli, and Birgit, for being there, and for being the way you are. A big Thank You also to Ann, Bill, and Joan; you know for what.

Last, but not at all least, I would like to thank my parents and my sister, who never stopped believing in me. Without your love and support the following pages would not exist.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Many phenomena occurring in industrial chemical plants can be described in terms of mathematical expressions, such as algebraic or differential equations. With the advances in computer technology and the invention of numerical methods for solving these models, it became possible to accurately predict the efficiency of a new plant design or the effect of new control strategies. This offers a very powerful instrument to process engineers, who may want to evaluate the usability and benefits of their ideas in a theoretical way before realizing them; in a safe, fast, and inexpensive manner. As a consequence, process engineering has become a very important discipline within chemical engineering over the past 30 years, avoiding the necessity of expensive pilot plants and yielding profit increases by improving existing processes.

Naturally extending the work of process simulation, engineers soon asked the question how certain parameters in a plant (e.g. equipment size) or controllable quantities (e.g. flow rates) should be chosen in order to optimize some objective. For example, one may want to reduce building costs by avoiding over-sizing certain equipment or to minimize the deviation from given specifications by finding an optimal control strategy. This optimization approach is more than just the automated execution of several simulations for different options that an engineer might have tried.

It can indeed lead to optimal solutions that at first sight seem counter-intuitive, and in this way offer new insight into the underlying chemical process.

Over the past 50 years, many numerical methods have been proposed to solve the resulting mathematical optimization problems and proved efficient in practice. However, the rapid advance of computer technology over the past decades enabled engineers and researchers to consider increasingly larger applications in increasingly detailed representation, leading to larger mathematical problem formulations. As a consequence, existing optimization algorithms and their software implementations keep reaching their practical limits, and new methods have to be devised that try to overcome the bottlenecks of the existing ones.

1.2 Problem Statement

Depending on the particular type of process engineering application, the mathematical optimization problem has different forms. Most of the quantities or variables appearing in process models are *continuous*, such as temperature, flow rates, etc. Even though in some cases, *discrete* decisions have to be made, e.g. whether to build a certain equipment in a new plant or not, only problem formulations with continuous variables will be addressed in this dissertation.

The relationships between the quantities in process models are expressed by equations, which are often nonlinear. In addition, many of the variables have to stay within certain intervals, either to ensure physical meaningfulness (e.g. mass has to be positive), or to guarantee certain operational constraints (e.g. maximum reactor temperature).

Therefore, this dissertation addresses the solution of nonconvex, nonlinear, continuous and smooth optimization problems, also called *Nonlinear Programs (NLPs)*,

which will be assumed to be stated in the following way:

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{1.1a}$$

$$\text{s.t.} \quad c(x) = 0 \tag{1.1b}$$

$$x_L \leq x \leq x_U. \tag{1.1c}$$

The objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and the equality constraints $c : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ with $m < n$ are assumed to be sufficiently smooth. $x_L \in (\mathbb{R}^n \cup \{-\infty\})^n$ and $x_U \in (\mathbb{R}^n \cup \{\infty\})^n$ are the lower and upper bounds on the optimization variables. By allowing infinite bounds we account for the option that some components $x^{(i)}$ can have no lower and/or upper bound.

A more general formulation of an NLP might include nonlinear inequality constraints such as "$d(x) \leq 0$" instead of only simple bounds on variables (1.1c). However, since nonlinear inequality constraints can be reformulated as equality constraints by adding slack variables to the problem statement, and since most inequality constraints in the application classes considered in this dissertation are indeed bounds on variables, we will focus on formulation (1.1).

Many numerical algorithms for the solution of NLP (1.1) have been developed and implemented over the past 50 years. Driven by the fact that larger problem formulations allow a more detailed and accurate representation of reality, increasingly larger instances were addressed, facilitated by the drastic advances in computer technology. During this development, bottlenecks of existing methods became apparent. For example, the currently most popular algorithms, SQP methods, require the explicit identification of variable bounds (1.1c) that are active at the solution, an NP-hard combinatorial problem. This can potentially lead to dramatic increase of computation time as the problem size grows. Furthermore, new related mathematical tools have been developed, for example the technique of automatic differentiation for the efficient and convenient computation of derivatives of f and c. Therefore, there is still a need for the development of new optimization methods, aiming to overcome bottlenecks and to exploit new advances. In addition, for the efficient solution par-

ticularly of very large problems it is important to tailor the methods at least in part to characteristics of the optimization problem. In the next section the characteristics for certain classes of process engineering applications will be discussed.

1.3 Challenges

Nonlinear programs (1.1) arise in process engineering for example in the following classes of applications.

- *Flowsheet Optimization:* Optimize the design or operating conditions of a plant in steady state, i.e. it is assumed that there is no change over time. Here, the variables x usually correspond to physical quantities such as temperature, pressure, flow rates etc., and the constraints c consists of the model equations, including balance equations, thermodynamic relationships etc. The Jacobian of the constraints is typically very sparse.

- *Multiperiod Design:* Find the optimal values of certain parameters p (such as reactor size), so that by adjusting control variables (such as flow rates) the resulting plant design can a) be operated under a given set of conditions (periods) and b) in some optimal way. In this case, the constraint functions c include "multiple copies" of the process model. These are "coupled" by the parameters p, which have to be the same for each copy.

- *Dynamic Optimization:* Find the optimal (time-dependent) profile of control variables for a plant, whose state is changing over time. Here, we may for example want to maximize the amount of product gained in a batch process, or to minimize the time required for the change-over between different steady states. If a discretization approach is used, the constraint functions c again include multiple blocks of the process model equations, now coupled through the condition that the state variables should have continuous profiles over time.

The mathematical problem formulations arising from those and other applications have certain characteristics, as discussed next.

The more details involved in the model formulation, the better will it represent the actual process. Similarly, the accuracy of a discretization in a dynamic optimization problem increases with the number of discretization points. As a consequence, it is desirable to solve very large NLPs, and the number of optimization variables, n, can exceed several millions. In addition, a plant model has typically only a few control variables compared to the number of state variables necessary to describe the process. As a consequence, the number of "degrees of freedom", $n - m$, is often very small compared to n, but it may still be on the order of thousands. Furthermore, frequently many of the optimization variables have bound constraints that are potentially active at the solution, and active set SQP methods may require solution times that grow exponentially with the problem size.

As indicated above, the constraint equations c often contain multiple blocks of model equations, so that as a consequence the Jacobian A^T of c is very sparse and structured (see e.g. Eq. (5.10) on page 142). If mathematical operations with this Jacobian are required by the optimization algorithm, such as solving linear systems involving a square submatrix C of $A^T = [C \quad N]$, it is important for efficiency that this can be done in a way that exploits the structure of C. For very large problems, this flexibility is even mandatory in order to obtain a solution at all.

Furthermore, one might want to re-use software that has been developed for the simulation of a process, i.e. for solving "$c(x) = 0$". In this case, the elements of the Jacobian are often not explicitly available to the optimizer, and only operations with the Jacobian or the submatrix C can be performed. Since most simulators are based on Newton's method, they can solve linear systems involving C, but in many cases not those involving C^T.

When process models are implemented from scratch using new modeling tools, automatic differentiation can provide second derivatives of the model equations, which should be exploited by the optimization method for fast convergence. On

the other hand, particularly when existing simulation software is to be used, second derivatives are typically not available and have to be approximated.

In summary, we are facing the following challenges:

- Very large problem formulations (n up to several millions);
- Large number of degrees of freedom ($n - m$ up to $10,000$);
- Avoiding the combinatorial bottleneck of identifying the correct active bound constraints;
- Providing flexibility regarding algorithmic requirements (such as access to second derivatives or availability of C^T) to adapt to individual applications and modeling environments;
- Providing flexibility regarding software interfaces to process models.

Many existing NLP solvers are currently not able to cope with these problem sizes, nor do they offer the flexibility to be tailored to specific applications.

Therefore, the goal of this Ph.D. project is the development and implementation of an efficient and robust NLP solver that can address the challenges above. Its theoretical convergence properties are to be explored and its performance is to be verified on many test cases.

1.4 Outline

The next chapter presents some background on nonlinear programming with the purpose of introducing the nomenclature and terminology used throughout this dissertation.

A detailed description of IPOPT, the method developed within this Ph.D. project, is given in Chapter 3. In order to avoid the combinatorial problem of identifying the correct active set, IPOPT is designed as an interior point method (Section 3.1). As a full-space approach (Section 3.2.1), it can be used as general purpose NLP solver

using exact second derivatives. If on the other hand problem structure is to be exploited, IPOPT computes search directions in a reduced space version using a coordinate decomposition (Section 3.2.2). Here, second derivatives can be approximated by quasi-Newton methods (Sections 3.2.3–3.2.4). Alternatively, Hessian-vector products (exact or approximated by finite differences) can be exploited using a preconditioned conjugate gradient method (Section 3.2.5), improving performance in problems with many degrees of freedom.

Global convergence is enforced in IPOPT by a line search approach. Here, several alternative merit functions are discussed (Sections 3.3.1–3.3.2). However, these options may fail to provide global convergence, as demonstrated on a counter example (Section 3.3.3). As a remedy, a novel filter line search method has been developed, which is described in detail in Section 3.4.

Chapter 4 addresses the theoretical aspects of the method. First, the counter example is analyzed in more detail in Section 4.1, including a discussion of other affected interior point methods. The global convergence properties of the filter line search algorithm are examined in Section 4.2. Here it is shown under mild assumptions, that every limit point of the sequence of iterates is feasible, and that at least one limit point is a first order optimal point for the barrier problem. Section 4.3 discusses the local convergence properties of the filter method. Due to a "switching condition" that is different from those in previously proposed filter methods, it can be shown that the Maratos effect can be prevented, if second order corrections are applied to the search directions. Therefore, fast local convergence is ensured under standard assumptions. Finally in Section 4.4 it is shown, how the obtained convergence results can be applied to active set SQP methods.

The practical performance of IPOPT is examined in Chapter 5. First, IPOPT is tested on a variety of standard test problems, in order to compare the performance and robustness of the individual line search options (Section 5.1.2). The results indicate that the new filter line search approach is indeed the most robust and efficient option. In addition, IPOPT is compared with two other recently developed

interior point NLP solvers in terms of robustness and efficiency (Section 5.1.3). These results are very encouraging and indicate a large potential of IPOPT as a general purpose NLP solver.

In Section 5.2, IPOPT is used to solve two dynamic optimization applications, using collocation on finite elements in a new implementation of the elemental decomposition (Section 5.2.1). The first example considers a continuous air separation distillation column (Section 5.2.2) which is solved for a varying number of finite elements, comparing the different options to handle second derivatives. The largest instance gives rise to an NLP with more than 2 million variables and $4,500$ degrees of freedom, which was solved in less than 7 hours on a Linux workstation. The same options are compared in Section 5.2.3 on the smaller example of a batch cooling crystallization process with different problem characteristics.

Finally, the conclusions in Chapter 6 summarize the contributions of this dissertation, and discuss directions for future work.

Chapter 2

Background on Nonlinear Programming

In this chapter some basic concepts and algorithms in nonlinear programming are revisited, in order to introduce terminology and notation for this dissertation and to lay foundations for the algorithm proposed in Chapter 3.

2.1 Notation

Throughout this dissertation, the i-th component of a vector $v \in \mathbb{R}^n$ will be denoted by $v^{(i)}$. Norms $\|\cdot\|$ will indicate a fixed vector norm and its compatible matrix norm unless otherwise specified. For brevity, we will use the convention $(x, \lambda) = (x^T, \lambda^T)^T$ for vectors x, λ. For a matrix A, we will denote by $\sigma_{\min}(A)$ the smallest singular value of A, and for a symmetric, positive definite matrix A we call the smallest eigenvalue $\lambda_{\min}(A)$. Given two vectors $v, w \in \mathbb{R}^n$, we define the convex segment $[v, w] := \{v + t(w - v) : t \in [0, 1]\}$. We will denote by $O(t_k)$ a sequence $\{v_k\}$ satisfying $\|v_k\| \leq \beta\ t_k$ for some constant $\beta > 0$ independent of k, and by $o(t_k)$ a sequence $\{v_k\}$ satisfying $\lim_k \|v_k\|/t_k = 0$. Finally, for a set S we will denote with $\mathrm{card}(S)$ the cardinality of S.

2.2 Optimality Conditions

In the next three chapters we consider the NLP formulation

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{2.1a}$$

$$\text{s.t.} \quad c(x) = 0 \tag{2.1b}$$

$$x^{(i)} \geq 0 \qquad \text{for } i \in \mathcal{I}, \tag{2.1c}$$

where the objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and the equality constraints $c : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ with $m < n$ are sufficiently smooth. Note, that in contrast to (1.1) we assume here that variables have only lower bounds of zero, where $\mathcal{I} \subseteq \{1, \ldots, n\}$ is the set of indices of bounded variables. This simplifies the notation, and the presented algorithms and results can be easily adapted to the problem formulation (1.1).

We will denote with $g(x) := \nabla f(x)$ the gradient of the objective function, and with $A(x) := \nabla c(x)$ the transpose of the Jacobian of the constraint functions. Furthermore, it is common to define the *Lagrangian function* associated with the

NLP (2.1) as

$$\mathcal{L}(x, \lambda, v) := f(x) + c(x)^T \lambda - \sum_{i \in \mathcal{I}} x^{(i)} v^{(i)} \tag{2.2}$$

for some *(Lagrangian) multipliers* $\lambda \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ (with $v^{(i)} = 0$ for $i \notin \mathcal{I}$) corresponding to the equality constraints (2.1b) and bound constraints (2.1c), respectively.

The algorithms discussed in this dissertation are iterative procedures that aim to generate a sequence of iterates $\{x_k\}$ which converges to a *local solution* x_* of the NLP (2.1). A point x_* is called a *local solution* of the NLP (2.1) if it is *feasible*, i.e. if it satisfies the constraints (2.1b)-(2.1c), and if in a sufficiently small neighborhood of x_* there is no other feasible point with a smaller value of the objective function. Furthermore, we will refer to a point x_* as *strict local solution*, if it is a local solution and all other feasible points in a sufficiently small neighborhood of x_* lead to strictly larger values of the objective function.

Assuming certain properties of the constraint functions (a *constraint qualification*), necessary and sufficient conditions for local optimality can be established. In this dissertation we will only employ the following constraint qualification at a local solution x_*.

Definition 2.1 (LICQ) *The linear independence constraint qualification (LICQ) holds at x_*, if the gradients of the equality constraints and active bound constraints are linearly independent, i.e. if the matrix*

$$[A(x_*) \;\; e_{j_1} \;\; \ldots \;\; e_{j_K}] \tag{2.3}$$

with

$$\mathcal{A}(x_*) := \{j : x_*^{(j)} = 0\} = \{j_1, \ldots, j_K\} \tag{2.4}$$

has full row rank.

With this, we are able to state the *first order necessary optimality conditions*, also called *Karush-Kuhn-Tucker (KKT) conditions.*

Definition 2.2 (KKT Conditions) *Let x_* be a point at which LICQ holds. Then, x_* satisfies the Karush-Kuhn-Tucker (KKT) conditions for (2.1), if there exist vectors $\lambda_* \in \mathbb{R}^m$ and $v_* \in \mathbb{R}^n$ with $v_* \geq 0$ and ${v_*}^{(i)} = 0$ for $i \notin \mathcal{I}$ so that*

$$\nabla_x \mathcal{L}(x_*, \lambda_*, v_*) = g(x_*) + A(x_*)\lambda_* - v_* = 0 \tag{2.5a}$$

$$c(x_*) = 0 \tag{2.5b}$$

$$x_* \geq 0 \tag{2.5c}$$

$$x_*^T v_* = 0. \tag{2.5d}$$

We may refer to condition (2.5a) as *dual feasibility*, to (2.5b)-(2.5c) as *(primal) feasibility*, and to (2.5d) as *complementarity*. We will call a point x_* satisfying the KKT conditions also *KKT point* or *critical point* for the NLP (2.1).

Proposition 2.1 *Let x_* be a local solution of the NLP (2.1) at which the LICQ holds. Then x_* satisfies the KKT conditions [69, Theorem 12.1].*

Using second derivative information it is also possible to formulate sufficient optimality conditions.

Definition 2.3 (SOS Conditions) *Let x_* be a point at which LICQ hold. Then, x_* satisfies the Second Order Sufficient (SOS) conditions for (2.1), if*

i) x_* satisfies the KKT conditions for (2.1),

ii) strict complementarity holds at x_*, i.e.

$$v_*^{(i)} > 0 \text{ for all } i \in \mathcal{I} \text{ with } x_*^{(i)} = 0,$$

and

iii) the Hessian $\nabla^2_{xx}\mathcal{L}(x_*, \lambda_*, v_*)$ of the Lagrangian is positive definite on the null space of the active constraints, i.e. if Z_* is a matrix whose columns form a basis of the null space of the matrix (2.3), then the reduced Hessian

$$Z_*^T \nabla^2_{xx}\mathcal{L}(x_*, \lambda_*, v_*) Z_* \tag{2.6}$$

is positive definite.

Proposition 2.2 *If x_* satisfies the SOS conditions, then x_* is a strict local solution of the NLP (2.1) [69, Theorem 12.5].*

2.3 SQP Methods

Many of the currently popular algorithms for solving NLPs belong to the class of *Sequential Quadratic Programming (SQP)* methods. Although there is a wide variety of those algorithms (see e.g. [10, 14, 17, 37, 44, 64]), their basic idea may be described as follows.

Assume for the moment that the NLP under consideration has no bound constraints, i.e. $\mathcal{I} = \emptyset$. Then, the KKT conditions are

$$g(x_*) + A(x_*)\lambda_* = 0 \tag{2.7a}$$

$$c(x_*) = 0. \tag{2.7b}$$

Therefore, in order to find a critical point x_* for the NLP with optimal multipliers λ_*, we may apply Newton's method to the nonlinear system of equations (2.7): Given an initial estimate (x_0, λ_0) of the solution of (2.7) we generate a sequence $\{(x_k, \lambda_k)\}$ by

$$(x_{k+1}, \lambda_{k+1}) := (x_k, \lambda_k) + (d_k, d_k^\lambda), \tag{2.8}$$

where the steps (d_k, d_k^λ) are obtained as solution from the linearization

$$\begin{bmatrix} W_k & A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} d_k \\ d_k^\lambda \end{pmatrix} = -\begin{pmatrix} g_k + A_k\lambda_k \\ c_k \end{pmatrix} \tag{2.9}$$

of the KKT conditions (2.7). Here, we define $c_k := c(x_k)$, $g_k := g(x_k)$, $A_k := A(x_k)$, and $W_k := \nabla^2_{xx}\mathcal{L}(x_k, \lambda_k)$. It is well known that if (x_0, λ_0) is sufficiently close to a strict local solution satisfying the SOS conditions, then the linear system (2.9) has a unique solution for all k, and the sequence $\{(x_k, \lambda_k)\}$ converges quadratically to (x_*, λ_*) [28, Theorem 15.2.1].

The important observation is that obtaining the step (d_k, d_k^λ) from the linear system (2.9) is equivalent to computing d_k as a critical point for the *Quadratic*

Program (QP)

$$\min_{d\in\mathbb{R}^n} \quad g_k^T d + \frac{1}{2} d^T W_k d \tag{2.10a}$$

$$\text{s.t.} \quad A_k^T d + c_k = 0 \tag{2.10b}$$

with corresponding multipliers λ_k^+, and setting

$$d_k^\lambda := \lambda_k^+ - \lambda_k. \tag{2.11}$$

If W_k is positive definite on the null space of A_k^T, for example in the neighborhood of a strict local solution x_* satisfying the SOS conditions, then d_k is indeed the optimal solution of the QP (2.10).

The QP (2.10) can be interpreted as a local model of the original NLP at x_k, consisting of a quadratic approximation of the objective function (including curvature information along the manifold $\{x : c(x) = c(x_k)\}$) and a linear approximation of the constraints. In order to handle NLPs with inequality constraints, we may therefore include a linear approximation of the inequality constraints in the local model and use the resulting QP

$$\min_{d\in\mathbb{R}^n} \quad g_k^T d + \frac{1}{2} d^T W_k d \tag{2.12a}$$

$$\text{s.t.} \quad A_k^T d + c_k = 0 \tag{2.12b}$$

$$x_k^{(i)} + d_k^{(i)} \geq 0 \qquad \text{for } i \in \mathcal{I} \tag{2.12c}$$

to obtain the steps $(d_k, d_k^\lambda, d_k^v)$, where d_k^v are now the steps for the bound multiplier estimates v_k.

If x_* is a strict local solution satisfying the SOS conditions with corresponding optimal multipliers λ_* and v_*, and if (x_0, λ_0, v_0) is sufficiently close to (x_*, λ_*, v_*) and the sequence $\{(x_k, \lambda_k, v_k)\}$ is generated in analogy to (2.8), then for sufficiently large k the QP (2.12) "identifies the correct active set" in the sense that the index set of bounds active at the solution of (2.12) coincides with $\mathcal{A}(x_*)$ defined in (2.4) [69, Theorem 18.1]. The algorithm then behaves as if all active bounds are included into the equality constraints and the inactive bounds are discarded. Therefore, the steps

again correspond to Newton steps for the system (2.9) (with "$c(x)$" now including the active bounds) and fast local convergence is achieved.

So far, we have only addressed the local convergence behavior of the basic SQP method. However, a sufficiently good starting guess (x_0, λ_0, v_0) is often unavailable, and strategies have to be added to guarantee *global convergence*, i.e. convergence to a solution (or at least a critical point) for the NLP (2.1) from any given starting point. The various SQP algorithms differ for example in the techniques used to achieve this goal. Further differences consist in the way the QP (2.12) is solved and in the choice of the Hessian matrix W_k in (2.12a), which, for example, might be a quasi-Newton approximation of $\nabla^2_{xx}\mathcal{L}(x_k, \lambda_k, v_k)$. In the next sections some of those options are revisited.

2.4 Global Convergence

In order to guarantee global convergence, the simple algorithm outlined in the previous section has to be modified, since otherwise the generated sequence of iterates might cycle or behave in some other undesired way. Currently, there are two main approaches to modify the steps: *Line search* methods and *trust region* methods. These techniques are not limited to the class of SQP methods; they are for example also used in other optimization algorithms or methods for solving systems of nonlinear equations. The optimization algorithm proposed in this dissertation follows the line search approach.

In a line search framework, a *search direction* d_k is computed for the current iterate x_k, in the case of SQP methods as a solution of the QP (2.12). Then, a *step length* $\alpha_k \in (0, 1]$ is determined, and the next iterate x_{k+1} is chosen as

$$x_{k+1} := x_k + \alpha_k d_k.$$

To find an appropriate step size α_k, we may perform a *backtracking line search*, i.e. a sequence of decreasing *trial step sizes* $\alpha_{k,0}, \alpha_{k,1}, \ldots$ (usually starting with $\alpha_{k,0} = 1$)

is tried, until a step size $\alpha_{k,l}$ is found, so that the resulting *trial point*

$$x_k(\alpha_{k,l}) := x_k + \alpha_{k,l} d_k \tag{2.13}$$

provides "sufficient progress" towards a local solution (or at least a critical point) for the NLP (2.1).

In order to judge whether a trial step provides sufficient progress, previously proposed SQP line search methods measure the quality of points x by means of a *merit function*, such as an *exact penalty function*

$$\phi_\nu(x) := f(x) + \nu \|c(x)\| \tag{2.14}$$

for some norm $\|\cdot\|$; popular choices are the ℓ_1- and ℓ_2-norm. This function captures the two goals of minimizing the objective function $f(x)$ and at the same time reducing the *constraint violation* $\|c(x)\|$, where the relative weight between these two measures is determined by the *penalty parameter* $\nu > 0$. (It is assumed here that the iterates x_k always satisfy the bound constraints (2.1c), so that only the violation of the equality constraints (2.1b) has to be penalized.) Therefore, a point x_{k+1} can be considered to be "better" than a point x_k, if it leads to a smaller value of the merit function (2.14). Indeed, the merit function $\phi_\nu(x)$ is *exact*, i.e. it can be shown that if x_* is a strict local solution of the NLP (2.1) satisfying the SOS conditions with optimal multipliers λ_* corresponding to the equality constraints (2.1b), and if the penalty parameter satisfies $\nu > \|\lambda_*\|_D$ (where $\|\cdot\|_D$ is the norm dual to $\|\cdot\|$), then x_* is also a strict local minimizer of the (unconstrained) penalty function $\phi_\nu(x)$ [28, Theorem 14.5.1]. Note, however, that $\phi_\nu(x)$ is not differentiable at x_* nor any other feasible point.

At the beginning of the optimization, the optimal value of λ_* and therefore also an appropriate value for ν are not known. Hence, most methods using the exact penalty function update a current estimate ν_k throughout the optimization process, for example

$$\nu_k := \max\{\nu_{k-1}, \|\lambda_k\|_D + \epsilon\}, \tag{2.15}$$

where λ_k is the current multiplier estimate, and $\epsilon > 0$ is a small number. In cases when multiplier estimates are unavailable, "multiplier-free" update rules have been proposed [15], as we will discuss in Section 3.3.1.

Having determined an appropriate value for the penalty parameter ν_k, many SQP algorithms performing a backtracking line search procedure using (2.14) accept a trial step size $\alpha_{k,l}$, if it satisfies the *Armijo condition*

$$\phi_{\nu_k}(x_k(\alpha_{k,l})) \leq \phi_{\nu_k}(x_k) + \eta \cdot \alpha_{k,l} \cdot D\phi_{\nu_k}(x_k, d_k) \tag{2.16}$$

for some fixed $\eta \in (0, \frac{1}{2})$, where $D\phi_{\nu_k}(x_k, d_k)$ denotes the directional derivative of ϕ_{ν_k} at x_k into direction d_k. In order to guarantee that such a step size exists and that d_k is indeed a *descent direction* (i.e. $D\phi_{\nu_k}(x_k, d_k) < 0$), certain assumptions on the Hessian W_k in (2.12a) (such as positive definiteness of a projection of W_k onto a certain subspace of $\mathbb{R}^n$) are usually made. Since these assumptions are often violated away from a strict local solution, precautions have to be taken or modifications on W_k have to be made.

Another popular merit function is the *augmented Lagrangian function*

$$\ell_\nu(x, \lambda) := f(x) + c(x)^T\lambda + \nu c(x)^T c(x), \tag{2.17}$$

which depends on the value of the multiplier estimates λ_k in addition to the primal variables x_k. This function is exact (in x), if the multipliers are chosen to be optimal, λ_*, but usually these values are not known. There is a variety of SQP algorithms employing the augmented Lagrangian function (e.g. [12, 44, 64]). In a line search context the progress is usually measured for trial points

$$(x_k, \lambda_k) + \alpha_{k,l}(d_k, d_l^\lambda)$$

that take steps in both primal and dual variables simultaneously.

The alternative to a line search approach, where a search direction d_k is computed only once and its length is reduced if a trial point is not accepted, is to use a *trust region* method. Here, the constraint

$$\|d\|_{[k]} \leq \Delta_k \tag{2.18}$$

is added to the QP (2.12), where $\|\cdot\|_{[k]}$ is some norm that might even change from one iteration to the next, and Δ_k is the *trust region* radius. In a sense, the trust region radius indicates, how far away from the current iterate we want to trust the QP (2.12) as a sufficiently accurate local model of the original NLP (2.1). At an iterate x_k a trial step d_k is obtained as a solution (or approximate solution) for the QP (2.12) augmented by (2.18). It is then checked if the resulting trial point $\bar{x}_k := x_k + d_k$ provides sufficient progress towards the solution. If so, the trial step is accepted as the new iterate $x_{k+1} := \bar{x}_k$, and the size of the trust region radius is adapted for the next iteration. If the trial step is not acceptable, $\bar{x}_k$ is discarded, no step is taken ($x_{k+1} := x_k$), and the trust region radius is reduced for the next iteration, $\Delta_{k+1} < \Delta_k$. (Note that here we also count unsuccessful trial points as iterates in contrast to the above outlined line search approach.)

In order to decide whether a trial point is acceptable, most trust region methods also employ a merit function. For example, in case of the exact penalty function (2.14) a model for $\phi_\nu(x)$ may be defined as

$$m^{\phi}_{\nu_k}(x_k; d) := f(x_k) + g_k^T d + \frac{1}{2} d^T W_k d + \nu \left\| c_k + A_k^T d \right\| .$$

Then the *actual reduction*

$$\mathrm{ared}_{\nu_k}(x_k, d_k) := \phi_{\nu_k}(x_k) - \phi_{\nu_k}(x_k + d_k)$$

is compared with the *predicted reduction*

$$\mathrm{pred}_{\nu_k}(x_k; d_k) := m^{\phi}_{\nu_k}(x_k; 0) - m^{\phi}_{\nu_k}(x_k; d_k).$$

Again, ν_k is an estimate of an appropriate value for the penalty parameter, usually chosen in a way that ensures $\mathrm{pred}_{\nu_k}(x_k; d_k) > 0$. A trial step is accepted if

$$\mathrm{ared}_{\nu_k}(x_k, d_k) \geq \eta \cdot \mathrm{pred}_{\nu_k}(x_k; d_k)$$

for some $\eta \in (0, 1)$. The degree of agreement between the predicted and actual reduction may also be used to decide whether to decrease or increase the trust region radius for the next iteration.

In the context of a trust region framework, Fletcher and Leyffer [37] recently proposed an alternative to merit functions as a tool to measure progress towards the solution. Within this Ph.D. project their *filter approach* has been applied to line search methods. Details will be discussed in Sections 3.4, 4.2 and 4.3.

2.5 Solution of the QP

In order to obtain a search direction or trial step in an SQP method, usually some variant of the QP (2.12) (possibly including the trust region constraint (2.18)) has to be solved.

There exists a variety of QP solvers [5, 43, 74, 76], most of which are *active set QP solvers*, i.e. they iteratively guess which of the bound constraints (2.12c) are active at the solution of the QP. As pointed out earlier, close to a solution of the NLP satisfying the SOS conditions, the active set of the QP does not change from one iteration to the next, so that a QP solver that is able to incorporate an initial guess of the active set (*warm start*) can obtain the solution to the QP very quickly. However, at the beginning of the optimization process a considerable amount of time might be necessary for the identification of the active set, which is known to be an NP-hard combinatorial problem, so that the solution time increases in the worst case exponentially with the size of the problem.

Alternatively, some QP solvers [76, 86] are based on the idea of an interior point approach (see also Section 2.7). Here, the combinatorial problem is circumvented, but at this point it is not clear whether those methods are able to efficiently exploit knowledge of a good initial guess for the active set in a warm start approach [90], or if even at late stages of the optimization process a certain minimal amount of computation time is required, which exceeds what an active set solver with warm starts would need.

In some SQP methods [37] the *full-space QP* (2.12) (or (2.12)+(2.18)) are solved "as is" by the QP solver. In the following, however, we concentrate on methods that

decompose this QP into smaller components.

Assuming that the Jacobian A_k^T of the constraints has full rank, we can choose a matrix $Z_k \in \mathbb{R}^{n\times(n-m)}$, whose columns form a basis of the null space of A_k^T; in particular we then have

$$A_k^T Z_k = 0. \tag{2.19}$$

We further choose a matrix $Y_k \in \mathbb{R}^{n\times m}$, so that the columns of $[Y_k\ Z_k]$ form a basis of $\mathbb{R}^n$ and can then decompose the solution d_k of the QP (2.12)

$$d_k = q_k + p_k \tag{2.20}$$

into a *(quasi-)normal component*

$$q_k := Y_k \bar{q}_k \tag{2.21}$$

for some $\bar{q}_k \in \mathbb{R}^m$, and a *tangential component*

$$p_k := Z_k \bar{p}_k \tag{2.22}$$

for some $\bar{p}_k \in \mathbb{R}^{n-m}$.

Substituting (2.20) and (2.21) into (2.12b) and remembering (2.19), we have that

$$\bar{q}_k = -\left[A_k^T Y_k\right]^{-1} c_k. \tag{2.23}$$

With this, we see together with (2.20), (2.23), and (2.12) that $\bar{p}_k$ is the solution of the *reduced QP*

$$\min_{\bar{p}\in\mathbb{R}^{n-m}} \quad \left(Z_k^T g_k + \zeta_k w_k\right)^T \bar{p} + \frac{1}{2}\bar{p}^T Z_k^T W_k Z_k \bar{p} \tag{2.24a}$$

$$\text{s.t.} \quad \left(Z_k \bar{p}\right)^{(i)} \geq -x_k^{(i)} - q_k^{(i)} \qquad \text{for } i \in \mathcal{I} \tag{2.24b}$$

where we defined the *cross term*

$$w_k := Z_k^T W_k q_k. \tag{2.25}$$

We also introduce the *damping parameter* $\zeta_k \in (0, 1]$, usually (close to) one, which will be discussed below. Note, that QP (2.24) is only in the space of $n-m$ variables,

but that the number of inequality constraints (2.24b) is the same as in the *full-space QP* (2.12), i.e. the combinatorial complexity for the identification of the correct active set has not changed.

The matrices Z_k and Y_k may be chosen in a way, so that $[Y_k\ Z_k]$ is an orthonormal matrix. For example, this can be done by performing an QR factorization of A_k (see e.g. [42] for details). This particular choice of Y_k and Z_k enhances the numerical stability of the resulting algorithm, but on the other hand is rather expensive, and prohibitive for large-scale problems.

Alternatively, we may follow a *coordinate decomposition* approach, which has been used in [14, 44], and partition the primal variables

$$x = P \begin{pmatrix} x^D \\ x^I \end{pmatrix}, \qquad \text{where } P \text{ is a permutation matrix,} \tag{2.26}$$

into dependent variables x_D and independent variables x_I, so that the columns in the constraint Jacobian

$$A(x)^T = [C(x)\ N(x)]\, P^T \tag{2.27}$$

corresponding to the dependent variables form a non-singular, well-conditioned submatrix $C(x)$. We then define the basis matrices as

$$Y_k := P \begin{bmatrix} I \\ 0 \end{bmatrix} \qquad \text{and} \qquad Z_k := P \begin{bmatrix} -C_k^{-1} N_k \\ I \end{bmatrix}, \tag{2.28}$$

where $C_k := C(x_k)$ and $N_k := N(x_k)$. Note, that for a fixed partition P the matrix $C(x)$ may not remain non-singular for all x. As a consequence, we may choose a partition at the beginning of the algorithm, estimate the condition number of C_k throughout the optimization process, and perform a repartitioning if necessary.

The advantage of this particular choice of the basis matrices becomes apparent if one considers the computation of the quasi-normal component from (2.23). The $m \times m$ matrix $\left[A_k^T Y_k\right]$ simplifies to the (usually sparse and often structured) submatrix C_k of A_k^T, so that we can employ special solution strategies for the large linear system

$$C_k \bar{q}_k = -c_k, \tag{2.29}$$

which can be tailored to specific problem structures (such as those described later in Section 5.2.1). In addition, the upper $m \times (n-m)$ part of Z_k can be computed column-wise by the same procedure (using the columns of N_k as right hand sides in the linear system (2.29)) and then passed to the QP solver addressing the reduced QP (2.24).

In the case that estimates for the equality constraint multipliers are required by the optimization algorithm, they might either be obtained from the full-space QP solver, or — if the reduced space approach is applied — approximated as *first order multipliers*

$$\lambda_k := -\left[Y_k^T A_k\right]^{-1} Y_k^T \left(g_k - v_k^+\right), \tag{2.30}$$

where v_k^+ are the optimal QP multipliers corresponding to the bound constraints (2.24b). This approximation is obtained by premultiplying (2.5a) by Y_k^T and re-ordering terms; it is therefore exact at a critical point of the problem. In a line search context with the exact penalty function (2.14) these multiplier estimates can be used in the penalty parameter update rule (2.15).

Note, that the term in the square brackets simplifies to C_k^T, so that as for the solution to (2.29) tailored procedures may be used to obtain λ_k. However, in some circumstances it is not possible to solve linear systems involving C_k^T, for example when existing simulation software is to be used [16]. In this case, multiplier estimates are not available, and *multiplier free* update rules for the penalty parameter, e.g.

$$\nu_k := \max\left\{\nu_{k-1}, \frac{|q_k^T(g_k - v_k^+)|}{\|c_k\|} + \epsilon\right\} \tag{2.31}$$

have been suggested [15]. In either case, it may be necessary to set the damping parameter ζ_k in (2.24a) to less than one, in order to ensure that the overall search direction d_k from (2.20) is indeed a descent direction for the merit function. In [14] the choice

$$\zeta_k := \begin{cases} 1 & \text{if } g_k^T Z_k B_k^{-1} w_k \geq 0 \\ \min\left\{1, \frac{-0.1 g_k^T Z_k B_k^{-1} Z_k^T g_k}{g_k^T Z_k B_k^{-1} w_k}\right\} & \text{otherwise} \end{cases} \tag{2.32}$$

has been suggested, where B_k is the (positive definite) approximation of $Z_k W_k Z_k$.

One issue that we have ignored so far is the fact that the QP (2.12) may be infeasible, in particular if it is augmented by the trust region constraints (2.18) with a small trust region radius Δ_k. In this case, some SQP methods relax the QP constraints (2.12b)-(2.12c) in order to be able to still obtain some search direction [15, 44], or switch to a different phase that tries to bring the iterates finally into a region in which the constraints are consistent again (such as the *feasibility restoration phase* in [35, 37]).

2.6 Hessian Approximations

As mentioned in Section 1.3, second derivative information may not always be available. In this case, it is necessary to approximate the Hessian W_k in (2.12a) or its reduced version $B_k := Z_k^T W_k Z_k$ in (2.24a). This is usually done by means of a *quasi-Newton method* where an estimate of the matrix involving the unavailable second derivative information is maintained and updated based on changes in first derivatives.

For simplicity we will assume until the end of this section, that the problem formulation (2.1) does not include bound constraints, i.e. that $\mathcal{I} = \emptyset$. Efficient approximation of second order information when active inequality constraints are present is more complicated (see e.g. [58]) and not relevant for this work.

If $\mathcal{I} = \emptyset$, then the optimal reduced Hessian from (2.6),

$$B_* = Z_*^T W_* Z_*$$

with $W_* := \nabla^2_{xx}\mathcal{L}(x_*, \lambda_*)$, is positive definite if x_* satisfies the SOS conditions. Therefore, it is popular to employ the *BFGS updating formula* (see e.g. [69])

$$B_{k+1} \leftarrow B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{2.33}$$

with certain vectors $s_k, y_k \in \mathbb{R}^{n-m}$ at every iteration in order to generate a sequence $\{B_k\}$ approximating B_*. Several choices for s_k and y_k have been proposed (see [68]),

among those

$$\begin{aligned} s_k &:= \bar{p}_k && (2.34a) \\ y_k &:= Z_{k+1}^T \left(\nabla_x \mathcal{L}(x_{k+1}\lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_k)\right) && (2.34b) \\ &= Z_{k+1}^T \left(g_{k+1} - g_k - A_k \lambda_k\right) \end{aligned}$$

(see e.g. [13]), and the *multiplier-free* version

$$\begin{aligned} s_k &:= \bar{p}_k && (2.35a) \\ y_k &:= Z_{k+1}^T g_{k+1} - Z_k^T g_k && (2.35b) \end{aligned}$$

(see [11, 15]), where no explicit knowledge of multiplier estimates λ_k is necessary.

Assuming that B_k is positive definite and $s_k^T y_k > 0$, it is guaranteed that B_{k+1} is positive definite as well. If at some iteration $s_k^T y_k \leq 0$, one might either skip the update, i.e. set $B_{k+1} := B_k$, or modify y_k according to

$$\begin{aligned} y_k &\leftarrow \theta_k y_k + (1-\theta_k) s_k^T B_k s_k \qquad \text{with} && (2.36a) \\ \theta_k &= \frac{0.8\ s_k^T B_k s_k}{s_k^T B_k s_k - s_k^T y_k} && (2.36b) \end{aligned}$$

whenever $s_k^T y_k < 0.2\ s_k^T B_k s_k$. This will guarantee that the new matrix B_{k+1} is still positive definite. We will refer to this procedure as *damped BFGS update* [69].

Second order information is also required for the computation of the cross term (2.25). Some methods simply ignore the cross term; in that case (at best) local two-step superlinear convergence can be shown [68]. Alternatively one might approximate the cross term by some finite difference of the *reduced gradient* $Z(x)^T g(x)$, which leads to local one-step superlinear convergence [11, 13].

For later reference we also recall the *SR1 update* [69]

$$B_{k+1} \leftarrow B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}, \tag{2.37}$$

which might be used if the approximated matrix B_* is indefinite.

2.7 Barrier Methods

As pointed out in Section 2.5, a bottleneck for SQP methods applied to large-scale NLPs is the combinatorial problem of identifying the active bound constraints during the solution of the QP. *Barrier methods*, based on earlier work by Fiacco and McCormick [34], completely avoid this problem by replacing the bound constraints (2.1c) by a logarithmic barrier term which is added to the objective function to give

$$\min \quad \varphi_\mu(x) := f(x) - \mu \sum_{i \in \mathcal{I}} \ln(x^{(i)}) \tag{2.38a}$$

$$\text{s.t.} \quad c(x) = 0 \tag{2.38b}$$

with a *barrier parameter* $\mu > 0$. Since the objective function of this *barrier problem* (2.38) becomes arbitrarily large as x approaches the boundary of the region $\{x \mid x^{(i)} \geq 0 \text{ for } i \in \mathcal{I}\}$ defined by the inequality constraints, it is clear that a local solution x_*^μ of this problem lies in the interior of this set, i.e., $(x_*^\mu)^{(i)} > 0$ for $i \in \mathcal{I}$. We will refer to $\varphi_\mu(x)$ as the *barrier function*. The degree of influence of the *barrier term* "$-\mu \sum_{i \in \mathcal{I}} \ln(x^{(i)})$" is determined by the size of μ, and under certain conditions x_*^μ converges to a local solution x_* of the original problem (2.1) as $\mu \to 0$ [34]. Consequently, a strategy for solving the original NLP (2.1) is to solve a sequence of barrier problems (2.38) for decreasing barrier parameters μ_l, where l is the counter for the sequence of subproblems. Since the exact solution $x_*^{\mu_l}$ is not of interest for large μ_l, the corresponding barrier problem is solved only to a relaxed accuracy ϵ_l, and the approximate solution is then used as a starting point for the solution of next barrier problem (with $\lim_{l \to \infty} \epsilon_l = 0$).

The KKT conditions for (2.38) are

$$\nabla \varphi_\mu(x) + A(x)\lambda = 0 \tag{2.39a}$$

$$c(x) = 0. \tag{2.39b}$$

Solving this system of equations directly by a Newton-type method, as in a straightforward application of an SQP method to (2.38), leads to a so-called *primal method*,

which treats only the primal variables x and possible the equality multipliers λ as iterates. However, the term "$\nabla\varphi_\mu(x)$" has components including $\mu/x^{(i)}$, i.e. the system (2.39) is not defined at a solution x_* of the NLP (2.1) with an active bound $x_*^{(i)} = 0$, and the radius of convergence of Newton's method applied to (2.39) converges to zero as $\mu \to 0$ [82]. It has been shown, that as a consequence after a change of μ the first search direction is not very good [85], but can be enhanced by an extrapolation approach [66].

Instead of following this primal approach, it has been more popular to devise *primal-dual* methods. Here, *dual variables* v are introduced, defined as

$$v^{(i)} := \frac{\mu}{x^{(i)}}.$$

With this definition, the KKT conditions (2.39) are equivalent to the *perturbed KKT conditions* or *primal-dual equations*

$$g(x) + A(x)\lambda - v = 0 \tag{2.40a}$$

$$c(x) = 0 \tag{2.40b}$$

$$x^{(i)}v^{(i)} - \mu = 0 \qquad \text{for } i \in \mathcal{I}. \tag{2.40c}$$

Note, that for $\mu = 0$ these conditions together with the inequalities

$$x^{(i)} \geq 0 \qquad \text{and} \qquad v^{(i)} \geq 0 \qquad \text{for } i \in \mathcal{I} \tag{2.41}$$

are actually the KKT conditions for the original problem (2.1), and that the dual variables v then correspond to the multipliers for the bound constraints (2.1c). *Primal-dual methods* solve the system (2.40) by a Newton-type approach, maintaining iterates for both x_k and v_k, and possibly λ_k. Since (2.41) holds strictly at the optimal solution of the barrier problem (2.38) for $\mu > 0$, x_k and v_k are always required to strictly satisfy the inequalities, that is

$$x_k^{(i)} > 0 \qquad \text{and} \qquad v_k^{(i)} > 0 \qquad \text{for } i \in \mathcal{I} \tag{2.42}$$

for all k, and can approach zero only asymptotically as $\mu \to 0$. For this reason, these methods are also often called *interior point (IP) methods.*

2.8 Previously Proposed Interior Point Algorithms for Nonlinear Programming

Within the past decade several interior point methods for nonlinear programming have been proposed (see e.g. [21, 22, 33, 39, 42, 65, 77, 78, 80, 88, 89]). These methods come in different varieties, such as primal or primal-dual methods, line search or trust region methods, with different merit functions, different strategies to update the barrier parameter, etc. In Section 5.1.3 we will compare the performance of the algorithm proposed in this dissertation with two of those methods, which are briefly described next.

The first method, KNITRO (formerly known as "NITRO"), has been developed by Byrd, Gilbert, Hribar, Liu, Nocedal, and Waltz [21, 22, 23, 24]; for details consult [22]. It addresses NLPs in the general formulation

$$\min_{x\in\mathbb{R}^n} \quad f(x) \tag{2.43a}$$

$$\text{s.t.} \quad c^{\mathcal{E}}(x) = 0 \tag{2.43b}$$

$$c^{\mathcal{I}}(x) \geq 0 \tag{2.43c}$$

by introducing non-negative slack variables s for the inequality constraints (2.43c). A sequence of corresponding barrier problems (similar to (2.38)) is solved for monotonically decreasing values of the barrier parameter μ, which are held constant for several iterations until the barrier problem is solved sufficiently well. Global convergence is ensured by a trust region approach (see Section 2.4) using the exact ℓ_2-penalty function (2.14) (with "$f(x)$" replaced by "$\varphi_\mu(x)$"). Steps are generated by a version of the Byrd-Omojokun trust region SQP algorithm [70] applied to the barrier problem: At an iterate (x_k, s_k) with $s_k > 0$ a trial step (d_k, d_k^s) is computed

as approximate solution of the subproblem

$$\min_{d,d^s} \quad g_k^T d + \frac{1}{2} d^T W_k d - \mu e^T S_k^{-1} d^s + \frac{1}{2} (d^s)^T \Sigma_k d^s \tag{2.44a}$$

$$\text{s.t.} \quad (A_k^{\mathcal{E}})^T d + c^{\mathcal{E}}(x_k) = r^{\mathcal{E}} \tag{2.44b}$$

$$(A_k^{\mathcal{I}})^T d - d^s + c^{\mathcal{I}}(x_k) - s_k = r^{\mathcal{I}} \tag{2.44c}$$

$$d^s \geq -\tau s_k \tag{2.44d}$$

$$\|(d, S_k^{-1} d^s)\|_2 \leq \Delta_k, \tag{2.44e}$$

where $e = (1, \ldots, 1)^T$, $A_k^{\mathcal{E}} := \nabla c^{\mathcal{E}}(x_k)$, $A_k^{\mathcal{I}} := \nabla c^{\mathcal{I}}(x_k)$, $S_k := \operatorname{diag}(s_k)$, and $\Sigma_k := S_k^{-1} V_k$ with $V_k := \operatorname{diag}(v_k)$ for some positive estimate v_k of the multipliers corresponding to (2.43c). W_k is the (exact) Hessian of the Lagrangian corresponding to (2.43). The *fraction-to-the-boundary rule* (2.44d) for some constant $\tau \in (0, 1)$ close to one ensures that also at the next iterate satisfies $s_{k+1} > 0$. Note the scaling of the step for the slack variables in the trust region constraint (2.44e), which takes into account the proximity of the individual components of the slack variables to their bound. Also note that in contrast to the SQP methods described earlier, the linearization of the constraints (2.44b) and (2.44c) do not have to be strictly satisfied by the trial steps (i.e. we might have $r^{\mathcal{E}}, r^{\mathcal{I}} \neq 0$), which allows to deal with the case where the "strict" QP with $r^{\mathcal{E}}, r^{\mathcal{I}} = 0$ is infeasible.

The approximate solution to the subproblem (2.44) is obtained by a decomposition of the overall trial step (d_k, d_k^s) into a normal component (q_k, q_k^s) and a tangential component (p_k, p_k^s) (in analogy to (2.20)). The normal component is an (approximate) solution of the *normal problem*

$$\min_{q,q^s} \quad \|(A_k^{\mathcal{E}})^T q + c^{\mathcal{E}}(x_k)\|_2^2 + \|(A_k^{\mathcal{I}})^T q - q^s + c^{\mathcal{I}}(x_k) - s_k\|_2^2 \tag{2.45a}$$

$$\text{s.t.} \quad q^s \geq -\tau s_k / 2 \tag{2.45b}$$

$$\|(q, S_k^{-1} q^s)\|_2 \leq 0.8 \Delta_k, \tag{2.45c}$$

which is obtained using the *dogleg method*. The resulting normal step is a linear combination of the steepest descent direction $(q_k^{\mathrm{sd}}, q_k^{s,\mathrm{sd}})$ and the least square solution $(q_k^{\mathrm{ls}}, q_k^{s,\mathrm{ls}})$ for the QP (2.45a) without the constraints (2.45b) and (2.45c). An

important point here is that both dogleg components are obtained after scaling the space of the slack variables by S_k^{-1} (as in (2.44e)), again incorporating information about the proximity to the boundary. For example, $(q_k^{\mathrm{ls}}, q_k^{s,\mathrm{ls}})$ is computed by first solving the (scaled) *augmented system*

$$\begin{bmatrix} I & 0 & A_k^{\mathcal{E}} & A_k^{\mathcal{I}} \\ 0 & I & 0 & -S_k \\ (A_k^{\mathcal{E}})^T & 0 & 0 & 0 \\ (A_k^{\mathcal{I}})^T & -S_k & 0 & 0 \end{bmatrix} \begin{pmatrix} y_k \\ y_k^s \\ z_k \\ z_k^s \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ c^{\mathcal{E}}(x_k) \\ c^{\mathcal{I}}(x_k) - s_k \end{pmatrix} \tag{2.46}$$

and then setting

$$\begin{pmatrix} q_k^{\mathrm{ls}} \\ q_k^{s,\mathrm{ls}} \end{pmatrix} := \begin{bmatrix} A_k^{\mathcal{E}} & A_k^{\mathcal{I}} \\ 0 & S_k^2 \end{bmatrix} \begin{pmatrix} z_k \\ z_k^s \end{pmatrix}.$$

The horizontal component is obtained by a *preconditioned conjugate gradient (PCG) method* as approximate solution of a reduced subproblem (similar to (2.24) without inequality constraints, but constraints similar to (2.44d) and (2.44e)). The PCG steps are computed using solutions of the (scaled) augmented system (2.46) with different right hand sides. If the projection of the Hessian in (2.44a) onto the null space of the constraints (2.44b)–(2.44c) is not positive definite, the PCG method may encounter *directions of negative curvature* which are followed until the boundary of the trust region is hit.

Estimates λ_k and v_k for the multipliers corresponding to (2.43b) and (2.43c), respectively, are computed as least square multipliers, possibly corrected to ensure $v_k > 0$.

Global convergence of KNITRO has been proven in [21], and superlinear local convergence has been shown in [23] using a superlinear decrease rate for the barrier parameter μ. However, the current implementation of KNITRO decreases μ only by a fixed factor of 0.2, therefore leading only to a linear local convergence rate.

The other interior point method, LOQO, with which we will compare the proposed algorithm, has been developed by Benson, Shanno, and Vanderbei [8, 9, 75, 80]. The

basic problem formulation addressed by LOQO has only inequality constraints, which after introducing slack variables becomes

$$\min_{x,s} \quad f(x) \tag{2.47a}$$

$$\text{s.t.} \quad c^{\mathcal{I}}(x) - s = 0 \tag{2.47b}$$

$$s \geq 0. \tag{2.47c}$$

Following a line search approach, LOQO maintains iterates for the primal variables x_k and s_k, as well as for the multiplier estimates v_k corresponding to (2.47b). Search directions are obtained from the linearization of the corresponding primal-dual equations (similar to (2.40)) and computed from

$$\begin{bmatrix} W_k & -\nabla c^{\mathcal{I}}(x_k) \\ -(\nabla c^{\mathcal{I}}(x_k))^T & -S_k V_k^{-1} \end{bmatrix} \begin{pmatrix} d_k \\ d_k^v \end{pmatrix} = -\begin{pmatrix} g_k - \nabla c^{\mathcal{I}}(x_k) v_k \\ c^{\mathcal{I}}(x_k) - s_k - S_k V_k^{-1}(v_k - \mu S_k^{-1} e) \end{pmatrix} \tag{2.48}$$

and $d_k^s := -S_k V_k^{-1}(v_k - \mu S_k^{-1} e + d_k^v)$. Here, we again use $S_k := \text{diag}(s_k)$, $V_k := \text{diag}(v_k)$, $e = (1, \ldots, 1)^T$, and W_k the Hessian of the Lagrangian associated with (2.47).

In contrast to KNITRO and the algorithm proposed in the next chapter, the value of the barrier parameter μ is updated non-monotonically in every iteration. In order to choose a step size α_k, LOQO originally used the (non-exact) merit function

$$\tilde{\phi}_\nu(x,s) := \underbrace{f(x) - \mu \sum_{i=1}^{n} \ln(s^{(i)})}_{=:\varphi_\mu(x,s)} + \nu \|c^{\mathcal{I}}(x) - s\|_2^2. \tag{2.49}$$

Recently, Benson, Shanno, and Vanderbei [8] have proposed alternative step acceptance mechanisms, which use elements of Fletcher and Leyffer's filter concept [37] in a heuristic manner, some of them reverting to the merit function (2.49) under certain circumstances. The option implemented in the current version LOQO 6.02 is the "Markov Filter" presented in [8] and accepts a trial step size, if it leads to sufficient reduction in either the barrier function $\varphi_\mu(x,s)$ or the constraint violation

$\|c^{\mathcal{I}}(x) - s\|$. In addition, some heuristics are included to avoid problems at (almost) feasible points.

As a line search method, LOQO needs to ensure that the generated search directions are descent directions, for example for the merit function (2.49). For this purpose it is guaranteed that the matrix $W_k + \nabla c^{\mathcal{I}}(x_k) S_k^{-1} V_k (\nabla c^{\mathcal{I}}(x_k))^T$ is positive definite by monitoring the sign of pivot elements during the factorization of the matrix in (2.48) and adding some multiple of the identity matrix I to W_k, if necessary. The method proposed in the next chapter borrows this idea (see Section 3.2.1).

In order to solve general NLPs (2.43) including equality constraints, LOQO introduces another pair of non-negative slack variables w and p; the above procedures are then applied to the following formulation:

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{2.50a}$$

$$\text{s.t.} \quad c^{\mathcal{I}}(x) - s = 0 \tag{2.50b}$$

$$c^{\mathcal{E}}(x) - w = 0 \tag{2.50c}$$

$$w + p = 0 \tag{2.50d}$$

$$s, w, p \geq 0. \tag{2.50e}$$

Interestingly, there are no values for (x, s, w, p) with $s, w, p > 0$ that satisfy (2.50b)–(2.50d). Therefore, the strict relative interior of the corresponding barrier problem is empty.

No theoretical global or local convergence proofs have been published for LOQO, but good practical performance has been observed [31].

Chapter 3

Description of IPOPT

The algorithm proposed in this dissertation, called IPOPT, is a primal-dual barrier method, solving a sequence of barrier problems (see Section 3.1). Search directions can be computed using a full-space (Section 3.2.1) or reduced space approach (Section 3.2.2), the latter allowing exploitation of problem-dependent structure and approximation of second derivative information (Sections 3.2.3–3.2.5). Global convergence is ensured by means of line search approaches, allowing the user to choose between different merit functions (Section 3.3) or a novel line search filter approach (Section 3.4).

3.1 The Outer Loop

IPOPT follows the primal-dual barrier approach described in Section 2.7. In an outer loop a sequence of barrier problems (2.38) is solved for a superlinearly decreasing sequence for barrier parameters $\{\mu_l\}$. Aggregating the terms on the left hand side of the primal-dual equations (2.40) into a function $F_\mu(x, v, \lambda)$, we require that each barrier problem is solved approximately, so that approximate solution $(\tilde{x}_{*,l+1}, \tilde{v}_{*,l+1}, \tilde{\lambda}_{*,l+1})$ satisfies

$$\left\| F_{\mu_l}\left(\tilde{x}_{*,l+1}, \tilde{v}_{*,l+1}, \tilde{\lambda}_{*,l+1}\right)\right\|_\infty \leq \epsilon_l \tag{3.1}$$

for an error tolerance $\epsilon_l > 0$ satisfying a certain relationship to μ_l. The precise algorithm is as follows.

Algorithm OUTER LOOP.

Given: Starting point $(\tilde{x}_{*,0}, \tilde{v}_{*,0}, \tilde{\lambda}_{*,0})$; initial value of barrier parameter μ_0; constants $\tau_\mu \in (0,1)$; $\tau_\epsilon, \epsilon_{\max} > 0$; $\theta_\mu \in (1,2)$; $\theta_\epsilon \in (1, 2/\theta_\mu)$.

1. Initialize outer iteration counter $l \leftarrow 0$.

2. Determine error tolerance

$$\epsilon_l := \min\left\{\epsilon_{\max}, \tau_\epsilon \min\left\{\mu_l,\ \mu_l^{\theta_\epsilon}\right\}\right\}.$$

3. Obtain an approximate solution $(\tilde{x}_{*,l+1}, \tilde{v}_{*,l+1}, \tilde{\lambda}_{*,l+1})$ to the barrier problem (2.38) for barrier parameter μ_l satisfying the error tolerance (3.1), using $(\tilde{x}_{*,l}, \tilde{v}_{*,l}, \tilde{\lambda}_{*,l})$ as starting point. If $l \neq 0$ make sure that at least one step is taken in the algorithm for the inner problem, even if the new error tolerance is already satisfied at the starting point.

4. Update barrier parameter

$$\mu_{l+1} := \min\left\{\tau_\mu \mu_l,\ \mu_l^{\theta_\mu}\right\}$$

and increase the outer iteration counter $l \leftarrow l + 1$.

5. Go back to Step 2.

This procedure was essentially proposed by Gould, Orban, Sartenaer, and Toint [47]. As a consequence, the local convergence analysis from [47] for $\mu_l \to 0$ should apply to IPOPT and guarantee superlinear local convergence to a local solution satisfying the SOS conditions, assuming that exact second derivative information is used. Note, that the first step during the solution of each barrier problem (explicitly enforced in Step 3) corresponds to the extrapolation step in [47].

The default settings in IPOPT for the constants are $\mu_0 = 10^{-1}$, $\tau_\mu = 0.2$, $\tau_\epsilon = 10$, $\epsilon_{\max} = 1000$, $\theta_\mu = 1.5$, and $\theta_\epsilon = 1.1$. In addition, in order to take into account the scaling of different problems, the error tolerance (3.1) is by default replaced by

$$\max\left\{\frac{\max\{\|\tilde{g}_{*,l+1} + \tilde{A}_{*,l+1}\tilde{\lambda}_{*,l+1} - \tilde{v}_{*,l+1}\|_\infty, \|\tilde{X}_{*,l+1}\tilde{v}_{*,l+1} - \mu_l e\|_\infty\}}{1 + \frac{\|\tilde{\lambda}_{*,l+1}\|_1 + \|\tilde{v}_{*,l+1}\|_1}{n+m}}, \frac{\|\tilde{c}_{*,l+1}\|_\infty}{1 + \frac{\|\tilde{x}_{*,l+1}\|_1}{n}}\right\} \le \epsilon_l.$$

In the remainder of this chapter we will discuss the algorithm for solving the barrier problem (2.38) for a *fixed* value of μ.

3.2 Computation of Search Directions

3.2.1 Full-Space Version

Since a solution of the barrier problem satisfies the primal-dual equations (2.40), we may apply Newton's method to this system of nonlinear equations. Then the search direction $(d_k, d_k^\lambda, d_k^v)$ at an iterate (x_k, λ_k, v_k) is obtained as a solution of the linearization of (2.40), that is

$$\begin{bmatrix} W_k & A_k & -I \\ A_k^T & 0 & 0 \\ V_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_k \\ d_k^\lambda \\ d_k^v \end{pmatrix} = - \begin{pmatrix} g_k + A_k\lambda_k - v_k \\ c_k \\ X_k v_k - \mu e \end{pmatrix}, \tag{3.2}$$

where e is the vector of ones of appropriate dimension, and $X_k := \mathrm{diag}(x_k)$ and $V_k := \mathrm{diag}(v_k)$; for notational convenience it is assumed that $\mathcal{I} = \{1, \ldots, n\}$. As

before, W_k denotes the Hessian of the Lagrangian (2.2), i.e.

$$W_k := \nabla^2_{xx}\mathcal{L}(x_k, \lambda_k, v_k).$$

We can obtain a solution to (3.2) by first solving

$$\begin{bmatrix} H_k & A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} d_k \\ \lambda_k^+ \end{pmatrix} = -\begin{pmatrix} \nabla\varphi_\mu(x_k) \\ c_k \end{pmatrix}, \tag{3.3}$$

where

$$H_k := W_k + \Sigma_k \tag{3.4}$$

with $\Sigma_k := X_k^{-1}V_k$, and then computing d_k^λ from (2.11) and d_k^v from

$$d_k^v := \mu X_k^{-1}e - v_k - \Sigma_k d_k. \tag{3.5}$$

As pointed out in Section 2.4, line search methods usually require that the projection of the Hessian (here H_k) onto some subspace of $\mathbb{R}^n$ is positive definite. IPOPT guarantees that H_k projected onto the null space of A_k^T is positive definite by adding some multiple of the identity, i.e.

$$H_k := W_k + \Sigma_k + \delta_1 I \tag{3.6}$$

for some $\delta_1 > 0$, if necessary. It is known (under the assumption that A_k has full rank) that the projection of H_k onto the null space of A_k^T is positive definite if and only if the iteration matrix in (3.3) has n positive and m negative eigenvalues [69, Theorem 16.6]. Therefore, in order to detect whether a modification of the Hessian is necessary, the inertia of this iteration matrix is monitored, and possibly several values of $\delta_1 \geq 0$ in (3.6) are tried, until the inertia is correct. This heuristic is similar to the one proposed by Vanderbei and Shanno [80].

In addition, the iteration matrix in (3.3) can only be non-singular, if the constraint Jacobian A_k^T has full rank. In order to avoid failure if this condition is not satisfied at some iterate, we use a standard trick [67] and add small pivot elements into the lower right corner of the iteration matrix, i.e. (3.3) is replaced by

$$\begin{bmatrix} H_k & A_k \\ A_k^T & -\delta_2 I \end{bmatrix} \begin{pmatrix} d_k \\ \lambda_k^+ \end{pmatrix} = -\begin{pmatrix} g_k - \mu X_k^{-1}e \\ c_k - \delta_2\lambda_k \end{pmatrix} \tag{3.7}$$

with some small $\delta_2 > 0$, whenever the inertia of the unmodified iteration matrix is incorrect.

In this full-space version, the initial multiplier estimates λ_0 are computed as least-square estimates, i.e. as solution of

$$\begin{bmatrix} I & A_0 \\ A_0^T & 0 \end{bmatrix} \begin{pmatrix} z \\ \lambda_0 \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_0) \\ 0 \end{pmatrix}.$$

3.2.2 Reduced Space Version

As mentioned in Section 2.3, the search direction d_k obtained from the linear system (3.3) is the solution of the QP

$$\min_{d \in \mathbb{R}^n} \quad \nabla \varphi_\mu(x_k)^T d + \frac{1}{2} d^T H_k d \tag{3.8a}$$

$$\text{s.t.} \quad A_k^T d + c_k = 0, \tag{3.8b}$$

assuming that H_k is positive definite in the null space of A_k^T. Therefore, the decomposition techniques previously developed for SQP methods (as described in Section 2.5) can be applied here as well.

In its reduced space version, IPOPT partitions the variables into dependent and independent variables as in (2.26), and defines basis matrices Y_k and Z_k by (2.28). The (primal) search direction d_k is decomposed as in (2.20) into its quasi-normal component (2.21) and tangential component (2.22). The quasi-normal component can again be computed by (2.23), which simplifies to (2.29) and as before allows efficient exploitation of problem structure. The reduced QP (2.24) now becomes

$$\min_{\bar{p} \in \mathbb{R}^{n-m}} \quad \left(Z_k^T \nabla \varphi_\mu(x_k) + \zeta_k w_k \right)^T \bar{p} + \frac{1}{2} \bar{p}^T Z_k^T H_k Z_k \bar{p},$$

and can be solved directly by solving the (usually dense) symmetric linear system

$$\bar{p}_k := -\tilde{B}_k^{-1} \left(Z_k^T \nabla \varphi_\mu(x_k) + \zeta_k w_k \right) \tag{3.9}$$

where $\tilde{B}_k$ is the overall reduced Hessian

$$Z_k^T H_k Z_k \overset{(3.4)}{=} Z_k^T W_k Z_k + Z_k^T \Sigma_k Z_k, \tag{3.10}$$

or some approximation to it.

Multiplier estimates may be obtained from

$$\lambda_k := -\left[Y_k^T A_k\right]^{-1} Y_k^T \left(g_k - v_k\right), \tag{3.11}$$

(in analogy to (2.30)), where v_k is now the current iterate for the dual variables.

3.2.3 Hessian Approximation

In IPOPT, several options have been implemented for the reduced space version to compute or approximate second derivative information, and to solve the linear system (3.9) efficiently.

Some of these options assume that products of the exact Hessian W_k (or the one-sided projection $Z_k^T W_k$) with vectors $y \in \mathbb{R}^n$ can be performed. In cases, where these products cannot be computed directly, they may be approximated by finite differences

$$Z_k^T W_k \cdot y \approx \frac{Z_k^T (\nabla \mathcal{L}(x_k + t \cdot y, \lambda_k) - \nabla \mathcal{L}(x_k, \lambda_k))}{t}, \tag{3.12}$$

where t is a small scalar. In the current implementation the value of t is chosen as large as possible, so that $\|t \cdot y\|_2 \leq 10^{-6}$ and $x_k + t \cdot y \geq 0$.

Let us first discuss the cross term w_k. If products with the Hessian are available, we set

$$w_k := Z_k^T H_k q_k, \tag{3.13}$$

otherwise we approximate the cross term by

$$w_k := Z_k^T \Sigma_k q_k, \tag{3.14}$$

which because of $\Sigma_k = X_k^{-1} V_k$ is readily available, and captures important curvature information corresponding to the barrier term.

Regarding the reduced Hessian $\tilde{B}_k$ in (3.9), the following options have been implemented:

1. Construct the exact reduced Hessian (3.10) column-wise by computing the products

$$Z_k^T \left(W_k + \Sigma_k\right) Z_k e_i \tag{3.15}$$

with the coordinate vectors e_i, possibly employing finite differences (3.12), and use this as $\tilde{B}_k$ in (3.9). However, it is not guaranteed that $\tilde{B}_k$ constructed in this fashion is positive definite, and modifications might be necessary, as described later in Section 3.2.4.

2. Note, that the reduced Hessian (3.10) consists of two parts. The first is the reduced Hessian of the *original* NLP (2.1) and contains in

$$W_k = \nabla^2 f(x_k) + \sum_{j=1}^{m} \lambda_k^{(j)} \nabla^2 c(x_k)$$

the second order information that might not be available. Therefore, we may approximate it with a quasi-Newton estimate B_k, updated by the BFGS formula (2.33), both with or without damping. The terms s_k and y_k in the update formula are then chosen as in (2.34) or (2.35), depending on whether multiplier estimates are available. The second term in (3.10) corresponds to the ("primal-dual") Hessian of the barrier term and can be computed explicitly. The approximation of the overall reduced Hessian used in (3.9) then is

$$\tilde{B}_k := B_k + Z_k^T \Sigma_k Z_k. \tag{3.16}$$

Since BFGS estimates are guaranteed to be positive definite, $\tilde{B}_k$ is also positive definite and can be used in (3.9) without modification.

Splitting the two terms in (3.10) and computing the primal-dual Hessian exactly has the advantage that the method can react quickly when components of the iterates $x^{(i)}$ come close to their bounds. The algorithm then immediately "feels" the barrier term and can be expected to generate search directions that are bent away from the boundary. Furthermore, after a change of the barrier parameter μ at the late stages of the optimization, the reduced Hessian

$Z_k^T W_k Z_k$ can be expected to change only slightly so that the quasi-Newton estimate from the previous barrier problem can be used effectively, whereas the overall reduced Hessian (3.10) will change significantly, if bounds (2.1c) are active at the solution; in this case it even becomes singular as $\mu \to 0$. Related approaches can be found in [1, 87].

A disadvantage of this approach is that the term $Z_k^T \Sigma_k Z_k$ has to be constructed explicitly, which can be computationally expensive for large dimensions n, even if the dimension of the reduced space $n-m$ is small. In this case, one can avoid the explicit construction of $Z_k^T \Sigma_k Z_k$ by solving the full-space system (3.3) with

$$H_k := \Sigma_k + P \begin{bmatrix} 0 & 0 \\ 0 & B_k \end{bmatrix} P^T,$$

where P is the permutation matrix in (2.26).

3. However, even if a local solution x_*^μ of the barrier problem (2.38) satisfies the SOS conditions, the term $Z_*^T W_* Z_*$, which is a projection of the Hessian of Lagrangian corresponding to the *original* NLP (2.1) only onto the null space of the equality constraints (2.1b), can be indefinite. Therefore, the previous option might produce inefficient estimates B_k, in particular, if the BFGS update is skipped whenever $s_k^T y_k \leq 0$. We therefore also implemented the option to update the quasi-Newton estimate B_k by the SR1 formula (2.37). Since now B_k may become indefinite, we again might have to perform modifications of $\tilde{B}_k$ obtained from (3.16), see Section 3.2.4.

   In our numerical experiments it often seemed that the estimates obtained by SR1 lead to better performance than those obtained by BFGS, unless too many correction have to be made to make $\tilde{B}_k$ positive definite. This is confirmed in the comparisons presented in Sections 5.2.2 and 5.2.3.

For the second approach, we showed in [84] local two-step superlinear convergence of the algorithm solving the barrier problem (2.38) for a fixed value of the

barrier parameter μ under the (strong) assumption, that the reduced Hessian of the Lagrangian corresponding to the *original* NLP is positive definite. Local one-step superlinear convergence can be shown, if the cross term w_k is computed using finite differences of the reduced gradients as proposed in [11, 13].

3.2.4 Solution of the Reduced System

Having discussed what matrix $\tilde{B}_k$ to use in (3.9), we now consider the question, how this dense, symmetric linear system can be solved.

If it is guaranteed that the reduced Hessian approximation $\tilde{B}_k$ is positive definite, like the BFGS case described as second option in the previous section, the standard (dense) Cholesky factorization is used.

For the other cases, the following options have been implemented:

1. Use a *modified Cholesky factorization* (see e.g. [69, p. 148]). This procedure implicitly makes the factorized matrix positive definite by increasing very small or negative pivot elements.

2. Compute the eigenvalues of $\tilde{B}_k$, and modify them to be (sufficiently) positive, if necessary. If $n - m \ll n$, as for many of the applications described in Section 1.3, the computational cost for this procedure is negligible compared to the factorization of the basis matrix C_k in (2.29).

3. Another implemented heuristic is similar to the one used in the full-space version described in Section 3.2.1. First, a standard Cholesky factorization is tried, and if no negative pivot element is encountered, $\tilde{B}_k$ is positive definite and is used as is. If the Cholesky factorization cannot be performed because $\tilde{B}_k$ has non-positive eigenvalues, we add (in analogy to (3.6)) a multiple of identity

$$\tilde{B}_k \leftarrow \tilde{B}_k + \delta_1 I$$

for several trial values $\delta_1 > 0$ until the resulting matrix $\tilde{B}_k$ is positive definite.

4. Since the effect of the previous modification depends on the particular partition P in (2.26), we also implemented the exact translation of the (partition independent) full-space heuristic (3.6) into the reduced space by choosing

$$\tilde{B}_k \leftarrow \tilde{B}_k + \delta_1 Z_k^T Z_k$$

and also modifying the cross term

$$w_k \leftarrow w_k + \delta_1 Z_k^T q_k.$$

However, this option can be considerably computationally more expensive than the previous one, since the dense matrix $Z_k^T Z_k$ has to be constructed explicitly.

5. Finally, we also implemented the option to solve the reduced system (3.9) first by a standard LU decomposition, ignoring the inertia of $\tilde{B}_k$. If, however, $\tilde{B}_k$ turns out to be singular, or the resulting solution is a direction of negative curvature, i.e.

$$\bar{p}_k^T \tilde{B}_k \bar{p}_k \leq 0,$$

we revert to one of the modifications above.

The third method described above is the default option in the reduced version of IPOPT.

3.2.5 Preconditioned Conjugate Gradients

The usage of exact second derivative information can considerably reduce the number of iterations compared to a quasi-Newton approach, particularly when the reduced space (with dimension $n - m$) is not very small. However, the explicit construction of the exact reduced Hessian using the Hessian-vector products (3.15) is computationally expensive.

Motivated by the desire to incorporate second order information without having to construct the exact reduced Hessian $\tilde{B}_k := Z_k^T H_k Z_k$ explicitly, the *Preconditioned Conjugate Gradient (PCG)* algorithm (see e.g. [69, p. 118-119]) has been implemented to solve the reduced system (3.9).

For efficient performance of the conjugate gradient method, it is essential to supply an effective preconditioner P_k, that tries to approximate the inverse of $\tilde{B}_k$ and to simplify the spectrum of the eigenvalues of $P_k\tilde{B}_k$ in order to speed up the convergence behavior of the PCG method.

At this point, the following preconditioners are available in IPOPT:

1. The first one has been proposed by Morales and Nocedal in the context of unconstrained optimization [63]. An approximation of the inverse of $\tilde{B}_k$ is maintained using the BFGS formula. At the beginning of the PCG procedure for (3.9) in a given iteration k the current estimate is copied and used as the (fixed) preconditioner. In order to prepare the preconditioner for the PCG procedure in the next IP iteration $k+1$, the inverse BFGS update formula [69, Eq. (8.16)] is applied at *every PCG iteration*, using $s := \Delta\bar{p}_{k,l}$ and $y := Z_k^T H_k Z_k \Delta\bar{p}_{k,l}$ for the PCG steps $\Delta\bar{p}_{k,l}$ (where l is the iteration counter for the PCG method). Note, that the product y has to be computed for the PCG method in any case. Hence we are collecting as much second order information as possible that is available without further computational effort.

2. For reasons similar to those mentioned in the discussion of the quasi-Newton option around Eq. (3.16), splitting the two terms in the reduced Hessian (3.10) has the advantage that the approximation can react quickly to changes in the reduced primal-dual Hessian of the barrier term, $Z_k^T\Sigma_k Z_k$, which is the component of $\tilde{B}_k$ that leads to the ill-conditioning of the reduced system (3.9), particularly as the barrier μ approaches zero. We therefore also implemented the option to construct the preconditioner P_k in an iteration k from a damped BFGS approximation $\tilde{P}_k$ of the reduced Hessian of the original NLP, $Z_k^T W_k Z_k$, and the exact term $Z_k^T\Sigma_k Z_k$, namely as

$$P_k := \left[\tilde{P}_k + Z_k^T\Sigma_k Z_k\right]^{-1}. \tag{3.17}$$

In addition to the updates of $\tilde{P}_k$ in every PCG iteration, we may optionally

perform a "regular" BFGS update based on changes of the reduced gradients, as in the second option in Section 3.2.3.

As we will see in Sections 5.2.2 and 5.2.3, this option provides better preconditioners than the first choice, in the sense that the number of PCG iterations is considerably reduced. On the other hand, the term $Z_k^T \Sigma_k Z_k$ has to be constructed explicitly, possibly requiring a substantial amount of computational work.

3. We also implemented the option to approximate the term $\tilde{P}_k$ in (3.17) using the SR1 updating formula (2.37). However, since the matrix sum in the square brackets can then become indefinite and the preconditioner has to be positive definite, some modification has to be performed. At this point we have not found an effective modification.

If the matrix in the linear system (3.9) is not positive definite, the PCG procedure may encounter steps $\Delta\bar{p}_{k,l}$ with

$$(\Delta\bar{p}_{k,l})^T [Z_k^T H_k Z_k] \Delta\bar{p}_{k,l} \leq 0.$$

When this occurs, we simply terminate the PCG procedure prematurely, and use the approximate solution for (3.9) that has been obtained so far.

As important differences to the standard PCG algorithm described in [69, p. 118-119], we should mention that the starting point for the PCG procedure in our implementation is not $\Delta\bar{p}_{k,0} = 0$ but

$$\Delta\bar{p}_{k,0} = -P_k \left(Z_k^T \nabla\varphi_\mu(x_k) + w_k \right),$$

in analogy to (3.9). The final horizontal step $\bar{p}_k$ can then be understood as the "traditional" quasi-Newton step (3.9) improved by explicit second order information. Also, in contrast to many PCG implementations, the termination criterion is based on the reduction of the unscaled residual, not the preconditioned residual, since the preconditioner can become very ill-conditioned. These modifications seem to work well in practice.

3.3 Merit Function Based Line Search

Having computed the primal-dual search directions (d_k, d_k^v), we need to choose step lengths $\alpha_k, \alpha_k^v \in (0,1]$ to obtain the next iterate

$$x_{k+1} := x_k + \alpha_k d_k \tag{3.18a}$$

$$v_{k+1} := v_k + \alpha_k^v d_k^v. \tag{3.18b}$$

Among other things, we need to ensure that the positivity constraints (2.42) are satisfied also for the new iterates, since a full step with $\alpha_k = 1$ and $\alpha_k^v = 1$ might violate these conditions. For this, we compute the largest step sizes $\alpha_k^{\max}, \alpha_k^{\max,v} \in (0,1]$ such that the "fraction-to-the-boundary-rule"

$$x_k^{(i)} + \alpha_k^{\max} d_k^{(i)} \geq (1-\tau) x_k^{(i)} \qquad \text{for } i \in \mathcal{I} \tag{3.19a}$$

$$v_k^{(i)} + \alpha_k^{\max,v} (d_k^v)^{(i)} \geq (1-\tau) v_k^{(i)} \qquad \text{for } i \in \mathcal{I} \tag{3.19b}$$

is satisfied, where $\tau \in (0,1)$ is some constant, usually chosen close to one (by default, $\tau = 0.99$ in IPOPT). In a backtracking line search procedure, a suitable value is then determined for the primal step size $\alpha_k \leq \alpha_k^{\max}$ in (3.18a). Regarding the dual step size α_k^v, numerical evidence suggests that the choice $\alpha_k^v := \alpha_k^{\max,v}$ is most efficient, but also the following choices have been implemented in IPOPT:

1. Choose the same step sizes for primal and dual variables, i.e. start the backtracking line from $\tilde{\alpha}_k^{\max} := \min\{\alpha_k^{\max}, \alpha_k^{\max,v}\}$ and choose $\alpha_k^v = \alpha_k$.
2. Choose $\alpha_k^v := \min\{\alpha_k^{\max,v}, \alpha_k\}$.

If steps d_k^λ are generated from (2.11) for equality constraint multiplier estimates λ_k, the new estimates are computed by

$$\lambda_{k+1} := \lambda_k + \alpha_k d_k^\lambda,$$

and used for example in the computation of the exact Hessian W_k.

In order to decide whether a trial step size $\alpha_{k,l}$ and the corresponding trial point (2.13) is acceptable, several options have been implemented in IPOPT. We first describe those procedures that use merit functions.

3.3.1 Exact Penalty Functions

The first set of line search options is based on the exact penalty function (2.14), which for the barrier problem (2.38) becomes

$$\phi_{\mu,\nu}(x) := \varphi_\mu(x) + \nu\|c(x)\|. \tag{3.20}$$

In the implementation of IPOPT, the norm in the infeasibility term can be chosen as the ℓ_1- (default), ℓ_2-, or ℓ_∞-norm. Since this function does not take the scaling of individual equality constraints into account, we also implemented the more general penalty function

$$\bar{\phi}_{\mu,\bar{\nu}}(x) := \varphi_\mu(x) + \sum_{j=1}^{m} \bar{\nu}^{(j)}|c^{(j)}(x)| \tag{3.21}$$

with individual positive penalty parameters $\bar{\nu} \in \mathbb{R}^m$. This function is exact as well, if $\bar{\nu}^{(j)} > |\lambda_*^{(j)}|$ for all j, where λ_* are the constraint multipliers corresponding to an optimal solution x_*^μ.

Trial points are accepted when the Armijo condition (2.16) is satisfied. It remains to describe how the penalty parameters are updated. Here, several options have been implemented.

1. In the full-space version, where multiplier estimates λ_k^+ are available from (3.3), we use

$$\nu_k := \begin{cases} \nu_{k-1} & \text{if } \nu_{k-1} \geq \|\lambda_k^+\|_D + \epsilon_\nu \\ \|\lambda_k^+\|_D + 2\epsilon_\nu & \text{otherwise} \end{cases} \tag{3.22}$$

for $\phi_{\mu,\nu}$, and

$$\bar{\nu}_k^{(j)} := \begin{cases} \bar{\nu}_{k-1}^{(j)} & \text{if } \bar{\nu}_{k-1}^{(j)} \geq \left|(\lambda_k^+)^{(j)}\right| + \epsilon_\nu \\ \left|(\lambda_k^+)^{(j)}\right| + 2\epsilon_\nu & \text{otherwise} \end{cases} \tag{3.23}$$

for $\bar{\phi}_{\mu,\bar{\nu}}$, where $\epsilon_\nu > 0$ is a safeguard parameter ($\epsilon_\nu = 10^{-6}$ by default in IPOPT).

However, due to the (undamped) cross term (3.13), which is implicitly incorporated in the solution of (3.3), the generated search direction d_k might not

be a descent direction for $\phi_{\mu,\nu}$ or $\bar{\phi}_{\mu,\bar{\nu}}$. We therefore increase ν_k or $\bar{\nu}_k$ by the factor 10, until

$$D\phi_{\mu,\nu_k}(x_k; d_k) \leq -\rho \cdot \nu_k \|c(x_k)\|$$

for some constant $\rho \in (0,1)$ (similarly for $\bar{\phi}_{\mu,\bar{\nu}}$; default in IPOPT is $\rho = 0.1$). This heuristic seems to work well in practice.

2. In the reduced space version of IPOPT, the same update rules (3.22) and (3.23) can be applied, if multiplier estimates are available from (3.11). Again it is not ensured that d_k is then a descent direction for the penalty function, and we may apply the same heuristic for increasing the penalty parameter(s) as described in the previous option.

   In analogy to the method described in [13], we may choose the damping parameter ζ_k for the cross term w_k from (2.32) (instead of always 1), with "g_k" replaced by "$\nabla\varphi_\mu(x_k)$" and "B_k" replaced by "$\tilde{B}_k$". However, since the multiplier estimates are obtained from (3.11) and not from

$$\lambda_k := -\left[Y_k^T A_k\right]^{-1} Y_k^T \nabla\varphi_\mu(x_k)$$

   as in [13], an additional increase of the penalty parameters might still be necessary to ensure descent properties.

3. If multipliers are not available, an update rule similar to (2.31) can be used to obtain estimates for the penalty parameter in $\phi_{\mu,\nu}$:

$$\nu_k := \begin{cases} \nu_{k+1} & \text{if } (1-\rho)\nu_k\|c_k\| \geq |\nabla\varphi_\mu(x_k)^T q_k| - \beta_k \\ \frac{|\nabla\varphi_\mu(x_k)^T q_k| - \beta_k}{(1-\rho)\|c_k\|} + \epsilon_\nu & \text{otherwise} \end{cases} \tag{3.24}$$

   with constants $\epsilon_\nu > 0$ and $\rho \in (0,1)$. Here, β_k is zero, if the damping parameter ζ_k for the cross term w_k in (3.9) is chosen according to (2.32), with "g_k" replaced by "$\nabla\varphi_\mu(x_k)$" and "B_k" replaced by "$\tilde{B}_k$". This choice is in accordance with earlier work in the context of an active set SQP method [15]. However, within the presented barrier method, w_k defined by (3.13) or (3.14) contains curvature

information for the barrier term, and damping might then result in less efficient search directions. Therefore, also a damping-free version of the above rule has been implemented. Here, we choose $\beta_k := \min\{0, w_k^T \bar{p}_k\}$. In [84] we analyze this option, alternatively with an update without absolute values in (3.24), leading to smaller values of the penalty parameter.

At the early stage of development (e.g. for the results reported in [26]), we instead used the following primal-dual penalty function

$$\phi_\nu^{\mathrm{dp}}(x, v) = \varphi_\mu(x) + \mathcal{V}_\mu(x, v) + \nu\|c(x)\|,$$

with the same penalty parameter updating rules, where the term

$$\mathcal{V}_\mu(x, v) = \sum_{i \in \mathcal{I}} \left(x^{(i)} v^{(i)} - \mu \ln(x^{(i)} v^{(i)}) \right)$$

penalizes the deviation of the primal-dual iterates from the relaxed complementarity condition (2.40c). Since the penalty function then measures the progress in both primal and dual variables, α_k and α_k^v have to be chosen identical. The properties of $\phi_\nu^{\mathrm{dp}}(x, v)$ are discussed in [84].

The function $\mathcal{V}_\mu$ is for example used in [1] to enforce complementarity with strictly convex inequality constraints. However, in the context of IPOPT, where all inequalities are simple bound constraints, the (relaxed) complementarity condition (2.40c) is bi-linear, and therefore convergence of the dual variables v_k follows automatically from convergence of the primal variables x_k, if the step sizes α_k and α_k^v in (3.18) become one for sufficiently large k. As a consequence, the term $\mathcal{V}$ is not necessary, and on the contrary in practice imposes a stronger requirement, leading in general to smaller (and less efficient) step sizes.

All of the above merit functions can suffer from the *Maratos effect*, where even arbitrarily close to a strict local solution the value of both the barrier function $\varphi_\mu(x)$ and the constraint violation $\|c(x)\|$ are increased by a full step. As a remedy, second order correction steps (as described in Section 3.4.3) are used. Alternatively, the

code also includes an implementation of *watchdog technique* by Chamberlain et. al. [27].

3.3.2 The Augmented Lagrangian Function

A merit function that does not suffer the Maratos effect is the augmented Lagrangian function (2.17), which for the barrier problem (2.38) becomes

$$\ell_{\mu,\nu}(x,\lambda) := \varphi_\mu(x) + \lambda^T c(x) + \nu c(x)^T c(x). \tag{3.25}$$

In IPOPT, the line search procedure proposed by Biegler and Cuthrell [12] using (3.25) has been implemented. As an important feature, the update of the penalty parameter ν is performed in a non-monotone manner, trying to allow step sizes α_k as large as possible. Global convergence for this approach has not been shown, although it seems to work well in practice (see Section 5.1.2).

When using (3.25), also iterates for λ have to be maintained. For the reduced space version, the corresponding search direction is obtained as $d_k^\lambda := \lambda_k - \tilde{\lambda}_k$ with λ_k from (3.11), and $\tilde{\lambda}_k$ is the current iterate.

3.3.3 Failure of Global Convergence

During the development and analysis of the merit function based line search options just described, we found a simple example problem, where IPOPT failed in an unexpected way. When trying to solve the problem

$$\begin{aligned}
\min_{x\in\mathbb{R}^3} \quad & x^{(1)} && \text{(3.26a)}\\
\text{s.t.} \quad & \left(x^{(1)}\right)^2 - x^{(2)} - 1 = 0 && \text{(3.26b)}\\
& x^{(1)} - x^{(3)} - 0.5 = 0 && \text{(3.26c)}\\
& x^{(2)}, x^{(3)} \geq 0 && \text{(3.26d)}
\end{aligned}$$

starting from the point $x_0 = (-2, 3, 1)$, which satisfies the first equality constraint (3.26b) and strictly satisfies the bound constraints (3.26d), IPOPT generates the

k	$x_k^{(1)}$	$x_k^{(2)}$	$x_k^{(3)}$	α_k
0	-2	3	1	–
1	-1.8077165354330709	2.2308661	0.01	0.338
2	-1.1941467232510745	0.0223087	0.0115512	0.264
3	-1.1827848033469259	0.223E-03	0.157E-02	0.125E-01
4	-1.1825077076013453	0.223E-05	0.160E-04	0.109E-02
5	-1.1825049104784042	0.223E-07	0.160E-06	0.111E-04
6	-1.1825048825066040	0.223E-09	0.160E-08	0.111E-06
7	-1.1825048822268860	0.223E-11	0.160E-10	0.111E-08
8	-1.1825048822240889	0.223E-13	0.160E-12	0.111E-10
9	-1.1825048822240609	0.223E-15	0.160E-14	0.111E-12

Table 3.1: Iterates for counter example

sequence of iterates as shown in Table 3.1, and aborts the optimization because the step size α_k becomes too small. Here, all previously described line search options always accept the first trial step (i.e. $\alpha_k = \alpha_k^{\max}$). The above numbers are obtained using exact second derivatives, but all other choices for the Hessian approximation lead to similar failures.

However, the disconcerting observation is that the example problem (3.26) is well posed: The objective and constraint functions are smooth, the Jacobian of the equality constraints has full rank everywhere, and there is only one stationary point $x_* = (1, 0, 0.5)^T$ with corresponding multipliers $\lambda_* = (-0.5, 0)^T$ and $(v_*^{(2)}, v_*^{(3)})^T = (0.5, 0)^T$, which satisfies the SOS conditions and is the global solution to the problem.

In order to discuss the reason for this failure, let us assume that we have an iterate x_k (such as x_0 from above) with $x_k^{(2)}, x_k^{(3)} > 0$ and $c_1(x_k) \geq 0$ as indicated in Figure 3.1, which shows a projection of the first equality constraint into the space of $x^{(1)}$ and $x^{(2)}$. Taking a full step $x_k + d_k$ with d_k satisfying the linearization of the equality constraints (3.8b) leads into the plotted tangent to the parabola.

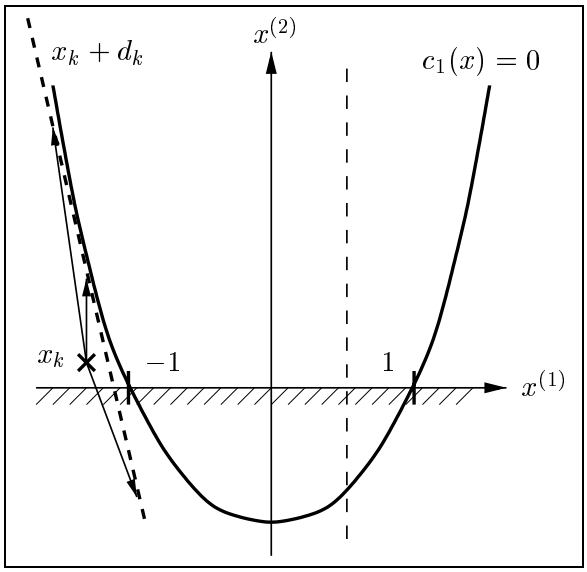

Figure 3.1: Example for new type of global convergence failure

Consequently, the next iterate of IPOPT will lie between the current point and this tangent, or on this tangent, but in order to respect the bound constraint for $x^{(2)}$ in the fraction-to-the-boundary rule (3.19a), it will again lie above the $x^{(1)}$-axis. In other words, from a starting point lying in the region above the $x^{(1)}$-axis and on the left side of the parabola, only points in that same region can be reached by IPOPT. A repetition of this argument shows that the iterates are confined to this region. However, for a point x_* to satisfy the second equality constraint (3.26c) and the bound constraint $x^{(3)} \geq 0$, $x_*^{(1)}$ has to be at least 0.5. Thus, IPOPT as described so far can never converge to a feasible point if started from a point in the described region.

The above argument does not depend on the particular choice of the objective function $f(x)$. Also, the quadratic term in (3.26b) can be replaced by any smooth function $g(x^{(1)})$, that is negative for $x^{(1)} \in (t_1, t_2)$ with $t_1 < t_2$, positive only for $x^{(1)} < t_1$ and $x^{(1)} > t_2$, and convex for $x^{(1)} < t_1$, if the remaining constraints make $x^{(1)} \leq t_1$ infeasible. Interestingly, a more detailed analysis of this example in Section 4.1 will show that the convergence problem also occurs in instances, in which

the parabola in Figure 3.1 lies entirely above the $x^{(1)}$-axis.

Clearly, any optimization method that takes steps that are fractions of search directions satisfying the linearization of the equality constraints (3.8b) has to fail in the same manner. As we will discuss in more detail in Section 4.1, most of the current interior point NLP algorithms in the line search category as well as some trust region methods fail on this innocuous example. As a consequence, the above example has been cited and discussed by other researchers [9, 57, 77].

In the remainder of this chapter we will present a different line search technique for IPOPT that does not suffer the described convergence problem.

3.4 Filter Based Line Search

3.4.1 Introduction

Recently, Fletcher and Leyffer [37] have proposed filter methods, offering an alternative to merit functions, as a tool to guarantee global convergence in algorithms for nonlinear programming. The underlying concept is that trial points are accepted if they improve the objective function *or* improve the constraint violation instead of a combination of those two measures defined by a merit function. The practical results reported for the filter trust region SQP method in [37] are encouraging, and recently global convergence results for related algorithms have been established [35, 38]. Other researchers have also proposed global convergence results for different trust region based filter methods, such as for another trust region SQP method [45], an interior point approach [78], a bundle method for non-smooth optimization [36], and a pattern search algorithm for derivative-free optimization [3].

In the following, a filter method framework based on line search is proposed and analyzed, which can be applied to barrier methods and active set SQP methods. The motivation given by Fletcher and Leyffer for the development of the filter method [37] is to avoid the necessity to determine a suitable value of the penalty parameter in the merit function (2.14). In addition, numerical evidence presented in Section 5.1.2

suggests that Newton directions are usually "good" directions (in particular if exact second derivative information is used), and that a filter approach has the potential to be more efficient than algorithms based on merit functions, as it generally accepts larger steps. However, in the context of a line search method, the filter approach offers another important advantage regarding robustness. It has been known for some time that line search methods can converge to "spurious solutions", infeasible points that are not even critical points for a measure of infeasibility, if the gradients of the constraints become linearly dependent at non-feasible points. In [71], Powell gives an example for this behavior. Another type of convergence problem has just been described in Section 3.3.3. Using a filter approach within a line search algorithm helps to overcome these problems: If the trial step size becomes too small in order to guarantee sufficient progress towards a solution of the problem, the filter method reverts to a feasibility restoration phase, whose goal is to deliver a new iterate that is sufficiently less infeasible. As a consequence, the global convergence problems mentioned above cannot occur.

The discussion of the line search filter approach will be as follows. The derivation will be presented in this chapter in the context of the barrier method IPOPT. Chapter 4 offers a convergence analysis, and in Sections 4.4.1 and 4.4.2 we will show how the presented techniques can be applied to active set SQP methods.

In the following section we will motivate and state the algorithm for the solution of the barrier problem (2.38) with a fixed value of the barrier parameter μ. The method is motivated by the trust region SQP method proposed and analyzed by Fletcher et. al. [35]. An important difference, however, lies in the condition that determines when to switch between certain sufficient decrease criteria; this allows us to show fast local convergence of the proposed line search filter method.

Later in Section 4.2 we prove that every limit point of the sequence of iterates generated by the algorithm is feasible, and that there is at least one limit point that satisfies the first order optimality conditions for the barrier problem. The assumptions made are less restrictive than those made for previously proposed line

search interior point methods for nonlinear programming (e.g. [33, 77, 88]).

As Fletcher and Leyffer pointed out in [37], filter methods can suffer from the so-called Maratos effect [56], which leads to short step sizes arbitrarily close to a solution of the problem, and hence to a poor local convergence behavior. In Section 3.4.3 we will present how second order corrections steps can be integrated into the filter method, and later in Section 4.3 we will show that this indeed guarantees that full steps for the resulting search directions will eventually be accepted in the neighborhood of a strict local solution of the barrier problem satisfying the SOS conditions. As a consequence, fast local convergence can be established for the solution of the barrier problem with a fixed value of the barrier parameter. In Section 3.4.5 we will then briefly describe how the filter method is applied as the barrier parameter is driven to zero.

In Section 3.4.6 we propose an alternative measure for the filter acceptance criteria. Here, a trial point is accepted if it reduces the infeasibility or the value of the Lagrangian function, instead of the (barrier) objective function. The global convergence results still hold for this modification. Having presented the line search filter framework on the example of a barrier method we will finally show in Sections 4.4.1 and 4.4.2 how it can be applied to active set SQP methods, preserving the same global and local convergence properties. In Section 4.4.3 we will briefly point out how our local convergence analysis can also be applied to a slightly modified version of the trust region filter SQP method proposed by Fletcher et. al. [35].

The presented approach differs from the trust region interior point filter algorithm proposed by M. Ulbrich, S. Ulbrich, and Vicente [78] in that the barrier parameter is kept constant for several iterations. This enables us to base the acceptance of trial steps directly on the barrier function instead of only on the norm of the optimality conditions. Therefore the presented method is less likely to converge to saddle points or maxima than the algorithm proposed in [78].

Recently, Benson, Shanno, and Vanderbei [8] proposed several heuristics based on the idea of filter methods, which were integrated into LOQO and for which improved

efficiency compared to the previous merit function approach are reported. Their approach is different from the one proposed here in many aspects, and no global convergence analysis is given.

3.4.2 Description of the Line Search Filter Approach

We now describe the filter line search procedure for solving the barrier problem (2.38) for a fixed value of the barrier parameter μ. Here, we are only concerned with the convergence of the primal variables x_k. If the primal variables converge and full steps are eventually taken for the dual variables v_k, also those converge to the corresponding multiplier estimates, because the relaxed complementarity condition (2.40c) is bi-linear.

Let us assume that the search direction d_k has been computed from (3.3), as well as the maximal step size

$$\alpha_k^{\max} := \max\{\alpha \in (0,1] : x_k + \alpha d_k \geq (1-\tau)x_k\} \tag{3.27}$$

satisfying the fraction-to-the-boundary rule (3.19a). Now we perform a back-tracking line search with decreasing trial step sizes $\alpha_{k,0} = \alpha_k^{\max}, \alpha_{k,1}, \alpha_{k,2}, \ldots$ until the resulting trial point $x_k(\alpha_{k,l})$ as defined in (2.13) satisfies certain conditions, which will be discussed next.

The underlying idea is to interpret the barrier problem (2.38) as a bi-objective optimization problem with two goals: Minimizing the constraint violation

$$\theta(x) = \|c(x)\|$$

and minimizing the barrier function $\varphi_\mu(x)$. A certain emphasis is placed on the first measure, since a point has to be feasible in order to be an optimal solution of the barrier problem. Here, we do not require that a trial point $x_k(\alpha_{k,l})$ provides progress in a merit function such as (3.20), which combines these two goals as a linear combination into one single measure. Instead, the trial point $x_k(\alpha_{k,l})$ is accepted if it improves feasibility, i.e. if $\theta(x_k(\alpha_{k,l})) < \theta(x_k)$, *or* if it improves the barrier function,

i.e. if $\varphi_\mu(x_k(\alpha_{k,l})) < \varphi_\mu(x_k)$. Note, that this criterion is less demanding than the enforcement of decrease in the penalty function (3.20) and will in general allow larger steps.

Of course, this simple concept is not sufficient to guarantee global convergence. Several precautions have to be added as we will outline in the following. (The overall line search filter algorithm is formally stated on page 62.)

1. *Sufficient Reduction.* Line search methods that use a merit function ensure *sufficient* progress towards the solution. For example, they may do so by enforcing the Armijo condition (2.16) for the exact penalty function (3.20) (as described in Sections 2.4 and 3.3.1). Here, we borrow the idea from [35, 38] and replace this condition by requiring that the next iterate provides at least as much progress in one of the measures θ or φ_μ that corresponds to a small fraction of the current constraint violation, $\theta(x_k)$. More precisely, for fixed constants $\gamma_\theta, \gamma_\varphi \in (0, 1)$, we say that a trial step size $\alpha_{k,l}$ provides sufficient reduction with respect to the current iterate x_k, if

$$\theta(x_k(\alpha_{k,l})) \leq (1-\gamma_\theta)\theta(x_k) \tag{3.28a}$$

$$\text{or} \qquad \varphi_\mu(x_k(\alpha_{k,l})) \leq \varphi_\mu(x_k) - \gamma_\varphi\theta(x_k). \tag{3.28b}$$

In a practical implementation, the constants $\gamma_\theta, \gamma_\varphi$ typically are chosen to be small. However, relying solely on this criterion would allow the acceptance of a sequence $\{x_k\}$ that always provides sufficient reduction with respect to the constraint violation (3.28a) and converges to a feasible, but non-optimal point. In order to prevent this, we change to a different sufficient reduction criterion whenever

i) the constraint violation becomes small, i.e.

$$\theta(x_k) \leq \theta_{\text{sml}} \tag{3.29}$$

for some fixed $\theta_{\text{sml}} \in (0, \infty]$, and

ii) for the current trial step size $\alpha_{k,l}$ the *switching condition*

$$m_k(\alpha_{k,l}) < 0 \qquad \text{and} \qquad [-m_k(\alpha_{k,l})]^{s_\varphi}[\alpha_{k,l}]^{1-s_\varphi} > \delta\,[\theta(x_k)]^{s_\theta} \tag{3.30}$$

holds with fixed constants $\delta > 0, s_\theta > 1, s_\varphi > 2s_\theta$, where

$$m_k(\alpha) := \alpha \nabla \varphi_\mu(x_k)^T d_k \tag{3.31}$$

is the linear model of the barrier function φ_μ into direction d_k. We choose to formulate the switching condition (3.30) in terms of a general model $m_k(\alpha)$ as it will allow us to define the algorithm for an alternative measure that replaces "$\varphi_\mu(x)$" in Section 3.4.6.

If both conditions (3.29) and (3.30) hold, instead of insisting on (3.28), we require that an Armijo-type condition for the barrier function,

$$\varphi_\mu(x_k(\alpha_{k,l})) \leq \varphi_\mu(x_k) + \eta_\varphi m_k(\alpha_{k,l}), \tag{3.32}$$

is satisfied (see [35]). Here, $\eta_\varphi \in (0, \frac{1}{2})$ is a fixed constant. It is possible that for several trial step sizes $\alpha_{k,l}$ with $l = 1, \ldots, \tilde{l}$ conditions (3.29) and (3.30), but not (3.32) are satisfied. In this case we note that for smaller step sizes the switching condition (3.30) may no longer be valid, so that the method reverts to the acceptance criterion (3.28).

The switching condition (3.30) deserves some discussion. If the current iterate is close to a feasible but non-optimal point we need to ensure that the accepted step satisfies the Armijo condition (3.32) and, therefore, leads to sufficient progress in the barrier function, so that convergence to a feasible but non-optimal point is prevented. Indeed, at a non-optimal point x_k with sufficiently small infeasibility, Lemma 4.2 on page 89 will show that $m_k(\alpha) \leq -\alpha\epsilon$ for some $\epsilon > 0$ and all $\alpha \in (0, 1]$. Condition (3.30) is then implied if $\alpha_{k,l} > \delta/\epsilon[\theta(x_k)]^{s_\varphi}$, where the right hand side is $o(\theta(x_k))$. This property is essential for the proof of Lemma 4.10 on page 97, which states that the next iterate x_{k+1} has to satisfy the Armijo condition (3.32). On the other hand, if the switching condition (3.30) is true close to a strict local solution x_*^μ of the barrier problem (2.38), then the progress in the barrier function, predicted by the model m_k, needs to be sufficiently large compared to the current constraint violation. This is important because we need to guarantee that a full step d_k, possibly

improved by a second order correction step (see Section 3.4.3), will be accepted. If condition (3.30) is true with $\alpha_{k,0} = 1$, then $\theta(x_k) = O(\|d_k\|^{\frac{s_\varphi}{s_\theta}}) = o(\|d_k\|^2)$. This property is crucial in the discussion of local convergence of the method in Section 4.3. Note that the switching conditions used in [35, 38] do not imply this relationship, but may be adapted; see Section 4.4.3.

2. *Filter as taboo-region.* It is also necessary to avoid cycling. For example, this may occur between two points that alternatingly improve one of the measures, θ and φ_μ, and worsen the other one. For this purpose, Fletcher and Leyffer [37] propose to define a "taboo region" in the half-plane $\{(\theta, \varphi_\mu) \in \mathbb{R}^2 : \theta \geq 0\}$. They maintain a list of $(\theta(x_p), \varphi_\mu(x_p))$-pairs (called *filter*) corresponding to some of the previous iterates x_p. In order to be accepted, they require that a point has to improve at least one of the two measures compared to those previous iterates. In other words, a trial step $x_k(\alpha_{k,l})$ can only be accepted, if

$$\begin{aligned} & \theta(x_k(\alpha_{k,l})) < \theta(x_p) \\ \text{or} \quad & \varphi_\mu(x_k(\alpha_{k,l})) < \varphi_\mu(x_p) \end{aligned}$$

for all $(\theta(x_p), \varphi_\mu(x_p))$ in the current filter.

In contrast to the notation in [35, 37], we will define the filter not as a list but as a *set* $\mathcal{F}_k \subseteq [0, \infty) \times \mathbb{R}$ containing *all* (θ, φ_μ)-pairs that are "prohibited" in iteration k. We will say, that a trial point $x_k(\alpha_{k,l})$ is *acceptable to the filter* if its (θ, φ_μ)-pair does not lie in the taboo-region, i.e. if

$$\Big(\theta(x_k(\alpha_{k,l})), \varphi_\mu(x_k(\alpha_{k,l}))\Big) \notin \mathcal{F}_k. \tag{3.33}$$

During the optimization we will make sure that the current iterate x_k is always acceptable to the current filter $\mathcal{F}_k$.

At the beginning of the optimization, the filter is initialized to be empty, $\mathcal{F}_0 := \emptyset$, or — if one wants to impose an explicit upper bound on the constraint violation — as $\mathcal{F}_0 := \{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq \theta_{\max}\}$ for some $\theta_{\max} > \theta(x_0)$. Throughout the optimization

the filter is then augmented in some iterations after the new iterate x_{k+1} has been accepted. For this, the updating formula

$$\mathcal{F}_{k+1} := \mathcal{F}_k \cup \Big\{(\theta,\varphi) \in \mathbb{R}^2 : \theta \geq (1-\gamma_\theta)\theta(x_k) \quad \text{and} \quad \varphi \geq \varphi_\mu(x_k) - \gamma_\varphi\theta(x_k)\Big\}. \tag{3.34}$$

is used (see also [35]). If the filter is not augmented, it remains unchanged, i.e. $\mathcal{F}_{k+1} := \mathcal{F}_k$. Note, that then $\mathcal{F}_k \subseteq \mathcal{F}_{k+1}$ for all k. This ensures that *all* later iterates will have to provide sufficient reduction with respect to x_k as defined by criterion (3.28), if the filter has been augmented in iteration k.

It remains to decide which iterations should augment the filter. Since one motivation of the filter method is to make it generally less conservative than a penalty function approach, we do not want to augment the filter in every iteration. In addition, as we will see in the discussion of the next safeguard below, it is important that we never include feasible points into the filter. The following rule from [35] is motivated by these considerations.

We will always augment the filter if for the accepted trial step size $\alpha_{k,l}$ the switching condition (3.30) or the Armijo condition (3.32) does not hold, even if $\theta(x_k) > \theta_{\min}$. Otherwise, if the filter is not augmented, the value of the barrier objective function is strictly decreased (see Eq. (4.31) on page 92). To see that this indeed prevents cycling, let us assume for the moment that the algorithms generates a cycle of length l,

$$x_K, x_{K+1}, \ldots, x_{K+l-1}, x_{K+l} = x_K, x_{K+l+1} = x_{K+1}, \ldots \tag{3.35}$$

Since a point x_k can never be reached again, if the filter is augmented in iteration k, the existence of a cycle would imply that the filter is not augmented for all $k \geq K$. However, this would imply that $\varphi_\mu(x_k)$ is a strictly decreasing sequence for $k \geq K$, so that (3.35) cannot be a cycle.

3. *Feasibility restoration phase.* Due to the fact that d_k satisfies the linearization of the constraints (from (3.3)), we have $\theta(x_k(\alpha_{k,l})) < \theta(x_k)$ whenever $\alpha_{k,l} > 0$ is

sufficiently small. It is not guaranteed, however, that there exists a trial step size $\alpha_{k,l} > 0$ that indeed provides *sufficient* reduction as defined by criterion (3.28). Furthermore, if the search direction d_k points outside of the non-negative orthant $\{x \in \mathbb{R} : x^{(i)} \geq 0 \text{ for } i \in \mathcal{I}\}$ and x_k is close to the boundary of this region, it is possible (e.g. in the example problem described in Section 3.3.3) that the first trial step size $\alpha_{k,0} = \alpha_k^{\max}$ is already too small to allow sufficient decrease in θ and φ_μ.

In this situation, where no admissible step size can be found, the method switches to a *feasibility restoration phase*, whose purpose is to find a new iterate x_{k+1} merely by decreasing the constraint violation θ, so that x_{k+1} satisfies (3.28) and is also acceptable to the current filter. The procedure for this feasibility restoration phase is not fixed. It could be any iterative algorithm for decreasing θ, possibly ignoring the objective function, and different methods could be used at different stages of the optimization procedure. In the current implementation of IPOPT we integrated TRON, a gradient projection method for bound constrained optimization developed by Lin and Moré [54], which follows a trust region approach using conjugate gradients. It is applied to the feasibility problem

$$\min_{x \in \mathbb{R}^n} \quad \|c(x)\|_2^2 \tag{3.36a}$$

$$x^{(i)} \geq \epsilon \qquad \text{for } i \in \mathcal{I} \tag{3.36b}$$

in order to obtain a sufficiently less infeasible new iterate x_{k+1} with $\theta(x_{k+1}) < 0.99 \cdot \theta(x_k)$ that both satisfies (3.28) and is acceptable to the current filter. Here, $\epsilon \in \mathbb{R}^n$ is a vector with small positive numbers, since also x_{k+1} has to satisfy the positivity requirement (2.42). However, before calling TRON within IPOPT, we first try the procedure described later in Section 3.4.4, in order to avoid switching to TRON very close to a local solution of the problem.

Since we will make sure that a feasible iterate is never included into the filter, the algorithm for the feasibility restoration phase usually should be able to find a new acceptable iterate x_{k+1} unless it converges to a stationary point of θ. The latter case may be important information for the user, as it indicates that the problem seems (at

least locally) infeasible. If the feasibility restoration phase terminates successfully by delivering a new admissible iterate x_{k+1}, the filter is augmented according to (3.34) to avoid cycling back to the problematic point x_k.

In order to detect the situation where no admissible step size can be found and the restoration phase has to be invoked, we propose the following rule. Consider the case when $\theta(x_k) \leq \theta_{\text{sml}}$ with θ_{sml} from (3.29), and the current trial step size $\alpha_{k,l}$ is still large enough so that the switching condition (3.30) holds for some $\alpha \leq \alpha_{k,l}$. In this case, we will not switch to the feasibility restoration phase, since there is still the chance that a shorter step length might be accepted by the Armijo condition (3.32). Therefore, we can see from the switching condition (3.30) and the definition of m_k (3.31) that we do not want to revert to the feasibility restoration phase if $\nabla\varphi_\mu(x_k)^T d_k < 0$ and

$$\alpha_{k,l} > \frac{\delta[\theta(x_k)]^{s_\theta}}{[-\nabla\varphi_\mu(x_k)^T d_k]^{s_\varphi}}.$$

However, if $\theta(x_k) > \theta_{\text{sml}}$ or if the switching condition (3.30) is not satisfied for the current trial step size $\alpha_{k,l}$ and all shorter trial step sizes, then the decision whether to switch to the feasibility restoration phase is based on the linear approximations

$$\theta(x_k + \alpha d_k) \approx \theta(x_k) - \alpha\theta(x_k) \qquad (\text{since } A_k^T d_k + c_k = 0) \tag{3.37a}$$

$$\varphi_\mu(x_k + \alpha d_k) \approx \varphi_\mu(x_k) + \alpha\nabla\varphi_\mu(x_k)^T d_k \tag{3.37b}$$

of the two measures. This predicts that the sufficient decrease condition for the infeasibility measure (3.28a) may not be satisfied for step sizes with $\alpha_{k,l} \leq \gamma_\theta$. Similarly, in case $\nabla\varphi_\mu(x_k)^T d_k < 0$, the sufficient decrease criterion for the barrier function (3.28b) may not be satisfied for step sizes satisfying

$$\alpha_{k,l} \leq \frac{\gamma_\varphi\theta(x_k)}{-\nabla\varphi_\mu(x_k)^T d_k}.$$

We can summarize this in the following formula for a minimal trial step size

$$\alpha_k^{\min} := \gamma_\alpha \cdot \begin{cases} \min\left\{\gamma_\theta, \frac{\gamma_\varphi\,\theta(x_k)}{-\nabla\varphi_\mu(x_k)^T d_k}, \frac{\delta[\theta(x_k)]^{s_\theta}}{[-\nabla\varphi_\mu(x_k)^T d_k]^{s_\varphi}}\right\} \\ \qquad\qquad \text{if } \nabla\varphi_\mu(x_k)^T d_k < 0 \text{ and } \theta(x_k) \le \theta_{\text{sml}} \\ \min\left\{\gamma_\theta, \frac{\gamma_\varphi\,\theta(x_k)}{-\nabla\varphi_\mu(x_k)^T d_k}\right\} \\ \qquad\qquad \text{if } \nabla\varphi_\mu(x_k)^T d_k < 0 \text{ and } \theta(x_k) > \theta_{\text{sml}} \\ \gamma_\theta \qquad\qquad \text{otherwise} \end{cases} \tag{3.38}$$

and switch to the feasibility restoration phase when $\alpha_{k,l}$ becomes smaller than $\alpha_k^{\min}$. Here, $\gamma_\alpha \in (0, 1]$ is a safety-factor that allows for inaccuracy of the linear prediction (3.37).

It is possible, however, to employ more sophisticated rules for the decision when to switch to the feasibility restoration phase while still retaining the convergence analysis in Chapter 4. These rules, for example, they could be based on higher order approximations of θ and/or φ_μ. We only need to ensure that the method does not switch to the feasibility restoration phase as long as (3.30) holds for a step size $\alpha \le \alpha_{k,l}$ where $\alpha_{k,l}$ is the current trial step size, and that the backtracking line search procedure is finite, i.e. it eventually either delivers a new iterate x_{k+1} or reverts to the feasibility restoration phase.

The proposed method also allows to switch to the feasibility restoration phase in any iteration, in which the infeasibility $\theta(x_k)$ is not too small. For example, this might be necessary, when the Jacobian of the constraints A_k^T is (nearly) rank-deficient, so that the linear system (3.3) is (nearly) singular and no search direction can be computed from (3.3). In the context of IPOPT, iterations with $\delta_2 > 0$ in (3.7) can be interpreted as such feasibility restoration steps.

We are now ready to formally state the overall algorithm for solving the barrier problem (2.38).

Algorithm FILTER

Given: Starting point $x_0 > 0$; constants $\theta_{\max} \in (\theta(x_0), \infty]$; $\theta_{\text{sml}} \in (0, \infty]$; $\gamma_\theta, \gamma_\varphi \in (0,1)$; $\delta > 0$; $\gamma_\alpha \in (0,1]$; $s_\theta > 1$; $s_\varphi > 2s_\theta$; $0 < \tau_1 \leq \tau_2 < 1$.

1. *Initialize.*

   Initialize the filter $\mathcal{F}_0 := \{(\theta, \varphi) \in \mathbb{R}^2 : \theta \geq \theta_{\max}\}$ and the iteration counter $k \leftarrow 0$.

2. *Check convergence.*

   Stop, if x_k is a local solution (or at least stationary point) of the barrier problem (2.38), i.e. if it satisfies the KKT conditions (2.39) for some $\lambda \in \mathbb{R}^m$.

3. *Compute search direction.*

   Compute the search direction d_k from the linear system (3.3). If this system is (almost) singular, go to the feasibility restoration phase in Step 9.

4. *Apply fraction-to-the-boundary rule.*

   Compute the maximal step size $\alpha_k^{\max}$ from (3.27).

5. *Backtracking line search.*

   5.1. *Initialize line search.*

   Set $\alpha_{k,0} = \alpha_k^{\max}$ and $l \leftarrow 0$.

   5.2. *Compute new trial point.*

   If the trial step size becomes too small, i.e. $\alpha_{k,l} < \alpha_k^{\min}$ with $\alpha_k^{\min}$ defined by (3.38), go to the feasibility restoration phase in Step 9.

   Otherwise, compute the new trial point $x_k(\alpha_{k,l}) = x_k + \alpha_{k,l} d_k$.

   5.3. *Check acceptability to the filter.*

   If $(\theta(x_k(\alpha_{k,l})), \varphi_\mu(x_k(\alpha_{k,l}))) \in \mathcal{F}_k$, reject the trial step size and go to Step 5.5.

   5.4. *Check sufficient decrease with respect to current iterate.*

5.4.1. *Case I. $\theta(x_k) \leq \theta_{\min}$ and the switching condition (3.30) holds:*

If the Armijo condition for the barrier function (3.32) holds, accept the trial step and go to Step 6.

Otherwise, go to Step 5.5.

5.4.2. *Case II. $\theta(x_k) > \theta_{\min}$ or the switching condition (3.30) is not satisfied:*

If (3.28) holds, accept the trial step and go to Step 6.

Otherwise, go to Step 5.5.

5.5. *Choose new trial step size.*

Choose $\alpha_{k,l+1} \in [\tau_1 \alpha_{k,l}, \tau_2 \alpha_{k,l}]$, set $l \leftarrow l+1$, and go back to Step 5.2.

6. *Accept trial point.*

Set $\alpha_k := \alpha_{k,l}$ and $x_{k+1} := x_k(\alpha_k)$.

7. *Augment filter if necessary.*

If one of the conditions (3.30) or (3.32) does not hold, augment the filter according to (3.34); otherwise leave the filter unchanged, i.e. set $\mathcal{F}_{k+1} := \mathcal{F}_k$.

(Note, that Step 5.3 and Step 5.4.2 ensure, that $(\theta(x_{k+1}), \varphi_\mu(x_{k+1})) \notin \mathcal{F}_{k+1}$.)

8. *Continue with next iteration.*

Increase the iteration counter $k \leftarrow k+1$ and go back to Step 2.

9. *Feasibility restoration phase.*

Compute a new iterate x_{k+1} by decreasing the infeasibility measure θ, so that x_{k+1} satisfies the sufficient decrease conditions (3.28) and is acceptable to the filter, i.e. $(\theta(x_{k+1}), \varphi_\mu(x_{k+1})) \notin \mathcal{F}_k$. Augment the filter according to (3.34) (for x_k) and continue with the regular barrier iteration in Step 8.

Remark 3.1 *From Step 5.5 it is clear that* $\lim_l \alpha_{k,l} = 0$. *In the case that* $\theta(x_k) > 0$ *it can be seen from (3.38) that* $\alpha_k^{\min} > 0$. *If* $\theta(x_k) = 0$ *then the positive definiteness of* H_k *on the null space of* A_k *implies that* $\nabla\varphi_\mu(x_k)^T d_k < 0$ *(see Lemma 4.4 on*

page 90) and, therefore, $\alpha_k^{\min} = 0$*, but Lemma 4.7 on page 95 will show that the Armijo condition (3.32) is satisfied for a sufficiently small step size* $\alpha_{k,l}$*. Thus, the inner loop in Step 5 will terminate in a finite number of trial steps.*

Remark 3.2 *The mechanisms of the filter ensure that* $(\theta(x_k), \varphi_\mu(x_k)) \notin \mathcal{F}_k$ *for all* k*. Furthermore, the initialization of the filter in Step 1 and the update rule (3.34) imply that for all* k *the filter has the following property.*

$$(\bar{\theta}, \bar{\varphi}) \notin \mathcal{F}_k \quad \Longrightarrow \quad (\theta, \varphi) \notin \mathcal{F}_k \text{ if } \theta \leq \bar{\theta} \text{ and } \varphi \leq \bar{\varphi}. \tag{3.39}$$

Remark 3.3 *In Step 3, the search directions* d_k *do not necessarily have to be obtained as solutions of the linear system (3.3) directly. All options discussed in Section 3.2 can also be interpreted as steps satisfying (3.3) by choosing the matrix* H_k *appropriately.*

3.4.3 Second Order Correction Steps

As it has been mentioned by Fletcher and Leyffer [37], the filter approach can still suffer from the Maratos effect [56], even though it is usually less restrictive in terms of accepting steps than a penalty function approach. The Maratos effect occurs if even arbitrarily close to a strict local solution of the barrier problem a full step d_k increases *both* the barrier function φ_μ and the constraint violation θ, and leads therefore not to sufficient progress with respect to the current iterate and is rejected. This can result in poor local convergence behavior. As a remedy, Fletcher and Leyffer propose to improve the step d_k, if it has been rejected, by means of a second order correction which aims to further reduce infeasibility.

In this section will describe how second order correction steps can be incorporated into the filter line search approach. Later in Section 4.3 we will show this indeed prevents the Maratos effect.

Let us now outline the procedure for the second order correction.

If in iteration k

i) $\alpha_k^{\max} = 1$ with $\alpha_k^{\max}$ defined in (3.27),

ii) the first trial step size $\alpha_{k,0} = 1$ has been rejected in Step 5.3 or Step 5.4 of Algorithm FILTER, and

iii) $\theta(x_k) \leq \theta_{\text{soc}}$ for some fixed constant $\theta_{\text{soc}} \in (0, \infty]$,

then, instead of immediately continuing with the selection of a shorter trial step size $\alpha_{k,1}$ in Step 5.5 of Algorithm FILTER, we first compute a second order correction step and accept it if it satisfies our usual acceptance criteria, as outlined next.

Algorithm SOC

5.1*. *Compute second order correction step.*

Solve the linear system

$$\begin{bmatrix} H_k^{\text{soc}} & A_k^{\text{soc}} \\ (A_k^{\text{soc}})^T & 0 \end{bmatrix} \begin{pmatrix} d_k^{\text{soc}} \\ \lambda_k^{\text{soc}} \end{pmatrix} = - \begin{pmatrix} g_k^{\text{soc}} \\ c(x_k + d_k) + c_k^{\text{soc}} \end{pmatrix}, \tag{3.40}$$

(particular admissible choices of $H_k^{\text{soc}}, A_k^{\text{soc}}, g_k^{\text{soc}}, c_k^{\text{soc}}$ are discussed below) to obtain the second order correction step d_k^{soc} and define $\bar{x}_{k+1} := x_k + d_k + d_k^{\text{soc}}$.

5.2*. *Check fraction-to-the-boundary rule.*

If

$$x_k + d_k + d_k^{\text{soc}} \geq (1 - \tau) x_k \tag{3.41}$$

is *not* satisfied, reject second order correction step and continue with Step 5.5 (of Algorithm FILTER).

5.3*. *Check acceptability to the filter.*

If $\bar{x}_{k+1} \in \mathcal{F}_k$, reject second order correction step and go to Step 5.5.

5.4*. *Check sufficient decrease with respect to current iterate.*

5.4.1*. *Case I. $\theta(x_k) \leq \theta_{\min}$ and the switching condition*

$$m_k(1) < 0 \qquad \text{and} \qquad [-m_k(1)]^{s_\varphi} > \delta\,[\theta(x_k)]^{s_\theta} \tag{3.42}$$

holds:

If Armijo condition for barrier function

$$\varphi_\mu(\bar{x}_{k+1}) - \varphi_\mu(x_k) \leq \eta_\varphi m_k(1) \tag{3.43}$$

holds, accept $\bar{x}_{k+1}$ and go to Step 6.

Otherwise, go to Step 5.5.

5.4.2*. *Case II. $\theta(x_k) > \theta_{\min}$ or the switching condition (3.42) is not satisfied:*

If

$$\theta(\bar{x}_{k+1}) \leq (1-\gamma_\theta)\theta(x_k) \tag{3.44a}$$

$$\text{or} \qquad \varphi_\mu(\bar{x}_{k+1}) \leq \varphi_\mu(x_k) - \gamma_\varphi\theta(x_k) \tag{3.44b}$$

hold, accept $x_{k+1} := \bar{x}_{k+1}$ and go to Step 7* below.

Otherwise, go to Step 5.5.

If $\bar{x}_{k+1}$ has been accepted by Algorithm SOC as the next iterate, we also replace Step 7 of Algorithm FILTER by

7*. If one of the conditions (3.42) or (3.43) does not hold, augment the filter according to (3.34); otherwise leave the filter unchanged, i.e. set $\mathcal{F}_{k+1} := \mathcal{F}_k$.

Second order correction steps of the form (3.40) are discussed by Conn, Gould, and Toint in [28, Section 15.3.2.3]. Here we assume that H_k^{soc} is uniformly positive definite on the null space of $(A_k^{\text{soc}})^T$, and that in a neighborhood of a strict local solution we have

$$g_k^{\text{soc}} = o(\|d_k\|), \qquad A_k - A_k^{\text{soc}} = O(\|d_k\|), \qquad c_k^{\text{soc}} = o(\|d_k\|^2). \tag{3.45}$$

In [28], the analysis is made for the particular choices $c_k^{\text{soc}} = 0$, $A_k^{\text{soc}} = A(x_k + p_k)$ for some $p_k = O(\|d_k\|)$, and $H_k = \nabla^2_{xx}\mathcal{L}_\mu(x_k, \lambda_k)$ in (3.3) for multiplier estimates

λ_k. However, the careful reader will be able to verify that the results that we will use from [28] still hold as long as

$$(W_k^\mu - H_k)d_k = o(\|d_k\|), \tag{3.46}$$

if x_k converges to a strict local solution x_*^μ of the barrier problem with corresponding multipliers λ_*^μ, where

$$W_k^\mu = \nabla_{xx}^2 \mathcal{L}_\mu(x_k, \lambda_*^\mu) = \nabla^2 \varphi_\mu(x_k) + \sum_{i=1}^{m} (\lambda_*^\mu)^{(i)} \nabla^2 c^{(i)}(x_k) \tag{3.47}$$

is the Hessian of the Lagrangian function

$$\mathcal{L}_\mu(x, \lambda) := \varphi_\mu(x) + \lambda^T c(x) \tag{3.48}$$

corresponding to the barrier problem (2.38).

Popular choices for the quantities in the computation of the second order correction step in (3.40) that satisfy (3.45) are for example the following.

(a) $H_k^{\text{soc}} = I$, $g_k^{\text{soc}} = 0$, $c_k^{\text{soc}} = 0$, and $A_k^{\text{soc}} = A_k$ or $A_k^{\text{soc}} = A(x_k + d_k)$, which corresponds to a least-square step for the constraints.

(b) $H_k^{\text{soc}} = X_k^2$, $g_k^{\text{soc}} = 0$, $c_k^{\text{soc}} = 0$, and $A_k^{\text{soc}} = A_k$ or $A_k^{\text{soc}} = A(x_k + d_k)$, which corresponds to a least-square step for the constraints in a different norm which takes the proximity to the boundary into account (similar to (2.46)).

(c) $H_k^{\text{soc}} = H_k$, $g_k^{\text{soc}} = 0$, $c_k^{\text{soc}} = 0$, and $A_k^{\text{soc}} = A_k$, which is very inexpensive since this choice allows to reuse the factorization of the linear system (3.3).

(d) H_k^{soc} being the Hessian approximation corresponding to $x_k + d_k$, $g_k^{\text{soc}} = \nabla\varphi_\mu(x_k + d_k) + A(x_k + d_k)^T \lambda_k^+$, $c_k^{\text{soc}} = 0$, and $A_k^{\text{soc}} = A(x_k + d_k)$ which corresponds to the step in the next iteration, supposing that $x_k + d_k$ has been accepted. This choice has the flavor of the watchdog technique [27].

(e) If d_k^{soc} is a second order correction step, and $\bar{d}_k^{\text{soc}}$ is an additional second order correction step (i.e. with "$c(x_k + d_k)$" replaced by "$c(x_k + d_k + d_k^{\text{soc}})$" in (3.40)),

then $d_k^{\text{soc}} + \bar{d}_k^{\text{soc}}$ can be understood as a single second order correction step for d_k (in that case with $c_k^{\text{soc}} \neq 0$). Similarly, several consecutive correction steps can be considered as a single one.

In IPOPT, the options (c) and (d) have been implemented. In contrast to Condition i) on page 65, we even try a second order correction step, if the fraction to the boundary is active for the normal step d_k and may cut the overall step $d_k + d_k^{\text{soc}}$.

3.4.4 Feasibility Restoration Close to a Strict Local Solution

Even close to a strict local solution x_*^μ of the barrier problem (2.38) the feasibility restoration phase may still be invoked (see Remark 4.1 on page 100). In order to avoid that the new iterate x_{k+1}, returned from the feasibility restoration phase, diverts from x_*^μ, we propose the following procedure. If the restoration phase is invoked at points where the KKT error (the norm of the left hand side of (2.39) or for a primal-dual method (2.40)) is small, continue to take steps into the usual search directions d_k from (3.3) (now within the restoration phase), as long as the KKT error is decreased by a fixed fraction. If this is not possible, we have to revert to a different algorithm for the feasibility restoration phase; e.g. TRON within IPOPT. If x_k is sufficiently close to a strict local solution x_*^μ for the barrier problem (2.38) satisfying the SOS conditions, then x_*^μ is a point of attraction for Newton's method, so that this procedure will be able to deliver a new iterate x_{k+1} which is sufficiently feasible in order to be accepted by the current filter and at the same time approaches x_*^μ, so that overall $\lim_k x_k = x_*^\mu$ is guaranteed.

In IPOPT, this reduction of the KKT error is always tried before calling TRON, even if the current error is not small.

3.4.5 Filter Modification as $\mu \to 0$

Local convergence of barrier methods has been discussed by other researchers, in particular by Nash and Sofer [66] for primal methods, and by Gould, Orban, Sarte-

naer, and Toint [48, 47] for primal-dual methods. In their approaches, the barrier problem (2.38) is solved to a certain tolerance $\epsilon > 0$ for a fixed value of the barrier parameter μ. The parameter μ is then decreased and the tolerance ϵ is tightened for the next barrier problem. It is shown that if the parameters μ and ϵ are updated in a particular fashion (see also Section 3.1), a starting point enhanced by extrapolation will eventually solve the next barrier problem already to the new tolerance ϵ. Then the barrier parameter μ will be decreased again immediately, thus leading to superlinear convergence rate of the overall interior point algorithm for solving the original NLP (2.1). These techniques can also be applied to the barrier algorithm proposed in this dissertation. For this, the current filter is simply deleted and Algorithm FILTER is re-started in Step 1 after the value of the barrier parameter μ has been changed and the improved starting point for the new barrier problem has been determined.

3.4.6 An Alternative Algorithm Based on Augmented Lagrangian Function

The two measures $\varphi_\mu(x)$ and $\theta(x)$ can be considered as the two components of the exact penalty function (3.20). Another popular merit function is the augmented Lagrangian function (3.25). If λ_*^μ are the multipliers corresponding to a strict local solution x_*^μ of the barrier problem, then there exists a penalty parameter $\nu > 0$, so that x_*^μ is a strict local minimizer of $\ell_{\mu,\nu}(x, \lambda_*^\mu)$.

In the line search filter method described in Section 3.4.2 we can alternatively follow an approach based on the augmented Lagrangian function rather than on the exact penalty function, by dividing the augmented Lagrangian function (3.25) into its two components $\mathcal{L}_\mu(x, \lambda)$ (defined in (3.48)) and $\theta(x)$. In Algorithm FILTER we then replace all occurrences of the measure "$\varphi_\mu(x)$" by "$\mathcal{L}_\mu(x, \lambda)$". In addition to the iterates x_k we now also keep iterates λ_k as estimates of the equality constraint multipliers, and compute in each iteration k a search direction d_k^λ for those variables,

see Eq. (2.11). Defining

$$\lambda_k(\alpha_{k,l}) := \lambda_k + \alpha_{k,l} d_k^\lambda,$$

the sufficient reduction criteria (3.28b) and (3.32) are then replaced by

$$\begin{aligned} \mathcal{L}_\mu(x_k(\alpha_{k,l}), \lambda_k(\alpha_{k,l})) &\leq \mathcal{L}_\mu(x_k, \lambda_k) - \gamma_\varphi \theta(x_k) \qquad \text{and} \\ \mathcal{L}_\mu(x_k(\alpha_{k,l}), \lambda_k(\alpha_{k,l})) &\leq \mathcal{L}_\mu(x_k, \lambda_k) + \eta_\varphi m_k(\alpha_{k,l}), \end{aligned}$$

respectively, where the model m_k for $\mathcal{L}_\mu$ is now defined as

$$\begin{aligned} m_k(\alpha) &:= \alpha \nabla \varphi_\mu(x_k)^T d_k - \alpha \lambda_k^T c(x_k) + \alpha(1-\alpha) c(x_k)^T d_k^\lambda \\ &\approx \mathcal{L}_\mu(x_k + \alpha d_k, \lambda_k + \alpha d_k^\lambda) - \mathcal{L}_\mu(x_k, \lambda_k) \end{aligned} \tag{3.49}$$

which is obtained by Taylor expansion of $\varphi_\mu(x)$ and $c(x)$ around x_k into direction d_k and the use of (3.3).

The switching condition (3.30) remains unchanged, but the definition of the minimum step size (3.38) has to be changed accordingly. The only requirements for this change are again that it is guaranteed that the method does not switch to the feasibility restoration phase in Step 5.2 as long as the switching condition (3.30) is satisfied for a trial step size $\alpha \leq \alpha_{k,l}$, and that the backtracking line search in Step 5 is finite.

In Section 4.2.4 we will briefly discuss how the global convergence analysis for the original approach can be adapted to this alternative approach.

Chapter 4

Convergence Analysis

This chapter addresses the theoretical aspects regarding the convergence of the method presented in the previous chapter. In Section 4.1 the counter example presented in Section 3.3.3 will be examined in more detail, including a discussion of the implications for algorithms proposed by other researchers. In Section 4.2, the global convergence properties of the line search filter method introduced in Section 3.4 will be proven under fairly mild assumptions. Section 4.3 examines its local convergence properties, showing that the Maratos effect is successfully prevented by the second order correction steps, allowing fast local convergence. Finally, Section 4.4 discusses how the presented filter line search techniques can be applied to active set SQP methods.

4.1 Discussion of a New Type of Global Convergence Failure

In Section 3.3.3, a simple, well-posed example has been presented, on which an unmodified version of IPOPT failed in an unexpected way. In this section a more general version of this example will be analyzed, and the implications for other interior point methods will be discussed. As mentioned in Section 2.8, a number of interior point methods for general nonlinear programming has been proposed over the past years. For some of those theoretical global convergence properties have been proved (see for example [21, 33, 77, 78, 88, 89]).

However, it will now be demonstrated that some of the current methods ([33, 42, 78, 80, 88, 89]), in particular many line search methods, can fail to converge to feasible points when applied to a well-posed problem. In those cases, the limit point of the sequence of iterates can depend arbitrarily on the choice of the starting point.

In the next section, the class of affected algorithms will be defined. The general example problem is introduced in Section 4.1.2, and it is shown that those methods fail to converge to feasible points when started from certain starting points. In Section 4.1.3 the implications of this result will be discussed. In particular, it will be examined how this result is compatible with the global analysis for some of the affected methods, and why some other methods do not fail.

4.1.1 Affected Class of Interior Point Methods for NLP

The algorithms that we want to consider fall into the class of barrier methods as described in Section 2.7. They can be understood as Newton-type methods that generate search directions by linearizing the first order optimality conditions for the barrier problem (2.38), or by linearizing the equivalent primal-dual equations (2.40). The barrier parameter μ is eventually driven to zero, and the iterates are always required to satisfy (2.42).

Let us now define the algorithmic framework of those methods for which we will

demonstrate the convergence problem:

Algorithm GIP (Generalized Interior Point).
Given a (primal) starting point $x_0 \in \mathbb{R}^n$ that strictly satisfies the bound constraints, i.e. $x_0^{(i)} > 0$ for $i \in \mathcal{I}$. Initialize $k := 0$.

1. Compute a (primal) search direction d_k that satisfies the linearization of the equality constraints (2.38b), i.e.

$$A_k^T d_k + c_k = 0. \tag{4.1}$$

2. Determine a step length $\alpha_k \in (0, 1]$, so that the next iterate will still strictly satisfy the bound constraints, i.e.

$$x_{k+1}^{(i)} := x_k^{(i)} + \alpha_k d_k^{(i)} > 0 \qquad \text{for } i \in \mathcal{I}. \tag{4.2}$$

3. Increase iteration counter $k \leftarrow k + 1$, and go to 1.

This framework is fairly general and covers many of the current interior point methods that use line searches (e.g. [26, 33, 42, 80, 88]). The method can be a primal-only or primal-dual method, and is independent of the rule for choosing the barrier parameter μ_k at every iteration, as well as the choice of a particular merit function or line search procedure. There are also trust region methods (e.g. [78, 89]), that can be understood to belong to the class of Algorithm GIP.

In general, however, trust region methods (such as KNITRO [22]) may not belong to this class, since the determination of direction and length of the step is usually not separated into two successive events as above. Also, penalty barrier methods as described by Forsgren and Gill [39] are not of the type Algorithm GIP, since in those line search methods the search directions are obtained by linearizing system (2.40) with equation (2.40b) replaced by

$$c^{(j)}(x) + \mu\lambda^{(j)} = 0 \qquad \text{for } j \in \{1, \ldots, m\},$$

so that the search directions usually do not satisfy condition (4.1).

We will argue in the following sections, that any method belonging to the class of Algorithm GIP cannot be guaranteed to be globally convergent.

4.1.2 Analysis of the General Counter Example

The general version of the counter example (3.26) that we want to consider here is

$$\min_{x\in\mathbb{R}^3} \quad f(x) \tag{4.3a}$$

$$\text{s.t.} \quad c(x) := \begin{pmatrix} c_1(x) \\ c_2(x) \end{pmatrix} = \begin{Bmatrix} \left(x^{(1)}\right)^2 - x^{(2)} + a \\ x^{(1)} - x^{(3)} - b \end{Bmatrix} = 0 \tag{4.3b}$$

$$x^{(2)}, x^{(3)} \geq 0, \tag{4.3c}$$

where $f(x)$ is some objective function, and $a, b \in \mathbb{R}$ are constants with $b \geq 0$. As we will see, the convergence problem is caused by the constraints only so that the particular choice of $f(x)$ is not important.

If for example $f(x) = x_1$, it can easily be seen that there is only one stationary point for this problem, which is the global minimizer of the problem. For $a \neq -b^2$, the sufficient second order optimality conditions and strict complementarity hold at this solution. In addition, the smallest singular value of the Jacobian of the equality constraints, $A(x)^T$, is at least 1 for all $x \in \mathbb{R}^3$, i.e. there is no degeneracy hidden in the equality constraints. In other words, this almost linear example problem is well-posed.

However, the following theorem demonstrates, that a method of the type Algorithm GIP will never converge to a feasible point, if started from within a fairly large region.

Theorem 4.1 *Let $\{x_k\}$ be a sequence generated by Algorithm* GIP *applied to problem (4.3) with a starting point x_0 satisfying $x_0^{(1)} < 0$, $x_0^{(2)}, x_0^{(3)} > 0$; note that this implies $c_2(x_0) < 0$. In addition, assume that $a - rb \leq \min\{0, -\frac{a}{2}\}$ with the ratio $r := \frac{c_1(x_0)}{|c_2(x_0)|} \geq 0$. Then, for all k, we have $x_k^{(1)} < -r$, $c_1(x_k) \geq (r+b)r$, and $c_2(x_k) < -(r+b)$.*

Before we prove this theorem, we want to emphasize that the assumptions made for the starting point x_0 are not very strong: In case $a \leq 0$, r can take any nonnegative value, so that all x_0 with $x_0^{(1)} \leq -\sqrt{x_0^{(2)} - a}$ and $x_0^{(2)}, x_0^{(3)} > 0$ are allowed. In case $a > 0$ (and consequently $b > 0$), we need to ensure $r \geq \frac{3a}{2b}$, which is valid for all

$$x_0^{(1)} \leq -\frac{3a}{4b} - \sqrt{\frac{9a^2}{16b^2} + \frac{a}{2} + x_0^{(2)} + \frac{3a}{2b} x_0^{(3)}} \qquad \text{and} \qquad x_0^{(2)}, x_0^{(3)} > 0,$$

such as $x_0 = (-3, 1, 1)^T$ for $a = b = 1$.

Proof. An essential ingredient for the proof is the ratio of the constraint values $\frac{c_1(x_k)}{|c_2(x_k)|}$, which will be shown to be bounded below by r. For later reference we note that the conditions

$$\frac{c_1(x_k)}{|c_2(x_k)|} \geq r \qquad \text{and} \qquad c_2(x_k) < 0 \tag{4.4}$$

imply

$$\begin{aligned} & (x_k^{(1)})^2 - x_k^{(2)} + a \;\geq\; -r\left(x_k^{(1)} - x_k^{(3)} - b\right) \\ \Longrightarrow \qquad & (x_k^{(1)})^2 + r x_k^{(1)} + (a - rb) \;\geq\; x_k^{(2)} + r x_k^{(3)} > 0 \end{aligned} \tag{4.5}$$

since $x_k^{(2)}, x_k^{(3)} > 0$. Denoting the roots of the quadratic function on the left hand side of (4.5) by $x_-^{(1)}$ and $x_+^{(1)}$ with $x_-^{(1)} \leq x_+^{(1)}$, we see that, since $a - rb \leq 0$, conditions (4.4) imply that

$$\text{either} \qquad x_k^{(1)} < x_-^{(1)} \qquad \text{or} \qquad x_k^{(1)} > x_+^{(1)}. \tag{4.6}$$

Note, that

$$x_-^{(1)} \leq -r \leq 0 \leq x_+^{(1)}, \tag{4.7}$$

where we again used the assumption $a - rb \leq 0$.

In the following we show by induction that for all k

$$(\mathrm{i}_k) \quad c_2(x_k) < 0 \qquad (\mathrm{ii}_k) \quad \frac{c_1(x_k)}{|c_2(x_k)|} \geq r \qquad (\mathrm{iii}_k) \quad x_k^{(1)} < x_-^{(1)}.$$

By assumption, (i_0) and (ii_0) are true, and (iii_0) follows from (4.6), (4.7), and the assumption $x_0^{(1)} < 0$.

Now assume that (i_k)-(iii_k) are true. It can easily be seen that the search directions satisfying condition (4.1) may be expressed in the form

$$d_k = \begin{pmatrix} 0 \\ (x_k^{(1)})^2 - x_k^{(2)} + a \\ x_k^{(1)} - x_k^{(3)} - b \end{pmatrix} + \theta_k \begin{pmatrix} 1 \\ 2x_k^{(1)} \\ 1 \end{pmatrix} \tag{4.8}$$

for some $\theta_k \in \mathbb{R}$, so that

$$x_{k+1}^{(1)} = x_k^{(1)} + \alpha_k \theta_k \tag{4.9a}$$

$$x_{k+1}^{(2)} = x_k^{(2)} + \alpha_k \left((x_k^{(1)})^2 - x_k^{(2)} + a \right) + 2x_k^{(1)} \alpha_k \theta_k \tag{4.9b}$$

$$x_{k+1}^{(3)} = x_k^{(3)} + \alpha_k \left(x_k^{(1)} - x_k^{(3)} - b \right) + \alpha_k \theta_k \tag{4.9c}$$

where $\alpha_k \in (0, 1]$ is the step length chosen to ensure (4.2).

We first show by contradiction that $\alpha_k < 1$. If $\alpha_k = 1$, $x_{k+1}^{(3)} = x_k^{(1)} - b + \theta_k > 0$ would imply that $x_k^{(1)} + \theta_k > b \geq 0$, and hence also $-\theta_k < x_k^{(1)} < -r < 0$ by (iii_k) and equation (4.7). These inequalities, together with $a - rb \leq 0$, yield

$$\begin{aligned} x_{k+1}^{(2)} + r x_{k+1}^{(3)} &= (x_k^{(1)})^2 + a + 2x_k^{(1)}\theta_k + r \left(x_k^{(1)} - b + \theta_k \right) \\ &= \left(x_k^{(1)} + \theta_k \right) \left(x_k^{(1)} + r \right) + (a - rb) + x_k^{(1)} \theta_k < 0. \end{aligned}$$

This is a contradiction to (4.2) and $r \geq 0$. Hence,

$$\alpha_k < 1. \tag{4.10}$$

Further, since c_2 is linear, it is

$$c_2(x_{k+1}) = (1 - \alpha_k)\, c_2(x_k), \tag{4.11}$$

so that $\alpha_k < 1$ and (i_k) validate (i_{k+1}).

Looking at the first constraint, it is easy to verify that

$$c_1(x_{k+1}) = \alpha_k^2 \theta_k^2 + (1 - \alpha_k)\, c_1(x_k) \geq (1 - \alpha_k)\, c_1(x_k). \tag{4.12}$$

Since from (4.11) it is $(1 - \alpha_k) = |c_2(x_{k+1})|/|c_2(x_k)|$ we obtain

$$\frac{c_1(x_{k+1})}{|c_2(x_{k+1})|} \geq \frac{c_1(x_k)}{|c_2(x_k)|} \overset{(\mathrm{ii}_k)}{\geq} r,$$

i.e. we have shown (ii$_{k+1}$).

Finally, we prove (iii$_{k+1}$) by contradiction. Suppose that $x_{k+1}^{(1)} \geq x_-^{(1)}$, so that from (4.6) $x_{k+1}^{(1)} > x_+^{(1)}$, and consequently from (4.9a)

$$\alpha_k \theta_k > x_+^{(1)} - x_k^{(1)}. \tag{4.13}$$

For the case $a \leq 0$ it is

$$x_+^{(1)} \overset{(4.7)}{\geq} 0 \geq -\frac{a}{2x_k^{(1)}}$$

whereas for the other case $a > 0$ we note that $x_-^{(1)} x_+^{(1)} = a - rb \leq -a/2$ by assumption, i.e. $x_-^{(1)} < 0$, and

$$x_+^{(1)} \geq -\frac{a}{2x_-^{(1)}} \overset{(\text{iii}_k)}{>} -\frac{a}{2x_k^{(1)}}.$$

Thus, from (4.13)

$$\begin{aligned} & \alpha_k \theta_k > -\frac{a}{2x_k^{(1)}} - x_k^{(1)} > -\frac{a}{2x_k^{(1)}} - \frac{1}{2} x_k^{(1)} \\ \Longrightarrow \quad & 2x_k^{(1)} \alpha_k \theta_k < -a - (x_k^{(1)})^2, \end{aligned}$$

so that with (4.9b)

$$\begin{aligned} x_{k+1}^{(2)} &< x_k^{(2)} + \alpha_k \left((x_k^{(1)})^2 - x_k^{(2)} + a \right) - a - (x_k^{(1)})^2 \\ &= -(1-\alpha_k) c_1(x_k) \\ &\overset{(\text{ii}_k)}{\leq} -(1-\alpha_k) r \, |c_2(x_k)| \\ &\leq 0, \end{aligned}$$

which contradicts (4.2). Thus, we have shown that (i$_k$)–(iii$_k$) are valid for all k.

Now, from (iii$_k$) and (4.7), we have $x_k^{(1)} < -r$ for all k, and consequently

$$c_2(x_k) = x_k^{(1)} - x_k^{(3)} - b < -r + 0 - b.$$

Also, by induction on (4.12) and (4.11), we have

$$\begin{aligned} c_1(x_k) &\geq \prod_{i=0}^{k-1} (1-\alpha_i) c_1(x_0) \\ -(r+b) > c_2(x_k) &= \prod_{i=0}^{k-1} (1-\alpha_i) c_2(x_0) \end{aligned} \tag{4.14}$$

so that

$$\prod_{i=0}^{k-1}(1-\alpha_i) > \frac{r+b}{|c_2(x_0)|}$$

and

$$c_1(x_k) \geq (r+b)\frac{c_1(x_0)}{|c_2(x_0)|}.$$

□

For the discussion in the next section we want to point out, that the sequence $\{x_k\}$ in Theorem 4.1 is bounded: From (4.14) we see that the linear function $c_2(x_k)$ is monotonically increasing. As a consequence of this and the fact that $x_k^{(1)}$ is bounded above and $x_k^{(3)}$ is bounded below, both $x_k^{(1)}$ and $x_k^{(3)}$ are bounded. The boundedness of $x_k^{(1)}$ and $c_1(x_k) \geq 0$ implies that $x_k^{(2)}$ is also bounded.

4.1.3 Discussion

Theorem 4.1 shows, that if the starting point x_0 is chosen such that $b + r > 0$, Algorithm GIP cannot reach any feasible points, even asymptotically, since the value of $c_2(x)$ is bounded away from zero. If in addition $r > 0$, the values of *both* equality constraints are bounded away from zero, and by choosing appropriate starting points, their values can be kept arbitrarily large.

One might hope that the limit points of the sequence of iterates $\{x_k\}$ generated by Algorithm GIP are at least minimizers for the constraint violation $\|c(x)\|$ in some norm, and that in this sense they are points at which the problem seems "locally infeasible". But for $r > 0$ it is easy to see, that at every limit point x_* of $\{x_k\}$ the violation of both equality constraints can be made smaller by increasing $x_*^{(1)}$ and keeping $x_*^{(2)}$ and $x_*^{(3)}$ constant. One can even show, that for all $1 \leq p \leq \infty$ the directional derivative at x_* of the infeasibility measure $\Theta_p(x) := \|c(x)\|_p$ in the direction $d_* = (1,0,0)^T$ is negative. In other words, d_* is a descent direction for the constraint violation (measured in *any* ℓ_p-norm), and since it does not affect the bounded variables, it is "compatible" with the bounds in the sense that a step into this direction does not lead to a violation of the bound constraints. There are,

however, methods (see below) that can be shown not to produce such limit points, which seem arbitrary and do not provide any useful insight to the user.

In numerical tests on this problem we observed that the methods that we applied and that belong to the class of Algorithm GIP (namely IPOPT with exact penalty function line search, see Section 3.3.3, and an earlier version of LOQO as described in [80]), converge quickly to points of the form $\bar{x} = (\bar{x}_1, 0, 0)$, i.e. the iterates "crashed into their bounds." The first component could be almost arbitrarily chosen within a sufficiently negative range by changing the starting point. Note, that those limit points are degenerate in the sense that the gradients of the active constraints (both equality constraints and both bound constraints) are linearly dependent.

What property of problem (4.3) could be responsible for the convergence problem? In the proof of Theorem 4.1 we showed for all iterates x_k that independent of the particular choice of a search direction d_k satisfying the linearized equality constraints (4.1), it is never possible to take a full step (see (4.10)). In other words, one can show that Algorithm GIP is confined to a region where at all points the linearized equality constraints (4.1) together with the linearized bound constraints

$$x_k^{(i)} + d_k^{(i)} \geq 0 \qquad \text{for } i \in \mathcal{I}$$

form an inconsistent system of equations and inequalities. This situation is a well-known difficulty for another class of optimization methods. In an SQP framework (see Section 2.3) this inconsistency would be detected immediately, since the QP (2.12), which needs to be solved at an iterate x_k in order to compute a search direction, would be infeasible. In order to continue with the optimization process at all, one would relax some of the constraints or switch to a restoration phase, which tries to find a feasible point. In an interior point framework, however, the immediate detection of such inconsistency is not obvious, and in many other cases the methods will eventually reach points where the constraints are consistent. But here it seems that search directions, which are confined to the affine subspace defined by equation (4.1), cannot take the bound constraints (2.1c) sufficiently into account.

The conflicting conditions (4.1) and (4.2) prevent us from taking steps with sufficient progress towards stationary points for feasibility.

In summary, we claim that interior point methods for solving NLPs of the form (2.1) following Algorithm GIP are not globally convergent since they can fail to converge to a meaningful point if applied to a well-posed problem. For some affected methods, as those in [33, 78, 88, 89], global convergence results have been published. In the following discussion, we analyze which of the assumptions made for those results are violated when the methods are applied to problem (4.3).

The algorithm proposed by El-Bakry et. al. [33] solves the problem formulation (4.3) with $f(x) = x_1$ according to Algorithm GIP after introducing slack variables (x_2 and x_3) for the original problem

$$\min_{x_1 \in \mathbb{R}} \quad x_1 \tag{4.15a}$$

$$\text{s.t.} \quad g(x) := \left\{ \begin{array}{c} (x^{(1)})^2 + a \\ x^{(1)} - b \end{array} \right\} \geq 0. \tag{4.15b}$$

It has been shown under certain assumptions (see Assumptions (C1)–(C4) on p. 532 in [33]) that all limit points of the sequence $\{x_k\}$ generated by this algorithm are KKT points for problem (4.15) or (4.3). Since $\{x_k\}$ is bounded — as shown in the last paragraph of Section 4.1.2 — three of those assumptions, namely (C1)–(C3), are readily satisfied. Thus, the remaining "Regularity Assumption" (C4) must be violated. This assumption basically states that the gradients of those inequality constraints (4.15b) belonging to slack variables that are not bounded away from zero, are linearly independent. Since the (one-dimensional) gradients of (4.15b) taken individually are always linearly independent, a violation of assumption (C4) implies that the slack variables for both inequality constraints are not bounded away from zero, i.e. this method will generate a sequence $\{x_k\}$ with $\liminf_{k\to\infty} x_k^{(2)} = \liminf_{k\to\infty} x_k^{(3)} = 0$, which is confined to an infeasible region.

The analysis for the filter method proposed by Ulbrich, Ulbrich, and Vicente [78] makes essentially the same assumptions, here directly stated as uniform boundedness

of inverse of the matrix in (3.2), a property implied by the assumptions made in [33]. We should point out, however, that also the feasibility restoration phase proposed in [78] takes steps according to Algorithm GIP. If a different method is chosen to perform this task, the convergence problem can be prevented.

The algorithm analyzed by Yamashita [88], which follows a barrier approach, is shown — for a fixed value of the barrier parameter μ — to generate a sequence of iterates, whose limit points are stationary points for the barrier problem (2.38). However, as shown above in Theorem 4.1, this cannot be the case for all admissible starting points. The assumptions made for the global convergence result (see Theorem 6.3 in [88]) can be guaranteed to be satisfied, except for the assumption, that the (fixed) penalty parameter ν for the exact ℓ_1-penalty function (3.20) is always larger than $\|\lambda_k\|_\infty$, where λ_k is the estimate of the multipliers for the equality constraints (2.38b) at iteration k. These multiplier estimates are treated as iterates, so that a violation of the latter assumption indicates that the sequence $\{\lambda_k\}$ must be unbounded. Thus, this assumption, like the "Regularity Assumption" in [33], pertains to the *behavior of the algorithm* rather than to the *problem statement* itself. A similar argument applies to the trust region method proposed by Yamashita, Yabe, and Tanabe [89].

There are, however, interior point methods that can be shown in theory as well as in practice to solve example (4.3) without any problems, such as the trust region method KNITRO analyzed by Byrd, Gilbert, and Nocedal [21]; see also Section 2.8 of a brief description. In contrast to Algorithm GIP, this method is not restricted to take steps that are fractions of a direction leading into the linearized constraints (4.1). Instead, at an iterate x_k, the generated step d_k is guaranteed to make at least as much progress in the linearized feasibility measure $\tilde{\Theta}_2^2(d) := \left\|A_k^T d + c_k\right\|_2^2$ as a *scaled Cauchy step* q_k^{cs} [21]. This step q_k^{cs} is defined as a step into the direction of steepest descent for $\tilde{\Theta}_2^2(d)$ after an affine scaling of the variables. As a consequence of this scaling, this direction is "bent away from the boundary" in the sense that it is the smaller for a component i of x_k, the closer $x_k^{(i)}$ is to its bound. In this way,

the proximity of x_k to the boundary of the region defined by the bound constraints (2.1c) is taken into account during the determination of the search *direction*. The length of q_k^{cs} is chosen to minimize $\tilde{\Theta}_2^2(d)$ subject to the trust region and a condition that ensures that $x_k + q_k^{\mathrm{cs}}$ is sufficiently within the bounds. Overall, this procedure allows to show under very mild assumptions, that any limit point x_* generated by this algorithm is a stationary point for the feasibility measure $\Theta_2^2(x) = \|c(x)\|_2^2$ under consideration of active bounds, i.e. $\frac{\partial(\Theta_2^2)}{\partial x^{(i)}}(x_*) = 0$ for all i with $x_*^{(i)} > 0$ or $i \notin \mathcal{I}$.

After the initial publication of the counter example and the above discussion in [83], several researchers have addressed this global convergence problem. It has been discussed by Marazzi and Nocedal [57] as an example where nonlinear optimization methods fail to exert "feasibility control". Benson, Shanno, and Vanderbei [9] propose a shifting of slack variables s, w, p in (2.50) as a remedy for LOQO, which seems to overcome the convergence problem in practice and is implemented in the current version of LOQO. Tits, Urban, Bakhtiari, and Lawrence [77] stress in their analysis, that their proposed line search interior point method does not suffer from the above convergence problem. Here, equality constraints are handled by relaxing them into inequality constraints that are penalized on the other side by a penalty term added to the objective function. As a consequence, the generated search directions do not necessarily satisfy the linearization of the equality constraints (4.1).

IPOPT encounters no problems solving (3.26) using the filter approach, requiring a total for 23 iterations, of which 4 are taken within TRON.

Interestingly, whereas the iterates in Table 3.1 have been obtained with $\delta_2 = 0$ in (3.7) for all iterations, IPOPT can solve the example problem with a merit function line search (e.g. using the augmented Lagrangian), when δ_2 is set to a small value as in the implemented heuristic — then the linearization (4.1) does no longer hold and the iterates are not trapped in an infeasible region.

4.2 Global Convergence Analysis of Line Search Filter Methods

In this section we will discuss the global convergence properties of the filter line search method stated as Algorithm FILTER in Section 3.4. As before we will assume here for the sake of simplified notation, that $\mathcal{I} = \{1, \ldots, n\}$ in (2.1c). However, it is straightforward to generalize the analysis for the case with fewer bounds.

4.2.1 Assumptions and Preliminary Results

Notation. In the remainder of this chapter we will denote the set of indices of those iterations, in which the filter has been augmented according to (3.34), by $\mathcal{A} \subseteq \mathbb{N}$; i.e.

$$\mathcal{F}_k \subsetneqq \mathcal{F}_{k+1} \qquad \Longleftrightarrow \qquad k \in \mathcal{A}. \tag{4.16}$$

The set $\mathcal{R} \subseteq \mathbb{N}$ will be defined as the set of all iteration indices in which the feasibility restoration phase is invoked. Since Step 9 in Algorithm FILTER makes sure that the filter is augmented in every iteration in which the restoration phase is invoked, we have $\mathcal{R} \subseteq \mathcal{A}$. We will denote with $\mathcal{R}_{\text{inc}} \subseteq \mathcal{R}$ the set of those iteration counters, in which the linear system (3.3) is too inconsistent and the restoration phase is invoked from Step 3.

Let us first state the assumptions necessary for the global convergence analysis of Algorithm FILTER. Since the barrier objective function (2.38a) and its derivatives become unbounded as x_k approaches the boundary of the non-negative orthant $\{x \in \mathbb{R}^n : x \geq 0\}$, it is more convenient to scale the first row and column of the linear system (3.3) by X_k to obtain

$$\begin{bmatrix} \tilde{H}_k & \tilde{A}_k \\ \tilde{A}_k^T & 0 \end{bmatrix} \begin{pmatrix} \tilde{d}_k \\ \lambda_k^+ \end{pmatrix} = - \begin{pmatrix} X_k \nabla f(x_k) - \mu e \\ c_k \end{pmatrix}, \tag{4.17}$$

where $\tilde{A}_k := \tilde{A}_k$ with $\tilde{A}(x) := XA(x)$, $\tilde{d}_k := X_k^{-1} d_k$, and $\tilde{H}_k := X_k H_k X_k$.

Assumptions G. *Let $\{x_k\}$ be the sequence generated by Algorithm* FILTER*, where we assume that the feasibility restoration phase in Step 9 always terminates successfully and that the algorithm does not stop in Step 2 at a first-order optimal point of the barrier problem.*

(G1) *There exists an open set $\mathcal{C} \subseteq \mathbb{R}^n$ with $[x_k, x_k + \alpha_k^{\max} d_k] \subseteq \mathcal{C}$ for all $k \notin \mathcal{R}_{\text{inc}}$, so that f and c are differentiable on $\mathcal{C}$, and their function values, as well as their first derivatives, are bounded and Lipschitz-continuous over $\mathcal{C}$.*

(G2) *The iterates are bounded, i.e. there exists $M_x > 0$ with $\|x_k\| \leq M_x$ for all k.*

(G3) *The matrices H_k approximating the Hessian of the Lagrangian in (3.3) are uniformly bounded for all $k \notin \mathcal{R}_{\text{inc}}$.*

(G4) *There exists a constant θ_{inc}, so that $k \notin \mathcal{R}_{\text{inc}}$ whenever $\theta(x_k) \leq \theta_{\text{inc}}$, i.e. the linear system (3.3) is "sufficiently consistent" at sufficiently feasible points.*

(G5) *There exists a constant $M_A > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$ we have*

$$\sigma_{\min}(\tilde{A}_k) \geq M_A.$$

(G6) *The scaled Hessian approximations $\tilde{H}_k$ are uniformly positive definite on the null space of the scaled Jacobian $\tilde{A}_k^T$. In other words, there exists a constant $M_H > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$*

$$\lambda_{\min}\left(\tilde{Z}_k^T \tilde{H}_k \tilde{Z}_k\right) \geq M_H, \tag{4.18}$$

where the columns of $\tilde{Z}_k \in \mathbb{R}^{n\times(n-m)}$ form an orthonormal basis matrix of the null space of $\tilde{A}_k^T$.

Assumptions (G1) and (G3) merely establish smoothness and boundedness of the problem data. Assumption (G2) may be considered rather strong since it explicitly excludes divergence of the iterates. In particular in an interior point framework this might constitute a problematic issue. However, it is necessary in our analysis to

make this assumption as it guarantees that the barrier objective function (2.38a) is bounded below.

As we will see later in Lemma 4.2, Assumption (G6) ensures a certain descent property and it is similar to common assumptions on the reduced Hessian in SQP line search methods (see e.g. [69]). To guarantee this requirement in a practical implementation, one could compute a QR-factorization of $\tilde{A}_k$ to obtain matrices $\tilde{Z}_k \in \mathbb{R}^{n\times(n-m)}$ and $\tilde{Y}_k \in \mathbb{R}^{n\times m}$ so that the columns of $[\tilde{Z}_k\ \tilde{Y}_k]$ form an orthonormal basis of $\mathbb{R}^n$, and the columns of $\tilde{Z}_k$ are a basis of the null space of $\tilde{A}_k^T$ (see e.g. [42]). Then, as in (2.20), the overall scaled search direction can be decomposed into

$$\tilde{d}_k = q_k + p_k, \tag{4.19}$$

where $q_k := \tilde{Y}_k \bar{q}_k$ and $p_k := \tilde{Z}_k \bar{p}_k$ with

$$\bar{q}_k := -\left[\tilde{A}_k^T \tilde{Y}_k\right]^{-1} c_k \tag{4.20a}$$

$$\bar{p}_k := -\left[\tilde{Z}_k^T \tilde{H}_k \tilde{Z}_k\right]^{-1} \tilde{Z}_k^T \left(X_k \nabla f(x_k) - \mu e + \tilde{H}_k q_k\right) \tag{4.20b}$$

(similar to (2.23) and (3.9)). The eigenvalues for the reduced scaled Hessian in (4.20b) (the term in square brackets) could be monitored and modified if necessary. However, this procedure is prohibitive for large-scale problems, and in those cases one instead might employ heuristics to ensure at least positive definiteness of the reduced Hessian, for example, by monitoring and possibly modifying the inertia of the iteration matrix in (3.3) or (4.17) (as for example proposed in Section 3.2.1). Note, on the other hand, that (4.18) holds in the neighborhood of a local solution x_*^μ satisfying the SOS conditions, if H_k approaches the exact Hessian (3.47) of the Lagrangian of the barrier problem. Then, close to x_*^μ, no eigenvalue correction will be necessary and fast local convergence can be expected, assuming that full steps are taken close to x_*^μ.

The regularity requirement (G5) ensures that, whenever the scaled gradients of the constraints become (nearly) linearly dependent, the method has to switch to the feasibility restoration phase in Step 3. In practice one could monitor the singular

values of $\tilde{Y}_k^T \tilde{A}_k$ in (4.20a), which are identical to the singular values of $\tilde{A}$, as a criterion when to switch to the restoration phase in Step 3.

Note that for $x \geq 0$, rank-deficiency of the scaled Jacobian $XA(x)$, that is $\sigma_{\min}(XA(x)) = 0$, is equivalent to the statement that the gradients of the equality constraints and of the bound constraints active at x,

$$\nabla c_1(x), \ldots, \nabla c_m(x), \qquad \text{and} \qquad e_i \text{ for } i \in \{j : x^{(j)} = 0\}, \tag{4.21}$$

are linearly dependent. With this in mind we can replace Assumptions (G4) and (G5) by the following assumption.

(G5*) *At all* feasible *points x the gradients of the active constraints (4.21) are linearly independent.*

If (G5*) holds, there exists constants $b_1, b_2 > 0$, so that

$$\theta(x_k) \leq b_1 \qquad \Longrightarrow \qquad \sigma_{\min}(\tilde{A}_k) \geq b_2$$

due to the continuity of $\sigma_{\min}(XA(x))$ as a function of x and the assumed boundedness of the iterates. If we now decide to invoke the feasibility restoration phase in Step 3 whenever $\sigma_{\min}(\tilde{A}_k) \leq b_3 \theta(x_k)$ for some fixed constant $b_3 > 0$, then Assumptions (G4) and (G5) hold.

In contrast to most previously analyzed interior point methods for general nonlinear programming (with the exception of [21]), this allows the treatment of degenerate constraints at non-feasible points. Assumption (G5*) is considerably less restrictive than those made in the analysis of [33, 78, 88, 89], where it is essentially required that the gradients of all equality constraints and active inequality constraints (4.21) are linearly independent at *all* points, and not only at all *feasible* points. The assumptions made in [77] are weaker than this, but still require at all points linear independence of the gradients of all *active* equality and inequality constraints. Also note that Assumption (G5*) is satisfied in the problematic example presented in Section 4.1.

Similar to the analysis in [35], we will make use of a *first order criticality measure* $\chi(x_k) \in [0, \infty]$ with the property that, if a subsequence $\{x_{k_i}\}$ of iterates with $\chi(x_{k_i}) \to 0$ converges to a feasible limit point x_*, then x_* corresponds to a first order optimal solution (assuming that certain constraint qualifications such as linear independence of the constraint gradients hold at x_*; see Assumption (G5)). In the case of the barrier method Algorithm FILTER this means that there exist λ_*, so that the KKT conditions (2.39) are satisfied for (x_*, λ_*).

For the convergence analysis of the barrier method we will define the criticality measure for iterations $k \notin \mathcal{R}_{\text{inc}}$ as

$$\chi(x_k) := \|\bar{p}_k\|_2 \,, \tag{4.22}$$

with $\bar{p}_k$ from (4.20b). Note that this definition is unique, since p_k in (4.19) is unique due to the orthogonality of $\tilde{Y}_k$ and $\tilde{Z}_k$, and since $\|\bar{p}_k\|_2 = \|p_k\|_2$ due to the orthonormality of $\tilde{Z}_k$. For completeness, we may define $\chi(x_k) := \infty$ for $k \in \mathcal{R}_{\text{inc}}$.

In order to see that $\chi(x_k)$ defined in this way is indeed a criticality measure under Assumptions G, let us consider a subsequence of iterates $\{x_{k_i}\}$ with $\lim_i \chi(x_{k_i}) = 0$ and $\lim_i x_{k_i} = x_*$ for some feasible limit point x_*. From Assumption (G4) we then have $k_i \notin \mathcal{R}_{\text{inc}}$ for i sufficiently large. Furthermore, from Assumption (G5) and (4.20a) we have $\lim_i \bar{q}_{k_i} = 0$, and then from $\lim_i \chi(x_{k_i}) = 0$, (4.22), (4.20b), and Assumption (G6) we have that

$$\lim_{i\to\infty} \|\tilde{Z}_{k_i}^T \left(X_{k_i} \nabla f(x_{k_i}) - \mu e\right)\| = \lim_{i\to\infty} \|\tilde{Z}_{k_i}^T X_{k_i} \nabla \varphi_\mu(x_{k_i})\| = 0. \tag{4.23}$$

$Z_{k_i} := X_{k_i}^{-1} \tilde{Z}_{k_i}$ is a null space matrix of the unscaled Jacobian $A_{k_i}^T$. If $x_* > 0$, then $X_{k_i}^{-1}$ is uniformly bounded, and from (4.23) we have $\lim_i \|Z_{k_i}^T \nabla \varphi_\mu(x_{k_i})\| = 0$, which is a well-known optimality measure (see e.g. [69]).

However, we also need to consider the case where the l-th component $x_*^{(l)}$ of the limit point x_* is zero. Since $\tilde{Y}_{k_i}$ and $\tilde{Z}_{k_i}$ in (4.19) have been chosen to be orthogonal and $\tilde{Z}_{k_i}$ is an orthonormal basis of the null space of $\tilde{A}_{k_i}^T$, premultiplying a vector by $\tilde{Z}_{k_i}^T$ gives the orthogonal projection of this vector onto the null space of the scaled

Jacobian $\tilde{A}_k^T$ in the scaled space. Therefore, we can write (4.23) equivalently as

$$\lim_{i\to\infty} \left\| \left((I - \tilde{A}_{k_i} \left[\tilde{A}_{k_i}^T \tilde{A}_{k_i} \right]^{-1} \tilde{A}_{k_i}^T \right) (X_{k_i} \nabla f(x_{k_i}) - \mu e) \right\| = 0.$$

Rearranging terms we then obtain

$$\lim_{i\to\infty} X_{k_i} \left(\nabla f(x_{k_i}) - A_{k_i} \left[\tilde{A}_{k_i}^T \tilde{A}_{k_i} \right]^{-1} \tilde{A}_{k_i}^T (X_{k_i} \nabla f(x_{k_i}) - \mu e) \right) = \mu e. \tag{4.24}$$

Since $\sigma_{\min}(\tilde{A}_{k_i})$ is uniformly bounded away from zero due to Assumption (G5), the expression in the large round brackets on the left hand side of (4.24) is bounded, so that the l-th component of the left hand side expression would converge to zero, whereas μ on the right hand side is non-zero. This contradiction shows that $x_* > 0$.

Before we begin the global convergence analysis, let us state some preliminary results.

Lemma 4.1 *Suppose Assumptions G hold. Then there exist constants $M_{\tilde{d}}$, M_d, M_λ, $M_m > 0$, such that*

$$\|\tilde{d}_k\| \le M_{\tilde{d}}, \qquad \|d_k\| \le M_d, \qquad \|\lambda_k^+\| \le M_\lambda, \qquad |m_k(\alpha)| \le M_m \alpha \tag{4.25}$$

for all $k \notin \mathcal{R}_{\text{inc}}$ and $\alpha \in (0, 1]$. Furthermore, there exists a constant $\bar{\alpha}^{\max} > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$ we have $\alpha_k^{\max} \ge \bar{\alpha}^{\max} > 0$.

Proof. From (G1) and (G2) it is clear that the right hand side of (4.17) is uniformly bounded. Additionally, Assumptions (G3), (G5), and (G6) guarantee that the inverse of the matrix in (4.17) exists and is uniformly bounded for all $k \notin \mathcal{R}_{\text{inc}}$. Consequently, the solution of (4.17), $(\tilde{d}_k, \lambda_k^+)$, as well as $d_k = X_k \tilde{d}_k$ are uniformly bounded. Then it also follows that

$$m_k(\alpha)/\alpha = \nabla \varphi_\mu(x_k)^T d_k = (X_k \nabla f(x_k) - \mu e)^T \tilde{d}_k$$

is uniformly bounded.

The fraction-to-the-boundary rule (3.27) can be reformulated as

$$\alpha_k^{\max} \tilde{d}_k \ge -\tau e.$$

Thus, since $\tilde{d}_k$ is uniformly bounded for all $k \notin \mathcal{R}_{\text{inc}}$, $\alpha_k^{\max}$ is uniformly bounded away from zero for all $k \notin \mathcal{R}_{\text{inc}}$. □

The following result shows that the search direction is a direction of sufficient descent for the barrier objective function at points that are sufficiently feasible and non-optimal.

Lemma 4.2 *Suppose Assumptions G hold. Then the following statement is true:*

If $\{x_{k_i}\}$ is a subsequence of iterates for which $\chi(x_{k_i}) \geq \epsilon$ with a constant $\epsilon > 0$ independent of i, then there exist constants $\epsilon_1, \epsilon_2 > 0$, such that

$$\theta(x_{k_i}) \leq \epsilon_1 \qquad \Longrightarrow \qquad m_{k_i}(\alpha) \leq -\epsilon_2 \alpha.$$

for all i and $\alpha \in (0, 1]$.

Proof. Consider a subsequence $\{x_{k_i}\}$ with $\chi(x_{k_i}) = \|\bar{p}_{k_i}\|_2 \geq \epsilon$. Then, by Assumption (G4), for all x_{k_i} with $\theta(x_{k_i}) \leq \theta_{\text{inc}}$ we have $k_i \notin \mathcal{R}_{\text{inc}}$. Furthermore, with $q_{k_i} = O(\|c(x_{k_i})\|)$ (from (4.20a) and Assumption (G5)) it follows that for $k_i \notin \mathcal{R}_{\text{inc}}$

$$\begin{aligned}
m_{k_i}(\alpha)/\alpha &= \nabla\varphi_\mu(x_{k_i})^T d_{k_i} \\
&= \nabla\varphi_\mu(x_{k_i})^T X_{k_i} \tilde{d}_{k_i} \\
&\overset{(4.19)}{=} \nabla\varphi_\mu(x_{k_i})^T X_{k_i} \tilde{Z}_{k_i} \bar{p}_{k_i} + \nabla\varphi_\mu(x_{k_i})^T X_{k_i} q_{k_i} \\
&\overset{(4.20b)}{=} -\bar{p}_{k_i}^T \left[\tilde{Z}_{k_i}^T \tilde{H}_{k_i} \tilde{Z}_{k_i}\right] \bar{p}_{k_i} - \bar{p}_{k_i}^T \tilde{Z}_{k_i}^T \tilde{H}_{k_i} q_{k_i} \\
&\qquad + \left(X_{k_i} \nabla f(x_{k_i}) - \mu e\right)^T q_{k_i} \qquad (4.26) \\
&\overset{(G3),(G6)}{\leq} -c_1 \|\bar{p}_{k_i}\|_2^2 + c_2 \|\bar{p}_{k_i}\|_2 \|c(x_{k_i})\| + c_3 \|c(x_{k_i})\| \\
&\leq \chi(x_{k_i}) \left(-\epsilon\, c_1 + c_2 \theta(x_{k_i}) + \frac{c_3}{\epsilon} \theta(x_{k_i})\right)
\end{aligned}$$

for some constants $c_1, c_2, c_3 > 0$, where we used $\chi(x_{k_i}) \geq \epsilon$ in the last inequality. If we now define

$$\epsilon_1 := \min\left\{\theta_{\text{inc}}, \frac{\epsilon^2\, c_1}{2(c_2\, \epsilon + c_3)}\right\},$$

it follows for all x_{k_i} with $\theta(x_{k_i}) \leq \epsilon_1$ that

$$m_{k_i}(\alpha) \leq -\alpha \frac{\epsilon\ c_1}{2} \chi(x_{k_i}) \leq -\alpha \frac{\epsilon^2\ c_1}{2} =: -\alpha\epsilon_2.$$

□

Lemma 4.3 *Suppose Assumptions (G1) and (G2) hold. Then there exist constants $C_\theta, C_\varphi > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$ and $\alpha \leq \alpha_k^{\max}$*

$$|\theta(x_k + \alpha d_k) - (1-\alpha)\theta(x_k)| \leq C_\theta \alpha^2 \|d_k\|^2 \tag{4.27a}$$

$$|\varphi_\mu(x_k + \alpha d_k) - \varphi_\mu(x_k) - m_k(\alpha)| \leq C_\varphi \alpha^2 \|\tilde{d}_k\|^2. \tag{4.27b}$$

Since the proof of this lemma is similar to the proof of Lemma 3.1 in [21], we omit it for the sake of brevity.

Finally, we show that Step 9 (feasibility restoration phase) of Algorithm FILTER is well-defined. Unless the feasiblity restoration phase terminates at a stationary point of the constraint violation it is essential that reducing the infeasibility measure $\theta(x)$ eventually leads to a point that is acceptable to the filter. This is guaranteed by the following lemma which shows that no (θ, φ)-pair corresponding to a feasible point is even included in the filter.

Lemma 4.4 *Suppose Assumptions G hold. Then*

$$\theta(x_k) = 0 \quad \Longrightarrow \quad m_k(\alpha) < 0 \qquad \textit{and} \tag{4.28}$$

$$\Theta_k := \min\{\theta : (\theta, \varphi) \in \mathcal{F}_k\} > 0 \tag{4.29}$$

for all k and $\alpha \in (0, 1]$.

Proof. If $\theta(x_k) = 0$, we have from Assumption (G4) that $k \notin \mathcal{R}_{\text{inc}}$. In addition, it then follows $\chi(x_k) > 0$ because Algorithm I would have terminated otherwise in Step 2. Considering the decomposition (4.19), it follows with (4.26) that

$$\frac{m_k(\alpha)}{\alpha} = \nabla\varphi_\mu(x_k)^T d_k \leq -c_1 \chi(x_k)^2 < 0,$$

i.e. (4.28) holds.

The proof of (4.29) is by induction. It is clear from Step 1 of Algorithm FILTER, that the claim is valid for $k = 0$ since $\theta_{\max} > 0$. Suppose the claim is true for k. Then, if $\theta(x_k) > 0$ and the filter is augmented in iteration k, it is clear from the update rule (3.34), that $\Theta_{k+1} > 0$, since $\gamma_\theta \in (0, 1)$. If $\theta(x_k) = 0$, Lemma 4.2 implies that the switching condition (3.30) is true for all trial step sizes, so that in Step 5.4 "Case I" is always considered and α_k must have been accepted, because it satisfies (3.32). Consequently, the filter is not augmented in Step 7. Hence, $\Theta_{k+1} = \Theta_k > 0$. □

4.2.2 Feasibility

In this section we will show that under Assumptions G the sequence $\theta(x_k)$ converges to zero, i.e. all limit points of $\{x_k\}$ are feasible.

Lemma 4.5 *Suppose that Assumptions G hold, and that the filter is augmented only a finite number of times, i.e.* $|\mathcal{A}| < \infty$. *Then*

$$\lim_{k\to\infty} \theta(x_k) = 0. \tag{4.30}$$

Proof. Choose K, so that for all iterations $k \geq K$ the filter is not augmented in iteration k; in particular, $k \notin \mathcal{R}_{\text{inc}} \subseteq \mathcal{A}$ for $k \geq K$. From Step 7 in Algorithm FILTER we then have, that for all $k \geq K$ both conditions (3.30) and (3.32) are satisfied for α_k. From (3.30) it follows with M_m from Lemma 4.1 that

$$\delta[\theta(x_k)]^{s_\theta} < [-m_k(\alpha_k)]^{s_\varphi}[\alpha_k]^{1-s_\varphi} \leq M_m^{s_\varphi}\,\alpha_k$$

and hence (since $1 - 1/s_\varphi > 1/2$)

$$c_4[\theta(x_k)]^{s_\theta - \frac{s_\theta}{s_\varphi}} < [\alpha_k]^{1-\frac{1}{s_\varphi}} \qquad \text{with} \qquad c_4 := \left(\frac{\delta}{M_m^{s_\varphi}}\right)^{1-\frac{1}{s_\varphi}},$$

which with (3.32) and again (3.30) implies

$$\begin{aligned}\varphi_\mu(x_{k+1}) - \varphi_\mu(x_k) &\leq \eta_\varphi m_k(\alpha_k) \\ &< -\eta_\varphi \delta^{\frac{1}{s_\varphi}} [\alpha_k]^{1-\frac{1}{s_\varphi}} [\theta(x_k)]^{\frac{s_\theta}{s_\varphi}} \\ &< -\eta_\varphi \delta^{\frac{1}{s_\varphi}} c_4 [\theta(x_k)]^{s_\theta}. \end{aligned} \tag{4.31}$$

Hence, for all $i = 1, 2, \ldots,$

$$\begin{aligned}\varphi_\mu(x_{K+i}) &= \varphi_\mu(x_K) + \sum_{k=K}^{K+i-1} (\varphi_\mu(x_{k+1}) - \varphi_\mu(x_k)) \\ &< \varphi_\mu(x_K) - \eta_\varphi \delta^{\frac{1}{s_\varphi}} c_4 \sum_{k=K}^{K+i-1} [\theta(x_k)]^{s_\theta}.\end{aligned}$$

Since $\varphi_\mu(x_{K+i})$ is bounded below (from Assumptions (G1) and (G2)), the series on the right hand side in the last line is bounded, which in turn implies (4.30). □

The following lemma considers a subsequence $\{x_{k_i}\}$ with $k_i \in \mathcal{A}$ for all i.

Lemma 4.6 *Let $\{x_{k_i}\}$ be a subsequence of iterates, so that the filter is augmented in iteration k_i, i.e. $k_i \in \mathcal{A}$ for all i. Furthermore assume that there exist constants $c_\varphi \in \mathbb{R}$ and $C_\theta > 0$, so that*

$$\varphi_\mu(x_{k_i}) \geq c_\varphi \qquad \text{and} \qquad \theta(x_{k_i}) \leq C_\theta$$

for all i (for example, if Assumptions (G1) and (G2) hold). It then follows that

$$\lim_{i \to \infty} \theta(x_{k_i}) = 0.$$

Proof. We will prove the claim by contradiction. Suppose, that there exists a constant $c_\theta > 0$ so that for a subsequence $\{x_{k_{i_j}}\}$ of $\{x_{k_i}\}$ we have $\theta(x_{k_{i_j}}) \geq c_\theta$. Without loss of generality we can assume that $\theta(x_{k_i}) \geq c_\theta$ for all i. Since $(\theta(x_{k_i}), \varphi_\mu(x_{k_i})) \notin \mathcal{F}_{k_i}$, we can see from the filter update formula (3.34), that the "area" of the new filter $\mathcal{F}_{k_i+1}$ (in the 2-dimensional (θ, φ) plane) is at least by $\Delta_\theta := \gamma_\theta \gamma_\varphi (c_\theta)^2 > 0$ larger than the area of $\mathcal{F}_{k_i}$. This increase of the filter in all iterations k_i by at least Δ_θ implies that the area of the filter becomes infinitely large, so that because of $\varphi_\mu(x_{k_i}) \geq c_\varphi$ and $c_\theta \leq \theta(x_{k_i}) \leq C_\theta$ we obtain $\limsup_i \varphi_\mu(x_{k_i}) = \infty$.

Without loss of generality we can now assume that $\lim_i \varphi_\mu(x_{k_i}) = \infty$ and that $\{\varphi_\mu(x_{k_i})\}$ is monotonically increasing. Since $(\theta(x_{k_{(i+1)}}), \varphi_\mu(x_{k_{(i+1)}})) \notin \mathcal{F}_{k_{(i+1)}} \supseteq \mathcal{F}_{(k_i+1)}$ and $\varphi_\mu(x_{k_{(i+1)}}) \geq \varphi_\mu(x_{k_i})$, it follows from the filter update rule (3.34) applied in iteration k_i that $\theta(x_{k_{(i+1)}}) \leq (1-\gamma_\theta)\theta(x_{k_i}) \leq \theta(x_{k_i}) - \gamma_\theta c_\theta$. From this, it follows that $\lim_i \theta(x_{k_i}) = -\infty$, which is a contradiction to $\theta(x_{k_i}) \geq c_\theta$. Thus, the assumption of the existence of $c_\theta > 0$ must have been wrong, which concludes this proof. □

The previous two lemmas prepare the proof of the following theorem.

Theorem 4.2 *Suppose Assumptions G hold. Then*

$$\lim_{k\to\infty} \theta(x_k) = 0.$$

Proof. In the case, that the filter is augmented only a finite number of times, Lemma 4.5 implies the claim. If in the other extreme there exists some $K \in \mathbb{N}$, so that the filter is updated by (3.34) in *all* iterations $k \geq K$, then the claim follows from Lemma 4.6. It remains to consider the case, where for all $K \in \mathbb{N}$ there exist $k_1, k_2 \geq K$ with $k_1 \in \mathcal{A}$ and $k_2 \notin \mathcal{A}$.

The proof is by contradiction. Suppose, $\limsup_k \theta(x_k) = M > 0$. Now construct two subsequences $\{x_{k_i}\}$ and $\{x_{l_i}\}$ of $\{x_k\}$ in the following way.

1. Set $i \leftarrow 0$ and $k_{-1} = -1$.

2. Pick $k_i > k_{i-1}$ with

$$\theta(x_{k_i}) \geq M/2 \tag{4.32}$$

   and $k_i \notin \mathcal{A}$. (Note that Lemma 4.6 ensures the existence of $k_i \notin \mathcal{A}$ since otherwise $\theta(x_{k_i}) \to 0$.)

3. Choose $l_i := \min\{l \in \mathcal{A} : l > k_i\}$, i.e. l_i is the first iteration in k_i in which the filter is augmented.

4. Set $i \leftarrow i + 1$ and go back to Step 2.

Thus, every x_{k_i} satisfies (4.32), and for each x_{k_i} the iterate x_{l_i} is the first iterate after x_{k_i} for which $(\theta(x_{l_i}), \varphi_\mu(x_{l_i}))$ is included into the filter.

Since (4.31) holds for all $k = k_i, \ldots, l_i - 1 \notin \mathcal{A}$, we obtain for all i

$$\varphi_\mu(x_{l_i}) \leq \varphi_\mu(x_{(k_i+1)}) < \varphi_\mu(x_{k_i}) - \eta_\varphi \delta^{\frac{1}{s_\varphi}} c_4 [M/2]^{s_\theta}. \tag{4.33}$$

This ensures that for all $K \in \mathbb{N}$ there exists some $i \geq K$ with $\varphi_\mu(x_{k_{(i+1)}}) \geq \varphi_\mu(x_{l_i})$ because otherwise (4.33) would imply

$$\varphi_\mu(x_{k_{(i+1)}}) < \varphi_\mu(x_{l_i}) < \varphi_\mu(x_{k_i}) - \eta_\varphi \delta^{\frac{1}{s_\varphi}} c_4 [M/2]^{s_\theta}$$

for all i and consequently $\lim_i \varphi_\mu(x_{k_i}) = -\infty$ in contradiction to the fact that $\{\varphi_\mu(x_k)\}$ is bounded below (from Assumptions (G1) and (G2)). Thus, there exists a subsequence $\{i_j\}$ of $\{i\}$ so that

$$\varphi_\mu(x_{k_{(i_j+1)}}) \geq \varphi_\mu(x_{l_{i_j}}). \tag{4.34}$$

Since $x_{k_{(i_j+1)}} \notin \mathcal{F}_{k_{(i_j+1)}} \supseteq \mathcal{F}_{l_{i_j}}$ and $l_{i_j} \in \mathcal{A}$, it follows from (4.34) and the filter update rule (3.34), that

$$\theta(x_{k_{(i_j+1)}}) \leq (1 - \gamma_\theta)\theta(x_{l_{i_j}}). \tag{4.35}$$

Since $l_{i_j} \in \mathcal{A}$ for all j, Lemma 4.6 yields $\lim_j \theta(x_{l_{i_j}}) = 0$, so that from (4.35) we obtain $\lim_j \theta(x_{k_{i_j}}) = 0$ in contradiction to (4.32). □

4.2.3 Optimality

In this section we will show that Assumptions G guarantee that at least one limit point of $\{x_k\}$ is a first order optimal point for the barrier problem (2.38).

The first lemma shows conditions under which it can be guaranteed that there exists a step length bounded away from zero so that the Armijo condition (3.32) for the barrier function is satisfied.

Lemma 4.7 *Suppose Assumptions G hold. Let $\{x_{k_i}\}$ be a subsequence with $k_i \notin \mathcal{R}_{\text{inc}}$ and $m_{k_i}(\alpha) \leq -\alpha\epsilon_2$ for a constant $\epsilon_2 > 0$ independent of k_i and for all $\alpha \in (0, 1]$. Then there exists some constant $\bar{\alpha} > 0$, so that for all k_i and $\alpha \leq \bar{\alpha}$*

$$\varphi_\mu(x_{k_i} + \alpha d_{k_i}) - \varphi_\mu(x_{k_i}) \leq \eta_\varphi m_{k_i}(\alpha). \tag{4.36}$$

Proof. Let $M_{\tilde{d}}, \bar{\alpha}^{\max}$ and C_φ be the constants from Lemma 4.1 and Lemma 4.3. It then follows from (4.27b) for all $\alpha \leq \bar{\alpha}$ with

$$\bar{\alpha} := \min\left\{\bar{\alpha}^{\max}, \frac{(1-\eta_\varphi)\epsilon_2}{C_\varphi M_{\tilde{d}}^2}\right\}$$

that

$$\begin{aligned} & \varphi_\mu(x_{k_i} + \alpha d_{k_i}) - \varphi_\mu(x_{k_i}) - m_{k_i}(\alpha) \\ \leq\ & C_\varphi \alpha^2 \|\tilde{d}_{k_i}\|^2 \leq \alpha(1-\eta_\varphi)\epsilon_2 \\ \leq\ & -(1-\eta_\varphi) m_{k_i}(\alpha), \end{aligned}$$

which implies (4.36). □

Let us again first consider the "easy" case, in which the filter is augmented only a finite number of times.

Lemma 4.8 *Suppose that Assumptions G hold and that the filter is augmented only a finite number of times, i.e. $|\mathcal{A}| < \infty$. Then*

$$\lim_{k\to\infty} \chi(x_k) = 0.$$

Proof. Since $|\mathcal{A}| < \infty$, there exists $K \in \mathbb{N}$ so that $k \notin \mathcal{A}$ for all $k \geq K$. Suppose, the claim is not true, i.e. there exists a subsequence $\{x_{k_i}\}$ and a constant $\epsilon > 0$, so that $\chi(x_{k_i}) \geq \epsilon$ for all i. From (4.30) and Lemma 4.2 there exist $\epsilon_1, \epsilon_2 > 0$ and $\tilde{K} \geq K$, so that for all $k_i \geq \tilde{K}$ we have $\theta(x_{k_i}) \leq \epsilon_1$ and

$$m_{k_i}(\alpha) \leq -\alpha\epsilon_2 \quad \text{for all} \quad \alpha \in (0, 1]. \tag{4.37}$$

It then follows from (3.32) that for $k_i \geq \tilde{K}$

$$\varphi_\mu(x_{k_i+1}) - \varphi_\mu(x_{k_i}) \leq \eta_\varphi m_{k_i}(\alpha_{k_i}) \leq -\alpha_{k_i}\eta_\varphi\epsilon_2.$$

Reasoning similarly as in proof of Lemma 4.5, one can conclude that $\lim_i \alpha_{k_i} = 0$, since $\varphi_\mu(x_{k_i})$ is bounded below and since $\varphi_\mu(x_k)$ is monotonically decreasing (from (4.31)) for all $k \geq \tilde{K}$. We can now assume without loss of generality that $\tilde{K}$ is sufficiently large, so that $\alpha_{k_i} < \bar{\alpha}^{\max}$ with $\bar{\alpha}^{\max}$ from Lemma 4.1. This means that for $k_i \geq \tilde{K}$ the first trial step $\alpha_{k,0} = \alpha_k^{\max}$ has not been accepted. The last rejected trial step size $\alpha_{k_i,l_i} \in [\alpha_{k_i}/\tau_2, \alpha_{k_i}/\tau_1]$ during the backtracking line search procedure then satisfies (3.30) since $k_i \notin \mathcal{A}$ and $\alpha_{k_i,l_i} > \alpha_{k_i}$. Thus, it must have been rejected because it violates (3.32), i.e. it satisfies

$$\varphi_\mu(x_{k_i} + \alpha_{k_i,l_i} d_{k_i}) - \varphi_\mu(x_{k_i}) > \eta_\varphi m_{k_i}(\alpha_{k_i,l_i}), \tag{4.38}$$

or it has been rejected because it is not acceptable to the current filter, i.e.

$$(\theta(x_{k_i} + \alpha_{k_i,l_i} d_{k_i}), \varphi_\mu(x_{k_i} + \alpha_{k_i,l_i} d_{k_i})) \in \mathcal{F}_{k_i} = \mathcal{F}_K. \tag{4.39}$$

We will conclude the proof by showing that neither (4.38) nor (4.39) can be true for sufficiently large k_i.

To (4.38): Since $\lim_i \alpha_{k_i} = 0$, we also have $\lim_i \alpha_{k_i,l_i} = 0$. In particular, for sufficiently large k_i we have $\alpha_{k_i,l_i} \leq \bar{\alpha}$ with $\bar{\alpha}$ from Lemma 4.7, i.e. (4.38) cannot be satisfied for those k_i.

To (4.39): Let $\Theta_K := \min\{\theta : (\theta, \varphi) \in \mathcal{F}_K\}$. From Lemma 4.4 we have $\Theta_K > 0$. Using Lemma 4.1 and Lemma 4.3, we then see that

$$\theta(x_{k_i} + \alpha_{k_i,l_i} d_{k_i}) \leq (1 - \alpha_{k_i,l_i})\theta(x_{k_i}) + C_\theta M_d^2 [\alpha_{k_i,l_i}]^2.$$

Since $\lim_i \alpha_{k_i,l_i} = 0$ and from Theorem 4.2 also $\lim_i \theta(x_{k_i}) = 0$, it follows that for k_i sufficiently large we have $\theta(x_{k_i} + \alpha_{k_i,l_i} d_{k_i}) < \Theta_K$ which contradicts (4.39). □

The next lemma establishes conditions under which a step size can be found that is acceptable to the current filter (see (3.33)).

Lemma 4.9 *Suppose Assumptions G hold. Let $\{x_{k_i}\}$ be a subsequence with $k_i \notin \mathcal{R}_{\text{inc}}$ and $m_{k_i}(\alpha) \leq -\alpha\epsilon_2$ for a constant $\epsilon_2 > 0$ independent of k_i and for all $\alpha \in$*

$(0, 1]$. *Then there exist constants* $c_5, c_6 > 0$ *so that*

$$(\theta(x_{k_i} + \alpha d_{k_i}), \varphi_\mu(x_{k_i} + \alpha d_{k_i})) \notin \mathcal{F}_{k_i}$$

for all k_i *and* $\alpha \leq \min\{c_5, c_6\theta(x_{k_i})\}$.

Proof. Let $M_d, M_{\tilde{d}}, \bar{\alpha}^{\max}, C_\theta, C_\varphi$ be the constants from Lemma 4.1 and Lemma 4.3. Define $c_5 := \min\{\bar{\alpha}^{\max}, \epsilon_2/(M_{\tilde{d}}^2\ C_\varphi)\}$ and $c_6 := 1/(M_d^2\ C_\theta)$.

Now choose an iterate x_{k_i}. The mechanisms of Algorithm FILTER ensure (see comment in Step 7), that

$$(\theta(x_{k_i}), \varphi_\mu(x_{k_i})) \notin \mathcal{F}_{k_i}. \tag{4.40}$$

For $\alpha \leq c_5$ we have $\alpha^2 \leq \frac{\alpha\epsilon_2}{M_{\tilde{d}}^2\ C_\varphi} \leq \frac{-m_{k_i}(\alpha)}{C_\varphi \|\tilde{d}_{k_i}\|^2}$, or equivalently

$$m_{k_i}(\alpha) + C_\varphi \alpha^2 \|\tilde{d}_{k_i}\|^2 \leq 0,$$

and it follows with (4.27b) that

$$\varphi_\mu(x_{k_i} + \alpha d_{k_i}) \leq \varphi_\mu(x_{k_i}), \tag{4.41}$$

since $\alpha \leq c_5 \leq \bar{\alpha}^{\max} \leq \alpha_k^{\max}$. Similarly, for $\alpha \leq c_6\theta(x_{k_i}) \leq \frac{\theta(x_{k_i})}{\|d_{k_i}\|^2\ C_\theta}$, we have $-\alpha\theta(x_{k_i}) + C_\theta\alpha^2\|d_{k_i}\|^2 \leq 0$ and thus from (4.27a)

$$\theta(x_{k_i} + \alpha d_{k_i}) \leq \theta(x_{k_i}). \tag{4.42}$$

The claim then follows from (4.40), (4.41) and (4.42) using (3.39). □

The last lemma in this section shows that in iterations corresponding to a subsequence with only non-optimal limit points the filter is eventually not augmented. This result will be used in the proof of the main global convergence theorem to yield a contradiction.

Lemma 4.10 *Suppose Assumptions G hold and let* $\{x_{k_i}\}$ *be a subsequence with* $\chi(x_{k_i}) \geq \epsilon$ *for a constant* $\epsilon > 0$ *independent of* k_i. *Then there exists* $K \in \mathbb{N}$, *so that for all* $k_i \geq K$ *the filter is not augmented in iteration* k_i, *i.e.* $k_i \notin \mathcal{A}$.

Proof. Since by Theorem 4.2 we have $\lim_i \theta(x_{k_i}) = 0$, it follows from Lemma 4.2 that there exist constants $\epsilon_1, \epsilon_2 > 0$, so that

$$\theta(x_{k_i}) \leq \epsilon_1 \quad \text{and} \quad m_{k_i}(\alpha) \leq -\alpha\epsilon_2 \tag{4.43}$$

for k_i sufficiently large and $\alpha \in (0, 1]$; without loss of generality we can assume that (4.43) is valid for all k_i. We can now apply Lemma 4.7 and Lemma 4.9 to obtain the constants $\bar{\alpha}, c_5, c_6 > 0$. Choose $K \in \mathbb{N}$, so that for all $k_i \geq K$

$$\theta(x_{k_i}) < \min\left\{\theta_{\text{sml}}, \theta_{\text{inc}}, \frac{\bar{\alpha}}{c_6}, \frac{c_5}{c_6}, \left[\frac{\tau_1 c_6 \epsilon_2^{s_\varphi}}{\delta}\right]^{\frac{1}{s_\theta - 1}}\right\} \tag{4.44}$$

with τ_1 from Step 5.5. Note, that this implies for all $k_i \geq K$ that $k_i \notin \mathcal{R}_{\text{inc}}$,

$$\frac{\delta\ [\theta(x_{k_i})]^{s_\theta}}{\epsilon_2^{s_\varphi}} < \tau_1 c_6 \theta(x_{k_i}) \tag{4.45}$$

(since $s_\theta > 1$), as well as

$$c_6 \theta(x_{k_i}) < \min\{\bar{\alpha}, c_5\}. \tag{4.46}$$

Now choose an arbitrary $k_i \geq K$ and define

$$\beta_{k_i} := c_6 \theta(x_{k_i}) \overset{(4.46)}{=} \min\{\bar{\alpha}, c_5, c_6 \theta(x_{k_i})\}. \tag{4.47}$$

Lemma 4.7 and Lemma 4.9 then imply, that a trial step size $\alpha_{k_i,l} \leq \beta_{k_i}$ will satisfy both

$$\varphi_\mu(x_{k_i}(\alpha_{k_i,l})) \leq \varphi_\mu(x_{k_i}) + \eta_\varphi m_{k_i}(\alpha_{k_i,l}) \tag{4.48}$$

and

$$\Big(\theta(x_{k_i}(\alpha_{k_i,l})), \varphi_\mu(x_{k_i}(\alpha_{k_i,l}))\Big) \notin \mathcal{F}_{k_i}. \tag{4.49}$$

If we now denote with $\alpha_{k_i,L}$ the first trial step size satisfying both (4.48) and (4.49), the backtracking line search procedure in Step 5.5 then implies that for $\alpha \geq \alpha_{k_i,L}$

$$\alpha \geq \tau_1 \beta_{k_i} \overset{(4.47)}{=} \tau_1 c_6 \theta(x_{k_i}) \overset{(4.45)}{>} \frac{\delta[\theta(x_{k_i})]^{s_\theta}}{\epsilon_2^{s_\varphi}}$$

and therefore from (4.43) for $\alpha \geq \alpha_{k_i,L}$

$$\delta[\theta(x_{k_i})]^{s_\theta} < \alpha\epsilon_2^{s_\varphi} = [\alpha]^{1-s_\varphi} (\alpha\epsilon_2)^{s_\varphi} \leq [\alpha]^{1-s_\varphi} [-m_{k_i}(\alpha)]^{s_\varphi}.$$

This means, the switching condition (3.30) is satisfied for $\alpha_{k_i,L}$ and all previous trial step sizes. Consequently, for all trial step sizes $\alpha_{k_i,l} \geq \alpha_{k_i,L}$, Case I is considered in Step 5.4. We also have $\alpha_{k_i,l} \geq \alpha_{k_i}^{\min}$, i.e. the method does not switch to the feasibility restoration phase in Step 5.2 for those trial step sizes. Consequently, $\alpha_{k_i,L}$ is indeed the accepted step size α_{k_i}. Since it satisfies both (3.30) and (4.48), the filter is not augmented in iteration k_i. □

We are now ready to prove the main global convergence result.

Theorem 4.3 *Suppose Assumptions G hold. Then*

$$\lim_{k\to\infty} \theta(x_k) = 0 \tag{4.50a}$$

$$\textit{and} \qquad \liminf_{k\to\infty} \chi(x_k) = 0. \tag{4.50b}$$

In other words, all limit points are feasible, and there exists a limit point $x_*^\mu > 0$ of $\{x_k\}$ which is a first order optimal point for the barrier problem (2.38).

Proof. (4.50a) follows from Theorem 4.2. In order to show (4.50b), we have to consider two cases:

i) The filter is augmented only a finite number of times. Then Lemma 4.8 proves the claim.

ii) There exists a subsequence $\{x_{k_i}\}$, so that $k_i \in \mathcal{A}$ for all i. Now suppose, that $\limsup_i \chi(x_{k_i}) > 0$. Then there exists a subsequence $\{x_{k_{i_j}}\}$ of $\{x_{k_i}\}$ and a constant $\epsilon > 0$, so that $\lim_j \theta(x_{k_{i_j}}) = 0$ and $\chi(x_{k_{i_j}}) > \epsilon$ for all k_{i_j}. Applying Lemma 4.10 to $\{x_{k_{i_j}}\}$, we see that there is an iteration k_{i_j}, in which the filter is not augmented, i.e. $k_{i_j} \notin \mathcal{A}$. This contradicts the choice of $\{x_{k_i}\}$, so that $\lim_i \chi(x_{k_i}) = 0$, which proves (4.50b).

That a limit point x_*^μ with $\theta(x_*^\mu) = \chi(x_*^\mu) = 0$ lies indeed in the interior of the non-negative orthant, i.e. $x_*^\mu > 0$, has been argued in the paragraph before the statement of Lemma 4.1. □

Remark 4.1 *It is not possible to obtain a stronger results in Theorem 4.3 such as "*$\lim_k \chi(x_k) = 0$*". The reason is that even arbitrarily close to a strict local solution the restoration phase might be invoked even though the search direction is very good. This can happen if the current filter contains "old historic information" corresponding to previous iterates that were in a different region of* $\mathbb{R}^n$ *but had values for* θ *and* φ_μ *similar to those for the current iterate. If for the current iterate* $(\theta(x_k), \varphi_\mu(x_k))$ *is very close to the current filter (e.g. there exists filter pairs* $(\bar{\theta}, \bar{\varphi}) \in \mathcal{F}_k$ *with* $\bar{\theta} < \theta(x_k)$ *and* $\bar{\varphi} \approx \varphi_\mu(x_k)$*) and the barrier function* φ_μ *has to be increased in order to approach the optimal solution, the trial step sizes can be repeatedly rejected in Step 5.3 so that finally* $\alpha_{k,l}$ *becomes smaller than* $\alpha_k^{\min}$ *and the restoration phase is triggered. Without making additional assumptions on the restoration phase we only know that the next iterate returned from the restoration phase is more feasible, but possibly far away from any KKT point. This is the reason why we proposed the particular method for the restoration phase close to KKT points in Section 3.4.4.*

Remark 4.2 *Performing SOC steps, as described in Section 3.4.3, does not affect the global convergence properties just proved, since it leads to the same steps if the full step size* $\alpha_k^{\max}$ *has not been accepted, so that the proofs of the critical Lemmas 4.8–4.10 still hold.*

4.2.4 Global Convergence with Measure $\mathcal{L}_\mu(x, \lambda)$

In order to see that the global convergence analysis in Section 4.2 still holds for the modifications in Section 3.4.6 under Assumptions G, let us quickly revisit the individual results. Lemma 4.1 remains valid and in particular ensures, that the estimates λ_k are uniformly bounded as well, and that hence the sequence $\{\mathcal{L}_\mu(x_k, \lambda_k)\}$ is uniformly bounded below. It is also easy to verify, that Lemma 4.2 and Lemma 4.4 are still valid for the model definition (3.49), since the first line in (4.26) then becomes

$$m_{k_i}(\alpha)/\alpha = \nabla\varphi_\mu(x_{k_i})^T d_{k_i} + O(\|c(x_k)\|),$$

and thus only the constant c_3 may change. Furthermore, Lemma 4.3 still holds for the model definition (3.49) and with the measure "φ_μ" replaced by "$\mathcal{L}_\mu$", still with "$C_\varphi \alpha^2 \|\tilde{d}_k\|^2$" on the right hand side of (4.27b), as can be verified easily using Taylor expansions. Finally, the analysis in Sections 4.2.2 and 4.2.3 then holds with replacing "φ_μ" by "$\mathcal{L}_\mu$" where appropriate. The only point that deserves special attention is the proof of Lemma 4.8. Here, it was essential that the last rejected trial step size $\alpha_{k_i,l_i} \in [\alpha_{k_i}/\tau_2, \alpha_{k_i}/\tau_1]$ satisfies the switching condition (3.30), at least for k_i sufficiently large. To see that this is also true for the model definition (3.49), which is no longer linear in α, let us define the function

$$h_{k_i}(\alpha) := [-m_{k_i}(\alpha)]^{s_\varphi}\ \alpha^{1-s_\varphi} - \delta[\theta(x_{k_i})]^{s_\theta}.$$

This function is well defined for all k_i due to (4.37), and we have $h_{k_i}(\alpha_{k,l}) > 0$ if and only if (3.30) holds. Since we assume $\lim_i \theta(x_{k_i}) = 0$ and $\chi(x_{k_i}) \geq \epsilon$ in the proof, it can then be shown (using arguments similar to those in the proof of Lemma 4.2) that $h'_{k_i}(0) \geq \epsilon_3$ for some $\epsilon_3 > 0$ and k_i sufficiently large, and that $h''_{k_i}(0)$ is uniformly bounded. Since $\alpha_{k_i,l_i} \to 0$ and $h_{k_i}(\alpha_{k_i}) > 0$, it then follows that the switching condition (3.30) holds for $\alpha_{k_i,l_i} \in [\alpha_{k_i}/\tau_2, \alpha_{k_i}/\tau_1]$ when k_i is sufficiently large.

Regarding local convergence, however, it is not clear at this point whether fast local convergence is also achieved when the measure "$\mathcal{L}_\mu(x,\lambda)$" is used, possibly even without the need of a second order correction step. This may require more expensive least square multiplier steps $d_k^\lambda := \lambda_k^{\mathrm{ls}} - \lambda_k$ with λ_k^{ls} from

$$\begin{bmatrix} X_k^2 & A(x_k + d_k) \\ A(x_k + d_k)^T & 0 \end{bmatrix} \begin{pmatrix} d_k^{\mathrm{ls}} \\ \lambda_k^{\mathrm{ls}} \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) \\ 0 \end{pmatrix}$$

(see e.g. [64]).

4.3 Local Convergence Analysis of Line Search Filter Methods

We start the analysis by stating the necessary assumptions.

Assumptions L. *Assume that $\{x_k\}$ converges to a local solution $x_*^\mu > 0$ of the barrier problem (2.38) and that the following holds.*

(L1) *The functions f and c are twice continuously differentiable in a neighborhood of x_*^μ.*

(L2) *x_*^μ satisfies the following sufficient second order optimality conditions.*

- *x_*^μ is feasible, i.e. $\theta(x_*^\mu) = 0$,*
- *there exists $\lambda_*^\mu \in \mathbb{R}^m$ so that the KKT conditions (2.39) are satisfied for (x_*^μ, λ_*^μ),*
- *the constraint Jacobian $A(x_*^\mu)$ has full rank, and*
- *the Hessian of the Lagrangian $W_*^\mu = \nabla_{xx}^2 \mathcal{L}_\mu(x_*^\mu, \lambda_*^\mu)$ is positive definite on the null space of $A(x_*^\mu)^T$.*

(L3) *In (3.40), H_k^{soc} is uniformly positive definite on the null space of $(A_k^{\mathrm{soc}})^T$, and (3.45) holds.*

(L4) *The Hessian approximations H_k in (3.3) satisfy (3.46).*

The assumption $x_k \to x_*^\mu$ might be considered to be rather strong, in particular since the feasibility restoration phase might be invoked arbitrarily close to x_*^μ and divert x_k away from x_*^μ, see Remark 4.1. However, in Section 3.4.4 we proposed a particular restoration phase which ensures that the method will converge to a solution x_*^μ satisfying (L2), once x_k is sufficiently close.

Assumption (L4) is reminiscent of the Dennis-Moré characterization of superlinear convergence [29]. However, this assumption is stronger than necessary for

superlinear convergence [18] which requires only that $Z_k^T(W_k^\mu - H_k)d_k = o(\|d_k\|)$, where Z_k is a null space matrix for A_k^T.

First we summarize some preliminary results.

Lemma 4.11 *Suppose Assumptions G and L hold. Then there exists a neighborhood U_1 of x_*^μ, so that for all $x_k \in U_1$ we have*

$$\begin{aligned} d_k^{\text{soc}} &= o(\|d_k\|) && \text{(4.51a)} \\ \alpha_k^{\max} &= 1 && \text{(4.51b)} \\ x_k + d_k + d_k^{\text{soc}} &\geq (1-\tau)x_k && \text{(4.51c)} \\ m_k(1) &= O(\|d_k\|) && \text{(4.51d)} \\ c(x_k + d_k + d_k^{\text{soc}}) &= o(\|d_k\|^2) && \text{(4.51e)} \end{aligned}$$

Proof. Since from Assumption (L3) the matrix in (3.40) has a uniformly bounded inverse and the right hand side is $o(\|d_k\|)$, claim (4.51a) follows. Furthermore, since $x_*^\mu > 0$ and $d_k, d_k^{\text{soc}} \to 0$ as $x_k \to x_*^\mu$, we have (4.51b) and (4.51c). (4.51d) follows from the boundedness of $\nabla\varphi_\mu(x_k)$ and (3.31). Finally, from

$$\begin{aligned} c(x_k + d_k + d_k^{\text{soc}}) &= c(x_k + d_k) + A(x_k + d_k)^T d_k^{\text{soc}} + O(\|d_k^{\text{soc}}\|^2) \\ &\overset{(3.40)}{=} -c_k^{\text{soc}} - (A_k^{\text{soc}})^T d_k^{\text{soc}} + (A(x_k) + O(\|d_k\|))^T d_k^{\text{soc}} \\ &\qquad + O(\|d_k^{\text{soc}}\|^2) \\ &\overset{(3.45)}{=} o(\|d_k\|^2) + O(\|d_k\|\|d_k^{\text{soc}}\|) + O(\|d_k^{\text{soc}}\|^2) \\ &\overset{(4.51a)}{=} o(\|d_k\|^2) \end{aligned}$$

for x_k close to x_*^μ the last claim (4.51e) follows. □

In order to prove the local convergence result we will make use of two results established in [28] regarding the effect of second order correction steps on the exact penalty function (3.20). Note, that we will employ the exact penalty function only as a technical device, but that the algorithm never refers to it explicitly. We will

also use the following model of the penalty function

$$q_{\mu,\nu}(x_k,d) = \varphi_\mu(x_k) + \nabla\varphi_\mu(x_k)^T d + \frac{1}{2}d^T H_k d + \nu \left\| A_k^T d + c(x_k) \right\|. \tag{4.52}$$

The first result follows from Theorem 15.3.7 in [28].

Lemma 4.12 *Suppose Assumptions G and L hold. Let $\phi_{\mu,\nu}$ be the exact penalty function (3.20) and $q_{\mu,\nu}$ defined by (4.52) with $\nu > \|\lambda_*\|_D$, where $\|\cdot\|_D$ is the dual norm to $\|\cdot\|$. Then,*

$$\lim_{k\to\infty} \frac{\phi_{\mu,\nu}(x_k + d_k + d_k^{\text{SOC}}) - \phi_{\mu,\nu}(x_k)}{q_{\mu,\nu}(x_k,d_k) - q_{\mu,\nu}(x_k,0)} = 1. \tag{4.53}$$

The next result follows from Theorem 15.3.2 in [28].

Lemma 4.13 *Suppose Assumptions G hold. Let (d_k,λ_k^+) be a solution of the linear system (3.3), and let $\nu > \|\lambda_k^+\|_D$. Then*

$$q_{\mu,\nu}(x_k,d_k) - q_{\mu,\nu}(x_k,0) \le 0. \tag{4.54}$$

The next lemma shows that in a neighborhood of x_*^μ Step 5.4.1* of Algorithm SOC will be successful if the switching condition (3.42) holds.

Lemma 4.14 *Suppose Assumptions G and L hold. Then there exists a neighborhood $U_2 \subseteq U_1$ of x_*^μ so that whenever the switching condition (3.42) holds, the Armijo condition (3.43) is satisfied for the step $d_k + d_k^{\text{SOC}}$.*

Proof. Choose U_1 to be the neighborhood from Lemma 4.11. It then follows that for $x_k \in U_1$ satisfying (3.42) that (3.41) holds and that

$$\theta(x_k) < \delta^{-\frac{1}{s_\theta}}[-m_k(1)]^{\frac{s_\varphi}{s_\theta}} \overset{(4.51\text{d})}{=} O(\|d_k\|^{\frac{s_\varphi}{s_\theta}}) = o(\|d_k\|^2), \tag{4.55}$$

since $\frac{s_\varphi}{s_\theta} > 2$.

Since $\eta_\varphi < \frac{1}{2}$, Lemma 4.12 and (4.54) imply that there exists $K \in \mathbb{N}$ such that for all $k \ge K$ we have for some constant $\nu > 0$ with $\nu > \|\lambda_k^+\|_D$ independent of k that

$$\phi_{\mu,\nu}(x_k + d_k + d_k^{\text{SOC}}) - \phi_{\mu,\nu}(x_k) \le \left(\frac{1}{2} + \eta_\varphi\right)\left(q_{\mu,\nu}(x_k,d_k) - q_{\mu,\nu}(x_k,0)\right).$$

Using the definitions of $\phi_{\mu,\nu}$ and $q_{\mu,\nu}$ as well as (4.55) and (4.51e) this gives

$$\varphi_\mu(x_k + d_k + d_k^{\text{soc}}) - \varphi_\mu(x_k) \leq \left(\frac{1}{2} + \eta_\varphi\right)\left(\nabla\varphi_\mu(x_k)^T d_k + \frac{1}{2} d_k^T H_k d_k\right) + o(\|d_k\|^2).$$

Since $m_k(1) = \nabla\varphi_\mu(x_k)^T d_k$, this implies with the boundedness of λ_k^+, $\bar{p}_k$, and q_k (from (4.19)) that

$$\begin{aligned}
& \varphi_\mu(x_k + d_k + d_k^{\text{soc}}) - \varphi_\mu(x_k) - \eta_\varphi m_k(1) \\
\leq\ & \frac{1}{2}\nabla\varphi_\mu(x_k)^T d_k + \left(\frac{1}{4} + \frac{\eta_\varphi}{2}\right) d_k^T H_k d_k + o(\|d_k\|^2) \\
\overset{(3.3)}{=}\ & -\left(\frac{1}{4} - \frac{\eta_\varphi}{2}\right) d_k^T H_k d_k + \frac{1}{2} c(x_k)^T \lambda_k^+ + o(\|d_k\|^2) \\
\overset{(4.55)}{=}\ & -\left(\frac{1}{4} - \frac{\eta_\varphi}{2}\right) d_k^T H_k d_k + o(\|d_k\|^2) \\
\overset{(4.19)}{=}\ & -\left(\frac{1}{4} - \frac{\eta_\varphi}{2}\right) \bar{p}_k^T \tilde{Z}_k^T \tilde{H}_k \tilde{Z}_k \bar{p}_k + O(\|q_k\|) + o(\|d_k\|^2). \qquad (4.56)
\end{aligned}$$

Finally, from (4.20a), Assumption (G5), (4.19), and the orthonormality of $\tilde{Z}_k$ and $\tilde{Y}_k$ we have

$$q_k = O(\theta(x_k)) \overset{(4.55)}{=} o(\|d_k\|^2) = o(\|\bar{p}_k\|^2) + o(\|q_k\|^2)$$

and therefore $q_k = o(\|\bar{p}_k\|^2)$, as well as $d_k = O(\|\bar{p}_k\|)$, so that (3.43) is implied by (4.56), Assumption (G6) and $\eta_\varphi < \frac{1}{2}$, if x_k is sufficiently close to x_*^μ. □

It remains to show that also the filter and the sufficient reduction criterion (3.28) do not interfere with the acceptance of full steps close to x_*^μ. The following technical lemmas address this issue and prepare the proof of the main local convergence theorem.

Lemma 4.15 *Suppose Assumptions G and L hold. Then there exists a neighborhood $U_3 \subseteq U_2$ (with U_2 from Lemma 4.14) and constants $\nu_1, \nu_2, \nu_3 > 0$ with*

$$\nu_3 = (1 - \gamma_\theta)\nu_2 - \gamma_\varphi \qquad (4.57a)$$

$$2\gamma_\theta\nu_2 < (1 + \gamma_\theta)(\nu_2 - \nu_1) - 2\gamma_\varphi \qquad (4.57b)$$

$$2\nu_3 \geq (1 + \gamma_\theta)\nu_1 + (1 - \gamma_\theta)\nu_2, \qquad (4.57c)$$

so that for all $x_k \in U_3$ *we have* $\|\lambda_k^+\|_D < \nu_i$ *for* $i = 1, 2, 3$*, and the second order correction step is always tried in Algorithm* SOC *if* $x_k + d_k$ *is rejected. Furthermore, for all* $x_k \in U_3$ *we have*

$$\phi_{\mu,\nu_i}(x_k + d_k + \bar{d}_k^{\text{soc}}) - \phi_{\mu,\nu_i}(x_k) \leq \frac{1+\gamma_\theta}{2}\left(q_{\mu,\nu_i}(x_k, d_k) - q_{\mu,\nu_i}(x_k, 0)\right) \overset{(4.54)}{\leq} 0 \quad (4.58)$$

for $i = 2, 3$ *and*

$$\bar{d}_k^{\text{soc}} = d_k^{\text{soc}}, \quad (4.59a)$$

$$\bar{d}_k^{\text{soc}} = \sigma_k d_k^{\text{soc}} + d_{k+1} + \sigma_{k+1} d_{k+1}^{\text{soc}}, \quad (4.59b)$$

$$\bar{d}_k^{\text{soc}} = \sigma_k d_k^{\text{soc}} + d_{k+1} + \sigma_{k+1} d_{k+1}^{\text{soc}} + d_{k+2} + \sigma_{k+2} d_{k+2}^{\text{soc}}, \quad (4.59c)$$

or
$$\bar{d}_k^{\text{soc}} = \sigma_k d_k^{\text{soc}} + d_{k+1} + \sigma_{k+1} d_{k+1}^{\text{soc}} + d_{k+2} + \sigma_{k+2} d_{k+2}^{\text{soc}} + d_{k+3} + \sigma_{k+3} d_{k+3}^{\text{soc}}, \quad (4.59d)$$

with $\sigma_k, \sigma_{k+1}, \sigma_{k+2}, \sigma_{k+3} \in \{0, 1\}$*, as long as* $x_{l+1} = x_l + d_l + \sigma_l d_k^{\text{soc}}$ *for* $l \in \{k, \ldots, k+j\}$ *with* $j \in \{-1, 0, 1, 2\}$*, respectively.*

Proof. Let $U_3 \subseteq U_2$ be a neighborhood of x_*^μ, so that for all $x_k \in U_3$ we have $\theta(x_k) \leq \theta_{\text{sml}}$ and $\theta(x_k) \leq \theta_{\text{soc}}$. Therefore, due to (4.51b) and (4.51c) the second order correction is always tried in Algorithm SOC if $x_k + d_k$ has been rejected. Since λ_k^+ is uniformly bounded for all k with $x_k \in U_3$, we can find $\nu_1 > \|\lambda_*^\mu\|$ with

$$\nu_1 > \|\lambda_k^+\|_D \quad (4.60)$$

for all k with $x_k \in U_3$. Defining now

$$\nu_2 := \frac{1+\gamma_\theta}{1-\gamma_\theta}\nu_1 + \frac{3\gamma_\varphi}{1-\gamma_\theta}$$

and ν_3 by (4.57a), it is then easy to verify that $\nu_2, \nu_3 \geq \nu_1 > \|\lambda_k^+\|_D$ and that (4.57b) and (4.57c) hold. Since $(1 + \gamma_\theta) < 2$, we have from Lemma 4.12 by possibly further reducing U_3 that (4.58) holds for $x_k \in U_3$, since according to (d) and (e) on page 67 all choices of $\bar{d}_k^{\text{soc}}$ in (4.59) can be understood as second order correction steps to d_k.

□

Before proceeding with the next lemma, let us introduce some more notation.

Let U_3 and ν_i be the neighborhood and constants from Lemma 4.15. Since $\lim_k x_k = x_*^\mu$, we can find $K_1 \in \mathbb{N}$ so that $x_k \in U_3$ for all $k \geq K_1$. Let us now define the level set

$$L := \{x \in U_3 : \phi_{\mu,\nu_3}(x) \leq \phi_{\mu,\nu_3}(x_*^\mu) + \kappa\}, \tag{4.61}$$

where $\kappa > 0$ is chosen so that for all $x \in L$ we have $(\theta(x), \varphi_\mu(x)) \notin \mathcal{F}_{K_1}$. This is possible, since $\Theta_{K_1} > 0$ from (4.29), and since $\max\{\theta(x) : x \in L\}$ converges to zero as $\kappa \to 0$, because x_*^μ is a strict local minimizer of ϕ_{μ,ν_3} [50]. Obviously, $x_*^\mu \in L$. For later reference let K_2 be the first iteration $K_2 \geq K_1$ with $x_{K_2} \in L$.

Furthermore, let us define for $k \in \mathbb{N}$

$$\mathcal{G}_k := \left\{(\theta, \varphi) : \theta \geq (1-\gamma_\theta)\theta(x_k) \quad \text{and} \quad \varphi \geq \varphi_\mu(x_k) - \gamma_\varphi \theta(x_k)\right\}$$

and $I_{k_1}^{k_2} := \{l = k_1, \ldots, k_2 - 1 : l \in \mathcal{A}\}$ for $k_1 \leq k_2$. Then it follows from the filter update rule (3.34) and the definition of $\mathcal{A}$ in (4.16) that for $k_1 \leq k_2$

$$\mathcal{F}_{k_2} = \mathcal{F}_{k_1} \cup \bigcup_{l \in I_{k_1}^{k_2}} \mathcal{G}_l. \tag{4.62}$$

Also note, that $l \in I_{k_1}^{k_2} \subseteq \mathcal{A}$ implies $\theta(x_l) > 0$. Otherwise, we would have from (4.28) that $m_k(\alpha_{k,l}) < 0$, so that (3.30) holds for all trial step sizes α, and the step must have been accepted in Step 5.4.1 or Step 5.4.1*, hence satisfying (3.32) or (3.43). This would contradict the filter update condition in Step 7 or 7*, respectively.

The last lemma will enable us to show in the main theorem of this section that, once the iterates have reached the level set L, the full step will always be acceptable to the current filter.

Lemma 4.16 *Suppose Assumptions G and L hold and let $x > 0$ and $l \geq K_1$ with $\theta(x_l) > 0$. Then the following statements hold.*

$$\left.\begin{aligned} &\text{If } \phi_{\mu,\nu_2}(x) - \phi_{\mu,\nu_2}(x_l) \leq \tfrac{1+\gamma_\theta}{2}\left(q_{\mu,\nu_2}(x_l, d_l) - q_{\mu,\nu_2}(x_l, 0)\right), \\ &\text{then } (\theta(x), \varphi_\mu(x)) \notin \mathcal{G}_l. \end{aligned}\right\} \tag{4.63}$$

$$\left.\begin{array}{l} \text{If } x \in L \text{ and } \phi_{\mu,\nu_2}(x) - \phi_{\mu,\nu_2}(x_{K_2}) \leq \frac{1+\gamma_\theta}{2}\left(q_{\mu,\nu_2}(x_{K_2}, d_{K_2}) - q_{\mu,\nu_2}(x_{K_2}, 0)\right), \\ \text{then } (\theta(x), \varphi_\mu(x)) \notin \mathcal{F}_{K_2}. \end{array}\right\} \tag{4.64}$$

Proof. To (4.63): Since $\nu_1 > \|\lambda_l^+\|_D$ we have from Lemma 4.13 that $q_{\mu,\nu_1}(x_l, d_l) - q_{\mu,\nu_1}(x_l, 0) \leq 0$, and hence using definition for $q_{\mu,\nu}$ (4.52) and

$A_l^T d_l + c(x_l) = 0$ (from (3.3)) that

$$\begin{aligned} \phi_{\mu,\nu_2}(x) - \phi_{\mu,\nu_2}(x_l) &\leq \frac{1+\gamma_\theta}{2}\left(q_{\mu,\nu_2}(x_l, d_l) - q_{\mu,\nu_2}(x_l, 0)\right) \\ &= \frac{1+\gamma_\theta}{2}\left(q_{\mu,\nu_1}(x_l, d_l) - q_{\mu,\nu_1}(x_l, 0) + (\nu_1 - \nu_2)\theta(x_l)\right) \\ &\leq \frac{1+\gamma_\theta}{2}(\nu_1 - \nu_2)\theta(x_l). \end{aligned} \tag{4.65}$$

If $\varphi_\mu(x) < \varphi_\mu(x_l) - \gamma_\varphi \theta(x_l)$, the claim follows immediately. Otherwise, suppose that $\varphi_\mu(x) \geq \varphi_\mu(x_l) - \gamma_\varphi \theta(x_l)$. In that case, we have together with (4.65), (3.20), (4.57b), and $\theta(x_l) > 0$ that

$$\begin{aligned} \theta(x) - \theta(x_l) &\leq \frac{1+\gamma_\theta}{2\nu_2}(\nu_1 - \nu_2)\theta(x_l) + \frac{1}{\nu_2}\left(\varphi_\mu(x_l) - \varphi_\mu(x)\right) \\ &\leq \frac{1+\gamma_\theta}{2\nu_2}(\nu_1 - \nu_2)\theta(x_l) + \frac{\gamma_\varphi}{\nu_2}\theta(x_l) \\ &< -\gamma_\theta \theta(x_l), \end{aligned}$$

so that $(\theta(x), \varphi_\mu(x)) \notin \mathcal{G}_l$.

To (4.64): Since $x \in L$, it follows by the choice of κ that $(\theta(x), \varphi_\mu(x)) \notin \mathcal{F}_{K_1}$. Thus, according to (4.62) it remains to show that for all $l \in I_{K_1}^{K_2}$ we have $(\theta(x), \varphi_\mu(x)) \notin \mathcal{G}_l$. Choose $l \in I_{K_1}^{K_2}$. As in (4.65) we can show that

$$\phi_{\mu,\nu_2}(x) - \phi_{\mu,\nu_2}(x_{K_2}) \leq \frac{1+\gamma_\theta}{2}(\nu_1 - \nu_2)\theta(x_{K_2}). \tag{4.66}$$

Since $x \in L$ it follows from (4.61), the definitions of K_2 and $\phi_{\mu,\nu}$, and the fact that $l < K_2$ that

$$\begin{aligned} \phi_{\mu,\nu_3}(x_l) &> \phi_{\mu,\nu_3}(x_{K_2}) = \phi_{\mu,\nu_2}(x_{K_2}) + (\nu_3 - \nu_2)\theta(x_{K_2}) \\ &\overset{(4.66)}{\geq} \phi_{\mu,\nu_2}(x) + \left(\nu_3 - \frac{1+\gamma_\theta}{2}\nu_1 - \frac{1-\gamma_\theta}{2}\nu_2\right)\theta(x_{K_2}) \\ &\overset{(4.57c)}{\geq} \phi_{\mu,\nu_2}(x). \end{aligned} \tag{4.67}$$

If $\varphi_\mu(x) < \varphi_\mu(x_l) - \gamma_\varphi\theta(x_l)$, we immediately have $(\theta(x), \varphi_\mu(x)) \notin \mathcal{G}_l$. Otherwise we have $\varphi_\mu(x) \geq \varphi_\mu(x_l) - \gamma_\varphi\theta(x_l)$ which together with (4.67) and the definition of $\phi_{\mu,\nu}$ yields

$$\begin{aligned}\theta(x) &< \frac{1}{\nu_2}\left(\varphi_\mu(x_l) + \nu_3\theta(x_l) - \varphi_\mu(x)\right)\\ &\leq \frac{\nu_3 + \gamma_\varphi}{\nu_2}\theta(x_l)\\ &\overset{(4.57a)}{=} (1-\gamma_\theta)\theta(x_l),\end{aligned}$$

so that $(\theta(x), \varphi_\mu(x)) \notin \mathcal{G}_l$ which concludes the proof of (4.64). □

After these preparations we are finally able to show the main local convergence theorem.

Theorem 4.4 *Suppose Assumptions G and L hold. Then, for k sufficiently large full steps of the form $x_{k+1} = x_k + d_k$ or $x_{k+1} = x_k + d_k + d_k^{\text{soc}}$ will be taken, and x_k converges to x_*^μ superlinearly.*

Proof. Recall that $K_2 \geq K_1$ is the first iteration after K_1 with $x_{K_2} \in L \subseteq U_3$. Hence, for all $k \geq K_2$ Lemma 4.11 and Lemma 4.15 imply that the second order correction step is always tried in Algorithm SOC if $x_k + d_k$ is rejected, and that $\alpha_k^{\max} = 1$ and (3.41) hold, i.e. the fraction-to-the-boundary rule is never active.

We now show by induction that the following statements are true for $k \geq K_2 + 2$:

$$\begin{aligned}&(\text{i}_k) \qquad \phi_{\mu,\nu_i}(x_k) - \phi_{\mu,\nu_i}(x_l) \leq \frac{1+\gamma_\theta}{2}\left(q_{\mu,\nu_i}(x_l, d_l) - q_{\mu,\nu_i}(x_l, 0)\right)\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for } i \in \{2,3\} \text{ and } K_2 \leq l \leq k-2\\ &(\text{ii}_k) \qquad x_k \in L\\ &(\text{iii}_k) \qquad x_k = x_{k-1} + d_{k-1} + \sigma_{k-1} d_{k-1}^{\text{soc}} \qquad \text{with } \sigma_{k-1} \in \{0,1\}.\end{aligned}$$

We start by showing that these statements are true for $k = K_2 + 2$.

Suppose, the point $x_{K_2} + d_{K_2}$ is not accepted by the line search. In that case, define $\bar{x}_{K_2+1} := x_{K_2} + d_{K_2} + d_{K_2}^{\text{soc}}$. Then, from (4.58) with $i = 3$, $k = K_2$, and (4.59a), we see from $x_{K_2} \in L$ and the definition of L that $\bar{x}_{K_2} \in L$. After applying

(4.58) again with $i = 2$ it follows from (4.64) that $(\theta(\bar{x}_{K_2+1}), \varphi_\mu(\bar{x}_{K_2+1})) \notin \mathcal{F}_{K_2}$, i.e. $\bar{x}_{K_2+1}$ is not rejected in Step 5.3*. Furthermore, if the switching condition (3.42) holds, we see from Lemma 4.14 that the Armijo condition (3.43) with $k = K_2$ is satisfied for the point $\bar{x}_{K_2+1}$. In the other case, i.e. if (3.42) is violated (note that then (4.28) and (3.42) imply $\theta(x_{K_2}) > 0$), we see from (4.58) for $i = 2$, $k = K_2$, and (4.59a), together with (4.63) for $l = K_2$, that (3.44) holds. Hence, $\bar{x}_{K_2+1}$ is also not rejected in Step 5.4* and accepted as next iterate. Summarizing the discussion in this paragraph we can write $x_{K_2+1} = x_{K_2} + d_{K_2} + \sigma_{K_2} d^{\text{soc}}_{K_2}$ with $\sigma_{K_2} \in \{0, 1\}$.

Let us now consider iteration $K_2 + 1$. For $\sigma_{K_2+1} \in \{0, 1\}$ we have from (4.63) for $k = K_2$ and (4.59b) that

$$\begin{aligned} & \phi_{\mu,\nu_i}(x_{K_2+1} + d_{K_2+1} + \sigma_{K_2+1} d^{\text{soc}}_{K_2+1}) - \phi_{\mu,\nu_i}(x_{K_2}) \\ \leq \; & \frac{1+\gamma_\theta}{2} \left(q_{\mu,\nu_i}(x_{K_2}, d_{K_2}) - q_{\mu,\nu_i}(x_{K_2}, 0) \right) \end{aligned} \tag{4.68}$$

for $i = 2, 3$, which yields

$$x_{K_2+1} + d_{K_2+1} + \sigma_{K_2+1} d^{\text{soc}}_{K_2+1} \in L. \tag{4.69}$$

If $x_{K_2+1} + d_{K_2+1}$ is accepted as next iterate x_{K_2+2}, we immediately obtain from (4.68) and (4.69) that (i_{K_2+2})–(iii_{K_2+2}) hold. Otherwise, we consider the case $\sigma_{K_2+1} = 1$. From (4.68), (4.69), and (4.64) we have for $\bar{x}_{K_2+2} := x_{K_2+1} + d_{K_2+1} + d^{\text{soc}}_{K_2+1}$ that $(\theta(\bar{x}_{K_2+2}), \varphi_\mu(\bar{x}_{K_2+2})) \notin \mathcal{F}_{K_2}$. If $K_2 \notin I^{K_2+1}_{K_2}$ it immediately follows from (4.62) that $(\theta(\bar{x}_{K_2+2}), \varphi_\mu(\bar{x}_{K_2+2})) \notin \mathcal{F}_{K_2+1}$. Otherwise, we have $\theta(x_{K_2}) > 0$. Then, (4.68) for $i = 2$ together with (4.63) implies $(\theta(\bar{x}_{K_2+2}), \varphi_\mu(\bar{x}_{K_2+2})) \notin \mathcal{G}_{K_2}$, and hence with (4.62) we have $(\theta(\bar{x}_{K_2+2}), \varphi_\mu(\bar{x}_{K_2+2})) \notin \mathcal{F}_{K_2+1}$, so that $\bar{x}_{K_2+2}$ is not rejected in Step 5.3*. Arguing similarly as in the previous paragraph we can conclude that $\bar{x}_{K_2+1}$ is also not rejected in Step 5.4*. Therefore, $x_{K_2+2} = \bar{x}_{K_2+2}$. Together with (4.68) and (4.69) this proves (i_{K_2+2})–(iii_{K_2+2}) for the case $\sigma_{K_2+1} = 1$.

Now suppose that (i_l)–(iii_l) are true for all $K_2 + 2 \leq l \leq k$ with some $k \geq K_2 + 2$. If $x_k + d_k$ is accepted by the line search, define $\sigma_k := 0$, otherwise $\sigma_k := 1$. Set

$\bar{x}_{k+1} := x_k + d_k + \sigma_k d_k^{\text{soc}}$. From (4.58) we then have for $i = 2, 3$

$$\phi_{\mu,\nu_i}(\bar{x}_{k+1}) - \phi_{\mu,\nu_i}(x_{k-1}) \leq \frac{1+\gamma_\theta}{2}\left(q_{\mu,\nu_i}(x_{k-1}, d_{k-1}) - q_{\mu,\nu_i}(x_{k-1}, 0)\right) \leq 0. \quad (4.70)$$

Choose l with $K_2 \leq l < k-1$ and consider two cases:

Case a): If $k = K_2 + 2$, then $l = K_2$, and it follows from (4.58) with (4.59d) that for $i = 2, 3$

$$\phi_{\mu,\nu_i}(\bar{x}_{k+1}) - \phi_{\mu,\nu_i}(x_l) \leq \frac{1+\gamma_\theta}{2}\left(q_{\mu,\nu_i}(x_l, d_l) - q_{\mu,\nu_i}(x_l, 0)\right) \leq 0. \quad (4.71)$$

Case b): If $k > K_2 + 2$, we have from (4.70) that $\phi_{\mu,\nu_i}(\bar{x}_{k+1}) \leq \phi_{\mu,\nu_i}(x_{k-1})$ and hence from (i_{k-1}) it follows that (4.71) also holds in this case.

In either case, (4.71) implies in particular that $\phi_{\mu,\nu_3}(\bar{x}_{k+1}) \leq \phi_{\mu,\nu_3}(x_{K_2})$, and since $x_{K_2} \in L$, we obtain

$$\bar{x}_{k+1} \in L. \quad (4.72)$$

If $x_k + d_k$ is accepted by the line search, (i_{k+1})–(iii_{k+1}) follow from (4.71), (4.70) and (4.72). If $x_k + d_k$ is rejected, we see from (4.72), (4.71) for $i = 2$ and $l = K_2$, and (4.64) that $(\theta(\bar{x}_{k+1}), \varphi_\mu(\bar{x}_{k+1})) \notin \mathcal{F}_{K_2}$. Furthermore, for $l \in I_{K_2}^k$ we have from (4.71) and (4.63) that $(\theta(\bar{x}_{k+1}), \varphi_\mu(\bar{x}_{k+1})) \notin \mathcal{G}_l$, and hence from (4.62) that $\bar{x}_{k+1}$ is not rejected in Step 5.3*. We can again show as before that $\bar{x}_{k+1}$ is not rejected in Step 5.4*, so that $x_{k+1} = \bar{x}_{k+1}$ which implies (i_{k+1})–(iii_{k+1}).

That $\{x_k\}$ converges to x_*^μ with a superlinear rate follows from (3.46) (see e.g. [68]). □

4.4 SQP Filter Methods

4.4.1 Line Search Filter SQP Method I

In this section we show how Algorithm FILTER can be applied to active set SQP methods for the solution of the NLP formulation (2.1). Here, at an iterate x_k with $x_k^{(i)} \geq 0$ for $i \in \mathcal{I}$, the search direction d_k is then computed as the solution of the QP (2.12).

Starting from an initial point x_0 with $x_0^{(i)} \geq 0$ for $i \in \mathcal{I}$ (and hence ensuring $x_k^{(i)} \geq 0$ for all k), we can still apply Algorithm FILTER as step acceptance mechanism with the following modifications.

1. All occurrences of "φ_μ" are replaced by "f".

2. The computation of the search direction in Step 3 is replaced by the solution of the QP (2.12). The restoration phase is invoked in this step, if the QP (2.12) is infeasible or not sufficiently consistent (see Assumption (G5**) below).

3. The fraction-to-the-boundary rule is no longer necessary, i.e. in Step 4 we always choose $\alpha_k^{\max} = 1$.

4. The feasibility restoration phase in Step 9 has to return an iterate x_{k+1} satisfying $x_{k+1}^{(i)} \geq 0$ for $i \in \mathcal{I}$.

In order to state the assumptions necessary for a global convergence analysis let us define the set of coordinates that are active at the current point x_k *and* at $x_k + d_k$,

$$\mathcal{S}_k := \left\{ i \in \mathcal{I} : x_k^{(i)} = 0 \quad \text{and} \quad d_k^{(i)} = 0 \right\},$$

and again consider a decomposition of the search direction as in (2.20), where q_k is now defined as the solution of the QP

$$\begin{aligned}
\min_{q \in \mathbb{R}^n} \quad & q^T q && && (4.73a)\\
\text{s.t.} \quad & A_k^T q + c_k = 0 && && (4.73b)\\
& q^{(i)} = 0 && \text{for } i \in \mathcal{S}_k && (4.73c)\\
& x_k^{(i)} + q^{(i)} \geq 0 && \text{for } i \in \mathcal{I} \setminus \mathcal{S}_k. && (4.73d)
\end{aligned}$$

whose solution is the shortest feasible step for the QP (2.12) that does not change the components corresponding to $\mathcal{S}_k$. We then choose Z_k as an orthonormal null space matrix for the matrix

$$\left[\begin{array}{cccc} A_k & e_{j_1} & \cdots & e_{j_{l_k}} \end{array} \right]^T, \qquad \text{where} \qquad \mathcal{S}_k = \{j_1, \ldots, j_{l_k}\},$$

i.e. Z_k is a basis of the null space of all equality constraints and bounds that are active at x_k and $x_k + d_k$. With this, we can obtain $p_k = Z_k \bar{p}_k$ with $\bar{p}_k$ as the solution of the reduced QP

$$\min_{\bar{p} \in \mathbb{R}^{n-m-l_k}} \quad \left(Z_k^T g_k + Z_k^T W_k q_k\right)^T \bar{p} + \frac{1}{2} \bar{p}^T Z_k^T W_k Z_k \bar{p} \tag{4.74a}$$

$$\text{s.t.} \quad x_k + q_k + Z_k \bar{p} \geq 0. \tag{4.74b}$$

Note that the set $\mathcal{S}_k$ is not known before the QP (2.12) has been solved. The QPs (4.73) and (4.74) are defined only to state the assumptions and are not a possible procedure to obtain the solution of (2.12).

With these definitions we can now replace Assumptions (G5) and (G6) by

(G5**) *There exists a constant $M_q > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$ we have*

$$\|q_k\| \leq M_q \theta(x_k).$$

(G6**) *There exists a constant $M_W > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$ we have*

$$\lambda_{\min}\left(Z_k^T W_k Z_k\right) \geq M_W, \tag{4.75}$$

where Z_k has been defined above.

We furthermore replace "H_k" by "W_k" in statement of Assumption (G3).

Assumption (G5**) is similar in spirit to the assumption expressed by Eq. (2.10) in [35]. Essentially, we assume that if the QPs (2.12) are becoming increasingly ill-conditioned, eventually the restoration phase will be triggered in Step 3. Together with Assumption (G4) this assumption also means that we suppose that the QP (2.12) is sufficiently consistent when feasible points are approached.

Assumption (G6**) again ensures descent in the objective function at sufficiently feasible points. Note that this assumption is natural in the sense that if the method converges to a strict local solution x_* of the NLP (2.1), the active set $\mathcal{S}_k$ finally becomes unchanged and Z_k is a null space matrix for the constraints active at x_*. Hence, no correction of the reduced Hessian will be necessary close to x_*, if exact second derivatives are used.

In order to see that the global convergence analysis in Section 4.2 still holds under the modified Assumptions G, let us first note that the objective function of the nonlinear problem solved by Algorithm FILTER is now bounded since no "ln"-terms appear in the NLP (2.1) in contrast to the barrier problem (2.38). Therefore, the scaling (4.17) of the linear system (3.3) is no longer necessary. After defining the criticality measure again as $\chi(x_k) := \|\bar{p}_k\|_2$ for $k \notin \mathcal{R}_{\text{inc}}$, the proofs are valid with minor straightforward modifications and with all occurrences of "φ_μ", "$\tilde{d}_k$", and "H_k" replaced by "f", "d_k", and "W_k", respectively. Only the proof of Lemma 4.2 deserves special attention. Let us first state the optimality conditions of the reduced QP (4.74).

$$
\begin{aligned}
Z_k^T W_k Z_k \bar{p}_k + (Z_k^T g_k + Z_k^T W_k q_k) - Z_k^T v_k^+ &= 0 && && \text{(4.76a)}\\
x_k + q_k + Z_k \bar{p}_k &= 0 && && \text{(4.76b)}\\
(x_k + q_k + Z_k \bar{p}_k)^T v_k^+ &= 0 && && \text{(4.76c)}\\
v_k^+ &\geq 0 && && \text{(4.76d)}\\
(v_k^+)^{(i)} &= 0 && \text{for } i \notin \mathcal{I} && \text{(4.76e)}
\end{aligned}
$$

For $k \notin \mathcal{R}_{\text{inc}}$ we then have from (4.76a) and (4.76c) that

$$
\begin{aligned}
Z_k^T g_k &= Z_k^T v_k^+ - Z_k^T W_k Z_k \bar{p}_k - Z_k^T W_k q_k\\
(x_k + q_k)^T v_k^+ &= -(v_k^+)^T Z_k \bar{p}_k
\end{aligned}
$$

and therefore

$$
\begin{aligned}
g_k^T Z_k \bar{p}_k &= -(x_k + q_k)^T v_k^+ - \bar{p}_k^T Z_k^T W_k Z_k \bar{p}_k - \bar{p}_k^T Z_k^T W_k q_k\\
&\leq -\bar{p}_k^T Z_k^T W_k Z_k \bar{p}_k - \bar{p}_k^T Z_k^T W_k q_k
\end{aligned}
$$

where we used (4.73c), (4.73d), and (4.76d) in the last inequality. This gives together with the definition of m_k (3.31), the decomposition (2.20), and Assumptions (G5**)

and (G6**)

$$\begin{aligned} m_k(\alpha)/\alpha &= g_k^T d_k \\ &= g_k^T Z_k \bar{p}_k + g_k^T q_k \\ &\leq -\bar{p}_k^T Z_k^T W_k Z_k \bar{p}_k - \bar{p}_k^T Z_k^T W_k q_k + g_k^T q_k \\ &\leq -M_W[\chi(x_k)]^2 + O\left(\chi(x_k)\theta(x_k)\right) + O(\theta(x_k)) \end{aligned}$$

which corresponds to the second last line in (4.26), and we can conclude the proof of Lemma 4.2 as before.

Finally, the discussion of local convergence in Section 4.3 also still applies if we assume that eventually the active set is not changed. To guarantee this, the computation of the second order correction step (3.40) and the feasibility restoration phase proposed in Section 3.4.4 have to be adapted in order to take the active bound constraints into account.

4.4.2 Line Search Filter SQP Method II

In this section we show how Algorithm FILTER can be applied to line search SQP methods for the solution of NLPs in the general formulation (2.43), where the functions f and $c := (c^{\mathcal{E}}, c^{\mathcal{I}})$ have the smoothness properties of f and c in Assumptions (G1) and (L1). Note that this formulation in contrast to (2.1) allows general inequality constraints. Furthermore, the assumptions necessary in order to ensure global convergence differ from the ones stated in the previous section.

Here, search directions d_k are obtained as the solution of the QP

$$\min_{d \in \mathbb{R}^n} \quad g_k^T d + \frac{1}{2} d^T W_k d \tag{4.77a}$$

$$\text{s.t.} \quad (A_k^{\mathcal{E}})^T d + c^{\mathcal{E}}(x_k) = 0 \tag{4.77b}$$

$$(A_k^{\mathcal{I}})^T d + c^{\mathcal{I}}(x_k) \geq 0, \tag{4.77c}$$

where $A_k^{\mathcal{E}} := \nabla c^{\mathcal{E}}(x_k)$, $A_k^{\mathcal{I}} := \nabla c^{\mathcal{I}}(x_k)$, and W_k is (an approximation of) the Hessian of the Lagrangian

$$\mathcal{L}(x, \lambda, v) = f(x) + (c^{\mathcal{E}}(x))^T \lambda - (c^{\mathcal{I}}(x))^T v$$

of the NLP (2.43) with the Lagrangian multipliers $v \geq 0$ corresponding to the inequality constraints (2.43c). We will denote the optimal QP multipliers corresponding to (4.77b) and (4.77c) with λ_k^+ and $v_k^+ \geq 0$, respectively.

We further define the infeasibility measure by

$$\theta(x) := \left\| \begin{pmatrix} c^{\mathcal{E}}(x) \\ c^{\mathcal{I}}(x)^{(-)} \end{pmatrix} \right\|,$$

where for a vector w the expression $w^{(-)}$ defines the vector with the components $\max\{0, -w^{(i)}\}$. Similarly to the discussion in the previous section, Algorithm FILTER can then be used with the following modifications.

1. All occurrences of "φ_μ" are replaced by "f".

2. The computation of the search direction in Step 3 is replaced by the solution of the QP (4.77). The restoration phase is invoked in this step, if the QP (4.77) is infeasible or not sufficiently consistent (see Assumption (G5***) below).

3. The fraction-to-the-boundary rule is no longer necessary, i.e. in Step 4 we always choose $\alpha_k^{\max} = 1$.

In order to state the assumptions necessary for a global convergence analysis let us again consider a decomposition of the search direction (2.20), where q_k is now defined as the solution of the QP

$$\begin{aligned} \min_{q \in \mathbb{R}^n} \quad & q^T q \\ \text{s.t.} \quad & (A_k^{\mathcal{E}})^T q + c^{\mathcal{E}}(x_k) = 0 \\ & (A_k^{\mathcal{I}})^T q + c^{\mathcal{I}}(x_k) \geq 0, \end{aligned}$$

i.e. q_k is the shortest vector satisfying the constraints in the QP (4.77). With these definitions we can now replace Assumptions (G5) and (G6) by

(G5***) *There exist constants $M_d, M_\lambda, M_v, M_q > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$ we have*

$$\|d_k\| \leq M_d, \quad \|\lambda_k^+\| \leq M_\lambda, \quad \|v_k^+\| \leq M_v, \quad \|q_k\| \leq M_q \theta(x_k)$$

(G6***) *There exists a constant $M_W > 0$, so that for all $k \notin \mathcal{R}_{\text{inc}}$ we have*

$$d_k^T W_k d_k \geq M_W d_k^T d_k. \tag{4.78}$$

Also, in Assumption (G3) we replace "H_k" by "W_k".

Assumption (G5***) is again similar to the assumption expressed by Eq. (2.10) in [35]. Essentially, we assume that if the constraints of the QPs (4.77) become increasingly ill-conditioned, eventually the restoration phase will be triggered in Step 3. Together with Assumption (G4), this assumption also means that we suppose that the QP (2.12) is sufficiently consistent when feasible points are approached.

Assumption (G6***) again ensures descent in the objective function at sufficiently feasible points. This assumption has been made previously for global convergence proofs of some SQP line search methods (see e.g. [73]). However, this assumption can be rather strong since even close to a strict local solution the exact Hessian might have to be modified in order to satisfy (4.78). For this reason in the previous section, an alternative and more natural assumption is considered for the NLP formulation (2.1) with only bound constraints as inequality constraints.

In order to see that the global convergence analysis in Section 4.2 still holds under the modified Assumptions G, let us first note that the objective function of the nonlinear problem solved by Algorithm FILTER is now bounded since no "ln"-terms appear in the NLP (2.43) in contrast to the barrier problem (2.38). Therefore, the scaling (4.17) of the linear system (3.3) is no longer necessary. After defining the criticality measure as $\chi(x_k) := \|d_k - q_k\|_2$ for $k \notin \mathcal{R}_{\text{inc}}$, the proofs are valid with minor straightforward modifications and with all occurrences of "φ_μ", "$\tilde{d}_k$", and "H_k" replaced by "f", "d_k", and "W_k", respectively. Only the proof of Lemma 4.2 deserves special attention. From the optimality conditions for the QP (4.77) it follows in

particular that

$$\begin{aligned} g_k + W_k d_k + A_k^{\mathcal{E}} \lambda_k^+ - A_k^{\mathcal{I}} v_k^+ &= 0 && (4.79a) \\ \left((A_k^{\mathcal{I}})^T d_k + c^{\mathcal{I}}(x_k)\right)^T v_k^+ &= 0 && (4.79b) \\ v_k^+ &\geq 0, && (4.79c) \end{aligned}$$

so that for all $k \notin \mathcal{R}_{\text{inc}}$

$$\begin{aligned} g_k^T d_k &\overset{(4.79a)}{=} -d_k^T W_k d_k - d_k^T A_k^{\mathcal{E}} \lambda_k^+ + d_k^T A_k^{\mathcal{I}} v_k^+ \\ &\overset{(4.77b),(4.79b)}{=} -d_k^T W_k d_k + c^{\mathcal{E}}(x_k)^T \lambda_k^+ - c^{\mathcal{I}}(x_k)^T v_k^+ \\ &\overset{(4.79c)}{\leq} -d_k^T W_k d_k + c^{\mathcal{E}}(x_k)^T \lambda_k^+ + \left(c^{\mathcal{I}}(x_k)^{(-)}\right)^T v_k^+ \\ &\overset{(2.20)}{\leq} -M_W[\chi(x_k)]^2 + O\left(\chi(x_k)\theta(x_k)\right) + O(\theta(x_k)) \end{aligned}$$

where we used Assumptions (G5***) and (G6***) in the last inequality. This corresponds to the second last line in (4.26), and we can conclude the proof of Lemma 4.2 as before.

Also the discussion of local convergence in Section 4.3 still applies if we assume that eventually the active set is not changed. To guarantee this, the computation of the second order correction step (3.40) and the feasibility restoration phase proposed in Section 3.4.4 have to be adapted in order to take the active inequality constraints into account.

4.4.3 Fast Local Convergence of a Trust Region Filter SQP Method

As briefly mentioned in Section 3.4.2, the switching rule used in the trust region SQP-filter algorithm proposed by Fletcher, Gould, Leyffer, Toint, and Wächter [35], for which global convergence has been proven, does not imply the relationship (4.55). Therefore the proof of Lemma 4.14 in our local convergence analysis does not hold for this method. However, it is easy to see that the global convergence analysis in [35] is still valid (in particular Lemma 3.7 and Lemma 3.10 in [35]), if the switching

rule Eq. (2.19) in [35] is modified in analogy to (3.30) and replaced by

$$[m_k(d_k)]^{s_\varphi} \Delta_k^{1-s_\varphi} \geq \kappa_\theta \theta_k^\varphi,$$

where m_k is now the change of the objective function predicted by a quadratic model of the objective function, Δ_k is the current trust region radius, $\kappa_\theta, \varphi > 0$ are constants from [35] satisfying certain relationships, and the new constant $s_\varphi > 0$ satisfies $s_\varphi > 2\varphi$. Then the local convergence analysis in Section 4.3 is still valid (also for the quadratic model formulation), assuming that

1. sufficiently close to a strict local solution the trust region is inactive,
2. the trust region radius Δ_k is uniformly bounded away from zero,
3. the (approximate) SQP steps s_k are computed (sufficiently) exactly, and
4. and a second order correction as discussed in Section 3.4.3 is performed.

Chapter 5

Numerical Results

In this chapter the practical performance of the algorithm presented in Chapter 3 will be explored. In Section 5.1.2, the different line search options are tested on a large set of NLP test problems. Section 5.1.3 presents a numerical comparison of IPOPT with two other recent interior point codes, demonstrating its potential as general purpose NLP solver.

The second half of this chapter will show on the example of dynamic optimization, how the optimization method can be tailored to particular problem structures, in this case using an elemental decomposition approach (Section 5.2.1). As example applications, a continuous air separation distillation column (Section 5.2.2) and a batch cooling crystallization process (Section 5.2.3) will be solved and used to compare the different options for handling second derivative information.

5.1 Performance of IPOPT on Test Problems

IPOPT has been interfaced to the AMPL Solver Library (ASL) [41], allowing to solve NLPs formulated in the AMPL modelling language [40]. In AMPL, problems are formulated as

$$\min_{x\in\mathbb{R}^n} \quad f(x) \tag{5.1a}$$

$$\text{s.t.} \quad c_L \leq c(x) \leq c_U \tag{5.1b}$$

$$x_L \leq x \leq x_U, \tag{5.1c}$$

with $c_L \in \{-\infty\} \cup \mathbb{R}^m$ and $c_U \in \{+\infty\} \cup \mathbb{R}^m$ satisfying $c_L \leq c_U$. For IPOPT, this statement is automatically reformulated as (1.1) by means of slack variables for those constraints (5.1b) with $c_L^{(j)} < c_U^{(j)}$. The ASL is able to provide first and second derivatives of objective and constraints functions by automatic differentiation.

5.1.1 The AMPL Test Sets and Performance Profiles

The first numerical results are obtained using the following collections of problems formulated in AMPL.

- **CUTE (Constrained and Unconstrained Test Environment)**; see Table A.1 in Appendix A. This collection of NLPs has been assembled by Bongartz, Conn, Gould, and Toint [20]. Originally, these problems were formulated in the Standard Input Format (SIF); the SIF file for each problem was translated into a set of Fortran routines that were linked to the optimization code. Recently, Benson and Vanderbei [79] translated most of those problems into AMPL. Some of the problems have been modified during this process, and whereas many of the original CUTE problem formulations in SIF were scalable, the AMPL translations are for a fixed size. However, we decided to use the AMPL formulation (as obtained from [79] in March 2001), since interfaces to the original CUTE software package for the solvers KNITRO and LOQO (with which IPOPT is compared in Section 5.1.3) are not available to us.

Number of variables (incl. slacks)	Number of problems in CUTE set
2 . . . 9	220
10 . . . 99	87
100 . . . 999	73
1000 . . . 9999	68
10000 . . . 50000	38

Table 5.1: Distribution of problem size in CUTE test set

We excluded those problems that have less than one degree of freedom, or that are Linear Programs (LPs) or QPs, since IPOPT has been developed to solved general nonlinear optimization problems. We also excluded problems with non-differentiable functions such as min, max, or $|\cdot|$.

- **COPS (Constrained Optimization Problems), Version 2.0**; see Table A.2 in Appendix A. This collection of 17 scalable NLP test problems has been assembled and implemented in AMPL by Bondarenko, Bortz, and Moré [19] and Dolan and Moré [30]. The test set "provides a modest selection of difficult nonlinearly constrained optimization problems from applications in optimal design, fluid dynamics, parameter estimation, and optimal control."

- **MITT ("Mittelmann Collection")**; see Table A.3 in Appendix A. These are some of the problems that have been collected and implemented by Mittelmann [60]. The scripts "`cont*`" implement parabolic boundary control problems from [61] and [62], "`ex*`" implement discretizations of elliptic control problems with control and state constraints [59], and "`lukvl*`" are linearly constrained NLPs from [55], both in equality constrained and inequality constrained formulation.

Tables A.1–A.3 list for each problem the number of variables (not including the slack variables for (5.1b)), how many of those have bounds, the number of equality and inequality (or range) constraints, as well as the number of non-zero elements in the Jacobian $(\nabla c(x))^T$ of the constraints in (5.1b) and the number non-zero elements in the Hessian $\nabla^2_{xx}\mathcal{L}$ of the Lagrangian (2.2).

From the CUTE selection we consider 486 problems. The sizes of those problems vary significantly; in Table 5.1 we see the distribution of the number of variables (after including the slack variables for the range constraints (5.1b)). In our COPS set, we included from each of the 17 problems two large-scale instances of different sizes, with the number of variables varying between $12,499$ and $160,000$, except for the two `elec*` instances, which have only 600 and $1,200$ variables, but where the Hessian of the Lagrangian is 100% dense. Note, that these problem sizes exceed the original formulations in [30] by up to 2 orders of magnitude. In MITT, the problem size varies between $12,097$ and $99,998$ variables (including slacks). For each of the `ex*` we have two instances, which both exceed the size of the original formulation.

In order to solve an NLP formulated in AMPL, the model file is usually read by the AMPL language interpreter, which translates it into an input file (`*.nl`) that is then read by the solver executable. For our experiments, those input files were created only once instead of being composed for each individual run, and passed to the solver executables directly. Therefore, the reported CPU times do not include the time required by AMPL for the creation of the input files.

The AMPL interpreter offers the option to perform a *pre-solve phase*, which tries to simplify the problem by eliminating variables and/or constraints, tightening bounds, and providing improved starting points. By default, this option is activated, and it was used to create the input files for the COPS and MITT problems. However, since some of the problems in the CUTE collection have purposely been formulated in a way that might make them difficult for NLP solvers (e.g. degeneracy), the pre-solve option was disabled when creating the input files for the CUTE set.

All numerical results have been obtained on PCs running Linux, imposing a 3 hour CPU time limit and an iteration limit of 3000. Those runs, for which CPU time is reported, were done on identical 1 GHz dual Pentium III machines with 1 GB of RAM. The presented CPU times are those reported as "User time" from the UNIX `time` command (the system time is negligible). Since we observed inconsistencies in those CPU times when multiple applications were running, it was ensured that

the solver executable was the only active program. Despite this precaution we still observed deviations in some cases of up to 15% and therefore recommend to use the reported CPU times with caution.

The IPOPT executable uses the ASL version 20010823, and was compiled with the GNU compiler suite version 2.96, using the optimization flag "`-O`".

In order to compare different options of IPOPT or IPOPT vs. other solvers, *performance profiles* as proposed by Dolan and Moré [31] will be used: Assume that we have a test set $\mathcal{P}$ containing n_p problems, as well as n_s runs (e.g. with different solvers or different options for one solver) for each problem, and that we want to compare quantities $t_{p,s}$ (such as number of iterations or required CPU times) obtained for each problem p and each run s. Then the *performance ratio* for a problem p and option s is defined as

$$r_{p,s} := \frac{t_{p,s}}{\min\{t_{p,s} : 1 \le s \le n_s\}}. \tag{5.2}$$

If the option s for problem p leads to a failure, we define $r_{p,s} := \infty$. With this,

$$\rho_s(\tau) := \frac{1}{n_p}\text{card}\{p \in \mathcal{P} : r_{p,s} \le \tau\}$$

is the probability that a performance ratio $r_{p,s}$ is within a factor $\tau \ge 1$ of the best obtained performance. The larger this number is for a run s, the better. For example, the value of $\rho_s(1)$ is the probability that the option or solver s is the winner, $\rho_s(2)$ indicates in what percentage of problems the option s was not more than twice as bad as the best options, and the limit $\lim_{\tau\to\infty}\rho_s(\tau)$ gives the probability that the option or solver s can successfully solve a problem from the test set $\mathcal{P}$.

The comparisons in the next section will be presented in terms of *performance plots* of ρ_s versus τ. (Note, that in our plots the τ-axis will be logarithmic.) Since the considered methods only guarantee convergence to local solutions of the NLP, it can happen that one option or solver converges to a different local solution than the other ones. Therefore, in our performance plots we exclude those problems for which the final values of the objective functions $f(x_*^1), \ldots, f(x_*^{n_s})$ were not sufficiently close,

that is we discard those problems for which

$$\frac{f_*^{\max} - f_*^{\min}}{1 + \max\{|f_*^{\min}|, |f_*^{\max}|\}} > 10^{-3}, \tag{5.3}$$

where $f_*^{\max} = \max\{f(x_*^1), \ldots, f(x_*^{n_s})\}$ and $f_*^{\min} = \min\{f(x_*^1), \ldots, f(x_*^{n_s})\}$. Here, objective functions values of unsuccessful runs are omitted. Eq. (5.3) is of course only a simple heuristic.

5.1.2 Comparison of Different Line Search Options

In Section 3.3 and Section 3.4 we presented different line search approaches that have been implemented in IPOPT. We will now compare the performance of the following options to solve the problems from the CUTE test set.

1. auglag: Using the augmented Lagrangian function (3.25) as merit function (with non-monotone update of the penalty parameter);

2. exact1: Using the exact penalty function (3.20) with one weight for all constraints as merit function (including second order correction);

3. exact2: Using the exact penalty function (3.21) with one weight for each constraint as merit function (including second order correction);

4. filter: Using the filter line search method as described in Section 3.4 (including second order correction);

5. fullstep: Taking no precautions, i.e. the accepted step sizes are $\alpha_k = \alpha_k^{\max}$ and $\alpha_k^v = \alpha_k^{\max,v}$ and are therefore only cut by the fraction to the boundary rule (3.19). If, however, the full step leads to a point with the IEEE numbers `NaN` or `Inf` in the objective or constraint function, α_k is reduced. This very simple option is the closest to the "pure" Newton method applied to perturbed KKT conditions (2.40). The performance of this option might give an idea of the quality of the search directions and the "degree of nonlinearity" of the considered problems.

All other IPOPT options are chosen according to the default; in particular, search directions have been generated by the full-space version (see Section 3.2.1) using exact second derivatives. Table B.1 on page 194 lists the outcome for each individual problem, together with the required number of function evaluations and iterations. For filter, also the number of iterates obtained from TRON within the feasibility restoration phase is reported for each problem.

In Table 5.2 we can see the number of problems with different outcomes for the individual options. Here, "$\surd$"' means that the error tolerance was reached, and the error codes "‡" and "†" are explained in Table B.5 on page 205. As we can see, all line search options could solve at least 80% of the entire set, with the filter line search procedure being the most robust option (solving 89.1%). There were 29 problems that could not be solved by any line search option, and in the third column of Table 5.2 we see the number of problems solved successfully only by a given method, confirming the superior robustness of filter over the other line search approaches. There are also 11 problems where only fullstep could converge to a point satisfying the error tolerance (but possibly reaching a saddle point of maximum).

When facing a relatively high failure rate of 11% we should keep in mind that the CUTE test set is a fairly colorful selection of problems, and that there are a number of problems that do not satisfy the assumptions made in Chapter 4 for the analysis of IPOPT, and have for example degenerate constraints and singular reduced Hessians.

Surprisingly, using no precautions as in the fullstep option still solves the majority of problems, and seems even more robust than line search with an exact penalty function, which in more than 40 cases failed due to line search failure, i.e. the trial step size is cut back by the merit function too much and is becoming smaller than 10^{-14}.

In 27 cases, at least two options converged to points where the convergence criterion was met and where the final values of the objective function were different (according to heuristic (5.3)). In Table 5.3 we see for each of those problems the values of $f_*^{\min}$ and $f_*^{\max}$, and which of the options converged to those values. For the

Line search option	Outcome		solved by no other option	like fullstep	#func/ #iter
auglag	421	$\surd$	1	41.1%	4.4666
	1	‡			
	42	†1			
	13	†2			
	8	†5			
	1	†9			
exact1	393	$\surd$	0	36.9%	7.2655
	1	‡			
	40	†1			
	48	†2			
	4	†5			
exact2	397	$\surd$	1	40.1%	5.8151
	3	‡			
	42	†1			
	39	†2			
	5	†5			
filter	433	$\surd$	10	45.0%	2.6469
	3	‡			
	24	†1			
	6	†3			
	1	†4			
	1	†5			
	3	†6			
	6	†7			
	9	†8			
fullstep	403	$\surd$	11	100%	1.0003
	6	‡			
	23	†1			
	10	†2			
	17	†5			
	26	†9			

Table 5.2: Outcome for individual line search options on CUTE problems

Problem	$f_*^{\min}$	Option	$f_*^{\max}$	Option
bt7	3.0650000e+02	a,e1,e2,f	3.6037977e+02	n
camel6	-1.0316285e+00	a,e1,e2,f	-2.1546382e-01	n
deconvc	5.6468218e-10	f,n	2.5694971e-03	a
drcavty2	6.5425099e-04	a	1.9243839e-03	e1
dtoc1nd*	1.2563879e+01	n	1.2786890e+01	a
eg3	9.5316949e-14	e1,e2	6.7179988e-02	a,f,n
fletcher	1.1656854e+01	a,f	1.9525366e+01	n
growth	1.0040406e+00	a,e1,e2,f	3.5421490e+03	n
growthls	1.0040406e+00	a,e1,e2,f	3.5421490e+03	n
haldmads	1.2238631e-04	a,e1	3.4659284e-02	f,n
himmelp2	-6.2053869e+01	e2	-8.1980317e+00	a,e1,f,n
hs059	-7.8027895e+00	a,f	-6.7495053e+00	e2
hs061	-1.4364614e+02	n	-8.1919096e+01	f
hs070	9.4019732e-03	a,e1,e2,f	1.6809117e-01	n
hs098	3.1358091e+00	e1,e2,n	4.0712464e+00	a,f
orthrds2*	3.0540043e+01	e2	1.5490713e+03	n
orthregd	1.5238997e+03	a,e1,e2	2.6066512e+03	f
orthrege*	6.7807800e-01	a,e1,e2	1.5173881e+00	n
orthrgds	1.5238997e+03	a,e1,f	7.8409872e+04	n
ssebnln	1.6844769e+07	e2	1.8047483e+07	f
steenbrc	1.8274717e+04	n	2.0357558e+04	a,f
steenbrd	9.2416335e+03	a,e1,e2,n	9.4372195e+03	f
steenbrf*	2.8267956e+02	f	3.1983506e+05	n
weeds	2.5872774e+00	n	9.2054352e+03	a,e1,e2,f
womflet	7.5177107e-09	a,e1,e2	6.0500000e+00	f,n
yfit	6.6697375e-13	a,e1,e2,f	5.9757165e+03	n
yfitu	6.6697206e-13	a,e1,e2,f	5.6141752e+03	n

Table 5.3: Diverting final objective function values; with "a" for auglag, "e1" for exact1, "e2" for exact2, "f" for filter, "n" for fullstep

problems marked by a start (*), there were additional final values of the objective function, lying in between the values given in the table. As a trend we can see, that pure Newton's method, taking no precautions, can be attracted to points with larger values of the objective function, possibly even saddle points or maxima, whereas the other options safely guide the algorithm to better points. On the other hand, for the weeds problem, the willingness to take risks pays off in a significantly smaller value of the objective function.

In order to see how conservative the individual line search mechanisms are, we list in the fourth column of Table 5.2 the percentage of successfully solved problems, for which the individual options took the same steps as the unguarded fullstep option. The last column shows the ratio of the total number of function evaluations over the total number of iterations for all successfully solved problems[1]. From this we see, that line search using the exact penalty function seems to be most conservative, as expected with exact1 being stricter than exact2, followed by the augmented Lagrangian line search (which in our implementation has the "advantage" to be allowed to decrease the value of the penalty parameter ν but might not be guaranteed to be globally convergent). The least conservative option seems to be the filter option, requiring on average only about 21/2 function evaluations per iteration, and in 45% of the successful cases behaving like the pure Newton's method. Among the 433 problems successfully solved by filter, TRON was called only in 30 instances to perform the feasibility restoration.

Finally, in order to evaluate the efficiency of the individual line search options, a performance plot comparing the required number of iterations is presented in Figure 5.1, excluding the 26 problems from Table 5.3. It shows the superiority of the filter method over the other options, followed by the augmented Lagrangian option and exact2. The observation, that fullstep shows the worst performance despite the fact that all options took in more than $37\% - 45\%$ of the problems steps identical to fullstep, seems to indicate that overall the step acceptance criteria interfere with pure Newton's method in the appropriate circumstances, at least for the considered test set.

Interestingly, exact2 is the winner, if one only considers the 335 problems solved by *all* options, with $\tilde{\rho}_{\text{exact2}}(1) = 0.79$, $\tilde{\rho}$ for auglag, exact1, filter around 0.74, and $\tilde{\rho}_{\text{exact2}}(1) = 0.67$. This might indicate, that there is a number of instances where it pays to be more cautious.

[1] The number of iterations was corrected by one, so that a value of 1.0 corresponds to always taking full steps.

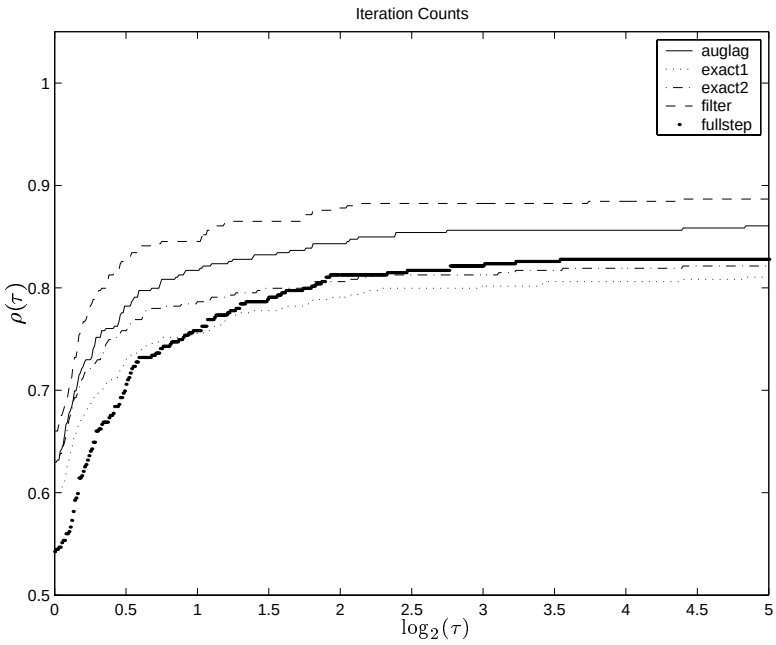

Figure 5.1: Performance plot comparing different line search options

5.1.3 Comparison with KNITRO and LOQO

In this section we present a numerical comparison of IPOPT with two other interior point solvers for general nonlinear programming, KNITRO 2.00 and LOQO 6.02. A brief description of the underlying algorithms has been presented in Section 2.8.

IPOPT and LOQO are relatively similar in the sense that they are both primal-dual interior point methods which follow a line search approach and generate Newton search directions for some formulation of the primal-dual equations. Both handle negative curvature by adding a multiple of the identity to the Hessian, and require the factorization of similar sparse indefinite symmetric matrices, see (3.7) and (2.48). Differences lie for example in the step acceptance criterion, the update procedure for the barrier parameter, and in the way equality constraints are handled. KNITRO on the other hand is a trust region based barrier method that employs only inexact Newton steps, which are obtained partly by a PCG method. All methods use exact second derivatives.

Before presenting the results, we should point out the following caveats: The

three codes have been developed by different people, so that as a consequence the implementations are very different: IPOPT and KNITRO are written in FORTRAN 77, whereas LOQO is coded in C; IPOPT and KNITRO use the generic Harwell routine `MA27` [32] to solve the occurring sparse symmetric indefinite linear systems, whereas LOQO uses its own linear solver; algorithmic constants such as pivot tolerances or small factors in decrease conditions etc. might be chosen differently. Furthermore, since the basic algorithms address different NLP formulations ((1.1) vs. (2.43) and (2.50)), the same AMPL formulation might lead to different formulations seen by the solvers. As a consequence, the presented results give only limited information on performance comparisons of the mathematical algorithms, measured for example in iteration count. In particular, implemented heuristics for special cases and ill-conditioning have a large impact on robustness. Therefore, the present results mainly compare the practical performance of software packages at a certain stage of development. Here, the degree of robustness and the required computation time are the important factors.

Appendices B–D present for each solver the details of the runs for the problems in the CUTE, COPS, and MITT test set, in particular the number of iterations and CPU seconds required, which are summarized in the performance plots in Figures 5.2–5.4.

All codes were run with default settings (i.e. line search option filter for IPOPT), except that the maximum number of iterations was increased to 3000. The termination criteria are not identical (due to different scaling), but we tried to choose them approximately similar by tightening the tolerances for KNITRO to `feastol` $= 10^{-8}$ and `opttol` $= 10^{-8}$ (as for IPOPT). LOQO terminates (by default) when both (scaled) primal and dual infeasibility are below 10^{-6} and the optimal value of the objective function is correct by at least 8 digits. Also, all codes have been compiled with GNU compilers (version 2.96) using the optimization flag "`-O`".

When comparing the number of iterations, we should keep in mind that the major amount of computational work required by IPOPT and LOQO per iteration is in principle very similar (factorization of a large symmetric indefinite matrix (2.48) and

(3.3)), unless the iteration in IPOPT is taking place within TRON when performing the restoration phase. (Then the amount of time spent per iteration depends on the number CG iterations and the number of rejected trial steps in TRON.) On the other hand, KNITRO counts each trial point as iteration, so that one KNITRO iteration always includes the solution of the reduced subproblem by PCG, but the factorization of the augmented system (2.46) (which is simpler and less dense than the matrices considered by IPOPT or LOQO) is only necessary after a successful iteration. Which of those two tasks is more time consuming depends on the characteristics of a particular problem. For this reason, we include in our iteration count comparison also an option for KNITRO (called "KNITRO (fact)"), where the number of factorizations (which equals the number of successful steps) is taken as iteration count. Finally, also the results for IPOPT's unguarded fullstep option are included in the comparison, in order to obtain an idea of the "difficulty" of the considered problems (in this case we count failure for fullstep if the final value of the objective function is different from the one used by the other options).

When creating the performance plot for the CUTE iteration count comparison, we excluded 50 of the total 486 instances, in which two methods successfully stopped at points with different values of the objective function (as defined by (5.3)). Since the computation time is negligible for small sized problems, we included in the second plot in Figure 5.2 only the 179 problems with at least 100 variables (including slacks; see Table 5.1). Among those, 19 were excluded because of (5.3), as well as 2 among the 34 COPS and 4 among the 56 MITT problems.

From the asymptotic behavior in the performance plots as $\tau \to \infty$ we can see that IPOPT seems to be the most robust solver (at least within the given timeframe of 3 hours), which is supported by Table 5.4 where the number of successfully solved problems for each test set are shown. In addition, the rows marked "only" show how many problems in each set were solved only by the given code. Among the CUTE problems there are 17 that could not be solved by any method, 4 among the COPS and 2 among the MITT problems.

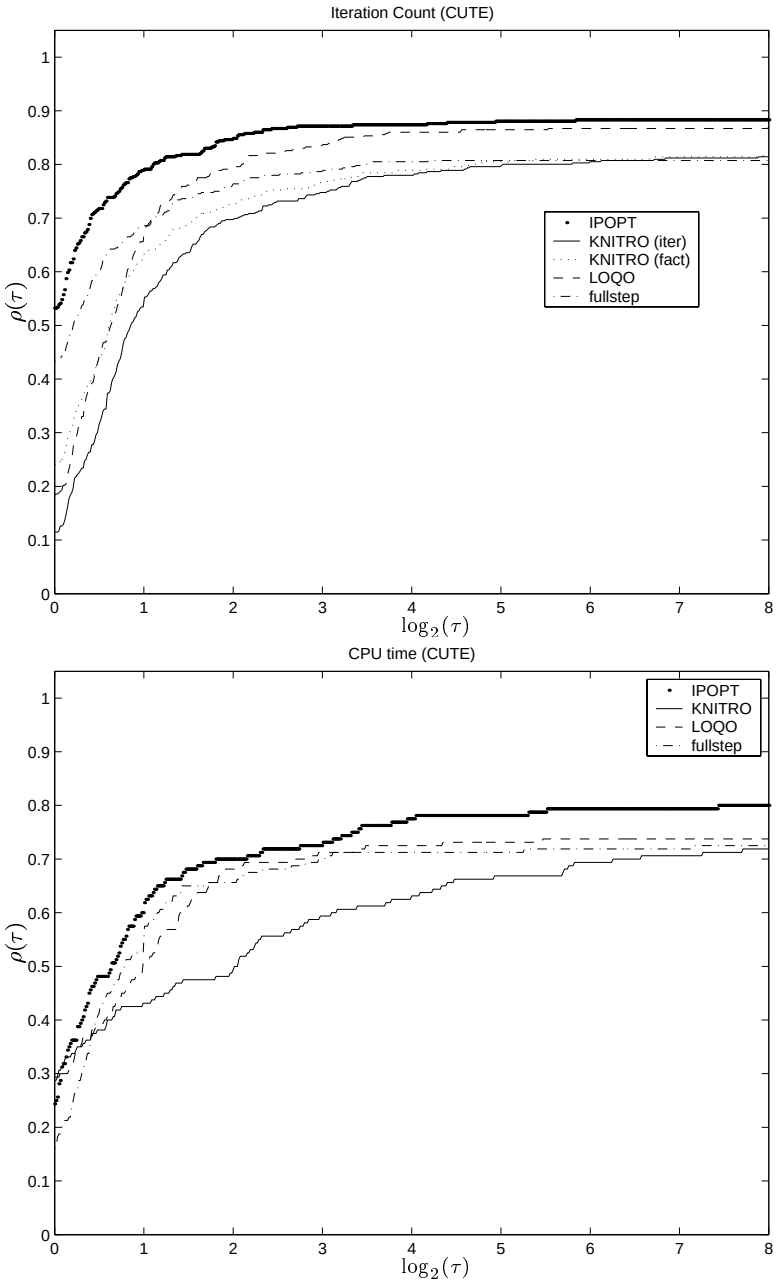

Figure 5.2: Performance plots for CUTE test set

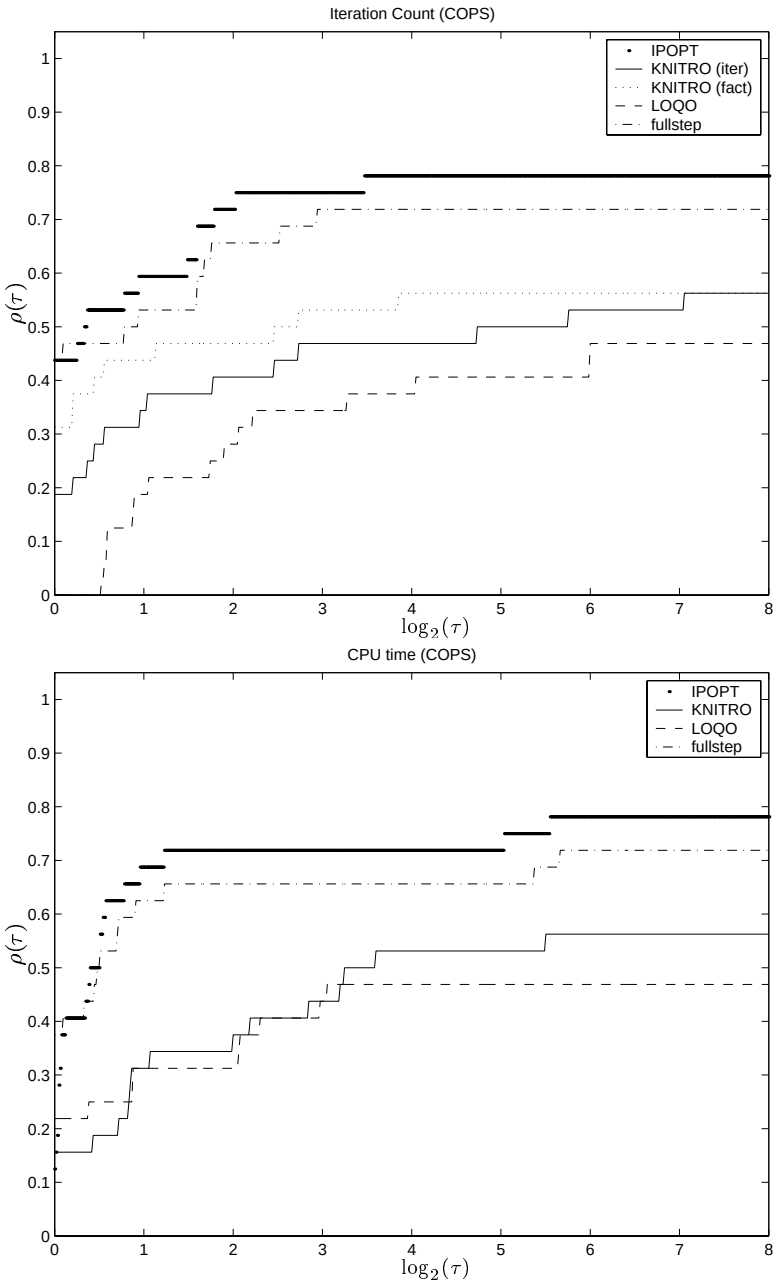

Figure 5.3: Performance plots for (large-scale) COPS test set

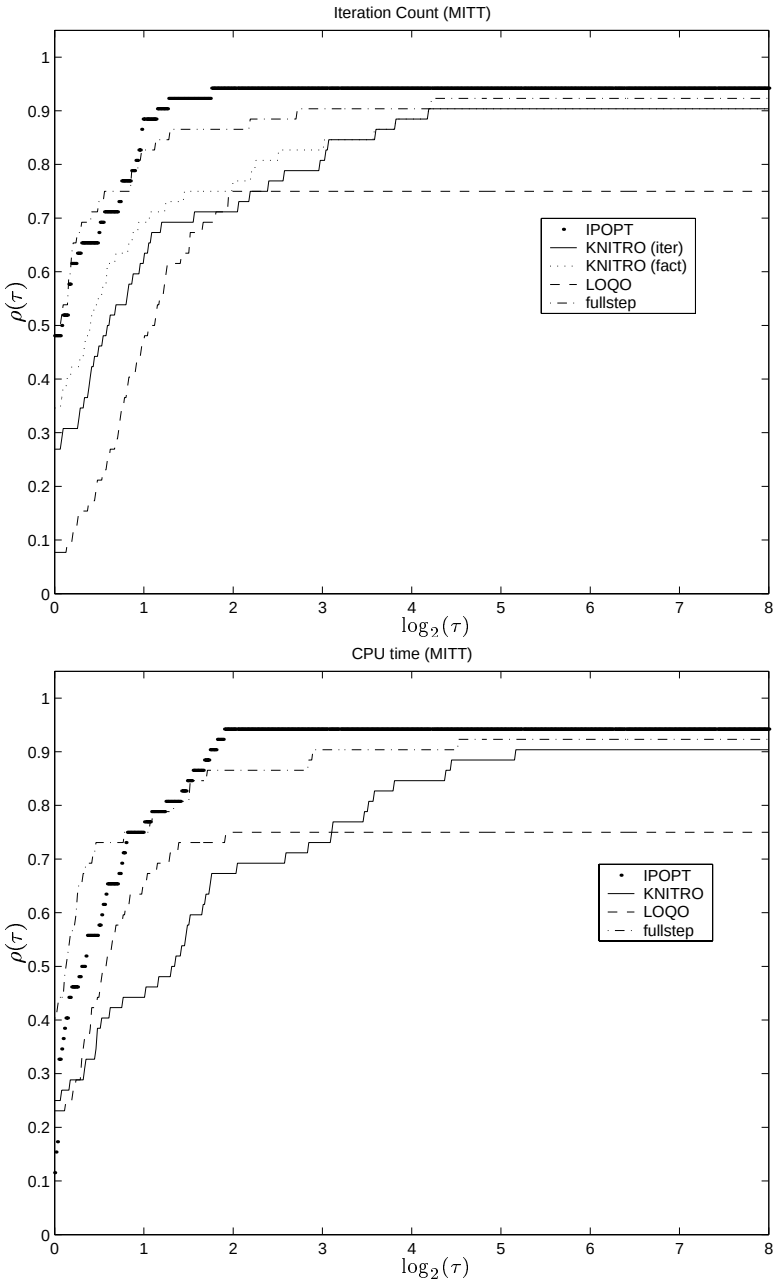

Figure 5.4: Performance plots for MITT test set

Solver	Code	CUTE	COPS	MITT
IPOPT	√	433	27	53
	‡	3	5	
	†1	24	1	1
	†3	6		
	†4	1		
	†5	1		
	†6	3	1	
	†7	6		1
	†8	9	1	1
	only	8	4	3
KNITRO	√	399	20	51
	‡	13	12	2
	†1	30	2	
	†2	42		3
	†3	2		
	only	9	2	1
LOQO	√	420	15	41
	‡	2	11	6
	†1	38	4	1
	†2	10		2
	†3	2		1
	†4	14	2	5
	†5		2	
	only	16	0	0

Table 5.4: Number of failures for test sets

We can further see that IPOPT is clearly superior in terms of iteration counts, even compared to "KNITRO (fact)". Also with respect to CPU time, IPOPT is seemingly the best method, but less outstandingly, closely followed by LOQO for the CUTE set. Since in most cases the task required per iteration by IPOPT and LOQO is relatively similar (factorization of (3.7) and (2.48), respectively), one could expect IPOPT to win clearly regarding computation time as well, but the performance plots do not support this conjecture. Possibly, this is due to the different solvers used to perform the factorization of the sparse, symmetric, indefinite matrices: In IPOPT, the generic Harwell routine `MA27` (with fixed pivot tolerance 10^{-8}) is used, whereas in LOQO this is done by a specialized implementation (see e.g. [75]) which seems to perform very well in many cases. Consider for example the MITT problem `ex4_160`,

for which LOQO requires only 5.1 CPU seconds per iteration, but IPOPT needs 19.3 CPU seconds per iteration, although it never had to factorize the KKT system (3.3) multiple times in order to find a suitable value for the regularization parameters δ_1 and δ_2 (#reg = 0). Profiling of the code shows, that IPOPT spends 90% of the computation time within the factorization routine `MA27BD`. Reducing the pivot tolerance to 10^{-12} leads only to a marginal decrease in CPU time. Therefore, it might be possible to enhance IPOPT's performance in the future by investigating alternative options for solving the linear system (3.7).

In comparison, the overall performance of KNITRO seems not as high. There are significant differences in the algorithmic principle compared to IPOPT and LOQO, that might be responsible for this. In particular, for large-scale problems with many degrees of freedom, KNITRO may perform a large number of PCG steps (and therefore many backsolves for the linear system (2.46)), in order to compute a trial step. On the other hand, one might therefore expect that KNITRO would outperform IPOPT on large-scale problems with only few degrees of freedom, such as the parameter estimation problems `gasoil`, `marine`, `methanol`, and `pinene` from COPS, but this is not supported by the presented results. In particular the case `pinene_5000` is puzzling, since KNITRO requires only 6 iterations, a very small fraction of the iteration count for IPOPT, and only a total of 10 PCG iterations, but 1.7 times the CPU time required by IPOPT. This weak performance might therefore be caused by some inefficiency hidden in the particular implementation. However, on the COPS problems `elec`, KNITRO clearly performs better than IPOPT; in this particular case the Hessian of the Lagrangian is dense, which leads to a considerable amount of work in IPOPT's factorization of (3.3), whereas KNITRO only has to perform products with the Hessian.

Finally, it is interesting to note, that also for the COPS and MITT problems the unguarded fullstep option of IPOPT performs surprisingly well; for the MITT suite it even outperforms the regular filter option. We can interpret this finding in two ways. Either the search directions generated by IPOPT, essentially Newton steps for the

perturbed KKT conditions, are in many cases very good and need not be modified, or the considered problems are relatively easy to solve.

5.1.4 Summary

In conclusion, IPOPT seems to be a robust and efficient method for solving large-scale NLPs. Among the implemented line search options, the novel filter approach seems overall to be the most robust and effective method.

The comparison presented in Section 5.1.3 is intended to demonstrate IPOPT's potential as a general purpose NLP solver, not as a thorough comparison of the three discussed methods, which is beyond the scope of this dissertation. Such a direct comparison should be performed by an independent party and would have to ensure that the conditions for each method (such as convergence criteria) are as similar as possible. Furthermore, one would have to examine in detail the individual reasons for failure of the solvers on the individual problems.

For example, there are some instances among the considered test problems, in which the error tolerance of 10^{-8} seems too tight, such as the MITT problem `ex4_2_80`, in which IPOPT after 22 iterations makes no progress in the objective function, the (unscaled) value of the infeasibility stays around 10^{-8}, but the error criterion does not go below 10^{-7}. LOQO fails on this problem in a similar manner.

5.2 Dynamic Optimization Problems in Process System Engineering

Many physical phenomena can be described in terms of *algebraic and differential equations (DAEs)*, and therefore models for chemical equipment such as reactors, distillation columns, and entire chemical processes are frequently formulated as

$$F(z(t), \dot{z}(t), y(t), u(t), p) = 0 \tag{5.4a}$$

$$G(z(t), y(t), u(t), p) = 0, \tag{5.4b}$$

where $z : [t_0, t_f] \longrightarrow \mathbb{R}^{n_z}$ and $y : [t_0, t_f] \longrightarrow \mathbb{R}^{n_y}$ are time-dependent *state variables*, whose profiles depend on the initial conditions

$$z(t_0) = z_0 \tag{5.5}$$

for some $z_0 \in \mathbb{R}^{n_z}$, time-dependent *control variables* $u : [t_0, t_f] \longrightarrow \mathbb{R}^{n_u}$, and time-independent *parameters* $p \in \mathbb{R}^{n_p}$. Here, $\dot{z}(t)$ denotes the derivative of $z(t)$ with respect to t. The (implicit) *differential equations* $F : \mathbb{R}^{n_z \times n_z \times n_y \times n_u \times p} \longrightarrow \mathbb{R}^{n_z}$ and *algebraic equations* $G : \mathbb{R}^{n_z \times n_y \times n_u \times p} \longrightarrow \mathbb{R}^{n_y}$ are assumed to be index one. This ensures that discretizations of the equations (as described below) are full rank. In many cases, t corresponds to time, and we will assume here that the length of the interval $[t_0, t_f]$ is fixed.

There is a variety of computational methods that have been developed for the solution of the DAE system (5.4)–(5.5) and are used for the *simulation* of chemical processes [2]. In this case it is assumed that the profiles of the control variables $u(t)$ as well as the parameters p are given.

As a natural extension one may then ask the question, how to choose the control variables and parameters in a way that optimizes some objective. In what follows we assume that such a *dynamic optimization problem* is of the form

$$\begin{aligned}
\min_{z,y,u,p} \quad & f(z(t_f), y(t_f), u(t_f), p) && \text{(5.6a)}\\
\text{s.t.} \quad & \text{implicit DAE system (5.4)} && \text{(5.6b)}\\
& \text{initial conditions (5.5)} && \text{(5.6c)}\\
& \text{bound constraints for } z, y, u, p && \text{(5.6d)}
\end{aligned}$$

for some scalar objective function f.

In a *sequential approach*, the profile of the control variables $u(t)$ is discretized and approximated by piece-wise polynomial functions. The "unknowns" in this optimization approach are the values of the coefficients in these polynomials as well as the parameters p, whose optimal values are computed by a nonlinear optimizer. For this, the DAE system (5.4)–(5.5) is solved for given values of these unknowns

by an integrator, providing the corresponding values of the objective function and the constraints. The derivatives can be obtained by solving either a DAE system of sensitivity or adjoint equations.

The advantage of this approach is that existing integrators and simulators can be used, and that the resulting NLP formulation is small. However, a possible inefficiency of this approach is that the DAE system has to be integrated in each iteration of the optimization process, although the exact profile of the state variables is not of interest at non-optimal points. In particular the evaluation of derivative information is computationally very expensive.

The alternative *simultaneous approach* tries to avoid this inefficiency by solving the DAE system at the same time as finding the optimal control profiles and parameters. For this, the profiles of the control variables as well as the state variables is discretized, leading to a finite dimensional approximation of the infinite dimensional problem (5.6). Such an approach based on orthogonal collocation [2] is described next.

5.2.1 Orthogonal Collocation and Elemental Decomposition

The DAE optimization problem (5.6) is converted into an NLP by approximating state and control variables by a family of polynomials on ne finite elements ($t_0 < t_1 < \ldots < t_{ne} = t_f$). Here, we use a monomial basis representation [4] for the differential state profiles, which is

$$z(t) = z_{i-1} + h_i \sum_{q=1}^{ncol} \Omega_q \left(\frac{t - t_{i-1}}{h_i} \right) \dot{z}_{i,q}. \tag{5.7}$$

Here, z_{i-1} is the value of the differential state variables at the beginning of element i, h_i is the length of element i, $\dot{z}_{i,q}$ is the value of its first derivative in element i at the collocation point q, and Ω_q is the polynomial of degree $ncol$, satisfying

$$\begin{aligned} \Omega_q(0) &= 0 \qquad && \text{for } q = 1, \ldots, ncol \\ \Omega_q'(\rho_r) &= \delta_{q,r} \qquad && \text{for } q, r = 1, \ldots, ncol, \end{aligned}$$

where $\rho_r \in [0,1]$ is the location of the r^{th} collocation point within a normalized element. Continuity of the differential variables is enforced by

$$z_i = z_{i-1} + h_i \sum_{q=1}^{ncol} \Omega_q(1)\, \dot{z}_{i,q}. \tag{5.8}$$

Radau collocation points are used because they allow us to set constraints easily at the end of each element and to stabilize the system more efficiently if high index DAEs are present.

In addition, the control and algebraic state profiles are approximated using a similar monomial basis representation, which takes the form

$$y(t) = \sum_{q=1}^{ncol} \psi_q\left(\frac{t - t_{i-1}}{h_i}\right) y_{i,q} \tag{5.9a}$$

$$u(t) = \sum_{q=1}^{ncol} \psi_q\left(\frac{t - t_{i-1}}{h_i}\right) u_{i,q}. \tag{5.9b}$$

Here, $y_{i,q}$ and $u_{i,q}$ represent the values of the algebraic state and control variables, respectively, in element i at collocation point q. ψ_q is the Lagrange polynomial of degree $ncol - 1$ satisfying

$$\psi_q(\rho_r) = \delta_{q,r} \qquad \text{for } q, r = 1, \ldots, ncol.$$

From (5.8), the differential state variables are required to be continuous throughout the time horizon, while the control and algebraic state variables are allowed to have discontinuities at the boundaries of the elements. For the presented results we assume that the number of finite elements, ne, and their lengths, h_i, are pre-determined.

The bound constraints (5.6d) are approximated by placing them on the coefficients $z_i, y_{i,q}, u_{i,q}, p$. It should be mentioned that with representation (5.7), the bounds on the differential variables are enforced directly only at element boundaries; however, they can be enforced at all collocation points by introducing identical algebraic state variables. Substitution of equations (5.9) into (5.6) finally leads to an NLP of the form (1.1) where $x = \{\dot{z}_{i,q}, z_i, y_{i,q}, u_{i,q}, p\}$, and $c(x)$ collects the DAEs (5.4) at all collocation points, together with (5.5) and (5.8).

Using the variable partition (2.26) within IPOPT's reduced space version (see Sections 3.2.2–3.2.5) allows us to perform a special decomposition of the Jacobian A_k^T that we will briefly explain. In the remainder of this section the iteration index k will be omitted for simplicity.

First, consider the Jacobian of the discretized DAE system

$$A^T = \begin{bmatrix}
Z_{\text{init}}^0 & & & & & & & & 0 & & 0 \\
Z_1^0 & \dot{Z}_1^1 & Y_1^1 & & & & & & U_1^1 & & P_1^1 \\
Z_2^0 & \dot{Z}_2^1 & Y_2^1 & & & & & & U_2^1 & & P_2^1 \\
\vdots & \vdots & \vdots & & & & & & \vdots & & \vdots \\
Z_{ncol}^0 & \dot{Z}_{ncol}^1 & Y_{ncol}^1 & & & & & & U_{ncol}^1 & & P_{ncol}^1 \\
I & D^1 & 0 & -I & & & & & 0 & & 0 \\
 & & & Z_1^1 & \dot{Z}_1^2 & Y_1^2 & & & 0 & U_1^2 & P_1^2 \\
 & & & & \ddots & \ddots & & & & \ddots & \vdots \\
 & & & & \ddots & \ddots & & & & \ddots & \vdots \\
 & & & & & Z_{ncol}^{ne-1} & \dot{Z}_{ncol}^{ne} & Y_{ncol}^{ne} & & U_{ncol}^{ne} & P_{ncol}^{ne} \\
 & & & & & I & D^{ne} & 0 & -I & &
\end{bmatrix} \tag{5.10}$$

where I represents the identity matrix of appropriate dimension, and D^i is a matrix containing the coefficients of the continuity equations of the i^{th} element. Z_q^i, $\dot{Z}_q^i$, Y_q^i, U_q^i and P_q^i represent the Jacobian of the collocation equations with respect to z_i, $\dot{z}_{i,q}$, $y_{i,q}$, $u_{i,q}$ and p, at collocation point q and element i. The factorization of this matrix is performed over smaller matrices, each one representing a finite element. To explore this decomposition, consider the rows and columns of A^T that correspond to element i

$$(A^i)^T = \begin{bmatrix}
Z_1^{i-1} & \dot{Z}_1^i & Y_1^i & 0 & U_1^i & P_1^i \\
Z_2^{i-1} & \dot{Z}_2^i & Y_2^i & 0 & U_2^i & P_2^i \\
\vdots & \vdots & \vdots & 0 & \vdots & \vdots \\
Z_{ncol}^{i-1} & \dot{Z}_{ncol}^i & Y_{ncol}^i & 0 & U_{ncol}^i & P_{ncol}^i \\
I & D^i & 0 & -I & 0 & 0
\end{bmatrix}. \tag{5.11}$$

In order to apply the reduced space algorithm, we need to partition the variables according to (2.26). The overall partition is obtained by selecting dependent and independent variables within each element. A user may specify this partition explicitly, for example choosing all control variables and parameters as independent variables. Alternatively, we can apply an LU factorization with partial pivoting on each rectangular system A^i. As discussed in [51], this LU factorization will yield a dichotomous system in each element. Thus, if an unstable mode is present in the DAE system, $(A^i)^T$ is required to be partitioned so that, in effect, the end conditions of any increasing mode are fixed or become decision variables. In the partitioning, if a differential variable z_j has an increasing mode, $\dot{z}_{j,ncol}$ would be identified and shifted from a column in C to a column in N. Correspondingly, a column corresponding to a control variable would be shifted from N to C. By considering the independent variables to be fixed, the decomposition approach then becomes equivalent to solving a discretized, linear boundary value problem.

In the current implementation of the elemental decomposition it is assume that the parameters are always chosen to be independent variables, so that the decomposition of this matrix can be performed directly, as all the variables can be eliminated locally. In the case that a parameter is present and chosen to be a dependent variable, the last column of $(A^i)^T$, which corresponds to the parameters, will be coupled to the entire system. For this case, a two stage decomposition will have to be implemented (see Section 6.2).

After the basis has been selected, we can represent the overall matrix A^T with

the following structure and partition

$$A^T = \left[\begin{array}{ccccccccccc} I & & & & & & & 0 & & & \\ T^1 & C^1 & & & & & & N^1 & & & \\ I & \widehat{C}^1 & -I & & & & & \widehat{N}^1 & & & \\ & & T^2 & C^2 & & & & & N^2 & & \\ & & I & \widehat{C}^2 & -I & & & & \widehat{N}^2 & & \\ & & & & T^3 & C^3 & & & & N^3 & \\ & & & & & \ddots & \ddots & & & & \ddots \end{array}\right] = \left[\begin{array}{c|c} C & N \end{array}\right],$$

and the corresponding right hand sides are

$$c^T = \left[\begin{array}{ccccccc} c^0 & c^1 & \widehat{c}^1 & c^2 & \widehat{c}^2 & c^3 & \cdots \end{array}\right].$$

Cervantes [25] had implemented a forward decomposition strategy for linear systems involving C, computing the entries of $\bar{q}$ in (2.29) and rows of $C^{-1}N$ in (2.28) for each element $i = 1, \ldots, ne$. The dense matrix $C^{-1}N$ was calculated and stored explicitly, and the factorization of the individual matrices C^i was discarded after the corresponding element had been addressed. The new approach, implemented within this Ph.D. project, instead stores the factorization of the individual C^i matrices and avoids the explicit storage of $C^{-1}N$. This leads to a considerable reduction in memory requirement and computation time for large-scale problems, and also allows the computation of multiplier estimates from (3.11), in contrast to the previous approach.

The current implementation of the elemental decomposition allows a user to model the process

1. as a set of FORTRAN subroutines, that provide function and derivative information of the model equations;

2. as a FORTRAN subroutine that implements only the model equations. The source file is then automatically translated into C++ and derivative information is obtained from the automatic differentiation package ADOL-C [49];

3. in the modeling environment gPROMS [72], which can communicate with the solver executable using an "equation solver object".

For convenient usage, IPOPT and the elemental decomposition has been integrated into DynoPC developed by Lang and Biegler [52], which provides a graphical user interface for easy definition of scenarios and monitoring of the optimization progress.

5.2.2 Continuous Air Separation Distillation

In the next two sections we compare the different options described in Sections 3.2.3–3.2.5 for approximating second order information and for solving the reduced system (3.9).

The first example simulates a simple air separation using a continuous distillation column with 15 trays. The feed is assumed to have a composition of 78.1% Nitrogen, 21% Oxygen, and 0.9% Argon. The purity of Nitrogen taken out at the top of the column is 99.8%. The complete model consists of 70 differential equations and 356 algebraic equations. Here the objective is to minimize the offset produced during the change from one steady state to another by controlling the feed flow rate F over a time horizon of $t_f = 10h$. These steady states are defined by the distillate flow rates operating at $D(0) = 301.8\ mol/h$ to $D^{set} = 256.0\ mol/h$. With this, we pose the optimization problem as follows.

$$\min \quad \int_0^{t_f} (D - D^{set})^2 dt \tag{5.12a}$$

$$\text{s.t.} \quad \text{DAE model (5.4)–(5.5)} \tag{5.12b}$$

$$0\ kmol/h \leq F \leq 2\ kmol/h. \tag{5.12c}$$

The starting point for this problem is the steady state value at $D(0)$ and the initial control profile is specified by $F = 1.5 kmol/h$.

For model equations (5.4) have been implemented in FORTRAN, as well as their (sparse) analytical gradients. Details of the model and the optimal solution can be found in [26], where an early version of IPOPT was used to perform the optimization.

The largest instance solved in [26] had 60 finite elements with each 2 collocation points, which corresponds to an NLP with $55,510$ optimization variables.

We now compare the following cases for solution of this problem.

- QN-BFGS: Use the BFGS update to approximate $Z_k^T W_k Z_k$ (option 2 on page 38).
- QN-SR1: Use the SR1 update to approximate $Z_k^T W_k Z_k$ (option 3 on page 39). Here the overall reduced Hessian is possibly modified by adding a multiple of the identity to ensure positive definite (option 3 on page 40).
- Explicit: Use finite difference approximations (3.12) to calculate $Z_k^T H_k Z_k$ explicitly (option 1 on page 38).
- PCG1: Use the PCG method (described in Section 3.2.5) employing finite differences (3.12). In order to obtain the preconditioner, approximate the overall reduced Hessian by BFGS (option 1 on page 42).
- PCG2: Use the PCG method (described in Section 3.2.5) employing finite differences (3.12). Here, approximate only the reduced Hessian corresponding to the *original* NLP by damped BFGS, and construct an estimate of the overall reduced Hessian by calculating $Z_k^T \Sigma_k Z_k$ explicitly to obtain the preconditioner (option 2 on page 42).

We note that for this example bounds are only imposed on control variables, which are chosen to be the independent variables. As a result we have $Z_k^T \Sigma_k Z_k = \Sigma_u$, which is a diagonal matrix corresponding to the control bounds and their multipliers, and which is therefore very easy to evaluate. This accounts for the fast performance in options PCG2, QN-BFGS and QN-SR1. To present performance behavior with the PCG2 case on more general control problems, we also provide the calculation of the complete matrix $Z_k^T \Sigma_k Z_k$, even though only Σ_u is relevant here. This option is labeled as PCG2*.

No. of elements	Problem size n / m	QN-BFGS Iter/ CPU	QN-SR1 Iter/ CPU	Explicit Iter/ CPU
50	67,620 / 67,470	205/ 3.43	82 / 1.39	10 / 5.04
100	135,170 / 134,870	239/ 8.10	100/ 3.47	12 / 23.87
200	270,270 / 269,670	283/ 21.30	123/ 9.30	11 / 87.83
500	675,570 / 674,070	314/ 99.87	209/ 69.22	11 / 539.16
900	1,215,970/1,213,270	449/527.25	287/369.70	11 /1721.08

No. of	PCG1	PCG2	PCG2*
elements	Iter/ CPU [#cg]	Iter/ CPU [#cg]	Iter/ CPU [#cg]
50	26 / 2.30 [558]	11 / 0.93 [169]	11 / 3.68 [169]
100	39 / 7.36 [897]	10 / 2.13 [199]	10 / 12.01 [199]
200	29 / 23.82 [1589]	12 / 8.65 [404]	12 / 56.44 [404]
500	76 /114.70 [2844]	12 / 33.10 [508]	12 / 325.51 [508]
900	39 /288.89 [4157]	12 /125.73 [871]	12 /1072.48 [871]

Table 5.5: Iteration count and CPU time (in minutes) for air separation example

All of these cases were run on an 1 GHz Pentium III Intel processor and were initialized to a feasible solution corresponding to the steady state at $t = 0$. The imposed convergence tolerance is 10^{-6}, and for the PCG options the maximal number of CG iterations per interior point iteration is limited to one third of the number of degrees of freedom. The CG convergence factor is chosen 10^{-1}.

Table 5.5 shows the computational performance for this example for three collocation points and for different numbers of elements. For each problem its size, the required number of iterations (Iter) and CPU time (CPU; in minutes), and for the PCG options also total number of conjugate gradient iterations (# cg) is listed.

From the results in this table, several features are worth noting. First it is clear that the Newton-based options (Explicit, PCG1, PCG2 and PCG2*) require far fewer interior point iterations than the quasi-Newton methods. In most cases this factor exceeds an order of magnitude. However, the Newton methods clearly require more

work per iteration, with the explicit approach incurring a large cost due to $n - m$ of Hessian vector products per IP iteration. An interesting feature can also be seen with the two PCG options. The first preconditioner is two to four times slower than the second one. This can be explained because the first preconditioner requires about three to six times more PCG iterations and Hessian vector products. PCG2 requires less work because its preconditioner has separated terms which are updated more accurately. On the other hand, PCG2 (as well as QN-BFGS and QN-SR1) is greatly aided by the simple form of $Z_k^T \Sigma_k Z_k = \Sigma_u$ for this particular example. For more generally constrained problems, $Z_k^T \Sigma_k Z_k$ is more costly to update. This can be seen with option PCG2*, which requires the same low number of PCG iterations as option PCG2 but is now about three times as expensive as PCG1.

The results also suggest that the SR1 update provides better quasi-Newton estimates than the BFGS formula, requiring only about half the number of iterations. In this example, QN-SR1 is almost competitive with PCG2. However, we should keep in mind that the required tolerance of 10^{-6} is not extremely tight, and a more significant difference can be expected choosing a stricter convergence criterion, where PCG2 has the advantage of providing very fast local convergence.

Overall, we see that the proposed interior point algorithm is quite attractive for solving very large nonlinear programming problems. The size of the problems in Table 5.5 is limited by the memory requirements, and the largest problem with over 1.2 million variables and $2,700$ degrees of freedom could be solved in a little more than 2 hours. We also solved a larger instance with $1,500$ finite elements on a 1 GHz dual-Pentium III machine with more memory, which corresponds to a problem with $2,026,570$ variables and $4,500$ degrees of freedom. This instance was solved by the PCG2 option in 11 iterations using $1,222$ PCG iterations. For this, a computation time of only 411.09 CPU minutes was necessary.

The optimization problems arising from this example are very large, and have only a moderate number of degrees of freedom compared to the total number of variables. In addition, the matrix $Z_k^T \Sigma_k Z_k$ is very easy to compute for this ap-

plication, so that certain options are favored, such as the quasi-Newton methods. In the following section, a considerably smaller example is presented with different characteristics.

5.2.3 Batch Cooling Crystallization

In this section we revisit the batch cooling crystallization process previously examined by Lang, Cervantes, and Biegler [53]. In this process, a solid-liquid separation occurs by cooling a jacketed stirred vessel. Here we need to determine an optimal cooling temperature profile. The driving force for growing crystals in the model is the *supercooling degree* $\Delta\theta$ defined by

$$\Delta\theta = \max\{0, T_{\text{equ}} - T\}, \tag{5.13}$$

where T is the temperature of the solution, and T_{equ} is the equilibrium temperature, whose dependency from the solution concentration C is expressed by a third order polynomial. Since (5.13) is a non-differentiable expression, it is approximated by a "smoothing function"

$$\Delta\theta = \frac{1}{2}\left(\Delta T + \sqrt{\Delta T^2 + \epsilon^2}\right) \tag{5.14}$$

with $\Delta T = T_{\text{equ}} - T$ and $\epsilon = 10^{-4}$. With this definition, the correlations between the growth rate (G), the mean size of the crystals (L_{s}), the nucleation rate (Nu), number of nuclei per liter of solvent (N), total length of the crystals per liter of solvent (L), total surface area of the crystals per liter of solvent (A), total volume of the crystals per liter of solvent (V_{c}), and the total mass of the crystals (M) are

expressed by the following system of ordinary differential equations:

$$G = \frac{\mathrm{d}L_s}{\mathrm{d}t} = K_\mathrm{g} \cdot L_\mathrm{s}^{0.5} \cdot (\Delta\theta)^\mathrm{en} \tag{5.15a}$$

$$\mathrm{Nu} = \frac{\mathrm{d}N}{\mathrm{d}t} = B_\mathrm{n} \cdot (\Delta\theta)^\mathrm{em} \tag{5.15b}$$

$$\frac{\mathrm{d}L}{\mathrm{d}t} = N \cdot G + \mathrm{Nu} \cdot L_\mathrm{n0} \tag{5.15c}$$

$$\frac{\mathrm{d}A}{\mathrm{d}t} = 2\alpha \cdot L \cdot G + \mathrm{Nu} \cdot L_\mathrm{n0}^2 \tag{5.15d}$$

$$\frac{\mathrm{d}V_\mathrm{c}}{\mathrm{d}t} = 3\beta \cdot A \cdot G + \mathrm{Nu} \cdot L_\mathrm{n0}^3 \tag{5.15e}$$

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \frac{3W_\mathrm{s0}}{L_\mathrm{s0}^3} L_\mathrm{s}^{2.5} \cdot G + \rho \cdot V \frac{\mathrm{d}V_\mathrm{c}}{\mathrm{d}t} \tag{5.15f}$$

$$\frac{\mathrm{d}C}{\mathrm{d}t} = -\frac{1}{V}\frac{\mathrm{d}M}{\mathrm{d}t}. \tag{5.15g}$$

Here, the volume of the solvent (V), the mass of seeds added (W_s0), the mean size of the seeds (L_s0), the specific gravity of the crystals (ρ), the shape factors α and β of the area and volume, as well as the empirical parameters en, em, K_g, and B_n are constants (see [53]).

When growing crystals, impurities are usually accumulated on the crystal surface. In order to improve product quality, one therefore strives to keep the total crystal surface area small, i.e. to grow the crystals as large as possible. Hence, the optimization problem consists in finding the optimal control profile for the temperature T_cool of the cooling water in the jacket of the crystallizer, so that the mean size of the crystals L_s after $t_f = 22$ h is maximized. The relationship between T_cool and the temperature T in the solution is characterized by the energy balance

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{K_\mathrm{c}\frac{\mathrm{d}M}{\mathrm{d}t} - K_\mathrm{E}\,(T - T_\mathrm{cool})}{W \cdot C_\mathrm{p}}, \tag{5.16}$$

where K_c is the specific melting latent heat of the product, W the total mass in the crystallizer, C_p the mean heat capacity, and K_E the heat-transfer coefficient (see [53]). Due to utility limitations, the temperature T_cool cannot go below 10 °C.

In order to prevent encrustation on the inside wall of the crystallizer, it is necessary to ensure that the cooling water temperature does not fall too much below

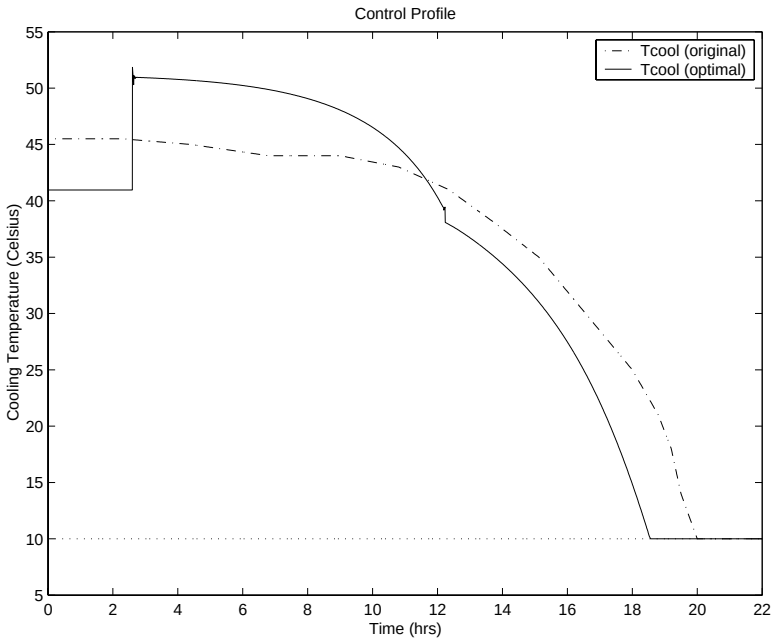

Figure 5.5: Control profiles for crystallization process

the temperature of the solution. For this, an allowable temperature T_{alw} is introduced, which depends on the concentration in the solution, expressed by a third order polynomial. An new algebraic (slack) variable y is then added to the problem,

$$y = T_{\text{cool}} - T_{\text{alw}}, \tag{5.17}$$

with a lower bound zero.

With this, the overall dynamic optimization problem consists of 8 differential state variables and 1 algebraic state variable, with the DAE system (5.15)–(5.17); Eq. (5.14) is directly substituted into (5.15). The model has been written in FORTRAN. First derivatives and possibly matrix-vector products with exact Hessians are obtained from ADOL-C [49].

As a starting point for the nonlinear optimization, the simulated profiles for the operating conditions originally applied in the plant are chosen, as presented in Figure 5.5. Here we also see the optimal temperature profile, using 500 finite elements and 3 collocation points. The control variables are restricted to be linear in each finite element. In [53], only 15 unequally spaced finite elements have been used,

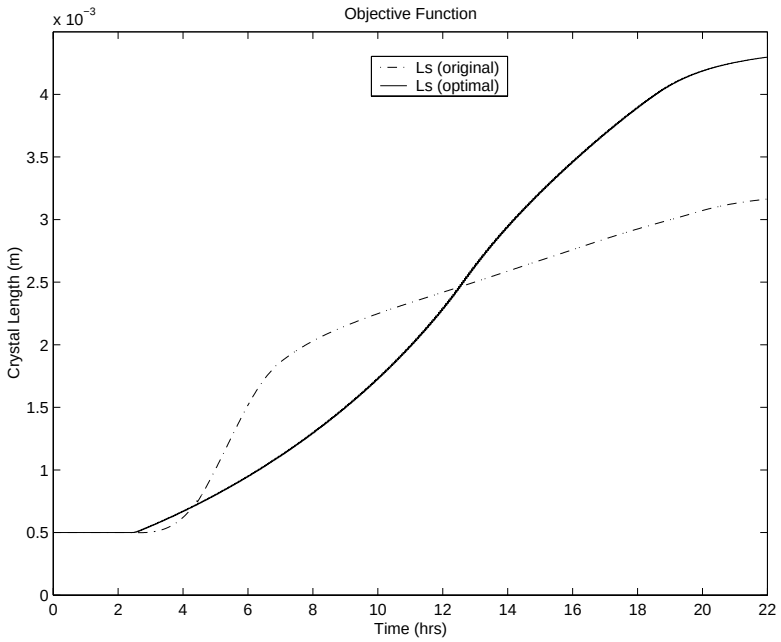

Figure 5.6: Crystal length for crystallization process

which required 130 iteration within rSQP [14]. The results obtained here confirm the finding in [53] and offer a very smooth representation of the optimal profiles.

In Figure 5.6 we see the profile of the mean crystal size L_s, both for the original and optimized operating conditions. The optimized control policy leads to a 50% increase of the final crystal size. Note, that the lower bound on the algebraic variables defined in (5.17) is active in several intervals of the time horizon, see Figure 5.7.

The model of the crystallization process is considerably smaller than the model of the distillation column discussed in the previous section. Also, in contrast to the problem statement (5.12), bounds are posed on *all* state and control variables for the optimization of the crystallizer. Some of those are active at the solution (as seen in Figures 5.5 and 5.7), and others are merely necessary to ensure that variables do not deviate to far from the physically meaningful region during the solution procedure, which could lead to convergence problems.

Therefore, the ratio of the numbers of degrees of freedom versus the total number of variables is much larger than in the previous example, and the reduced primal-

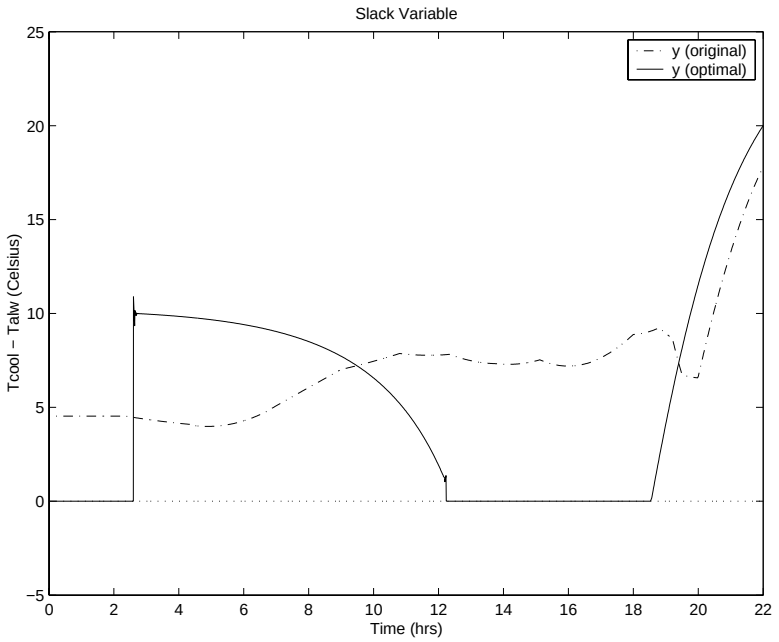

Figure 5.7: Active temperature constraint for crystallization process

dual Hessian of the barrier term, $Z_k^T \Sigma_k Z_k$, is not only a diagonal matrix but has to be computed explicitly. Furthermore, function and derivative evaluations are more time consuming (relative to the model size), since they are obtained by automatic differentiation in ADOL-C, which requires a repeated interpretation of the "tape" storing the description of the functions.

In order to see the performance of the individual options introduced in the previous section under these problem characteristics, the model was solved for different numbers of collocation points, as shown in Table 5.6. For options, for which products of the Hessian of the Lagrangian with vectors are required, either ADOL-C functions were used which perform multiple Hessian-vector products with exact second derivatives (Explicit, PCG1, PCG2), or the finite difference formula (3.12) was used employing only first derivatives provided by ADOL-C (fd-Explicit, fd-PCG1, fd-PCG2).

As in the comparison in Section 5.2.2, the SR1 updating formula seems to provide better estimates than BFGS, and both require iteration counts significantly increas-

No. of elements	Problem size n / m	QN-BFGS Iter / CPU	QN-SR1 Iter / CPU
50	1,908 / 1,808	478 / 70.40	339 / 48.78
100	3,808 / 3,608	833 / 617.75	555 / 404.80
220	8,368 / 7,928	1323/ 5728.65	1090/ 4755.10
500	19,008/18,008	1953/50360.78	1712/46258.57

No. of elements	Explicit Iter/ CPU	PCG1 Iter/ CPU [#cg]	PCG2 Iter/ CPU [#cg]
50	32 / 67.43	53 / 23.67 [955]	32 / 15.39 [398]
100	34 / 283.60	46 / 82.05 [1702]	37 / 69.01 [746]
220	35 /1435.03	45 / 418.90 [3806]	35 / 406.40 [1456]
500	42 /9063.01	47 /2574.31 [9180]	40 /2542.30 [2318]

No. of elements	fd-Explicit Iter/ CPU	fd-PCG1 Iter/ CPU [#cg]	fd-PCG2 Iter/ CPU [#cg]
50	32 / 33.39	65 / 15.04 [1097]	35 / 11.95 [429]
100	34 / 143.96	44 / 47.65 [1882]	35 / 49.46 [715]
220	35 / 742.84	46 / 269.49 [4282]	34 / 322.20 [1402]
500	53 /6010.54	45 /1848.38 [10470]	39 /2267.24 [2513]

Table 5.6: Iteration count and CPU time (in seconds) for crystallization example

ing with the problem size. However, for this example these quasi-Newton options are not at all competitive in terms for CPU time with the remaining Newton-type methods, for which the iteration counts increase only slightly. As before we see that the PCG methods lead to significant time savings compared to the Explicit option, where the preconditioner in PCG2 is again more effective than the one in PCG1 in the sense that overall less CG iterations are required.

Interestingly, when we compare the options using exact Hessian-vector products from ADOL-C with those using the finite difference formula (3.12), we see that the

approximations lead for the PCG options to an increase in CG iterations, due to the in-exactness of the products, but are faster in terms of computation time. This speed-up is particularly visible for the options Explicit and PCG1 which require many Hessian-vector products; for the first the time savings are about 50%, except for the largest instance in which the number of iterations is significantly increased when inexact products are used. At this point it is not clear why the usage of exact second derivatives from ADOL-C is not competitive in this example; it might be partly due to the fact that during the computation of second derivatives, the first derivatives are computed anyway. However, in other applications (e.g. [6]) Hessian-vector products approximated via finite differences were too inaccurate, leading to failures, and the ability of obtain exact products from ADOL-C proved to be essential.

Finally, when the costs for evaluating Hessian-vector products are not very high as for fd-PCG1 and fd-PCG2, we see that the first option actually becomes faster than the second one; PCG2 requires less CG iterations, but the matrix $Z_k^T \Sigma_k Z_k$ has to be constructed in every iteration and is relatively large in this example.

In conclusion, the results in the last two sections indicate that none of the options for handling or approximating second order information in the reduced space version of IPOPT is clearly superior to the others. On the contrary, the best choice depends on problem characteristics, such as size, relative number of degrees of freedom, number of bounds, costs for function and derivative evaluations, and of course the difficulty of the problem. This emphasizes the importance to be able to adapt optimization strategies and their implementation to individual applications.

Chapter 6

Conclusions

Nonlinear programming remains a very active research area, particularly since existing optimization methods and software codes reach their practical limits. In the field of process engineering, the emerging bottlenecks include limitations in problem size and missing flexibility of algorithms and implementations for the exploitation of problem dependent structures. The objective of this research project was the development and implementation of a new algorithm for nonlinear optimization, aiming to overcome those challenges.

The next section presents a summary of this work, and lists its contributions. Suggestions for future work are made in the final Section 6.2.

6.1 Thesis Summary and Contributions

After a short introduction to nonlinear programming in Chapter 2, a novel algorithm for large-scale nonlinear, nonconvex optimization has been proposed in Chapter 3. The challenges mentioned in Section 1.3 currently encountered for the solution of optimization problems arising in process engineering, have motivated its design in the following ways:

- The method follows a barrier approach, avoiding the potential bottleneck of active set identification in SQP methods.
- Search directions can be computed in different ways, allowing to choose the most appropriate option for individual purposes:
  - In its full-space version, the method is a powerful general purpose optimization solver, as demonstrated on a large number of test problems in Section 5.1. Interfaced to the modeling language AMPL, NLPs with up to 160,000 variables have been solved; these are not limited to a small number of degrees of freedom. Furthermore, the code compares favorably with two other recent interior point codes in terms of robustness, computation time, and iteration count on a large set of test problems (Section 5.1.3).
  - Alternatively, in the reduced space version a step decomposition based on variable partition allows the exploitation of constraint Jacobian structure. The effectiveness of this approach has been demonstrated in Section 5.2 on dynamic optimization examples, where the linear algebra is tailored to the problem structure by means of an improved implementation of the elemental decomposition.
  - The reduced space version of the proposed algorithm can also potentially be used to work with process simulators in a *tailored approach* [16]. Here, linear systems are solved within the simulator, and the proposed optimization algorithm can adapt to different levels of "openness" of the

simulation program, depending on whether systems involving the transpose of the constraint Jacobian can be solved or not. In the latter case, multiplier estimates are not available, but the algorithm can be applied in its multiplier-free form.

- Also with respect to second order information, several options are available:
  - In the full space version, the exact Hessian of the Lagrangian is used, exploiting its sparsity structure. Alternatively, one may use a sparse approximation, such as a Gauss-Newton approximation.
  - If second order information is not available or too expensive, the reduced space approach allows the approximation of the reduced Hessian by means of BFGS or SR1. In the comparisons presented in Sections 5.2.2 and 5.2.3, the latter update seems to provide faster performance.
  - If the number of degrees of freedom is large, quasi-Newton approximations may only provide slow convergence rates. In this case, Hessian-vector products can be used by the proposed algorithm within a preconditioned conjugate gradient (PCG) method. Here, two different preconditioners for the reduced barrier Hessian are proposed. If exact Hessian-vector products are not available, they can be approximated by finite differences. The effectiveness of the PCG approach has been demonstrated on a dynamic optimization problem with many finite elements for a process model with 426 DAEs, resulting in an NLP with more than 2 million variables and $4,500$ degrees of freedom. This was solved within 6.85 hours on a Linux workstation (Section 5.2.2).
  - The comparison of those different options on two different applications in Sections 5.2.2 and 5.2.2 indicate that the best choice depends on the particular problem characteristics. This emphasizes the importance of flexibility of an optimization algorithm.

- Global convergence is enforced by a line search approach, since it allows an easy decomposition of the steps. Here, several merit functions have been implemented, for the purpose of comparison. However, since these approaches can fail to guarantee global convergence on some well-posed problems (Section 3.3.3), a novel filter line search method (Section 3.4) has been devised and integrated into the code. A comparative study of the different line search options in Section 5.1.2 indicates that the new approach is overall the most robust and efficient one among the implemented strategies.

The theoretical contributions of this dissertation presented in Chapter 4 are:

- The discovery, analysis, and discussion of a new type of global convergence failure inherent to many current interior point methods for nonlinear programming (Section 4.1).
- First global convergence analysis for a line search filter method under very mild assumptions (Section 4.2). The results also hold for filter line search active set SQP methods.
- First local convergence analysis of a filter method, showing that second order correction steps indeed prevent the Maratos effect if the "switching condition" is chosen in a way different than previously proposed (Section 4.3).

6.2 Directions for Future Work

While the practical results of the proposed algorithm are very encouraging, there is always room for improvement. Some possible aspects worthy of further investigation are as follows:

Optimization Algorithm

- For convenience, the current implementation of the filter line search method uses TRON as the method performing the feasibility restoration phase, which

is a general purpose code for bound constrained optimization employing conjugate gradients. While it is performing well in most of the considered test problems, improvements regarding robustness are necessary, particularly since the restoration phase is the only step where the filter line search method can fail, and therefore inherits all the difficult cases. In addition, in some instances the conjugate gradient procedure within TRON seems very time-consuming, and an alternative direct approach, possibly based on a barrier method instead of gradient projection, might be more efficient. Furthermore, since the objective function is currently ignored during the restoration phase, it has been observed in some instances that the new iterates delivered from the restoration phase can lead to extremely bad objective function values, so that safeguards in this respect are necessary.

- The heuristic described in Section 3.2.1 for handling KKT matrices with wrong inertia in the full space approach seems to work efficiently in many cases. However, there are instances among the considered test problems, in which corrections are made in a large number of successive iterations, during which only small progress is made towards the solution. It might therefore be worthwhile to explore inertia-controlling symmetric indefinite factorizations and explicit exploitation of directions of negative curvature (see [39]).

- As a related topic, the current strategy for handling negative curvature within the preconditioned gradient method is not yet sufficiently robust and efficient. Here, a strategy similar to the one proposed in [46] for line search methods in the context of unconstrained optimization might be helpful.

- Although the preconditioners proposed in this dissertation for the conjugate gradient method work well in the considered examples, further investigation might lead to improved efficiency and robustness. In particular, even though the preconditioner PCG2 is superior to PCG1 with regard to CG iteration count, its construction can be prohibitive for very large problems, as indicated by the

long computation times of option PCG2* in Table 5.5 on page 147. On the other hand, the cheaper preconditioner PCG1 has been observed to impair robustness of the overall algorithm in other applications.

- In the full space method, degeneracy in the equality constraints is handled by introducing small pivot elements into the iteration matrix (see Eq. (3.7)). While this procedure is a practical heuristic at iterates away from a solution, it will not overcome impoverished local convergence to a solution of the barrier problem, if the constraint gradients are linearly dependent at that point. Similarly, degeneracy might also be introduced through bound constraints. In these cases, stabilization techniques as those proposed in [81] might be necessary to yield fast local convergence.

- In interior point algorithms for linear and quadratic programming, update schemes of the barrier parameter μ, that change its value in each iteration, have been proven to be very efficient. To follow such an approach in nonlinear and nonconvex cases, the mechanisms for global convergence have to be adapted. This also opens interesting theoretical questions.

Software Implementation

- IPOPT has been coded in FORTRAN 77, and will be available under an open source license. Since it is a relatively new piece of software, continued testing and improvements will be necessary. Also, continued usage is expected to reveal bottlenecks, either in the implementation or in the underlying mathematical algorithm, and might raise interesting research questions.

- As mentioned in Section 5.1.3, the choice of the linear solver used for the factorization of the KKT matrix impacts the computational speed and robustness of the full space version of IPOPT. Here, it might be worthwhile to compare different solvers, and to investigate how they can be tailored to the specific

task in IPOPT. For example, the heuristic for handling negative curvature described in Section 3.2.1 tries different values of the regularization parameter δ_1 in (3.6), until the inertia of the KKT matrix is correct. In the current implementation, each trial corresponds to a complete factorization of the KKT matrix. However, this search would be less time consuming if the factorization could be aborted prematurely, as soon as a pivot element with a wrong sign is detected (see also [39]).

- In the implementation of IPOPT, all operations with the constraints are requested through one single subroutine, in order to make it easy to tailor decomposition techniques etc. to particular applications. However, an even larger degree of flexibility could be obtained by a re-implementation in C++, employing object-oriented concepts. Using vector abstraction concepts proposed by Bartlett [5], this would also allow a very easy parallelization of the code.

Applications

- IPOPT and the new implementation of the elemental decomposition has already been integrated into DynoPC [52]. A process model can be provided as a FORTRAN 77 file, which is automatically preprocessed, so that derivatives such as the sparse Jacobian of the model equations and exact Hessian-vector products are accessible to the optimizer. The two examples presented in Chapter 5 are representations of many other problems that have been solved with this tool.

- Another promising new modeling environment for dynamic optimization and nonlinear model predictive control (NMPC) is OCOMA [7]. Using this tool, it is planned to investigate the potential of IPOPT as optimization code for NMPC, in a comparison with rSQP++ [5].

- In the implementation of the elemental decomposition it is currently not possible to impose additional constraints, that couple the entire system. Here, a two-stage decomposition strategy might provide the answer.

- Finally, since IPOPT has been designed with the intention to make it possible to tailor it to different engineering applications, it would be worthwhile exploring its potential in fields like PDE-constrained optimization and circuit tuning.

Bibliography

[1] P. Armand, J. Ch. Gilbert, and S. Jan-Jégou. A feasible BFGS interior point algorithm for solving strongly convex minimization problems. *SIAM Journal on Optimization*, 11(1):199–222, 2000.

[2] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, PA, USA, 1998.

[3] C. Audet and J. E. Dennis. A pattern search filter method for nonlinear programming without derivatives. Technical Report TR00-09, Department of Computational and Applied Mathematics, Rice University, Houston, TX, USA, March 2000.

[4] G. Bader and U. Ascher. A new basis implementation for mixed order boundary value ODE solver. *SIAM Journal on Scientific Computing*, 8:483–500, 1987.

[5] R. A. Bartlett. *Object-Oriented Approaches to Large-Scale Nonlinear Programming for Process Systems Engineering*. PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, August 2001.

[6] R. A. Bartlett, A. Wächter, and L. T. Biegler. Active set vs. interior point methods for nonlinear model predictive control. In *Proceedings of American Control Conference*, page 3505, Chicago, IL, USA, 2000.

[7] J. Bausa and G. Tsatsaronis. Dynamic optimization of start-up and load-increasing processes in power plants – methodology and application. In *Proceedings of the ASME Advanced Energy Systems Division*, volume 38, pages 165–173, 1998.

[8] H. Y. Benson, D. F. Shanno, and R. J. Vanderbei. Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions. Technical report, Operations Research and Financial Engineering, Princeton University, Princeton, NJ, USA, December 2000.

[9] H. Y. Benson, D. F. Shanno, and R. J. Vanderbei. Interior-point methods for nonconvex nonlinear programming: Jamming and comparative numerical testing. Technical report, Operations Research and Financial Engineering, Princeton University, Princeton, NJ, USA, August 2000.

[10] J. T. Betts and P. D. Frank. A sparse nonlinear optimization algorithm. *Journal of Optimization Theory and Application*, 82(3):519–541, 1994.

[11] L. T. Biegler. Convergence analysis for a multiplier-free reduced Hessian method. Technical Report 06–203–95, EDRC, Carnegie Mellon University, Pittsburgh, PA, USA, 1995. Revised 1999.

[12] L. T. Biegler and J. E. Cuthrell. Improved infeasible path optimization for sequential modular simulators – II: The optimization algorithm. *Computers and Chemical Engineering*, 9(3):257–267, 1985.

[13] L. T. Biegler, J. Nocedal, and C. Schmid. A reduced Hessian method for large-scale constrained optimization. *SIAM Journal on Optimization*, 5(2):314–347, 1995.

[14] L. T. Biegler, J. Nocedal, C. Schmid, and D. Ternet. Numerical experience with a reduced Hessian method for large-scale constrained optimization. *Computational Optimization and Applications*, 15(1):45–67, 2000.

[15] L. T. Biegler, C. Schmid, and D. Ternet. A multiplier-free, reduced Hessian method for process optimization. In L. T. Biegler, T. F. Coleman, A. R. Conn, and F. N. Santosa, editors, *Large-Scale Optimization with Applications, Part II: Optimal Design and Control*, IMA Volumes in Mathematics and Applications, page 101. Springer Verlag, 1997.

[16] L. T. Biegler and A. Wächter. SQP SAND strategies that link to existing modeling systems. Presentation at the First CSRI Workshop on PDE-based Optimization, Santa Fe, NM, USA, March 2001.

[17] P. T. Boggs, A. J. Kearsley, and J. W. Tolle. A practical algorithm for general large scale nonlinear optimization problems. *SIAM Journal on Optimization*, 9(3):755–778, 1999.

[18] P. T. Boggs, J. W. Tolle, and P. Wang. On the local convergence of quasi-Newton methods for constrained optimization. *SIAM Journal on Control and Optimization*, 20:161–171, 1982.

[19] A. S. Bondarenko, D. M. Bortz, and J. J. Moré. COPS: Large-scale nonlinearly constrained optimization problems. Technical Report ANL/MCS-TM-237, Argonne National Laboratory, Argonne, IL, USA, 1998. revised October 1999.

[20] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[21] R. H. Byrd, J. Ch. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89:149–185, 2000.

[22] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[23] R. H. Byrd, G. Liu, and J. Nocedal. On the local behavior of an interior point method for nonlinear programming. In D. F. Griffiths and D. J. Higham, editors, *Numerical Analysis 1997*. Addison–Wesley Longman, Reading, MA, USA, 1997.

[24] R. H. Byrd, J. Nocedal, and R. A. Waltz. Feasible interior methods using slacks for nonlinear optimization. Technical Report OTC 2000/11, Optimization Technology Center, Northwestern University, Evanston, IL, USA, December 2000. (submitted to Mathematical Programming).

[25] A. M. Cervantes. *Stable large-scale DAE optimization using simultaneous approaches*. PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA ,USA, 2000.

[26] A. M. Cervantes, A. Wächter, R. H. Tütüncü, and L. T. Biegler. A reduced space interior point strategy for optimization of differential algebraic systems. *Computers and Chemical Engineering*, 24(1):39–51, 2000.

[27] R. M. Chamberlain, C. Lemarechal, H. C. Pedersen, and M. J. D. Powell. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Mathematical Programming Study*, 16:1–17, 1982.

[28] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. SIAM, Philadelphia, PA, USA, 2000.

[29] J. E. Dennis and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19(1):46–89, 1977.

[30] E. D. Dolan and J. J. Moré. Benchmarking optimization software with COPS. Technical Report ANL/MCS-246, Argonne National Laboratory, Argonne, IL, November 2000.

[31] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. Technical Report ANL/MCS-P861-1200, Argonne National Laboratory, Argonne, IL, November 2001. Updated March 2001.

[32] I. S. Duff and J. K. Reid. MA27: A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Technical report, Computer Science and Systems Division, AERE Harwell, Oxford, England, 1982.

[33] A. S. El-Bakry, R. A. Tapia, T. Tsuchiya, and Y. Zhang. On the formulation and theory of the Newton interior-point method for nonlinear programming. *Journal of Optimization Theory and Application*, 89(3):507–541, 1996.

[34] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley, New York, USA, 1968. Reprinted by SIAM Publications, 1990.

[35] R. Fletcher, N. I. M. Gould, S. Leyffer, Ph. L. Toint, and A. Wächter. Global convergence of a trust-region SQP-filter algorithms for general nonlinear programming. Technical Report 99/03, Department of Mathematics, University of Namur, Belgium, May 1999. Revised October 2001.

[36] R. Fletcher and S. Leyffer. A bundle filter method for nonsmooth nonlinear optimization. Technical Report NA/195, Department of Mathematics, University of Dundee, Scotland, December 1999.

[37] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 2002. To appear.

[38] R. Fletcher, S. Leyffer, and Ph. L. Toint. On the global convergence of a filter-SQP algorithm. Technical Report NA/197, Department of Mathematics, University of Dundee, Scotland, October 2000.

[39] A. Forsgren and P. E. Gill. Primal-dual interior methods for nonconvex nonlinear programming. *SIAM Journal on Optimization*, 8(4):1132–1152, 1998.

[40] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language For Mathematical Programming.* Thomson Publishing Company, Danvers, MA, USA, 1993.

[41] D. M. Gay. AMPL solver library. See `http://www.netlib.org/ampl`.

[42] D. M. Gay, M. L. Overton, and M. H. Wright. A primal-dual interior method for nonconvex nonlinear programming. In Y. Yuan, editor, *Advances in Nonlinear Programming*, pages 31–56. Kluwer Academic Publishers, Dordretch, 1998.

[43] P. E. Gill, W. Murray, and M. A. Saunders. User's guide for SQOPT 5.3: A Fortran package for large-scale linear and quadratic programming. Numerical Analysis Report 97-4, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1997.

[44] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 2002. To appear.

[45] C. C. Gonzaga, E. Karas, and M. Vanti. A globally convergent filter method for nonlinear programming. Technical report, Department of Mathematics, Federal University of Santa Catarina, Florianópolis, SC, Brazil, November 2001.

[46] N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Exploiting negative curvature directions in linesearch methods for unconstrained optimization. Technical Report 97/18, Department of Mathematics, FUNDP, Namur, Belgium, 1997.

[47] N. I. M. Gould, D. Orban, A. Sartenaer, and P. L. Toint. Componentwise fast convergence in the solution of full-rank systems of nonlinear equations. Technical Report TR/PA/00/56, CERFACS, Toulouse, France, 1999. Revised May 2001.

[48] N. I. M. Gould, D. Orban, A. Sartenaer, and P. L. Toint. Superlinear convergence of primal-dual interior point algorithms for nonlinear programming. *SIAM Journal on Optimization*, 11(4):974–1002, 2001.

[49] A. Griewank, D. Juedes, and J. Utke. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, 1996.

[50] S.-P. Han. A globally convergent method for nonlinear programming. *Journal of Optimization Theory and Application*, 22:297–309, 1977.

[51] F. R. De Hoog and R. M. M. Mattheij. On dichotomy and well conditioning in BVP. *SIAM Journal on Numerical Analysis*, 24(1):89–105, 1987.

[52] Y.-D. Lang, A. M. Cervantes, and L. T. Biegler. DynoPC: A dynamic optimization tool for process engineering. Presentation at INFORMS National Meeting, Seattle, WA, USA, October 1998.

[53] Y.-D. Lang, A. M. Cervantes, and L. T. Biegler. Dynamic optimization of a batch cooling crystallization process. *Indutrial and Engineering Chemistry Research*, 38(4):1469–1477, 1999.

[54] C.-J. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[55] L. Lukšan and J. Vlček. Sparse and paritally separable test problems for unconstrained and equality constrained optimization. Technical Report 767, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic, January 1999.

[56] N. Maratos. *Exact Penalty Function Algorithms for Finite Dimensional and Control Optimization Problems.* PhD thesis, University of London, London, UK, 1978.

[57] M. Marazzi and J. Nocedal. Feasibility control in nonlinear optimization. Technical Report OTC 2000/04, Optimization Technology Center, March 2000. To appear in Foundations of Computational Mathematics, Cambridge University Press.

[58] Frode Martinsen. *The Optimization Algorithm rFSQP with Application to Nonlinear Model Predictive Control of Grate Sintering.* PhD thesis, Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway, August 2001.

[59] H. Maurer and H. D. Mittelmann. Optimization techniques for solving elliptic control problems with control and state constraints. Part 2: Distributed control. *Computational Optimization and Applications*, 18:141–160, 2001.

[60] H. D. Mittelmann. AMPL models. See `ftp://plato.la.asu.edu/pub/ampl_files/`.

[61] H. D. Mittelmann. Sufficient optimality for discretized parabolic and elliptic control problems. In K.-H. Hoffmann, R. H. W. Hoppe, and V. Schulz, editors, *Fast solution of discretized optimization problems.* Birkhaeuser, Basel, 2001.

[62] H. D. Mittelmann and F. Troeltzsch. Sufficient optimality in a parabolic control problem. In P. Manchanda, A. H. Siddiqi, and M. Kocvara, editors, *Proceedings of the first International Conference on Industrial and Applied Mathematics in Indian Subcontinent*, Dordrecht, The Netherlands, 2001. Kluwer.

[63] J. L. Morales and J.Nocedal. Automatic preconditioning by limited memory quasi-Newton updating. *SIAM Journal on Optimization*, 10(4):1079–1096, 2000.

[64] W. Murray and F. J. Prieto. A sequential quadratic programming algorithm using an incomplete solution of the subproblem. *SIAM Journal on Optimization*, 5(3):590–640, 1995.

[65] S. G. Nash and A. Sofer. A barrier method for large-scale constrained optimization. *ORSA Journal on Computing*, 5(1):40–53, 1993.

[66] S. G. Nash and A. Sofer. Why extrapolation helps barrier methods. Technical report, Operations Research and Engineering Department, George Mason University, Fairfax, A 22030, USA, September 1998.

[67] J. Nocedal. Personal communication.

[68] J. Nocedal and M. Overton. Projected Hessian updating algorithms for nonlinearly constrained optimization. *SIAM Journal on Numerical Analysis*, 22:821–850, 1985.

[69] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 1999.

[70] E. Omojokun. *Trust Region Algorithms for Optimization with Nonlinear Equality and Inequality Constraints*. PhD thesis, University of Colorado, Boulder, CO, USA, 1989.

[71] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–114. Gordon and Breach Science Publishers, London, New York, Paris, 1970.

[72] Process Systems Enterprise Limited. See `http://www.psenterprise.com`.

[73] K. Schittkowski. The nonlinear programming method of Wilson, Han and Powell with an augmented Lagrangian type line search function. *Numerische Mathematik*, 38, 1981.

[74] C. Schmid and L. T. Biegler. Quadratic programming methods for reduced Hessian SQP. *Computers and Chemical Engineering*, 18(9), 1994.

[75] D. F. Shanno and R. J. Vanderbei. Interior-point methods for nonconvex nonlinear programming: Orderings and higher-order methods. Technical report, Operations Research and Financial Engineering, Princeton University, Princeton, NJ, USA, 1999.

[76] D. J. Ternet and L. T. Bielger. Interior-point methods for reduced Hessian successive quadratic programming. *Computers and Chemical Engineering*, 23:859–873, 1999.

[77] A.L. Tits, T.J. Urban, S. Bakhtiari, and C.T. Lawrence. A primal-dual interior-point method for nonlinear programming with strong global and local convergence properties. Technical Report TR 2001-3 R2, Department of Electrical and Computer Engineering and Institute for Systems Research, Univerity of Maryland, College Park, MD, July 2001.

[78] M. Ulbrich, S. Ulbrich, and L. N. Vicente. A globally convergent primal-dual interior-point filter method for nonconvex nonlinear programming. Technical Report TR00-12, Department of Computational and Applied Mathematics, Rice University, Houston, TX, USA, 2000.

[79] R. J. Vanderbei. Nonlinear optimization models. See `http://www.sor.princeton.edu/~rvdb/ampl/nlmodels/`.

[80] R. J. Vanderbei and D. F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–252, 1999.

[81] L. Vicente and S. J. Wright. Local convergence of a primal-dual method for degenerate nonlinear programming. Technical report, May 2001. To appear in Computational Optimization and Applications.

[82] C. Villalobos, R. Tapia, and Y. Zhang. The sphere of convergence of Newton's method on two equivalent systems from nonlinear programming. Technical Report TR99–15, Department of Computational and Applied Mathematics, Rice University, Houston, TX, USA, 1999.

[83] A. Wächter and L. T. Biegler. Failure of global convergence for a class of interior point methods for nonlinear programming. *Mathematical Programming*, 88(2):565–574, 2000.

[84] A. Wächter and L. T. Biegler. Global and local convergence of a reduced space quasi-Newton barrier algorithm for large-scale nonlinear programming. Tech-

nical Report B-00-06, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, 2000.

[85] M. H. Wright. Why a pure primal Newton barrier step may be infeasible. *SIAM Journal on Optimization*, 5(1):1–12, 1995.

[86] S. Wright. *Primal-dual interior-point methods*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[87] H. Yabe and H. Yamashita. Q-superlinear convergence of primal-dual interior point quasi-Newton methods for constrained optimization. Technical report, Mathematical Systems Inc., Tokio, Japan, February 1995. Revised August 1995.

[88] H. Yamashita. A globally convergent primal-dual interior-point method for constrained optimization. *Optimization Methods and Software*, 10:443–469, 1998.

[89] H. Yamashita, H. Yabe, and T. Tanabe. A globally and superlinearly convergent primal-dual interior point trust region method for large scale constrained optimization. Technical report, Mathematical System Institute, Inc., Tokyo, Japan, July 1997. Revised July 1998.

[90] E. A. Yildirim and S. J. Wright. Warm-start strategies in interior-point methods for linear programming. Technical Report ANL/MCS-P799-0300, Argonne National Laboratory, Argonne, IL, USA, March 2000. To appear in SIAM Journal on Optimization.

Appendix A

Characteristics of NLP Test Problems

The tables in this appendix list the sizes of the individual problems in the CUTE, COPS, and MITT test sets. For each problem, we show the number of variables (#var; not including slack variables for inequality and range constraints (5.1b)) and how many of those have bounds (#bdd); the number of equality (#eq) and inequality or range constraints (#ineq), as well as the number of non-zero elements in constraint Jacobian (#nzJac), and number of non-zero elements in Hessian of Lagrangian (#nzHes), counting symmetric off-diagonal elements only once.

For COPS and MITT, these are the characteristics of the problem formulations after AMPL's pre-solve phase.

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
airport	84	84	0	42	84	1806
aljazzaf	3	3	1	0	3	3
allinit	4	0	1	2	3	10
allinitc	4	0	1	3	5	10
allinitu	4	0	0	0	0	10
alsotame	2	0	1	2	4	3
arwhead	5000	0	0	0	0	9999
avion2	49	49	15	0	43	123
bard	3	0	0	0	0	6
batch	46	46	12	61	178	33
bdexp	5000	0	0	0	0	14997
bdqrtic	1000	0	0	0	0	4990
beale	2	0	0	0	0	3
bigbank	2230	1674	1112	0	4460	1366
biggs3	6	0	3	0	3	21
biggs5	6	0	1	0	1	21
biggs6	6	0	0	0	0	21
box2	3	0	1	0	1	6
box3	3	0	0	0	0	6
brainpc0	6905	6905	6900	0	41395	10478
brainpc1	6905	6905	6900	0	41395	10478
brainpc2	13805	13805	13800	0	82795	20828
brainpc3	6905	6905	6900	0	41395	10478
brainpc4	6905	6905	6900	0	41395	10478
brainpc5	6905	6905	6900	0	41395	10478
brainpc6	6905	6905	6900	0	41395	10478
brainpc7	6905	6905	6900	0	41395	10478
brainpc8	6905	6905	6900	0	41395	10478
brainpc9	6905	6905	6900	0	41395	10478
bratu1d	1001	0	0	0	0	2001
britgas	450	450	360	0	1576	735
brkmcc	2	0	0	0	0	3
brownal	10	0	0	0	0	55
brownbs	2	0	0	0	0	3
broydn7d	1000	0	0	0	0	3497
brybnd	5000	0	0	0	0	34979
bt1	2	0	1	0	2	2
bt2	3	0	1	0	3	5
bt4	3	0	2	0	6	3
bt5	3	0	2	0	6	5
bt6	5	0	2	0	5	9
bt7	5	0	3	0	8	6
bt8	5	0	2	0	6	5
bt9	4	0	2	0	6	3
bt11	5	0	3	0	8	9
bt12	5	0	3	0	8	5
bt13	5	0	1	1	6	8
byrdsphr	3	0	2	0	6	3
camel6	2	0	0	2	2	3
cantilvr	5	5	0	1	5	5
catena	32	0	11	0	62	61
catenary	496	0	166	0	826	989
cb2	3	0	0	3	9	3
cb3	3	0	0	3	9	3
chaconn1	3	0	0	3	9	3
chaconn2	3	0	0	3	9	3
chebyqad	50	50	0	0	0	1275
chnrosnb	50	0	0	0	0	99
cliff	2	0	0	0	0	3
clnlbeam	1499	998	1000	0	3994	1000
clplatea	4970	0	0	0	0	14700
clplateb	4970	0	0	0	0	14700
clplatec	4970	0	0	0	0	14700

Table A.1: Characteristics of CUTE test set (*continued on next page*)

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
concon	15	5	11	0	26	11
congigmz	3	0	0	5	13	2
core1	65	0	41	74	212	24
corkscrw	8997	4000	6000	1000	20988	5000
coshfun	61	0	0	20	118	80
cosine	10000	0	0	0	0	19999
cragglvy	5000	0	0	0	0	9999
cresc100	6	4	0	200	1100	17
cresc132	6	4	0	2654	14597	17
cresc4	6	4	0	8	44	17
cresc50	6	4	0	100	550	17
csfi1	5	5	2	2	11	7
csfi2	5	5	2	2	11	7
cube	2	0	0	0	0	3
curly10	10000	0	0	0	0	109945
curly20	10000	0	0	0	0	209790
curly30	10000	0	0	0	0	309535
dallasl	906	906	667	0	1812	888
dallasm	196	196	151	0	392	190
dallass	46	46	31	0	92	41
deconvc	51	51	1	0	11	891
demymalo	3	0	0	3	9	2
denschna	2	0	0	0	0	3
denschnb	2	0	0	0	0	3
denschnc	2	0	0	0	0	3
denschnd	3	0	0	0	0	6
denschne	3	0	0	0	0	3
denschnf	2	0	0	0	0	3
dipigri	7	0	0	4	19	9
disc2	28	6	17	6	81	43
discs	36	12	21	48	399	234
dittert	327	327	264	0	2431	1088
dixchlng	10	0	5	0	30	55
dixchlnv	100	100	50	0	2550	296
dixmaana	3000	0	0	0	0	6000
dixmaanb	3000	0	0	0	0	8999
dixmaanc	3000	0	0	0	0	8999
dixmaand	3000	0	0	0	0	8999
dixmaane	3000	0	0	0	0	6000
dixmaanf	3000	0	0	0	0	8999
dixmaang	3000	0	0	0	0	8999
dixmaanh	3000	0	0	0	0	8999
dixmaani	3000	0	0	0	0	6000
dixmaanj	3000	0	0	0	0	8999
dixmaank	3000	0	0	0	0	8999
dixmaanl	3000	0	0	0	0	8999
djtl	2	0	0	0	0	3
dnieper	61	60	24	0	128	79
dqrtic	5000	0	0	0	0	5000
drcavty1	10816	0	816	0	816	216802
drcavty2	10816	0	816	0	816	216802
drcavty3	10816	0	816	0	816	216802
dtoc1l	14985	0	9990	0	82889	14985
dtoc1na	1485	0	990	0	15740	6385
dtoc1nb	1485	0	990	0	15740	6385
dtoc1nc	1485	0	990	0	15740	6385
dtoc1nd	735	0	490	0	7740	3135
dtoc2	5994	0	3996	0	15980	14977
dtoc4	14997	0	9998	0	34989	19995
dtoc5	9998	0	4999	0	14996	9997
dtoc6	10000	0	5000	0	14999	14998
edensch	2000	0	0	0	0	3999
eg1	3	2	0	0	0	6

Table A.1: Characteristics of CUTE test set (*continued on next page*)

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
eg2	1000	0	0	0	0	1998
eg3	101	100	1	199	595	301
eigena	110	110	0	0	0	6105
eigena2	110	0	55	0	1000	660
eigenaco	110	0	55	0	1000	6105
eigenals	110	0	0	0	0	6105
eigenb	110	0	0	0	0	6105
eigenb2	110	0	55	0	1000	660
eigenbco	110	0	55	0	1000	6105
eigenbls	110	0	0	0	0	6105
eigenc2	462	0	231	0	9261	5313
eigencco	30	0	15	0	125	465
engval1	5000	0	0	0	0	9999
engval2	3	0	0	0	0	6
errinros	50	0	0	0	0	99
expfit	2	0	0	0	0	3
expfita	5	0	0	22	71	15
expfitb	5	0	0	102	351	15
expfitc	5	0	0	502	1751	15
explin	120	120	0	0	0	21
explin2	120	120	0	0	0	21
expquad	120	0	0	10	10	239
extrosnb	10	0	0	0	0	19
fletcbv2	100	0	0	0	0	199
fletcbv3	10000	0	0	0	0	19999
fletchbv	10000	0	0	0	0	19999
fletchcr	100	0	0	0	0	199
fletcher	4	0	1	4	11	10
flosp2hh	691	41	0	0	0	10011
flosp2hl	691	41	0	0	0	10011
flosp2hm	691	41	0	0	0	10011
flosp2th	691	41	0	0	0	10009
flosp2tl	691	41	0	0	0	10009
flosp2tm	691	41	0	0	0	10009
fminsrf2	1024	0	0	0	0	4930
fminsurf	1024	0	0	0	0	524800
freuroth	5000	0	0	0	0	9999
gausselm	1495	0	1240	2722	11145	1704
genhumps	5	0	0	0	0	9
genrose	500	0	0	0	0	999
gigomez1	3	0	0	3	9	2
gilbert	1000	0	1	0	1000	1000
gpp	250	0	0	498	996	31375
growth	3	0	0	0	0	6
growthls	3	0	0	0	0	6
gulf	3	0	0	0	0	6
hadamals	100	100	0	0	0	5050
hadamard	65	1	64	192	1280	289
hager2	10000	0	5000	0	14999	14999
hager4	10000	5000	5000	0	14999	14999
haifam	85	0	0	150	711	437
haifas	7	0	0	9	31	11
hairy	2	0	0	0	0	3
haldmads	6	0	0	42	244	15
hanging	300	12	0	180	1080	840
hart6	6	6	0	0	0	21
hatflda	4	4	0	0	0	7
hatfldb	4	4	0	1	1	7
hatfldc	4	0	0	3	3	6
hatfldd	3	0	0	0	0	6
hatfldе	3	0	0	0	0	6
heart6ls	6	0	0	0	0	21
heart8ls	8	0	0	0	0	36

Table A.1: Characteristics of CUTE test set (*continued on next page*)

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
helix	3	0	0	0	0	6
himmelbb	2	0	0	0	0	3
himmelbf	4	0	0	0	0	10
himmelbg	2	0	0	0	0	3
himmelbh	2	0	0	0	0	2
himmelbi	100	100	0	12	135	143
himmelbj	45	45	16	0	88	290
himmelbk	24	24	14	0	336	144
himmelp1	2	0	0	2	2	3
himmelp2	2	0	0	3	4	3
himmelp3	2	0	0	4	6	3
himmelp4	2	0	0	5	8	3
himmelp5	2	0	0	5	8	3
himmelp6	2	2	0	5	10	3
hong	4	4	1	0	4	4
hs001	2	0	0	1	1	3
hs002	2	0	0	1	1	3
hs004	2	0	0	2	2	1
hs005	2	0	0	2	2	3
hs006	2	0	1	0	2	1
hs007	2	0	1	0	2	2
hs009	2	0	1	0	2	3
hs010	2	0	0	1	2	3
hs011	2	0	0	1	2	2
hs012	2	0	0	1	2	3
hs013	2	2	0	1	2	2
hs014	2	0	1	1	4	2
hs015	2	0	0	3	5	3
hs016	2	0	0	4	6	3
hs017	2	0	0	4	6	3
hs018	2	0	0	4	6	3
hs019	2	0	0	4	6	2
hs020	2	0	0	4	7	3
hs023	2	2	0	5	10	2
hs024	2	2	0	3	6	3
hs025	3	0	0	3	3	6
hs026	3	0	1	0	3	5
hs027	3	0	1	0	2	4
hs029	3	0	0	1	3	6
hs030	3	0	0	4	5	3
hs031	3	0	0	4	5	4
hs032	3	3	1	1	6	6
hs033	3	3	0	3	7	3
hs034	3	3	0	5	7	2
hs036	3	3	0	4	6	3
hs037	3	3	0	2	6	3
hs038	4	4	0	0	0	7
hs039	4	0	2	0	6	3
hs040	4	0	3	0	7	9
hs041	4	4	1	4	8	3
hs042	4	4	2	0	3	4
hs043	4	0	0	3	12	4
hs045	5	5	0	0	0	10
hs046	5	0	2	0	6	9
hs047	5	0	3	0	8	10
hs049	5	0	2	0	6	6
hs050	5	0	3	0	9	9
hs054	6	6	1	0	2	7
hs056	7	7	4	0	10	7
hs057	2	0	0	3	4	3
hs059	2	2	0	3	6	3
hs060	3	3	1	0	3	5
hs061	3	0	2	0	4	3

Table A.1: Characteristics of CUTE test set (*continued on next page*)

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
hs062	3	3	1	0	3	6
hs063	3	3	2	0	6	5
hs064	3	3	0	1	3	3
hs065	3	0	0	4	6	4
hs066	3	0	0	5	7	2
hs067	10	3	7	14	35	15
hs070	4	4	0	1	2	10
hs071	4	4	1	1	8	10
hs072	4	4	0	6	12	4
hs073	4	4	1	2	12	10
hs074	4	4	3	1	10	5
hs075	4	4	3	1	10	5
hs077	5	0	2	0	6	9
hs078	5	0	3	0	11	15
hs079	5	0	3	0	8	10
hs080	5	5	3	0	11	15
hs081	5	5	3	0	11	15
hs083	5	5	0	3	13	8
hs084	5	5	0	3	15	4
hs085	5	0	0	48	129	15
hs086	5	5	0	10	37	15
hs087	11	11	6	0	22	6
hs088	2	0	0	1	2	3
hs089	3	0	0	1	3	6
hs090	4	0	0	1	4	10
hs091	5	0	0	1	5	15
hs092	6	0	0	1	6	21
hs093	6	6	0	2	12	19
hs095	6	6	0	4	20	6
hs096	6	6	0	4	20	6
hs097	6	6	0	4	20	6
hs098	6	6	0	4	20	6
hs099	23	7	18	0	53	28
hs100	7	0	0	4	19	9
hs100lnp	7	0	2	0	10	9
hs100mod	7	0	0	4	21	9
hs101	7	7	0	6	38	28
hs102	7	7	0	6	38	28
hs103	7	7	0	6	38	28
hs104	8	8	0	6	21	16
hs105	8	0	0	9	10	36
hs106	8	0	0	14	25	5
hs107	9	0	6	8	42	17
hs108	9	0	0	14	40	25
hs109	9	9	6	4	42	19
hs110	10	10	0	0	0	55
hs111	10	10	3	0	14	55
hs111lnp	10	0	3	0	14	55
hs112	10	10	3	0	14	55
hs113	10	0	0	8	32	11
hs114	10	10	3	8	31	15
hs116	13	13	0	28	59	15
hs117	15	15	0	5	62	15
hs119	16	16	8	0	53	47
hs99exp	31	10	21	0	70	8
hubfit	2	1	0	1	2	3
humps	2	0	0	0	0	3
hvycrash	202	102	150	0	600	300
hypcir	2	0	2	0	4	3
indef	1000	0	0	0	0	2997
jensmp	2	0	0	0	0	3
kissing	127	0	42	861	6153	2709
kiwcresc	3	0	0	2	6	2

Table A.1: Characteristics of CUTE test set (*continued on next page*)

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
kowosb	4	0	0	0	0	10
lakes	90	18	78	0	240	102
launch	25	25	9	20	127	78
lch	600	0	1	0	400	2000
liarwhd	10000	0	0	0	0	19999
lminsurf	15625	0	496	0	496	77377
loadbal	31	31	11	20	91	41
loghairy	2	0	0	0	0	3
logros	2	2	0	0	0	3
lootsma	3	3	0	3	7	3
lsnnodoc	5	3	4	0	10	9
madsen	3	0	0	6	14	3
madsschj	81	0	0	158	12797	80
makela1	3	0	0	2	6	2
makela2	3	0	0	3	9	2
makela3	21	0	0	20	40	20
mancino	100	0	0	0	0	100
manne	1094	1094	0	730	2187	729
maratos	2	0	1	0	2	2
matrix2	6	4	0	2	6	11
maxlika	8	0	0	8	8	36
mccormck	50000	50000	0	0	0	99999
mdhole	2	1	0	0	0	3
methanb8	31	0	0	0	0	256
methanl8	31	0	0	0	0	256
mexhat	2	0	0	0	0	3
meyer3	3	0	0	0	0	6
mifflin1	3	0	0	2	5	2
mifflin2	3	0	0	2	6	2
minc44	311	311	262	0	2399	1016
minmaxbd	5	0	0	20	100	6
minmaxrb	3	0	0	4	10	1
minperm	1113	1113	1033	0	11433	5110
minsurf	64	0	32	0	32	274
mistake	9	1	0	13	39	25
morebv	5002	0	2	0	2	15003
msqrtals	1024	0	0	0	0	524800
msqrtbls	1024	0	0	0	0	524800
mwright	5	0	3	0	8	10
ngone	100	100	0	1273	4996	2698
noncvxu2	1000	0	0	0	0	3991
noncvxun	1000	0	0	0	0	1000
nondia	9999	0	0	0	0	19997
nondquar	10000	0	0	0	0	29997
nonmsqrt	9	0	0	0	0	18
nonscomp	10000	10000	0	0	0	19999
odfits	10	10	6	0	15	10
oet2	3	0	0	1002	3004	2
oet7	7	0	0	1002	7008	6
optcdeg2	1199	799	800	0	2795	799
optcdeg3	1199	799	800	0	2795	799
optcntrl	32	23	20	1	81	21
optctrl3	122	3	80	1	360	120
optctrl6	122	3	80	1	360	120
optmass	66	0	44	11	170	26
optprloc	30	30	0	30	170	5
orthrdm2	4003	0	2000	0	10000	18006
orthrds2	203	0	100	0	500	906
orthrega	517	0	256	0	1792	2304
orthregb	27	0	6	0	72	108
orthregc	10005	0	5000	0	35000	45000
orthregd	10003	0	5000	0	25000	45006
orthrege	36	1	20	0	120	77

Table A.1: Characteristics of CUTE test set (*continued on next page*)

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
orthrgdm	10003	0	5000	0	25000	45006
orthrgds	10003	0	5000	0	25000	45006
osbornea	5	0	0	0	0	15
osborneb	11	0	0	0	0	66
oslbqp	8	0	0	8	8	8
palmer1	4	3	0	0	0	10
palmer1a	6	2	0	0	0	21
palmer1b	4	2	0	0	0	10
palmer1e	8	0	0	0	0	36
palmer2	4	3	0	0	0	10
palmer2a	6	2	0	0	0	21
palmer2b	4	2	0	0	0	10
palmer2e	8	0	0	0	0	36
palmer3	4	3	0	0	0	10
palmer3a	6	2	0	0	0	21
palmer3b	4	2	0	0	0	10
palmer3e	8	0	0	0	0	36
palmer4	4	3	0	0	0	10
palmer4a	6	2	0	0	0	21
palmer4b	4	2	0	0	0	10
palmer4e	8	0	0	0	0	36
palmer5a	8	2	0	0	0	36
palmer5b	9	2	0	0	0	45
palmer5e	8	1	0	0	0	36
palmer6a	6	2	0	0	0	21
palmer6e	8	1	0	0	0	36
palmer7a	6	2	0	0	0	21
palmer7e	8	1	0	0	0	36
palmer8a	6	2	0	0	0	21
palmer8e	8	1	0	0	0	36
penalty1	1000	0	0	0	0	500500
penalty2	100	0	0	0	0	5050
pentagon	6	0	0	15	27	21
pfit1ls	3	0	0	0	0	6
pfit2ls	3	0	0	0	0	6
pfit3ls	3	0	0	0	0	6
pfit4ls	3	0	0	0	0	6
polak1	3	0	0	2	6	3
polak2	11	0	0	2	22	55
polak3	12	0	0	10	120	11
polak4	3	0	0	3	9	2
polak5	3	0	0	2	6	3
polak6	5	0	0	4	20	7
power	1000	0	0	0	0	1000
probpenl	500	500	0	0	0	125250
prodpl0	60	60	20	9	112	31
prodpl1	60	60	20	9	112	31
pspdoc	4	0	0	1	1	9
qr3d	155	10	0	0	0	9120
qr3dbd	127	10	0	0	0	5872
qr3dls	155	10	0	0	0	9120
qrtquad	120	0	0	10	10	239
quartc	10000	0	0	0	0	10000
reading1	10001	10001	5000	0	19999	10000
reading3	202	202	101	1	604	202
rk23	17	6	11	0	43	11
robot	14	0	9	0	21	21
rosenbr	2	0	0	0	0	3
s365mod	7	0	0	9	26	26
s368	100	100	0	0	0	5050
sawpath	593	0	590	196	3325	588
scon1dls	1000	1000	0	0	0	2997
scosine	10000	0	0	0	0	19999

Table A.1: Characteristics of CUTE test set (*continued on next page*)

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
scurly10	10000	0	0	0	0	109945
scurly20	10000	0	0	0	0	209790
scurly30	10000	0	0	0	0	309535
sineali	20	20	0	0	0	39
sineval	2	0	0	0	0	3
sinquad	10000	0	0	0	0	29997
sinrosnb	1000	0	0	1000	1999	1999
sisser	2	0	0	0	0	3
smbank	117	85	64	0	234	85
smmpsf	720	720	240	23	1213	56
snake	2	0	0	2	4	1
spanhyd	97	97	33	0	194	513
spiral	3	0	0	2	6	3
sreadin3	10001	10001	5001	0	20000	10000
srosenbr	10000	0	0	0	0	15000
ssebnln	194	194	72	24	360	24
ssnlbeam	33	22	20	0	80	22
stancmin	3	3	0	2	6	6
steenbrb	468	468	108	0	864	3276
steenbrc	540	540	126	0	1008	216
steenbrd	468	468	108	0	864	3276
steenbre	540	540	126	0	1008	4320
steenbrf	468	468	108	0	864	216
steenbrg	540	540	126	0	1008	4320
svanberg	5000	5000	0	5000	42504	5000
swopf	83	10	78	14	364	98
synthes1	6	6	0	6	16	3
trainf	20008	10008	10002	0	50010	10002
trainh	20008	10008	10002	0	60012	15004
trimloss	142	142	20	55	963	36
try-b	2	2	1	0	2	2
twirism1	343	343	224	89	4084	3738
twobars	2	2	0	2	4	3
ubh5	19997	6003	14000	0	63981	6003
vardim	100	0	0	0	0	5050
watson	31	0	0	0	0	496
weeds	3	1	0	0	0	6
womflet	3	0	0	3	9	2
woods	10000	0	0	0	0	17500
yfit	3	1	0	0	0	6
yfitu	3	0	0	0	0	6
zecevic3	2	0	0	4	6	3
zecevic4	2	0	0	4	6	3
zigzag	64	36	40	10	140	10
zy2	3	3	0	2	4	3

Table A.1: Characteristics of problems in the CUTE test set

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
bearing_200	40000	40000	0	0	0	119600
bearing_400	160000	160000	0	0	0	479200
camshape_10000	10000	10000	0	20000	49997	19998
camshape_20000	20000	20000	0	40000	99997	39998
catmix_10000	30001	10001	20000	0	119996	20000
catmix_20000	60001	20001	40000	0	239996	40000
chain_20000	40000	0	20001	0	99999	40000
chain_40000	80000	0	40001	0	199999	80000
channel_5000	39998	0	39998	0	309974	159989
channel_10000	79998	0	79998	0	619974	319989
elec_200	600	0	200	0	600	180300
elec_400	1200	0	400	0	1200	720600
gasoil_2500	25001	3	24998	0	219974	75292
gasoil_5000	50001	3	49998	0	439974	150292
glider_2500	12499	7501	10000	0	69983	74973
glider_5000	24999	15001	20000	0	139983	149973
marine_1000	24015	15	23992	0	165968	45208
marine_2000	48015	15	47992	0	331968	90208
methanol_5000	60002	5	59997	0	674964	340143
methanol_10000	120002	5	119997	0	1349964	680143
minsurf_200_200	40000	40000	0	0	0	159201
minsurf_300_300	90000	90000	0	0	0	358801
pinene_2500	50000	5	49995	0	422446	50395
pinene_5000	100000	5	99995	0	844946	100395
polygon_200	398	398	0	20098	79598	79401
polygon_400	798	798	0	80198	319198	318801
robot_5000	44999	30001	30001	0	179978	129980
robot_10000	89999	60001	60001	0	359978	259980
rocket_10000	40001	40001	30000	0	189992	259985
rocket_20000	80001	80001	60000	0	379992	519985
steering_10000	50000	10002	40001	0	199991	40000
steering_20000	100000	20002	80001	0	399991	80000
torsion_200_200	40000	40000	0	0	0	119600
torsion_400_400	160000	160000	0	0	0	479200

Table A.2: Characteristics of problems in the COPS test set

Problem	#var	#bdd	#eq	#ineq	#nzJac	#nzHes
cont5_1	40400	40400	40200	0	239603	600
cont5_2_1	40400	40400	40200	0	239603	401
cont5_2_2	40400	40400	40200	0	239603	600
cont5_2_3	40400	40400	40200	0	239603	600
cont5_2_4	40400	200	40200	0	239603	120200
cont_p	12096	192	11904	1	82188	12096
ex1_80	12482	12482	6241	0	37130	12482
ex1_160	50562	50562	25281	0	151050	50562
ex2_80	12482	12482	6241	0	37130	6241
ex2_160	50562	50562	25281	0	151050	25281
ex3_80	12482	12482	6241	0	37130	12482
ex3_160	50562	50562	25281	0	151050	50562
ex4_80	12798	12798	6557	0	38078	12482
ex4_160	51198	51198	25917	0	152958	50562
ex4_2_80	12798	12798	6557	0	38078	18723
ex4_2_160	51198	51198	25917	0	152958	75843
ex5_80	12798	12798	6557	0	38078	6241
ex5_160	51198	51198	25917	0	152958	25281
ex6_80	12798	12798	6557	0	38078	18723
ex6_160	51198	51198	25917	0	152958	75843
lukvle1	50000	0	49998	0	149994	99999
lukvle2	50002	0	49993	0	399929	100003
lukvle3	50002	0	2	0	4	125003
lukvle4	50002	0	49998	0	149994	100003
lukvle5	50000	0	49996	0	249980	149997
lukvle6	49999	0	24999	0	74997	349972
lukvle7	50002	0	4	0	14	50005
lukvle8	50000	0	49998	0	149994	150000
lukvle9	50000	0	6	0	30	75002
lukvle10	50000	0	49998	0	149994	75000
lukvle11	49997	0	33330	0	99990	99993
lukvle12	49997	0	37497	0	99992	112492
lukvle13	49997	0	33330	0	133320	83327
lukvle14	49997	0	33330	0	99990	66662
lukvle15	49997	0	37497	0	112491	99993
lukvle16	49997	0	37497	0	87493	74995
lukvle17	49997	0	37497	0	87493	74995
lukvle18	49997	0	37497	0	87493	74995
lukvli1	50000	0	0	49998	149994	99999
lukvli2	50000	0	0	49993	399929	99999
lukvli3	50000	0	0	2	4	124998
lukvli4	50000	0	0	49998	149994	99999
lukvli5	50000	0	0	49996	249980	149997
lukvli6	49999	0	0	24999	74997	349972
lukvli7	50000	0	0	4	14	50003
lukvli8	50000	0	0	49998	149994	150000
lukvli9	50000	0	0	6	30	75002
lukvli10	50000	0	0	49998	149994	75000
lukvli11	49997	0	0	33330	99990	99993
lukvli12	49997	0	0	37497	99992	112492
lukvli13	49997	0	0	33330	133320	83327
lukvli14	49997	0	0	33330	99990	66662
lukvli15	49997	0	0	37497	112491	99993
lukvli16	49997	0	0	37497	87493	74995
lukvli17	49997	0	0	37497	87493	74995
lukvli18	49997	0	0	37497	87493	74995

Table A.3: Characteristics of problems in the MITT test set

Appendix B

Results for IPOPT

Table B.1 lists the number of function evaluations ($\#f$) and iteration counts (#iter) for different line search options discussed in Section 5.1.2 on each problem in the CUTE test set. For the filter option, in addition the total number of successful iterations taken within TRON in the restoration phase (#Tit) is shown. A star (*) after a problem name indicates that some options converged to points with different final values of the objective function, and the error codes ‡ and † are explained in Table B.5.

Tables B.2–B.4 document the runs of IPOPT on the test sets CUTE, COPS, and MITT for the comparison in Section 5.1.3. For each problem, they list the following numbers: number of iterations (#iter) and error code (see Table B.5), if failed; number of evaluations of objective function ($\#f$) and constraints ($\#c$); required CPU time in seconds; number of iterations in which a regularization (δ_1 or δ_2 in (3.6) and (3.7) non-zero) was necessary (#ref); how often TRON was called to perform feasibility restoration phase (#call) and the total number of successful iterations (#it) and CG steps (#cg) within TRON; final value of the objective function ($f(x_*)$) and (unscaled) constraint violation ($\|c(x_*)\|$). A problem name is marked with a star (*), if KNITRO or LOQO successfully terminated at a point with a different value of the objective function (see Eq. (5.3)).

Problem	auglag	exact1	exact2	filter		fullstep
	#f/#iter	#f/#iter	#f/#iter	#f/#iter	[#Tit]	#f/#iter
airport	14/13	14/13	14/13	14/13	[0]	14/13
aljazzaf	42/38	98/28	98/28	87/29	[0]	35/34
allinit	16/12	24/12	24/12	18/12	[0]	18/17
allinitc	*33/30*$^{\dagger 5}$	*37/32*$^{\dagger 5}$	*36/30*$^{\dagger 5}$	52/39	[0]	*31/30*$^{\dagger 5}$
allinitu	15/14	15/14	15/14	15/14	[0]	15/14
alsotame	11/10	11/10	11/10	11/10	[0]	11/10
arwhead	7/6	7/6	7/6	7/6	[0]	7/6
avion2	*17/16*$^{\dagger 5}$	24036/946	*17/16*$^{\dagger 5}$	79/23	[0]	*17/16*$^{\dagger 5}$
bard	9/8	9/8	9/8	9/8	[0]	9/8
batch	46/45	50/44	45/44	46/45	[0]	45/44
bdexp	18/17	18/17	18/17	18/17	[0]	18/17
bdqrtic	11/10	11/10	11/10	11/10	[0]	11/10
beale	12/6	14/6	14/6	13/6	[0]	*1000/999*$^{\dagger 5}$
bigbank	27/26	36/28	31/27	27/26	[0]	27/26
biggs3	17/11	26/9	26/9	23/9	[0]	14/13
biggs5	28/23	31/20	31/20	27/20	[0]	49/48
biggs6	38/34	44/34	44/34	41/34	[0]	*151/150*$^{\dagger 9}$
box2	9/8	9/8	9/8	9/8	[0]	9/8
box3	11/9	13/9	13/9	12/9	[0]	10/9
brainpc0	*70487/3000*$^{\dagger 1}$	*12625/643*$^{\dagger 2}$	*3645/231*$^{\dagger 2}$	*47610/3000*$^{\dagger 1}$	[*0*]	37/36
brainpc1	*42327/3000*$^{\dagger 1}$	*1053/72*$^{\dagger 2}$	*66/5*$^{\dagger 2}$	*0/0*‡	[*0*]	111/110
brainpc2	295/81	*1767/41*$^{\dagger 2}$	*110/14*$^{\dagger 2}$	*700/284*$^{\dagger 6}$	[*178*]	60/59
brainpc3	*39734/3000*$^{\dagger 1}$	*272/28*$^{\dagger 2}$	*568/23*$^{\dagger 2}$	76/61	[0]	*792/791*$^{\dagger 2}$
brainpc4	*99167/3000*$^{\dagger 1}$	*2327/133*$^{\dagger 2}$	*626/25*$^{\dagger 2}$	73/57	[0]	145/144
brainpc5	*34344/3000*$^{\dagger 1}$	*3068/154*$^{\dagger 2}$	*494/21*$^{\dagger 2}$	*1562/129*$^{\dagger 8}$	[*52*]	*515/514*$^{\dagger 5}$
brainpc6	*38607/3000*$^{\dagger 1}$	*2185/130*$^{\dagger 2}$	*502/21*$^{\dagger 2}$	*47942/3000*$^{\dagger 1}$	[*24*]	*2830/2829*$^{\dagger 2}$
brainpc7	*99076/3000*$^{\dagger 1}$	*2142/130*$^{\dagger 2}$	*745/27*$^{\dagger 2}$	92/57	[0]	161/160
brainpc8	*99136/3000*$^{\dagger 1}$	*267/28*$^{\dagger 2}$	*221/15*$^{\dagger 2}$	120/66	[1]	117/116
brainpc9	*99662/3000*$^{\dagger 1}$	*2892/158*$^{\dagger 2}$	*66/5*$^{\dagger 2}$	191/91	[1]	*499/498*$^{\dagger 2}$
bratu1d	*62819/3000*$^{\dagger 1}$	*68807/3000*$^{\dagger 1}$	*68807/3000*$^{\dagger 1}$	*65813/3000*$^{\dagger 1}$	[*0*]	412/411
britgas	*34819/3000*$^{\dagger 1}$	200/31	222/35	55/34	[0]	*646/645*$^{\dagger 5}$
brkmcc	4/3	4/3	4/3	4/3	[0]	4/3
brownal	8/7	8/7	8/7	8/7	[0]	8/7
brownbs	10/8	12/8	12/8	11/8	[0]	10/9
broydn7d	96/93	98/93	98/93	97/93	[0]	89/88
brybnd	9/8	9/8	9/8	9/8	[0]	9/8
bt1	*48/0*$^{\dagger 2}$	*11700/3000*$^{\dagger 1}$	*11700/3000*$^{\dagger 1}$	*161/15*$^{\dagger 4}$	[*4*]	19/18
bt2	13/12	26/11	26/11	13/12	[0]	13/12
bt4	34/6	41365/2088	143/16	14/13	[0]	14/13
bt5	8/6	10/6	10/6	8/7	[0]	8/7
bt6	11/9	23/9	23/9	18/13	[0]	14/13
bt7*	24/16	75/18	47/16	22/14	[0]	27/26
bt8	53/52	53/52	55/53	53/52	[0]	53/52
bt9	16/12	19/11	16/10	14/13	[0]	14/13
bt11	9/8	21/7	21/7	9/8	[0]	9/8
bt12	5/4	6/4	5/4	5/4	[0]	5/4
bt13	24/23	29/23	29/23	25/23	[0]	24/23
byrdsphr	364/33	*92688/3000*$^{\dagger 1}$	*92688/3000*$^{\dagger 1}$	25/24	[0]	25/24
camel6*	15/13	18/13	18/13	17/13	[0]	14/13
cantilvr	17/14	15/14	15/14	15/14	[0]	15/14
catena	8/6	7/6	7/6	7/6	[0]	7/6
catenary	*12205/3000*$^{\dagger 1}$	188/47	8654/539	95/46	[0]	*3001/3000*$^{\dagger 1}$
cb2	11/10	11/10	16/10	11/10	[0]	11/10
cb3	10/9	10/9	10/9	10/9	[0]	10/9
chaconn1	9/8	9/8	9/8	9/8	[0]	9/8
chaconn2	8/7	8/7	8/7	8/7	[0]	8/7
chebyqad	86/78	94/78	94/78	90/78	[0]	111/110
chnrosnb	55/43	71/43	71/43	63/43	[0]	43/42
cliff	28/27	28/27	28/27	28/27	[0]	28/27
clnlbeam	287/286	*56/18*$^{\dagger 2}$	*212/103*$^{\dagger 2}$	288/286	[0]	287/286
clplatea	8/6	10/6	10/6	9/6	[0]	9/8

Table B.1: Comparison of IPOPT's line search options (*continued on next page*)

Problem	auglag	exact1	exact2	filter		fullstep
	#f/#iter	#f/#iter	#f/#iter	#f/#iter	[#Tit]	#f/#iter
clplateb	9/6	11/6	11/6	10/6	[0]	10/9
clplatec	3/2	3/2	3/2	3/2	[0]	3/2
concon	12/11	15/12	13/12	12/11	[0]	13/12
congigmz	30/27	*349/39*$^{\dagger 2}$	*54787/3000*$^{\dagger 1}$	36/29	[0]	45/44
core1	552/150	*414/136*$^{\dagger 2}$	*1390/178*$^{\dagger 2}$	802/280	[147]	139/138
corkscrw	*389/388*$^{\dagger 2}$	*314/294*$^{\dagger 2}$	*335/295*$^{\dagger 2}$	2829/2099	[214]	*299/298*$^{\dagger 5}$
coshfun	*71445/3000*$^{\dagger 1}$	*57646/3000*$^{\dagger 1}$	*9165/470*$^{\dagger 2}$	*57740/3000*$^{\dagger 1}$	[*1182*]	*877/822*$^{\dagger 2}$
cosine	11/10	11/10	11/10	11/10	[0]	11/10
cragglvy	16/15	16/15	16/15	16/15	[0]	16/15
cresc100	*32023/3000*$^{\dagger 1}$	*6494/235*$^{\dagger 2}$	*5308/239*$^{\dagger 2}$	*6684/1090*$^{\dagger 8}$	[*192*]	*286/285*$^{\dagger 5}$
cresc132	*501/278*$^{\dagger 2}$	*440/51*$^{\dagger 2}$	*419/46*$^{\dagger 2}$	*20077/3000*$^{\dagger 1}$	[*1258*]	*661/660*$^{\dagger 5}$
cresc4	158/66	*66547/3000*$^{\dagger 1}$	*66170/3000*$^{\dagger 1}$	*15644/1304*$^{\dagger 8}$	[*140*]	139/138
cresc50	*32973/3000*$^{\dagger 1}$	*70002/2583*$^{\dagger 2}$	*886/67*$^{\dagger 2}$	*9748/1170*$^{\dagger 8}$	[*132*]	*748/747*$^{\dagger 2}$
csfi1	82/31	*810/54*$^{\dagger 2}$	94/26	22/21	[0]	18/17
csfi2	33/28	*1315/77*$^{\dagger 2}$	*1968/97*$^{\dagger 2}$	55/32	[2]	38/37
cube	38/27	48/27	48/27	43/27	[0]	9/8
curly10	23/22	23/22	23/22	23/22	[0]	23/22
curly20	27/21	31/21	31/21	29/21	[0]	24/23
curly30	27/26	27/26	27/26	27/26	[0]	27/26
dallasl	429/79	*911/58*$^{\dagger 2}$	*335/45*$^{\dagger 2}$	*583/181*$^{\dagger 7}$	[*4*]	*89/84*$^{\dagger 2}$
dallasm	198/40	*265/28*$^{\dagger 2}$	*522/44*$^{\dagger 2}$	56/33	[0]	39/38
dallass	168/42	698/55	255/32	*110/30*$^{\dagger 7}$	[*3*]	40/30
deconvc*	183/146	*52750/3000*$^{\dagger 1}$	*52952/3000*$^{\dagger 1}$	158/105	[0]	93/92
demymalo	14/13	21/13	21/13	14/13	[0]	14/13
denschna	7/6	7/6	7/6	7/6	[0]	7/6
denschnb	21/7	23/7	23/7	22/7	[0]	27/26
denschnc	11/10	11/10	11/10	11/10	[0]	11/10
denschnd	49/47	51/47	51/47	50/47	[0]	53/52
denschne	15/10	17/10	17/10	16/10	[0]	117/116
denschnf	7/6	7/6	7/6	7/6	[0]	7/6
dipigri	19/11	21/11	21/11	22/12	[0]	28/27
disc2	104/95	*164/54*$^{\dagger 2}$	*116/53*$^{\dagger 2}$	91/79	[1]	145/144
discs	*205/204*$^{\dagger 5}$	*202/201*$^{\dagger 5}$	*202/201*$^{\dagger 5}$	*4449/1071*$^{\dagger 8}$	[*809*]	*202/201*$^{\dagger 5}$
dittert	33/31	8717/329	280/38	37/31	[0]	37/36
dixchlng	25/10	23/9	23/9	11/10	[0]	11/10
dixchlnv	23/21	21/20	21/20	21/20	[0]	21/20
dixmaana	7/6	7/6	7/6	7/6	[0]	7/6
dixmaanb	8/7	8/7	8/7	8/7	[0]	8/7
dixmaanc	10/9	10/9	10/9	10/9	[0]	10/9
dixmaand	10/9	10/9	10/9	10/9	[0]	10/9
dixmaane	9/8	9/8	9/8	9/8	[0]	9/8
dixmaanf	23/22	23/22	23/22	23/22	[0]	23/22
dixmaang	17/16	17/16	17/16	17/16	[0]	17/16
dixmaanh	23/22	23/22	23/22	23/22	[0]	23/22
dixmaani	12/11	12/11	12/11	12/11	[0]	12/11
dixmaanj	21/20	21/20	21/20	21/20	[0]	21/20
dixmaank	29/24	33/24	33/24	31/24	[0]	25/24
dixmaanl	30/29	30/29	30/29	30/29	[0]	30/29
djtl	76/26	94/26	94/26	*73/17*$^{\dagger 3}$	[*0*]	*3001/3000*$^{\dagger 1}$
dnieper	31/30	*71055/3000*$^{\dagger 1}$	388/60	33/30	[0]	31/30
dqrtic	39/38	39/38	39/38	39/38	[0]	39/38
drcavty1	228/203	840/327	*0/0*‡	544/264	[0]	*0/0*‡
drcavty2*	285/247	867/349	*0/0*‡	*0/0*‡	[*0*]	*0/0*‡
drcavty3	*0/0*‡	*0/0*‡	*0/0*‡	*0/0*‡	[*0*]	*0/0*‡
dtoc1l	7/6	7/6	7/6	7/6	[0]	7/6
dtoc1na	7/6	7/6	7/6	7/6	[0]	7/6
dtoc1nb	7/6	7/6	7/6	7/6	[0]	7/6
dtoc1nc	15/11	72/14	14/10	20/15	[0]	12/11
dtoc1nd*	33/22	*5520/228*$^{\dagger 2}$	2127/176	35/26	[0]	40/39
dtoc2	16/15	91/15	17/9	17/10	[0]	16/15
dtoc4	4/3	4/3	4/3	4/3	[0]	4/3

Table B.1: Comparison of IPOPT's line search options (*continued on next page*)

Problem	auglag	exact1	exact2	filter		fullstep
	#f/#iter	#f/#iter	#f/#iter	#f/#iter	[#Tit]	#f/#iter
dtoc5	5/4	18/4	5/4	5/4	[0]	5/4
dtoc6	12/11	25/11	12/11	12/11	[0]	12/11
edensch	8/7	8/7	8/7	8/7	[0]	8/7
eg1	11/10	11/10	11/10	11/10	[0]	11/10
eg2	4/3	4/3	4/3	4/3	[0]	4/3
eg3*	35/33	85/26	99/26	37/36	[0]	31/30
eigena	30/28	32/28	32/28	31/28	[0]	34/33
eigena2	4/3	4/3	4/3	4/3	[0]	4/3
eigenaco	4/3	4/3	4/3	4/3	[0]	4/3
eigenals	27/25	29/25	29/25	28/25	[0]	31/30
eigenb	143/116	179/116	179/116	161/116	[0]	127/126
eigenb2	47633/1902	177/64	14371/516	73/62	[0]	56/55
eigenbco	85/73	30300/1982	*62244/2013*$^{\dagger 2}$	91/71	[0]	71/70
eigenbls	154/114	194/114	194/114	174/114	[0]	101/100
eigenc2	19/18	40/18	21/18	19/18	[0]	19/18
eigencco	11/10	13/10	11/10	11/10	[0]	11/10
engval1	9/8	9/8	9/8	9/8	[0]	9/8
engval2	21/20	21/20	21/20	21/20	[0]	21/20
errinros	49/29	65/29	65/29	57/29	[0]	22/21
expfit	9/8	9/8	9/8	9/8	[0]	9/8
expfita	33/32	35/31	35/31	33/31	[0]	33/32
expfitb	76/75	76/75	76/75	76/75	[0]	76/75
expfitc	157/156	160/156	160/156	158/156	[0]	157/156
explin	22/21	22/21	22/21	22/21	[0]	22/21
explin2	22/21	22/21	22/21	22/21	[0]	22/21
expquad	24/23	24/23	24/23	24/23	[0]	24/23
extrosnb	1/0	1/0	1/0	1/0	[0]	1/0
fletcbv2	3/2	3/2	3/2	3/2	[0]	3/2
fletcbv3	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*[0]*	*3001/3000*$^{\dagger 1}$
fletchbv	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*[0]*	*3001/3000*$^{\dagger 1}$
fletchcr	50/48	52/48	52/48	51/48	[0]	46/45
fletcher*	29/25	*357/9*$^{\dagger 2}$	*1066/29*$^{\dagger 2}$	55/20	[3]	128/127
flosp2hh	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*[0]*	*3001/3000*$^{\dagger 1}$
flosp2hl	4/3	4/3	4/3	4/3	[0]	4/3
flosp2hm	*9298/3000*$^{\dagger 1}$	*10140/3000*$^{\dagger 1}$	*10140/3000*$^{\dagger 1}$	*2579/2578*$^{\dagger 3}$	*[0]*	*2932/2931*$^{\dagger 9}$
flosp2th	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*[0]*	*3001/3000*$^{\dagger 1}$
flosp2tl	3/2	3/2	3/2	3/2	[0]	3/2
flosp2tm	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*3001/3000*$^{\dagger 1}$	*1017/1016*$^{\dagger 3}$	*[0]*	*3001/3000*$^{\dagger 1}$
fminsrf2	100/17	126/17	126/17	113/17	[0]	*3001/3000*$^{\dagger 1}$
fminsurf	228/33	284/33	284/33	256/33	[0]	*0/0*‡
freuroth	12/8	14/8	14/8	13/8	[0]	15/14
gausselm	*10119/2119*$^{\dagger 5}$	*1136/150*$^{\dagger 2}$	*24536/3000*$^{\dagger 1}$	*1721/1599*$^{\dagger 5}$	*[1]*	*1203/1202*$^{\dagger 5}$
genhumps	202/194	210/194	210/194	206/194	[0]	380/379
genrose	849/744	1057/744	1057/744	953/744	[0]	666/665
gigomez1	18/15	36/14	77/18	19/16	[0]	20/19
gilbert	20/19	20/19	20/19	20/19	[0]	20/19
gpp	26/25	357/54	28/26	26/25	[0]	26/25
growth*	102/70	136/70	136/70	119/70	[0]	51/50
growthls*	107/74	145/74	145/74	126/74	[0]	51/50
gulf	28/22	36/22	36/22	32/22	[0]	33/32
hadamals	99/98	99/98	99/98	99/98	[0]	99/98
hadamard	7/6	7/6	7/6	13/9	[0]	7/6
hager2	2/1	2/1	2/1	2/1	[0]	2/1
hager4	14/13	14/13	14/13	14/13	[0]	14/13
haifam	7158/885	*68338/3000*$^{\dagger 1}$	*65547/3000*$^{\dagger 1}$	42/31	[0]	33/32
haifas	36/18	12/10	12/10	14/12	[0]	13/12
hairy	53/46	61/46	61/46	57/46	[0]	69/68
haldmads*	30/21	7900/567	*7184/513*$^{\dagger 2}$	65/58	[0]	169/168
hanging	35/17	24/18	21/18	20/19	[0]	20/19
hart6	12/10	14/10	14/10	13/10	[0]	13/12
hatflda	12/11	12/11	12/11	12/11	[0]	12/11
hatfldb	13/12	13/12	13/12	13/12	[0]	13/12

Table B.1: Comparison of IPOPT's line search options (*continued on next page*)

Problem	auglag	exact1	exact2	filter		fullstep
	#f/#iter	#f/#iter	#f/#iter	#f/#iter	[#Tit]	#f/#iter
hatfldc	8/7	8/7	8/7	8/7	[0]	8/7
hatfldd	28/21	34/21	34/21	31/21	[0]	40/39
hatflde	31/26	37/26	37/26	34/26	[0]	*12/11*$^{\dagger 9}$
heart6ls	1012/828	1204/828	1204/828	1108/828	[0]	*3001/3000*$^{\dagger 1}$
heart8ls	111/88	129/88	129/88	120/88	[0]	85/84
helix	13/11	15/11	15/11	14/11	[0]	13/12
himmelbb	15/10	17/10	17/10	16/10	[0]	69/68
himmelbf	11/9	13/9	13/9	12/9	[0]	62/61
himmelbg	10/6	12/6	12/6	11/6	[0]	21/20
himmelbh	20/4	22/4	22/4	21/4	[0]	21/20
himmelbi	31/30	31/30	31/30	31/30	[0]	31/30
himmelbj	110/52	90/46	489/52	*62/37*$^{\dagger 6}$	[*3*]	47/46
himmelbk	22/20	158/24	61/22	20/19	[0]	20/19
himmelp1	12/11	12/11	12/11	12/11	[0]	12/11
himmelp2*	167/41	60/23	32/24	15/14	[0]	15/14
himmelp3	39/19	14/13	14/13	14/13	[0]	14/13
himmelp4	59/25	70/22	24/17	18/17	[0]	18/17
himmelp5	*42/36*$^{\dagger 5}$	371/44	*54613/3000*$^{\dagger 1}$	30/29	[0]	30/29
himmelp6	10/9	10/9	10/9	10/9	[0]	10/9
hong	14/13	14/13	14/13	14/13	[0]	14/13
hs001	38/30	49/30	49/30	45/30	[0]	15/14
hs002	11/10	11/10	11/10	11/10	[0]	11/10
hs004	7/6	7/6	7/6	7/6	[0]	7/6
hs005	11/10	15/10	15/10	12/10	[0]	11/10
hs006	10/7	14/6	14/6	7/5	[0]	6/5
hs007	20/10	61/11	17/8	10/9	[0]	10/9
hs009	5/3	7/3	7/3	6/3	[0]	8/7
hs010	14/13	14/13	14/13	14/13	[0]	14/13
hs011	9/8	9/8	9/8	9/8	[0]	9/8
hs012	17/11	14/13	14/13	14/13	[0]	14/13
hs013	34/33	34/33	35/33	34/33	[0]	34/33
hs014	10/9	10/9	10/9	10/9	[0]	10/9
hs015	17/16	57/16	40/25	32/17	[0]	17/16
hs016	11/10	11/10	11/10	11/10	[0]	11/10
hs017	26/23	70/24	30/23	30/24	[0]	24/23
hs018	23/18	56/15	56/15	20/16	[0]	16/15
hs019	12/11	44/14	44/14	12/11	[0]	14/13
hs020	13/12	13/12	13/12	13/12	[0]	13/12
hs023	12/11	14/10	12/10	12/10	[0]	12/11
hs024	12/11	16/12	16/12	14/12	[0]	12/11
hs025	40/38	49/34	49/34	46/34	[0]	66/65
hs026	25/24	25/24	25/24	25/24	[0]	25/24
hs027	*4388/3000*$^{\dagger 1}$	*43576/3000*$^{\dagger 1}$	*44389/3000*$^{\dagger 1}$	214/75	[2]	*300/252*$^{\dagger 2}$
hs029	13/10	279/27	13/11	10/9	[0]	12/11
hs030	50/48	69/50	196/49	47/32	[0]	46/45
hs031	12/10	11/10	11/10	11/10	[0]	11/10
hs032	18/16	21/16	21/16	20/16	[0]	16/15
hs033	22/8	20/11	20/11	12/11	[0]	12/11
hs034	13/9	11/9	11/10	11/10	[0]	11/10
hs036	12/11	12/11	12/11	12/11	[0]	12/11
hs037	11/10	11/10	11/10	11/10	[0]	11/10
hs038	56/42	72/42	72/42	64/42	[0]	30/29
hs039	16/12	19/11	16/10	14/13	[0]	14/13
hs040	4/3	4/3	4/3	4/3	[0]	4/3
hs041	12/11	102/17	102/17	14/12	[0]	12/11
hs042	8/7	8/7	8/7	8/7	[0]	8/7
hs043	11/10	11/10	11/10	11/10	[0]	11/10
hs045	22/21	22/21	22/21	22/21	[0]	22/21
hs046	25/23	37/19	27/23	19/18	[0]	19/18
hs047	21/20	24/19	24/19	21/19	[0]	21/20
hs049	21/20	21/20	21/20	21/20	[0]	21/20
hs050	10/9	10/9	10/9	10/9	[0]	10/9

Table B.1: Comparison of IPOPT's line search options (*continued on next page*)

Problem	auglag	exact1	exact2	filter		fullstep
	#f/#iter	#f/#iter	#f/#iter	#f/#iter	[#Tit]	#f/#iter
hs054	9/8	9/8	9/8	9/8	[0]	9/8
hs056	23/17	*89236/3000*$^{\dagger 1}$	41/17	103/70	[0]	60/59
hs057	1883/141	*318/36*$^{\dagger 2}$	294/45	*1151/48*$^{\dagger 8}$	[*2*]	*445/390*$^{\dagger 2}$
hs059*	112/41	*52222/3000*$^{\dagger 1}$	5114/414	164/46	[1]	*30/29*$^{\dagger 5}$
hs060	9/8	9/8	9/8	9/8	[0]	9/8
hs061*	*164/27*$^{\dagger 2}$	*1162/46*$^{\dagger 2}$	*1040/42*$^{\dagger 2}$	19969/1847	[188]	2281/2280
hs062	10/9	10/6	10/6	8/6	[0]	10/9
hs063	10/9	15/9	13/9	10/9	[0]	10/9
hs064	29/28	41/22	42/22	31/20	[0]	29/28
hs065	17/16	*68815/3000*$^{\dagger 1}$	*58053/3000*$^{\dagger 1}$	17/16	[0]	48/47
hs066	12/8	9/8	9/8	9/8	[0]	9/8
hs067	20/10	243/18	147/14	10/9	[0]	10/9
hs070*	29/21	61/19	61/19	31/20	[0]	26/25
hs071	10/9	15/9	15/9	10/9	[0]	10/9
hs072	17/16	17/16	17/16	17/16	[0]	17/16
hs073	10/9	10/9	10/9	10/9	[0]	10/9
hs074	12/11	15/11	18/11	12/11	[0]	12/11
hs075	11/10	10677/762	11016/784	11/10	[0]	11/10
hs077	11/9	23/9	23/9	13/11	[0]	12/11
hs078	5/4	5/4	5/4	5/4	[0]	5/4
hs079	5/4	5/4	5/4	5/4	[0]	5/4
hs080	9/8	9/8	9/8	9/8	[0]	9/8
hs081	9/8	11/8	14/8	9/8	[0]	9/8
hs083	18/17	18/17	18/17	18/17	[0]	18/17
hs084	20/19	231/50	259/53	20/19	[0]	33/32
hs085	7126/809	157/19	15/14	15/14	[0]	15/14
hs086	12/11	12/11	12/11	12/11	[0]	12/11
hs087	20/17	121/24	27/22	19/17	[0]	22/21
hs088	26/20	24/17	24/17	21/16	[0]	24/23
hs089	14/12	29/17	24/17	22/17	[0]	17/16
hs090	17/13	26/15	26/15	19/13	[0]	*3001/3000*$^{\dagger 1}$
hs091	22/17	27/15	24/16	49/34	[0]	*172/171*$^{\dagger 5}$
hs092	23/22	25/16	26/18	23/22	[0]	23/22
hs093	8/7	9/7	9/7	8/7	[0]	8/7
hs095	15/14	21/14	19/14	15/14	[0]	15/14
hs096	21/17	42/17	34/16	27/19	[0]	17/16
hs097	19/18	143/44	30/23	19/18	[0]	31/30
hs098*	25/23	77/40	87/40	25/23	[0]	38/37
hs099	7/6	16/7	7/6	7/6	[0]	7/6
hs100	19/11	21/11	21/11	22/12	[0]	28/27
hs100lnp	7/6	7/6	7/6	7/6	[0]	7/6
hs100mod	23/11	28/10	28/10	35/11	[0]	24/23
hs101	634/117	707/85	114/28	100/32	[0]	40/39
hs102	838/141	1147/96	57/21	82/27	[0]	117/116
hs103	195/55	2007/166	32/21	88/27	[0]	31/30
hs104	20/15	35/13	19/11	53/43	[0]	32/31
hs105	27/26	27/26	27/26	27/26	[0]	27/26
hs106	19/15	46/17	15/14	15/14	[0]	15/14
hs107	*131/39*$^{\dagger 5}$	*27686/741*$^{\dagger 5}$	*57/22*$^{\dagger 5}$	55/39	[2]	*124/123*$^{\dagger 9}$
hs108	22/17	38/17	34/18	25/22	[0]	21/20
hs109	76/32	548/39	*55475/3000*$^{\dagger 1}$	28/17	[0]	20/19
hs110	8/7	8/7	8/7	8/7	[0]	8/7
hs111	14/11	17/11	17/11	24/23	[0]	24/23
hs111lnp	15/11	29/12	28/12	24/23	[0]	24/23
hs112	18/17	18/17	18/17	18/17	[0]	18/17
hs113	13/12	16/11	13/12	13/12	[0]	13/12
hs114	20/19	365/44	20/19	20/19	[0]	20/19
hs116	24/23	36/23	32/23	24/23	[0]	24/23
hs117	46/25	*12259/3000*$^{\dagger 1}$	*11555/325*$^{\dagger 2}$	51/24	[1]	27/26
hs119	17/16	17/16	17/16	17/16	[0]	17/16
hs99exp	17/16	13/12	13/12	17/16	[0]	13/12
hubfit	10/9	10/9	10/9	10/9	[0]	10/9

Table B.1: Comparison of IPOPT's line search options (*continued on next page*)

Problem	auglag	exact1	exact2	filter		fullstep
	#f/#iter	#f/#iter	#f/#iter	#f/#iter	[#Tit]	#f/#iter
humps	359/346	371/346	371/346	365/346	[0]	406/405
hvycrash	*25/21*$^{\dagger 9}$	*1021/137*$^{\dagger 2}$	*50/11*$^{\dagger 2}$	*5840/741*$^{\dagger 7}$	[*409*]	*736/735*$^{\dagger 5}$
hypcir	6/4	8/5	8/5	8/5	[0]	6/5
indef	*3002/3000*$^{\dagger 1}$	*3004/3000*$^{\dagger 1}$	*3004/3000*$^{\dagger 1}$	*3003/3000*$^{\dagger 1}$	[*0*]	*3001/3000*$^{\dagger 1}$
jensmp	11/10	11/10	11/10	11/10	[0]	11/10
kissing	*445/130*$^{\dagger 2}$	*58/8*$^{\dagger 2}$	*117/36*$^{\dagger 2}$	256/178	[7]	548/547
kiwcresc	36/15	22/10	22/10	12/10	[0]	12/11
kowosb	15/8	19/8	19/8	17/8	[0]	*255/254*$^{\dagger 9}$
lakes	*55/1*$^{\dagger 2}$	*2419/54*$^{\dagger 2}$	*778/18*$^{\dagger 2}$	353/117	[63]	*107/106*$^{\dagger 9}$
launch	*4021/241*$^{\dagger 5}$	*7253/500*$^{\dagger 2}$	*24/23*$^{\dagger 5}$	*2257/762*$^{\dagger 8}$	[*675*]	*24/23*$^{\dagger 5}$
lch	56/54	157/57	159/57	56/55	[0]	56/55
liarwhd	13/12	13/12	13/12	13/12	[0]	13/12
lminsurf	3/2	3/2	3/2	3/2	[0]	3/2
loadbal	18/16	20/16	20/16	19/16	[0]	21/20
loghairy	842/809	878/809	878/809	860/809	[0]	2949/2948
logros	*94287/3000*$^{\dagger 1}$	*10019/3000*$^{\dagger 1}$	*10019/3000*$^{\dagger 1}$	*97243/3000*$^{\dagger 1}$	[*0*]	98/97
lootsma	22/8	20/11	20/11	12/11	[0]	12/11
lsnnodoc	13/12	19/13	19/13	16/13	[0]	13/12
madsen	23/22	125/26	50/18	23/22	[0]	23/22
madsschj	239/95	6535/508	4612/404	69/47	[0]	*231/230*$^{\dagger 2}$
makela1	16/15	56/20	55/20	16/15	[0]	18/17
makela2	10/9	11/9	10/9	10/9	[0]	10/9
makela3	36/18	46/20	*65683/3000*$^{\dagger 1}$	19/18	[0]	19/18
mancino	*11370/3000*$^{\dagger 1}$	*11969/3000*$^{\dagger 1}$	*11969/3000*$^{\dagger 1}$	*7/6*$^{\dagger 3}$	[*0*]	*3001/3000*$^{\dagger 1}$
manne	17038/2663	*2482/428*$^{\dagger 2}$	1343/1124	1336/1324	[0]	1341/1340
maratos	5/4	5/4	5/4	5/4	[0]	5/4
matrix2	18/17	18/17	18/17	18/17	[0]	18/17
maxlika	30/26	40/25	40/25	30/25	[0]	29/28
mccormck	10/9	10/9	10/9	10/9	[0]	10/9
mdhole	64/48	80/48	80/48	72/48	[0]	12/11
methanb8	8/7	8/7	8/7	8/7	[0]	8/7
methanl8	66/46	82/46	82/46	74/46	[0]	*125/124*$^{\dagger 9}$
mexhat	5/4	5/4	5/4	5/4	[0]	5/4
meyer3	*34174/3000*$^{\dagger 1}$	*39924/3000*$^{\dagger 1}$	*39924/3000*$^{\dagger 1}$	*37049/3000*$^{\dagger 1}$	[*0*]	*102/101*$^{\dagger 9}$
mifflin1	9/8	9/8	9/8	9/8	[0]	9/8
mifflin2	36/15	23/12	16/12	17/16	[0]	17/16
minc44	26/25	146/28	111/26	27/25	[0]	26/25
minmaxbd	148/63	*89369/3000*$^{\dagger 1}$	*10646/3000*$^{\dagger 1}$	202/129	[2]	235/234
minmaxrb	38/15	25/9	25/9	15/11	[0]	12/11
minperm	9/8	9/8	9/8	9/8	[0]	9/8
minsurf	4/3	4/3	4/3	4/3	[0]	4/3
mistake	194/39	*82473/3000*$^{\dagger 1}$	57/19	21/20	[0]	21/20
morebv	2/1	2/1	2/1	2/1	[0]	2/1
msqrtals	31/24	37/24	37/24	34/24	[0]	194/193
msqrtbls	41/28	49/28	49/28	45/28	[0]	*0/0*‡
mwright	10/8	28/9	26/10	10/9	[0]	10/9
ngone	50/49	*58617/3000*$^{\dagger 1}$	*45461/3000*$^{\dagger 1}$	50/49	[0]	50/49
noncvxu2	451/404	481/404	481/404	466/404	[0]	530/529
noncvxun	36/30	40/30	40/30	38/30	[0]	57/56
nondia	6/5	6/5	6/5	6/5	[0]	6/5
nondquar	25/24	25/24	25/24	25/24	[0]	25/24
nonmsqrt	*90331/3000*$^{\dagger 1}$	*95729/3000*$^{\dagger 1}$	*95729/3000*$^{\dagger 1}$	*93030/3000*$^{\dagger 1}$	[*0*]	565/564
nonscomp	42/23	50/23	50/23	46/23	[0]	30/29
odfits	13/12	13/12	13/12	13/12	[0]	13/12
oet2	89/63	*490/32*$^{\dagger 2}$	*10094/3000*$^{\dagger 1}$	157/128	[3]	*68/67*$^{\dagger 9}$
oet7	1564/379	*13520/3000*$^{\dagger 1}$	*287/97*$^{\dagger 2}$	150/129	[1]	*375/374*$^{\dagger 5}$
optcdeg2	32/31	109/32	32/31	32/31	[0]	32/31
optcdeg3	29/28	142/36	29/28	29/28	[0]	29/28
optcntrl	52/50	84/76	75/73	78/50	[0]	75/74
optctrl3	1027/141	260/27	241/27	1058/90	[51]	56/55
optctrl6	1027/141	260/27	241/27	1058/90	[51]	56/55

Table B.1: Comparison of IPOPT's line search options (*continued on next page*)

Problem	auglag	exact1	exact2	filter		fullstep
	#f/#iter	#f/#iter	#f/#iter	#f/#iter	[#Tit]	#f/#iter
optmass	24/22	134/28	33/23	23/22	[0]	23/22
optprloc	78/32	105/28	45/25	21/20	[0]	24/23
orthrdm2	7/5	8/6	10/5	8/6	[0]	9/8
orthrds2*	*70/14*$^{\dagger 2}$	*1798/90*$^{\dagger 2}$	57/53	1320/143	[31]	288/287
orthrega	52/45	994/99	186/63	71/49	[0]	45/44
orthregb	3/2	3/2	3/2	3/2	[0]	3/2
orthregc	15/14	32/14	30/14	28/14	[0]	15/14
orthregd*	8/6	14/7	11/6	448/81	[1]	*362/361*$^{\dagger 9}$
orthrege*	951/188	52/17	47046/2301	57/41	[0]	46/45
orthrgdm	*63/13*$^{\dagger 2}$	*153/43*$^{\dagger 2}$	1313/132	*832/71*$^{\dagger 6}$	[*25*]	*0/0*‡
orthrgds*	399/61	1966/194	*93880/3000*$^{\dagger 1}$	24/16	[0]	68/67
osbornea	50/35	62/35	62/35	56/35	[0]	*28/27*$^{\dagger 9}$
osborneb	19/17	21/17	21/17	20/17	[0]	*17/15*$^{\dagger 9}$
oslbqp	15/14	15/14	15/14	15/14	[0]	15/14
palmer1	1280/893	1792/893	1792/893	*1518/875*$^{\dagger 3}$	[*0*]	218/217
palmer1a	*47340/3000*$^{\dagger 1}$	*53268/3000*$^{\dagger 1}$	*53268/3000*$^{\dagger 1}$	*50304/3000*$^{\dagger 1}$	[*0*]	*59/58*$^{\dagger 9}$
palmer1b	25/22	27/22	27/22	23/22	[0]	23/22
palmer1e	53/42	67/42	67/42	60/42	[0]	*77/76*$^{\dagger 9}$
palmer2	25/23	27/23	27/23	26/23	[0]	*65/64*$^{\dagger 9}$
palmer2a	203/153	277/153	277/153	240/153	[0]	*24/23*$^{\dagger 9}$
palmer2b	23/20	27/20	27/20	25/20	[0]	*22/21*$^{\dagger 9}$
palmer2e	51/39	63/39	63/39	57/39	[0]	*77/75*$^{\dagger 9}$
palmer3	*4420/3000*$^{\dagger 1}$	*6292/3000*$^{\dagger 1}$	*6292/3000*$^{\dagger 1}$	*5356/3000*$^{\dagger 1}$	[*0*]	*31/30*$^{\dagger 9}$
palmer3a	180/135	244/135	244/135	212/135	[0]	59/58
palmer3b	18/15	22/15	22/15	20/15	[0]	*22/21*$^{\dagger 9}$
palmer3e	109/74	153/74	153/74	131/74	[0]	105/104
palmer4	60/38	78/38	78/38	55/35	[0]	*51/50*$^{\dagger 9}$
palmer4a	166/124	230/124	230/124	198/124	[0]	*27/26*$^{\dagger 9}$
palmer4b	21/15	23/15	23/15	16/15	[0]	16/15
palmer4e	35/27	41/27	41/27	38/27	[0]	*28/27*$^{\dagger 9}$
palmer5a	*7360/3000*$^{\dagger 1}$	*10198/3000*$^{\dagger 1}$	*10198/3000*$^{\dagger 1}$	*8779/3000*$^{\dagger 1}$	[*0*]	1907/1906
palmer5b	481/237	657/237	657/237	569/237	[0]	200/199
palmer5e	*4453/3000*$^{\dagger 1}$	*6335/3000*$^{\dagger 1}$	*6335/3000*$^{\dagger 1}$	*5394/3000*$^{\dagger 1}$	[*0*]	301/300
palmer6a	427/308	587/308	587/308	507/308	[0]	185/184
palmer6e	43/29	55/29	55/29	49/29	[0]	88/87
palmer7a	*4418/3000*$^{\dagger 1}$	*6302/3000*$^{\dagger 1}$	*6302/3000*$^{\dagger 1}$	*5360/3000*$^{\dagger 1}$	[*0*]	*78/77*$^{\dagger 9}$
palmer7e	*6993/3000*$^{\dagger 1}$	*9659/3000*$^{\dagger 1}$	*9659/3000*$^{\dagger 1}$	*8326/3000*$^{\dagger 1}$	[*0*]	*3001/3000*$^{\dagger 1}$
palmer8a	*64453/3000*$^{\dagger 1}$	*70333/3000*$^{\dagger 1}$	*70333/3000*$^{\dagger 1}$	*67393/3000*$^{\dagger 1}$	[*0*]	73/72
palmer8e	19/16	23/16	23/16	21/16	[0]	24/23
penalty1	44/41	48/41	48/41	46/41	[0]	40/39
penalty2	20/19	20/19	20/19	20/19	[0]	20/19
pentagon	17/16	17/16	17/16	17/16	[0]	17/16
pfit1ls	20/19	20/19	20/19	20/19	[0]	20/19
pfit2ls	21/20	21/20	21/20	21/20	[0]	21/20
pfit3ls	22/21	22/21	22/21	22/21	[0]	22/21
pfit4ls	22/21	22/21	22/21	22/21	[0]	22/21
polak1	23/11	12/10	12/10	12/11	[0]	12/11
polak2	278/28	*7/0*$^{\dagger 2}$	*7/0*$^{\dagger 2}$	38/23	[0]	*54/1*$^{\dagger 2}$
polak3	82/39	98/27	640/65	*2635/200*$^{\dagger 8}$	[*60*]	*16/11*$^{\dagger 9}$
polak4	*45/43*$^{\dagger 5}$	20/11	26/14	41/18	[0]	25/24
polak5	32/31	32/31	32/31	32/31	[0]	32/31
polak6	448/81	*93/8*$^{\dagger 2}$	*247/21*$^{\dagger 2}$	236/95	[5]	*22/21*$^{\dagger 9}$
power	2/1	2/1	2/1	2/1	[0]	2/1
probpenl	421/416	429/416	429/416	425/416	[0]	408/407
prodpl0	20/19	18/17	18/17	20/19	[0]	18/17
prodpl1	*26984/3000*$^{\dagger 1}$	*14992/494*$^{\dagger 2}$	28/26	*6638/1216*$^{\dagger 7}$	[*70*]	27/26
pspdoc	*564/400*$^{\dagger 2}$	25/11	25/11	29/10	[0]	*3001/3000*$^{\dagger 1}$
qr3d	75/53	95/53	95/53	85/53	[0]	202/201
qr3dbd	40/26	54/26	54/26	47/26	[0]	244/243
qr3dls	75/53	95/53	95/53	85/53	[0]	189/188
qrtquad	27/24	32/23	32/23	29/23	[0]	33/32

Table B.1: Comparison of IPOPT's line search options (*continued on next page*)

Problem	auglag	exact1	exact2	filter		fullstep
	#f/#iter	#f/#iter	#f/#iter	#f/#iter	[#Tit]	#f/#iter
quartc	41/40	41/40	41/40	41/40	[0]	41/40
reading1	240/71	*59252/3000*$^{\dagger 1}$	*54875/3000*$^{\dagger 1}$	28/27	[0]	28/27
reading3	*232/33*$^{\dagger 2}$	*186/36*$^{\dagger 2}$	543/111	11011/799	[0]	61/60
rk23	13/12	634/59	577/56	16/14	[0]	18/17
robot	*59/11*$^{\dagger 2}$	*67/11*$^{\dagger 2}$	*67/11*$^{\dagger 2}$	516/141	[11]	*3001/3000*$^{\dagger 1}$
rosenbr	29/21	37/21	37/21	33/21	[0]	7/6
s365mod	17/14	287/54	18/16	37/20	[0]	16/15
s368	149/147	151/147	151/147	150/147	[0]	149/148
sawpath	144/50	843/50	1847/87	*18/17*$^{\dagger 3}$	[*0*]	42/41
scon1dls	1220/370	1774/370	1774/370	1497/370	[0]	*62/61*$^{\dagger 2}$
scosine	133/132	133/132	133/132	133/132	[0]	133/132
scurly10	*3012/3000*$^{\dagger 1}$	*3020/3000*$^{\dagger 1}$	*3020/3000*$^{\dagger 1}$	*3016/3000*$^{\dagger 1}$	[*0*]	*3001/3000*$^{\dagger 1}$
scurly20	*3007/3000*$^{\dagger 1}$	*3011/3000*$^{\dagger 1}$	*3011/3000*$^{\dagger 1}$	*3009/3000*$^{\dagger 1}$	[*0*]	*3001/3000*$^{\dagger 1}$
scurly30	*3003/3000*$^{\dagger 1}$	*3005/3000*$^{\dagger 1}$	*3005/3000*$^{\dagger 1}$	*3004/3000*$^{\dagger 1}$	[*0*]	*3001/3000*$^{\dagger 1}$
sineali	*4363/3000*$^{\dagger 1}$	*6141/3000*$^{\dagger 1}$	*6141/3000*$^{\dagger 1}$	*5252/3000*$^{\dagger 1}$	[*0*]	68/67
sineval	66/42	88/42	88/42	77/42	[0]	3/2
sinquad	24/19	30/19	30/19	27/19	[0]	21/20
sinrosnb	8/7	8/7	8/7	8/7	[0]	8/7
sisser	19/18	19/18	19/18	19/18	[0]	19/18
smbank	18/17	22/18	18/17	18/17	[0]	18/17
smmpsf	*69958/3000*$^{\dagger 1}$	*5157/320*$^{\dagger 2}$	*81410/3000*$^{\dagger 1}$	*2003/379*$^{\dagger 8}$	[*43*]	*3001/3000*$^{\dagger 1}$
snake	18/15	29/8	29/8	14/12	[0]	9/8
spanhyd	76/47	46/45	46/45	47/46	[0]	46/45
spiral	80/67	162/67	149/67	63/59	[0]	59/58
sreadin3	9/8	12/8	12/8	9/8	[0]	9/8
srosenbr	29/21	37/21	37/21	33/21	[0]	7/6
ssebnln*	*282/259*$^{\dagger 2}$	*409/248*$^{\dagger 2}$	607/377	263/205	[14]	*360/359*$^{\dagger 5}$
ssnlbeam	23/22	23/19	23/19	23/22	[0]	20/19
stancmin	12/11	12/11	12/11	12/11	[0]	12/11
steenbrb	60/59	60/59	60/59	60/59	[0]	60/59
steenbrc*	532/511	*599/261*$^{\dagger 2}$	*613/251*$^{\dagger 2}$	1633/1236	[7]	552/551
steenbrd*	124/122	120/116	120/116	247/235	[3]	117/116
steenbre	201/199	269/226	271/226	*2337/2203*$^{\dagger 7}$	[*25*]	215/214
steenbrf*	493/485	*803/139*$^{\dagger 2}$	*132/62*$^{\dagger 2}$	795/472	[5]	565/564
steenbrg	149/148	167/163	167/163	149/146	[0]	168/167
svanberg	31/28	45/28	39/28	30/29	[0]	30/29
swopf	268/47	874/58	449/40	87/30	[1]	29/28
synthes1	14/12	14/12	13/12	13/12	[0]	13/12
trainf	32/31	*497/57*$^{\dagger 2}$	*548/60*$^{\dagger 2}$	32/31	[0]	*3001/3000*$^{\dagger 1}$
trainh	76/75	*164/64*$^{\dagger 2}$	*163/56*$^{\dagger 2}$	76/75	[0]	*3001/3000*$^{\dagger 1}$
trimloss	*29105/3000*$^{\dagger 1}$	*33952/1403*$^{\dagger 2}$	*58861/3000*$^{\dagger 1}$	1005/174	[36]	*3001/3000*$^{\dagger 1}$
try-b	15/13	22/14	22/14	18/14	[0]	15/14
twirism1	*122/91*$^{\dagger 2}$	*56/9*$^{\dagger 2}$	*81/28*$^{\dagger 2}$	*6484/999*$^{\dagger 7}$	[*630*]	*3001/3000*$^{\dagger 1}$
twobars	14/13	20/12	20/12	14/13	[0]	14/13
ubh5	8/7	8/7	8/7	8/7	[0]	8/7
vardim	26/25	26/25	26/25	26/25	[0]	26/25
watson	14/13	14/13	14/13	14/13	[0]	14/13
weeds*	27/24	29/24	29/24	28/24	[0]	56/55
womflet*	20/13	176/28	32/13	13/12	[0]	13/12
woods	59/41	83/41	83/41	71/41	[0]	39/38
yfit*	63/51	77/51	77/51	70/51	[0]	17/16
yfitu*	45/36	55/36	55/36	50/36	[0]	526/525
zecevic3	*11313/3000*$^{\dagger 1}$	*444/26*$^{\dagger 5}$	*319/18*$^{\dagger 2}$	36/26	[4]	*52/51*$^{\dagger 5}$
zecevic4	13/12	15/12	15/12	13/12	[0]	13/12
zigzag	28/27	265/54	29/24	28/27	[0]	28/27
zy2	12/11	15/11	15/11	12/11	[0]	12/11

Table B.1: Comparison of IPOPT's line search options.

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]	$f(x_*)$	$\|\|c(x_*)\|\|$
airport	13	14/14	0.17	0	0/0 [0]	4.79527017E+04	8.2E-12
aljazzaf	29	87/87	0.03	1	0/0 [0]	7.50050000E+01	1.1E-16
allinit	12	18/18	0.01	0	0/0 [0]	1.67059684E+01	0.0E+00
allinitc	39	52/52	0.02	1	0/0 [0]	3.04965452E+01	2.2E-16
allinitu	14	15/0	0.02	9	0/0 [0]	5.74438491E+00	0.0E+00
alsotame	10	11/11	0.02	0	0/0 [0]	8.20850011E-02	2.2E-16
arwhead	6	7/0	0.99	0	0/0 [0]	-2.66453525E-15	0.0E+00
avion2	23	79/79	0.03	16	0/0 [0]	9.46801295E+07	7.9E-12
bard	8	9/0	0.02	3	0/0 [0]	8.21487730E-03	0.0E+00
batch	45	46/46	0.05	0	0/0 [0]	2.59180350E+05	1.7E-09
bdexp	17	18/0	1.53	1	0/0 [0]	1.32368539E-06	0.0E+00
bdqrtic	10	11/0	0.43	0	0/0 [0]	3.98381795E+03	0.0E+00
beale	6	13/0	0.00	1	0/0 [0]	2.24577204E-19	0.0E+00
bigbank	26	27/27	1.19	26	0/0 [0]	-4.20569614E+06	4.2E-10
biggs3	9	23/23	0.01	2	0/0 [0]	1.30661717E-26	0.0E+00
biggs5	20	27/27	0.02	13	0/0 [0]	1.07536634E-19	0.0E+00
biggs6	34	41/0	0.02	19	0/0 [0]	5.98496860E-21	0.0E+00
box2	8	9/9	0.00	3	0/0 [0]	1.36456705E-27	0.0E+00
box3	9	12/0	0.01	1	0/0 [0]	1.69127241E-25	0.0E+00
brainpc0	†1 *3000*	*47610/47610*	*1942.68*	*64*	*0/0* [*0*]	3.43138764E-01	5.7E+02
brainpc1	‡ *0*	*0/0*	*0.00*	*0*	*0/0* [*0*]	--	--
brainpc2	†6 *284*	*497/700*	*3028.74*	*111*	*25/178* [*304270*]	4.12445060E-04	6.4E-10
brainpc3	61	76/76	31.39	37	0/0 [0]	4.13823164E-04	4.9E-08
brainpc4	57	73/73	29.27	30	0/0 [0]	4.38684365E-04	3.1E-06
brainpc5	†8 *129*	*456/1562*	*180.70*	*96*	*38/52* [*9193*]	3.67226284E-04	1.9E-12
brainpc6	†1 *3000*	*47912/47942*	*1834.37*	*77*	*3/24* [*4082*]	2.68997665E-04	5.6E-02
brainpc7	57	92/92	30.00	32	0/0 [0]	3.93138275E-04	1.4E-06
brainpc8	66	118/120	33.02	27	1/1 [79]	3.56039939E-04	8.5E-08
brainpc9	91	189/191	43.36	45	1/1 [1]	4.53505951E-04	1.2E-02
bratu1d	†1 *3000*	*65813/0*	*389.14*	*0*	*0/0* [*0*]	-8.51892727E+00	0.0E+00
britgas	34	55/55	0.54	9	0/0 [0]	6.02538457E-08	2.2E-08
brkmcc	3	4/0	0.00	0	0/0 [0]	1.69042679E-01	0.0E+00
brownal	7	8/0	0.00	0	0/0 [0]	1.49563507E-16	0.0E+00
brownbs	8	11/0	0.00	1	0/0 [0]	1.97215226E-31	0.0E+00
broydn7d*	93	97/0	2.51	87	0/0 [0]	3.45014948E+02	0.0E+00
brybnd	8	9/0	2.86	0	0/0 [0]	1.22478636E-26	0.0E+00
bt1	†4 *15*	*153/161*	*0.01*	*16*	*4/4* [*4*]	-1.00000000E+00	0.0E+00
bt2	12	13/13	0.00	0	0/0 [0]	3.25682003E-02	2.2E-12
bt4	13	14/14	0.01	3	0/0 [0]	-4.55105507E+01	2.5E-09
bt5	7	8/8	0.02	0	0/0 [0]	9.61715172E+02	5.0E-14
bt6	13	18/18	0.01	0	0/0 [0]	2.77044788E-01	5.3E-11
bt7*	14	22/22	0.00	7	0/0 [0]	3.06499999E+02	1.8E-13
bt8	52	53/53	0.02	47	0/0 [0]	1.00000000E+00	4.4E-16
bt9	13	14/14	0.02	1	0/0 [0]	-1.00000000E+00	1.7E-12
bt11	8	9/9	0.01	0	0/0 [0]	8.24891778E-01	1.1E-16
bt12	4	5/5	0.02	0	0/0 [0]	6.18811881E+00	3.2E-13
bt13	23	25/25	0.02	1	0/0 [0]	2.50590355E-09	9.1E-10
byrdsphr	24	25/25	0.00	2	0/0 [0]	-4.68330013E+00	9.4E-09
camel6	13	17/17	0.00	6	0/0 [0]	-1.03162845E+00	0.0E+00
cantilvr	14	15/15	0.01	0	0/0 [0]	1.33995636E+00	6.7E-16
catena	6	7/7	0.01	0	0/0 [0]	-2.30777462E+04	1.0E-10
catenary	46	95/95	3.15	9	0/0 [0]	-3.48403157E+05	3.8E-09
cb2	10	11/11	0.01	0	0/0 [0]	1.95222449E+00	8.4E-15
cb3	9	10/10	0.02	0	0/0 [0]	2.00000000E+00	7.3E-15
chaconn1	8	9/9	0.01	0	0/0 [0]	1.95222449E+00	9.2E-15
chaconn2	7	8/8	0.00	0	0/0 [0]	2.00000000E+00	1.8E-15
chebyqad	78	90/0	69.21	72	0/0 [0]	5.38631531E-03	0.0E+00
chnrosnb	43	63/0	0.03	1	0/0 [0]	1.82240476E-26	0.0E+00
cliff	27	28/0	0.01	0	0/0 [0]	1.99786613E-01	0.0E+00
clnlbeam*	286	288/288	9.05	161	0/0 [0]	3.44876218E+02	4.2E-09
clplatea	6	9/0	1.59	0	0/0 [0]	-1.25920948E-02	0.0E+00
clplateb	6	10/0	1.57	0	0/0 [0]	-6.98822201E+00	0.0E+00

Table B.2: Numerical results of IPOPT on CUTE test set (*continued on next page*)

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]	$f(x_*)$	$\|\|c(x_*)\|\|$
clplatec	2	3/0	0.88	0	0/0 [0]	-5.02072422E-03	0.0E+00
concon	11	12/12	0.00	0	0/0 [0]	-6.23079556E+03	5.8E-10
congigmz	29	36/36	0.02	7	0/0 [0]	2.80000000E+01	3.1E-09
core1	280	524/802	0.37	25	58/147 [1704]	9.10562400E+01	2.0E-06
corkscrw	2099	2532/2829	4212.90	80	82/214 [1408526]	9.06878264E+01	1.3E-09
coshfun	†1 *3000*	*55380/57740*	*6.50*	*2931*	*1178/1182* [*5736*]	-7.27681504E-01	4.6E-06
cosine	10	11/0	2.35	8	0/0 [0]	-9.99900000E+03	0.0E+00
cragglvy	15	16/0	1.76	0	0/0 [0]	1.68821530E+03	0.0E+00
cresc100	†8 *1090*	*5434/6684*	*12.14*	*907*	*23/192* [*3465*]	5.57632039E-01	5.6E-02
cresc132	†1 *3000*	*17645/20077*	*817.26*	*2100*	*369/1258* [*6960*]	2.33108592E+02	1.5E+02
cresc4	†8 *1304*	*14351/15644*	*0.98*	*1141*	*133/140* [*1238*]	1.97926739E+00	2.6E-06
cresc50	†8 *1170*	*8936/9748*	*5.66*	*1043*	*50/132* [*1822*]	7.36572566E-01	9.0E-02
csfi1	21	22/22	0.01	7	0/0 [0]	-4.90752000E+01	6.2E-09
csfi2	32	51/55	0.02	11	2/2 [4]	5.50176056E+01	2.7E-12
cube	27	43/0	0.01	0	0/0 [0]	1.75356784E-24	0.0E+00
curly10	22	23/0	13.25	16	0/0 [0]	-1.00316290E+06	0.0E+00
curly20	21	29/0	24.88	16	0/0 [0]	-1.00316290E+06	0.0E+00
curly30	26	27/0	47.04	20	0/0 [0]	-1.00316290E+06	0.0E+00
dallasl	†7 *181*	*583/535*	*4.94*	*94*	*3/4* [*234*]	curly30	Inf
dallasm	33	56/56	0.16	14	0/0 [0]	-4.81981886E+04	5.2E-10
dallass	†7 *30*	*110/106*	*0.03*	*15*	*3/3* [*45*]	dallasm	Inf
deconvc*	105	158/158	0.42	62	0/0 [0]	5.64682184E-10	9.9E-13
demymalo	13	14/14	0.01	1	0/0 [0]	-2.99999999E+00	2.4E-15
denschna	6	7/0	0.00	0	0/0 [0]	1.10283709E-23	0.0E+00
denschnb	7	22/0	0.01	1	0/0 [0]	9.86076131E-32	0.0E+00
denschnc	10	11/0	0.00	0	0/0 [0]	2.17767937E-20	0.0E+00
denschnd	47	50/0	0.02	9	0/0 [0]	6.97456686E-14	0.0E+00
denschne	10	16/0	0.01	5	0/0 [0]	2.59994751E-20	0.0E+00
denschnf	6	7/0	0.00	0	0/0 [0]	6.51324621E-22	0.0E+00
dipigri	12	22/22	0.01	0	0/0 [0]	6.80630057E+02	1.4E-14
disc2	79	89/91	0.07	60	1/1 [1]	1.56250000E+00	6.3E-10
discs	†8 *1071*	*2176/4449*	*3.09*	*284*	*91/809* [*55652*]	8.50831859E+01	4.0E+00
dittert	31	37/37	3.74	27	0/0 [0]	-1.99759674E+00	6.5E-10
dixchlng	10	11/11	0.01	0	0/0 [0]	2.47189781E+03	2.2E-16
dixchlnv	20	21/21	0.34	0	0/0 [0]	0.00000000E+00	0.0E+00
dixmaana	6	7/0	0.51	2	0/0 [0]	1.00000000E+00	0.0E+00
dixmaanb	7	8/0	0.99	2	0/0 [0]	1.00000000E+00	0.0E+00
dixmaanc	9	10/0	1.16	4	0/0 [0]	1.00000000E+00	0.0E+00
dixmaand	9	10/0	1.11	5	0/0 [0]	1.00000000E+00	0.0E+00
dixmaane	8	9/0	0.68	5	0/0 [0]	1.00000000E+00	0.0E+00
dixmaanf	22	23/0	2.43	17	0/0 [0]	1.00000000E+00	0.0E+00
dixmaang	16	17/0	1.86	12	0/0 [0]	1.00000000E+00	0.0E+00
dixmaanh	22	23/0	2.38	17	0/0 [0]	1.00000000E+00	0.0E+00
dixmaani	11	12/0	0.81	7	0/0 [0]	1.00000000E+00	0.0E+00
dixmaanj	20	21/0	2.27	17	0/0 [0]	1.00000000E+00	0.0E+00
dixmaank	24	31/0	2.77	20	0/0 [0]	1.00000000E+00	0.0E+00
dixmaanl	29	30/0	3.15	26	0/0 [0]	1.00000000E+00	0.0E+00
djtl	†3 *17*	*73/0*	*0.01*	*3*	*0/0* [*0*]	-8.95154472E+03	0.0E+00
dnieper	30	33/33	0.05	2	0/0 [0]	1.87440146E+04	5.0E-08
dqrtic	38	39/0	1.41	0	0/0 [0]	1.06993424E-09	0.0E+00
drcavty1	264	544/544	9668.83	178	0/0 [0]	1.96643634E-04	1.9E-17
drcavty2	‡ *0*	*0/0*	*0.00*	*0*	*0/0* [*0*]	--	--
drcavty3	‡ *0*	*0/0*	*0.00*	*0*	*0/0* [*0*]	--	--
dtoc1l	6	7/7	4.15	0	0/0 [0]	1.25338129E+02	2.6E-14
dtoc1na	6	7/7	2.93	0	0/0 [0]	1.27020299E+01	8.8E-14
dtoc1nb	6	7/7	3.01	0	0/0 [0]	1.59377776E+01	3.3E-16
dtoc1nc	15	20/20	5.87	4	0/0 [0]	2.49698127E+01	1.1E-13
dtoc1nd*	26	35/35	4.67	19	0/0 [0]	1.27515615E+01	2.6E-14
dtoc2	10	17/17	3.71	9	0/0 [0]	5.08676207E-01	2.9E-10
dtoc4	3	4/4	2.13	0	0/0 [0]	2.86853822E+00	1.6E-10
dtoc5	4	5/5	1.30	0	0/0 [0]	1.53511153E+00	2.0E-13
dtoc6	11	12/12	2.42	0	0/0 [0]	1.34850616E+05	1.1E-11

Table B.2: Numerical results of IPOPT on CUTE test set (*continued on next page*)

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]	$f(x_*)$	$\|\|c(x_*)\|\|$
edensch	7	8/0	0.39	0	0/0 [0]	1.20032845E+04	0.0E+00
eg1	10	11/0	0.01	0	0/0 [0]	-1.42930675E+00	0.0E+00
eg2	3	4/0	0.08	1	0/0 [0]	-9.98947393E+02	0.0E+00
eg3*	36	37/37	0.18	8	0/0 [0]	6.71799881E-02	1.7E-11
eigena	28	31/0	0.56	4	0/0 [0]	1.16995304E-07	0.0E+00
eigena2	3	4/4	0.08	1	0/0 [0]	8.68979590E-30	0.0E+00
eigenaco	3	4/4	0.11	1	0/0 [0]	0.00000000E+00	0.0E+00
eigenals	25	28/0	0.78	16	0/0 [0]	2.53328105E-24	0.0E+00
eigenb	116	161/0	2.65	61	0/0 [0]	4.73897786E-20	0.0E+00
eigenb2*	62	73/73	1.42	56	0/0 [0]	5.44579126E-20	1.2E-10
eigenbco*	71	91/91	3.38	54	0/0 [0]	3.96581044E-19	4.0E-11
eigenbls	114	174/0	3.65	56	0/0 [0]	1.20963254E-17	0.0E+00
eigenc2	18	19/19	52.22	12	0/0 [0]	4.11036638E-23	1.2E-12
eigencco	10	11/11	0.02	5	0/0 [0]	8.09393237E-23	1.3E-12
engval1	8	9/0	0.79	0	0/0 [0]	5.54866841E+03	0.0E+00
engval2	20	21/0	0.01	7	0/0 [0]	2.02010021E-28	0.0E+00
errinros*	29	57/0	0.02	6	0/0 [0]	4.04044907E+01	0.0E+00
expfit	8	9/0	0.01	4	0/0 [0]	2.40510593E-01	0.0E+00
expfita	31	33/33	0.01	11	0/0 [0]	1.13662181E-03	2.8E-14
expfitb	75	76/76	0.15	31	0/0 [0]	5.01937570E-03	4.3E-14
expfitc*	156	158/158	1.83	78	0/0 [0]	2.33025810E-02	3.6E-14
explin	21	22/0	0.03	0	0/0 [0]	-7.23756265E+05	0.0E+00
explin2	21	22/0	0.02	0	0/0 [0]	-7.24459142E+05	0.0E+00
expquad	23	24/24	0.04	13	0/0 [0]	-3.62459988E+06	8.3E-25
extrosnb	0	1/0	0.01	0	0/0 [0]	0.00000000E+00	0.0E+00
fletcbv2	2	3/0	0.02	0	0/0 [0]	-5.14006786E-01	0.0E+00
fletcbv3	†1*3000*	*3001/0*	*530.25*	*2999*	*0/0* [*0*]	-6.63200592E+07	0.0E+00
fletchbv	†1*3000*	*3001/0*	*525.27*	*2999*	*0/0* [*0*]	-6.63728505E+15	0.0E+00
fletchcr	48	51/0	0.06	46	0/0 [0]	1.02489796E-16	0.0E+00
fletcher*	20	49/55	0.01	11	1/3 [13]	1.16568542E+01	2.1E-12
flosp2hh	†1*3000*	*3001/0*	*324.27*	*2999*	*0/0* [*0*]	3.88854290E+01	0.0E+00
flosp2hl	3	4/0	0.39	2	0/0 [0]	3.88705439E+01	0.0E+00
flosp2hm	†3*2578*	*2579/0*	*214.05*	*1571*	*0/0* [*0*]	3.88712559E+01	0.0E+00
flosp2th	†1*3000*	*3001/0*	*319.45*	*3000*	*0/0* [*0*]	1.93640772E+01	0.0E+00
flosp2tl	2	3/0	0.26	0	0/0 [0]	1.00000000E+01	0.0E+00
flosp2tm	†3*1016*	*1017/0*	*85.84*	*601*	*0/0* [*0*]	1.00000000E+01	0.0E+00
fminsrf2	17	113/0	0.98	3	0/0 [0]	1.00000000E+00	0.0E+00
fminsurf	33	256/0	632.75	17	0/0 [0]	1.00000000E+00	0.0E+00
freuroth	8	13/0	1.51	1	0/0 [0]	6.08159189E+05	0.0E+00
gausselm	†5*1599*	*1719/1721*	*409.08*	*1487*	*1/1* [*265*]	-1.72755430E+01	1.7E-08
genhumps	194	206/0	0.05	179	0/0 [0]	1.28669686E-29	0.0E+00
genrose	744	953/0	4.25	5	0/0 [0]	1.00000000E+00	0.0E+00
gigomez1	16	19/19	0.01	2	0/0 [0]	-2.99999999E+00	1.1E-15
gilbert	19	20/20	0.35	1	0/0 [0]	4.82027299E+02	2.2E-10
gpp	25	26/26	8.92	3	0/0 [0]	1.44009271E+04	1.5E-10
growth	70	119/0	0.02	9	0/0 [0]	1.00404058E+00	0.0E+00
growthls	74	126/0	0.02	9	0/0 [0]	1.00404058E+00	0.0E+00
gulf	22	32/0	0.02	7	0/0 [0]	3.89353644E-28	0.0E+00
hadamals*	98	99/0	1.58	85	0/0 [0]	2.53164161E+01	0.0E+00
hadamard	9	13/13	0.19	9	0/0 [0]	1.00000016E+00	1.1E-12
hager2	1	2/2	1.09	0	0/0 [0]	4.32082250E-01	2.9E-12
hager4	13	14/14	3.58	0	0/0 [0]	2.79403733E+00	9.5E-12
haifam	31	42/42	0.26	9	0/0 [0]	-4.50003603E+01	1.4E-07
haifas	12	14/14	0.01	0	0/0 [0]	-4.49999992E-01	5.7E-11
hairy	46	57/0	0.01	34	0/0 [0]	2.00000000E+01	0.0E+00
haldmads*	58	65/65	0.07	50	0/0 [0]	3.46592839E-02	9.0E-13
hanging	19	20/20	0.18	0	0/0 [0]	-6.20176046E+02	2.3E-13
hart6	10	13/0	0.01	2	0/0 [0]	-3.32288689E+00	0.0E+00
hatflda	11	12/0	0.01	0	0/0 [0]	7.23718446E-16	0.0E+00
hatfldb	12	13/13	0.01	0	0/0 [0]	5.57281150E-03	0.0E+00
hatfldc	7	8/8	0.00	0	0/0 [0]	2.94596279E-18	0.0E+00
hatfldd	21	31/0	0.01	3	0/0 [0]	6.61511391E-08	0.0E+00

Table B.2: Numerical results of IPOPT on CUTE test set (*continued on next page*)

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]	$f(x_*)$	$\|\|c(x_*)\|\|$
hatflde	26	34/0	0.01	2	0/0 [0]	4.43440070E-07	0.0E+00
heart6ls	828	1108/0	0.22	804	0/0 [0]	1.50485244E-30	0.0E+00
heart8ls	88	120/0	0.04	77	0/0 [0]	2.71111944E-27	0.0E+00
helix	11	14/0	0.02	5	0/0 [0]	8.10653032E-29	0.0E+00
himmelbb	10	16/0	0.02	8	0/0 [0]	1.13629984E-21	0.0E+00
himmelbf	9	12/0	0.02	2	0/0 [0]	3.18571748E+02	0.0E+00
himmelbg	6	11/0	0.00	1	0/0 [0]	3.63299957E-22	0.0E+00
himmelbh	4	21/0	0.02	1	0/0 [0]	-1.00000000E+00	0.0E+00
himmelbi	30	31/31	0.07	14	0/0 [0]	-1.75499999E+03	1.9E-09
himmelbj	†6 *37*	*57/62*	*0.05*	*0*	*2/3* [*20*]	-1.90897461E+03	3.0E-12
himmelbk	19	20/20	0.04	0	0/0 [0]	5.18143882E-02	4.4E-13
himmelp1	11	12/12	0.01	5	0/0 [0]	-6.20538693E+01	0.0E+00
himmelp2*	14	15/15	0.00	3	0/0 [0]	-8.19803173E+00	0.0E+00
himmelp3	13	14/14	0.01	2	0/0 [0]	-5.90131235E+01	9.1E-13
himmelp4	17	18/18	0.01	3	0/0 [0]	-5.90131235E+01	9.1E-13
himmelp5	29	30/30	0.03	2	0/0 [0]	-5.90131235E+01	1.1E-14
himmelp6*	9	10/10	0.01	0	0/0 [0]	-5.90131235E+01	9.1E-13
hong	13	14/14	0.01	0	0/0 [0]	1.34730660E+00	0.0E+00
hs001	30	45/45	0.01	0	0/0 [0]	1.00724019E-18	0.0E+00
hs002	10	11/11	0.02	0	0/0 [0]	4.94122933E+00	0.0E+00
hs004	6	7/7	0.01	0	0/0 [0]	2.66666670E+00	0.0E+00
hs005	10	12/12	0.01	1	0/0 [0]	-1.91322295E+00	2.2E-16
hs006	5	7/7	0.01	2	0/0 [0]	0.00000000E+00	1.8E-15
hs007	9	10/10	0.00	2	0/0 [0]	-1.73205080E+00	2.7E-15
hs009	3	6/6	0.01	1	0/0 [0]	-4.99999999E-01	0.0E+00
hs010	13	14/14	0.00	0	0/0 [0]	-9.99999997E-01	2.2E-16
hs011	8	9/9	0.01	0	0/0 [0]	-8.49846420E+00	1.2E-12
hs012	13	14/14	0.02	0	0/0 [0]	-2.99999999E+01	0.0E+00
hs013*	33	34/34	0.01	0	0/0 [0]	1.00006519E+00	3.5E-14
hs014	9	10/10	0.00	0	0/0 [0]	1.39346498E+00	2.2E-16
hs015*	17	32/32	0.01	5	0/0 [0]	3.06500003E+02	5.8E-13
hs016	10	11/11	0.00	0	0/0 [0]	2.31446609E+01	3.3E-13
hs017	24	30/30	0.00	0	0/0 [0]	1.00000000E+00	1.3E-11
hs018	16	20/20	0.01	2	0/0 [0]	5.00000000E+00	2.8E-14
hs019	11	12/12	0.00	2	0/0 [0]	-6.96181387E+03	1.1E-13
hs020	12	13/13	0.01	0	0/0 [0]	4.01987298E+01	2.9E-14
hs023	10	12/12	0.01	0	0/0 [0]	2.00000000E+00	4.4E-16
hs024	12	14/14	0.00	4	0/0 [0]	-9.99999994E-01	8.9E-16
hs025*	34	46/46	0.04	19	0/0 [0]	8.52763232E-16	0.0E+00
hs026	24	25/25	0.01	0	0/0 [0]	6.53763051E-16	1.4E-08
hs027	75	211/214	0.02	8	1/2 [2]	3.99999999E-02	1.7E-11
hs029	9	10/10	0.01	1	0/0 [0]	-2.26274169E+01	0.0E+00
hs030	32	47/47	0.01	0	0/0 [0]	1.00000000E+00	2.0E-28
hs031	10	11/11	0.01	0	0/0 [0]	6.00000000E+00	2.2E-16
hs032	16	20/20	0.01	0	0/0 [0]	1.00000000E+00	5.3E-14
hs033	11	12/12	0.00	0	0/0 [0]	-4.58578638E+00	6.4E-10
hs034	10	11/11	0.01	0	0/0 [0]	-8.34032437E-01	3.2E-14
hs036	11	12/12	0.01	4	0/0 [0]	-3.29999999E+03	1.4E-14
hs037	10	11/11	0.00	3	0/0 [0]	-3.45599999E+03	0.0E+00
hs038	42	64/0	0.02	3	0/0 [0]	1.45502447E-22	0.0E+00
hs039	13	14/14	0.01	1	0/0 [0]	-1.00000000E+00	1.7E-12
hs040	3	4/4	0.01	0	0/0 [0]	-2.50000000E-01	1.9E-10
hs041	12	14/14	0.01	0	0/0 [0]	1.92592592E+00	1.5E-10
hs042	7	8/8	0.01	0	0/0 [0]	1.38578643E+01	4.4E-16
hs043	10	11/11	0.01	0	0/0 [0]	-4.39999999E+01	8.9E-16
hs045	21	22/0	0.01	11	0/0 [0]	1.00000001E+00	0.0E+00
hs046	18	19/19	0.00	0	0/0 [0]	4.33010833E-15	1.6E-08
hs047	19	21/21	0.01	0	0/0 [0]	6.57516035E-14	1.6E-09
hs049	20	21/21	0.02	0	0/0 [0]	2.09381950E-12	0.0E+00
hs050	9	10/10	0.01	0	0/0 [0]	0.00000000E+00	0.0E+00
hs054	8	9/9	0.01	0	0/0 [0]	1.92857142E-01	0.0E+00
hs056	70	103/103	0.02	54	0/0 [0]	-3.45600000E+00	1.8E-15

Table B.2: Numerical results of IPOPT on CUTE test set (*continued on next page*)

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]	$f(x_*)$	$\|\|c(x_*)\|\|$
hs057	†8 *48*	*195/1151*	*0.06*	*2*	*2/2* [*954*]	3.06487083E-02	3.2E+01
hs059*	46	162/164	0.02	2	1/1 [1]	-7.80278946E+00	1.4E-14
hs060	8	9/9	0.01	0	0/0 [0]	3.25682002E-02	0.0E+00
hs061*	1847	19592/19969	0.51	1576	160/188 [471]	-8.19190960E+01	1.3E-10
hs062	6	8/8	0.00	0	0/0 [0]	-2.62725144E+04	0.0E+00
hs063	9	10/10	0.02	0	0/0 [0]	9.61715172E+02	4.3E-14
hs064	20	31/31	0.01	8	0/0 [0]	6.29984241E+03	3.5E-09
hs065	16	17/17	0.01	4	0/0 [0]	9.53528859E-01	1.4E-14
hs066	8	9/9	0.01	0	0/0 [0]	5.18163279E-01	8.0E-14
hs067	9	10/10	0.01	0	0/0 [0]	-1.16202700E+03	1.7E-07
hs070	20	31/31	0.03	11	0/0 [0]	9.40197325E-03	0.0E+00
hs071	9	10/10	0.01	0	0/0 [0]	1.70140172E+01	7.1E-15
hs072	16	17/17	0.02	0	0/0 [0]	7.27679361E+02	4.5E-13
hs073	9	10/10	0.01	0	0/0 [0]	2.98943782E+01	7.7E-12
hs074	11	12/12	0.00	0	0/0 [0]	5.12649810E+03	2.3E-13
hs075	10	11/11	0.01	0	0/0 [0]	5.17441269E+03	1.6E-12
hs077	11	13/13	0.01	0	0/0 [0]	2.41505128E-01	2.3E-10
hs078	4	5/5	0.01	0	0/0 [0]	-2.91970040E+00	5.6E-12
hs079	4	5/5	0.01	0	0/0 [0]	7.87768209E-02	3.9E-09
hs080	8	9/9	0.01	0	0/0 [0]	5.39498477E-02	1.8E-15
hs081	8	9/9	0.00	0	0/0 [0]	5.39498477E-02	8.9E-16
hs083	17	18/18	0.01	0	0/0 [0]	-3.06655386E+04	1.8E-15
hs084	19	20/20	0.01	12	0/0 [0]	-5.28033513E+06	2.3E-10
hs085	14	15/15	0.03	0	0/0 [0]	-1.90515524E+00	4.7E-10
hs086	11	12/12	0.01	0	0/0 [0]	-3.23486788E+01	5.6E-17
hs087	17	19/19	0.02	0	0/0 [0]	8.82759773E+03	3.3E-13
hs088	16	21/21	0.06	0	0/0 [0]	1.36265681E+00	2.2E-16
hs089	17	22/22	0.10	0	0/0 [0]	1.36265865E+00	7.7E-13
hs090	13	19/19	0.11	4	0/0 [0]	1.36265789E+00	7.2E-10
hs091	34	49/49	0.28	28	0/0 [0]	1.36265682E+00	1.1E-11
hs092	22	23/23	0.21	6	0/0 [0]	1.36265681E+00	8.3E-17
hs093	7	8/8	0.01	0	0/0 [0]	1.35075962E+02	1.1E-12
hs095	14	15/15	0.01	2	0/0 [0]	1.56196375E-02	2.1E-09
hs096	19	27/27	0.01	4	0/0 [0]	1.56196375E-02	2.1E-09
hs097*	18	19/19	0.01	5	0/0 [0]	4.07124637E+00	2.2E-09
hs098*	23	25/25	0.01	3	0/0 [0]	4.07124637E+00	1.3E-09
hs099	6	7/7	0.02	0	0/0 [0]	-8.31079891E+08	2.0E-10
hs100	12	22/22	0.01	0	0/0 [0]	6.80630057E+02	1.4E-14
hs100lnp	6	7/7	0.00	2	0/0 [0]	6.80630057E+02	1.4E-14
hs100mod	11	35/35	0.02	0	0/0 [0]	6.78754727E+02	7.6E-14
hs101	32	100/100	0.04	11	0/0 [0]	1.80976476E+03	5.7E-11
hs102	27	82/82	0.03	5	0/0 [0]	9.11880576E+02	2.8E-10
hs103	27	88/88	0.02	4	0/0 [0]	5.43667958E+02	8.3E-12
hs104	43	53/53	0.01	23	0/0 [0]	3.95116345E+00	1.7E-15
hs105*	26	27/27	1.02	5	0/0 [0]	1.13630730E+03	0.0E+00
hs106	14	15/15	0.01	0	0/0 [0]	7.04924801E+03	7.8E-08
hs107	39	50/55	0.01	5	2/2 [10]	5.05501180E+03	4.5E-13
hs108*	22	25/25	0.02	6	0/0 [0]	-6.74981427E-01	1.0E-13
hs109	17	28/28	0.02	2	0/0 [0]	5.32685133E+03	2.3E-10
hs110	7	8/0	0.01	0	0/0 [0]	-4.57784697E+01	0.0E+00
hs111	23	24/24	0.02	13	0/0 [0]	-4.77610908E+01	9.5E-15
hs111lnp	23	24/24	0.02	14	0/0 [0]	-4.77610914E+01	2.2E-08
hs112	17	18/18	0.01	0	0/0 [0]	-4.77610908E+01	2.2E-16
hs113	12	13/13	0.00	0	0/0 [0]	2.43062090E+01	1.4E-14
hs114	19	20/20	0.02	0	0/0 [0]	-1.76880696E+03	4.5E-13
hs116	23	24/24	0.02	0	0/0 [0]	9.75875095E+01	1.6E-13
hs117	24	49/51	0.02	6	1/1 [9]	3.23486789E+01	1.4E-14
hs119	16	17/17	0.01	0	0/0 [0]	2.44899697E+02	4.4E-16
hs99exp	16	17/17	0.01	2	0/0 [0]	-1.00806250E+09	2.8E-10
hubfit	9	10/10	0.01	0	0/0 [0]	1.68934964E-02	0.0E+00
humps	346	365/0	0.05	338	0/0 [0]	1.18677014E-24	0.0E+00
hvycrash	†7 *741*	*5147/5840*	*9.10*	*519*	*194/409* [*24399*]	-2.18500000E-01	4.6E-12

Table B.2: Numerical results of IPOPT on CUTE test set (*continued on next page*)

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]		$f(x_*)$	$\|\|c(x_*)\|\|$
hypcir	5	8/8	0.01	0	0/0	[0]	0.00000000E+00	6.8E-13
indef	†1*3000*	*3003/0*	*37.95*	*3000*	*0/0*	[*0*]	-8.12220389E+12	0.0E+00
jensmp	10	11/0	0.01	0	0/0	[0]	1.24362182E+02	0.0E+00
kissing	178	242/256	14.38	167	4/7	[36]	8.44457950E-01	1.8E-09
kiwcresc	10	12/12	0.00	1	0/0	[0]	5.01180482E-09	8.9E-16
kowosb	8	17/0	0.03	3	0/0	[0]	3.07505603E-04	0.0E+00
lakes	117	267/353	0.16	34	3/63	[873]	3.50524793E+05	5.8E-11
launch	†8*762*	*1365/2257*	*0.96*	*106*	*65/675*	[*11379*]	1.21534320E+00	2.2E+02
lch	55	56/56	1.72	53	0/0	[0]	-4.31828879E+00	1.7E-12
liarwhd	12	13/0	3.55	0	0/0	[0]	8.19834757E-22	0.0E+00
lminsurf	2	3/3	8.05	0	0/0	[0]	8.99999999E+00	0.0E+00
loadbal	16	19/19	0.02	0	0/0	[0]	4.52851064E-01	3.5E-12
loghairy	809	860/0	0.13	786	0/0	[0]	1.82321556E-01	0.0E+00
logros	†1*3000*	*97243/0*	*0.73*	*2*	*0/0*	[*0*]	0.00000000E+00	0.0E+00
lootsma	11	12/12	0.02	0	0/0	[0]	1.41421361E+00	9.0E-10
lsnnodoc	13	16/16	0.01	13	0/0	[0]	1.23112448E+02	5.3E-14
madsen	22	23/23	0.02	7	0/0	[0]	6.16432440E-01	8.4E-15
madsschj	47	69/69	3.28	15	0/0	[0]	-7.97283702E+02	5.6E-13
makela1	15	16/16	0.01	4	0/0	[0]	-1.41421355E+00	1.6E-15
makela2	9	10/10	0.01	0	0/0	[0]	7.20000000E+00	1.4E-14
makela3	18	19/19	0.00	0	0/0	[0]	5.01180711E-08	1.1E-22
mancino	†3*6*	*7/0*	*1.27*	*0*	*0/0*	[*0*]	8.29262700E-22	0.0E+00
manne	1324	1336/1336	35.10	1298	0/0	[0]	-9.74203446E-01	5.7E-11
maratos	4	5/5	0.00	0	0/0	[0]	-1.00000000E+00	1.8E-15
matrix2	17	18/18	0.01	0	0/0	[0]	3.61019574E-08	7.2E-09
maxlika*	25	30/30	1.00	4	0/0	[0]	1.13630730E+03	0.0E+00
mccormck	9	10/0	9.99	0	0/0	[0]	-4.56616135E+04	0.0E+00
mdhole	48	72/0	0.01	9	0/0	[0]	2.50590355E-09	0.0E+00
methanb8	7	8/0	0.02	3	0/0	[0]	6.51263182E-24	0.0E+00
methanl8	46	74/0	0.10	38	0/0	[0]	6.10062715E-26	0.0E+00
mexhat	4	5/0	0.01	1	0/0	[0]	-4.01000000E-02	0.0E+00
meyer3	†1*3000*	*37049/0*	*0.94*	*7*	*0/0*	[*0*]	8.79458551E+01	0.0E+00
mifflin1	8	9/9	0.01	0	0/0	[0]	-9.99999994E-01	3.9E-17
mifflin2	16	17/17	0.01	3	0/0	[0]	-9.99999994E-01	3.1E-15
minc44	25	27/27	2.15	13	0/0	[0]	2.57302897E-03	6.3E-12
minmaxbd	129	198/202	0.07	51	2/2	[2]	1.15706439E+02	1.7E-13
minmaxrb	11	15/15	0.00	0	0/0	[0]	1.00236130E-08	3.8E-15
minperm	8	9/9	105.88	6	0/0	[0]	3.62880000E-04	2.7E-14
minsurf	3	4/4	0.00	3	0/0	[0]	1.00000000E+00	2.2E-16
mistake	20	21/21	0.01	5	0/0	[0]	-9.99999989E-01	1.6E-12
morebv	1	2/2	0.64	0	0/0	[0]	5.83437924E-15	0.0E+00
msqrtals	24	34/0	691.11	19	0/0	[0]	4.22369695E-16	0.0E+00
msqrtbls	28	45/0	789.91	22	0/0	[0]	8.02930283E-19	0.0E+00
mwright	9	10/10	0.01	3	0/0	[0]	2.49788095E+01	4.2E-10
ngone*	49	50/50	3.00	28	0/0	[0]	-6.35969004E-01	7.7E-11
noncvxu2	404	466/0	828.96	401	0/0	[0]	2.31921213E+03	0.0E+00
noncvxun	30	38/0	0.33	25	0/0	[0]	2.31680841E+03	0.0E+00
nondia	5	6/0	2.30	0	0/0	[0]	4.76317222E-25	0.0E+00
nondquar	24	25/0	4.33	0	0/0	[0]	1.24495101E-13	0.0E+00
nonmsqrt	†1*3000*	*93030/0*	*1.09*	*2758*	*0/0*	[*0*]	7.51800518E-01	0.0E+00
nonscomp	23	46/0	4.45	0	0/0	[0]	5.56279901E-06	0.0E+00
odfits	12	13/13	0.00	0	0/0	[0]	-2.38002677E+03	2.8E-14
oet2	128	150/157	11.80	36	3/3	[7]	8.71596414E-02	1.3E-14
oet7*	129	147/150	6.46	106	1/1	[10]	4.45540319E-05	1.2E-09
optcdeg2	31	32/32	0.45	0	0/0	[0]	2.29573418E+02	6.4E-09
optcdeg3	28	29/29	0.45	0	0/0	[0]	4.61456698E+01	2.7E-09
optcntrl	50	78/78	0.03	0	0/0	[0]	5.49999999E+02	4.0E-11
optctrl3	90	962/1058	0.43	0	21/51	[1727]	2.04801654E+03	4.5E-13
optctrl6	90	962/1058	0.42	0	21/51	[1727]	2.04801654E+03	4.5E-13
optmass	22	23/23	0.05	11	0/0	[0]	-1.89542472E-01	4.6E-11
optprloc	20	21/21	0.03	0	0/0	[0]	-1.64197737E+01	1.1E-06
orthrdm2	6	8/8	1.52	0	0/0	[0]	1.55532815E+02	1.4E-13

Table B.2: Numerical results of IPOPT on CUTE test set (*continued on next page*)

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]		$f(x_*)$	$\|\|c(x_*)\|\|$
orthrds2	143	1250/1320	1.54	131	26/31	[260]	5.27758427E+02	3.4E-13
orthrega*	49	71/71	1.18	39	0/0	[0]	1.41405588E+03	1.1E-15
orthregb	2	3/3	0.01	2	0/0	[0]	4.52460763E-20	2.0E-10
orthregc	14	28/28	14.76	8	0/0	[0]	1.89597674E+02	6.7E-13
orthregd	81	446/448	58.83	42	1/1	[7]	2.60665123E+03	1.4E-13
orthrege*	41	57/57	0.02	13	0/0	[0]	1.28604709E+00	2.8E-09
orthrgdm	†6 *71*	*782/832*	*94.80*	*70*	*24/25*	[*174*]	3.63666639E+04	1.2E-08
orthrgds	16	24/24	11.13	7	0/0	[0]	1.52389973E+03	1.0E-10
osbornea	35	56/0	0.03	7	0/0	[0]	5.46489469E-05	0.0E+00
osborneb*	17	20/0	0.05	7	0/0	[0]	4.01377362E-02	0.0E+00
oslbqp	14	15/15	0.00	0	0/0	[0]	6.25000002E+00	1.4E-20
palmer1	†3 *875*	*1518/0*	*0.23*	*5*	*0/0*	[*0*]	1.17546025E+04	0.0E+00
palmer1a	†1 *3000*	*50304/0*	*1.49*	*15*	*0/0*	[*0*]	8.98836290E-02	0.0E+00
palmer1b	22	23/0	0.02	3	0/0	[0]	3.44735461E+00	0.0E+00
palmer1e	42	60/0	0.03	14	0/0	[0]	8.35268268E-04	0.0E+00
palmer2	23	26/0	0.02	21	0/0	[0]	3.65108950E+03	0.0E+00
palmer2a	153	240/0	0.06	25	0/0	[0]	1.71607394E-02	0.0E+00
palmer2b	20	25/0	0.01	7	0/0	[0]	6.23394652E-01	0.0E+00
palmer2e	39	57/0	0.02	12	0/0	[0]	2.15352481E-04	0.0E+00
palmer3	†1 *3000*	*5356/0*	*0.72*	*9*	*0/0*	[*0*]	2.26595849E+03	0.0E+00
palmer3a	135	212/0	0.04	9	0/0	[0]	2.04314229E-02	0.0E+00
palmer3b	15	20/0	0.01	5	0/0	[0]	4.22764725E+00	0.0E+00
palmer3e	74	131/0	0.02	4	0/0	[0]	5.07408418E-05	0.0E+00
palmer4*	35	55/0	0.01	20	0/0	[0]	2.42401641E+03	0.0E+00
palmer4a	124	198/0	0.04	13	0/0	[0]	4.06061393E-02	0.0E+00
palmer4b	15	16/0	0.01	6	0/0	[0]	6.83513859E+00	0.0E+00
palmer4e	27	38/0	0.01	12	0/0	[0]	1.48004219E-04	0.0E+00
palmer5a	†1 *3000*	*8779/0*	*0.73*	*1610*	*0/0*	[*0*]	3.90202187E-02	0.0E+00
palmer5b	237	569/0	0.05	89	0/0	[0]	9.75249263E-03	0.0E+00
palmer5e	†1 *3000*	*5394/0*	*0.67*	*6*	*0/0*	[*0*]	2.09737858E-02	0.0E+00
palmer6a	308	507/0	0.06	10	0/0	[0]	5.59488389E-02	0.0E+00
palmer6e	29	49/0	0.01	6	0/0	[0]	2.23955033E-04	0.0E+00
palmer7a	†1 *3000*	*5360/0*	*0.61*	*13*	*0/0*	[*0*]	1.03348583E+01	0.0E+00
palmer7e	†1 *3000*	*8326/0*	*0.79*	*1856*	*0/0*	[*0*]	6.57509547E+00	0.0E+00
palmer8a	†1 *3000*	*67393/0*	*1.03*	*16*	*0/0*	[*0*]	7.40096979E-02	0.0E+00
palmer8e	16	21/0	0.01	4	0/0	[0]	6.33930743E-03	0.0E+00
penalty1	41	46/0	400.92	0	0/0	[0]	9.68617543E-03	0.0E+00
penalty2	19	20/0	0.18	0	0/0	[0]	9.70960839E+04	0.0E+00
pentagon	16	17/17	0.00	8	0/0	[0]	1.36532463E-04	1.1E-16
pfit1ls*	19	20/0	0.02	0	0/0	[0]	9.46567901E+02	0.0E+00
pfit2ls*	20	21/0	0.01	0	0/0	[0]	9.42143209E+03	0.0E+00
pfit3ls*	21	22/0	0.01	0	0/0	[0]	3.97340795E+04	0.0E+00
pfit4ls*	21	22/0	0.01	0	0/0	[0]	1.13934428E+05	0.0E+00
polak1	11	12/12	0.01	0	0/0	[0]	2.71828183E+00	1.5E-16
polak2	23	38/38	0.01	0	0/0	[0]	5.45981500E+01	6.1E-15
polak3	†8 *200*	*1491/2635*	*0.45*	*187*	*48/60*	[*1621*]	9.44865066E+01	6.8E-14
polak4	18	41/41	0.01	5	0/0	[0]	7.51800808E-09	1.5E-11
polak5	31	32/32	0.01	1	0/0	[0]	5.00000000E+01	8.4E-09
polak6	95	226/236	0.02	30	5/5	[10]	-4.39999999E+01	1.4E-14
power	1	2/0	0.05	0	0/0	[0]	3.55601741E-24	0.0E+00
probpenl	416	425/0	1282.77	411	0/0	[0]	-4.23825959E+03	0.0E+00
prodpl0	19	20/20	0.01	3	0/0	[0]	6.09192371E+01	9.7E-09
prodpl1	†7 *1216*	*6503/6638*	*0.93*	*8*	*65/70*	[*2491*]	5.30370216E+01	7.4E-13
pspdoc	10	29/29	0.00	0	0/0	[0]	2.41421356E+00	0.0E+00
qr3d	53	85/0	1.74	36	0/0	[0]	1.01655251E-16	0.0E+00
qr3dbd	26	47/0	0.46	12	0/0	[0]	1.01655250E-16	0.0E+00
qr3dls	53	85/0	2.00	36	0/0	[0]	1.01655248E-16	0.0E+00
qrtquad	23	29/29	0.03	12	0/0	[0]	-3.64808836E+06	0.0E+00
quartc	40	41/0	2.96	0	0/0	[0]	1.33691192E-09	0.0E+00
reading1	27	28/28	7.41	0	0/0	[0]	-1.60457843E-01	2.5E-12
reading3	799	11011/11011	11.01	3	0/0	[0]	-6.01814883E-34	8.4E-11
rk23	14	16/16	0.01	14	0/0	[0]	8.33333458E-02	1.9E-13

Table B.2: Numerical results of IPOPT on CUTE test set (*continued on next page*)

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]		$f(x_*)$	$\|\|c(x_*)\|\|$
robot*	141	495/516	0.07	129	8/11	[27]	5.46284122E+00	1.3E-10
rosenbr	21	33/0	0.02	0	0/0	[0]	3.74397564E-21	0.0E+00
s365mod*	20	37/37	0.01	9	0/0	[0]	5.21890765E+01	1.1E-10
s368*	147	150/0	17.87	137	0/0	[0]	-7.96874997E+01	0.0E+00
sawpath	†3 *17*	*18/18*	*0.43*	*4*	*0/0*	[*0*]	1.81572992E+02	1.8E-14
scon1dls*	370	1497/0	10.20	117	0/0	[0]	1.25660022E-11	0.0E+00
scosine	132	133/0	23.98	128	0/0	[0]	-9.99748214E+03	0.0E+00
scurly10	†1 *3000*	*3016/0*	*1039.66*	*221*	*0/0*	[*0*]	-1.00316290E+06	0.0E+00
scurly20	†1 *3000*	*3009/0*	*1886.71*	*199*	*0/0*	[*0*]	-1.00316290E+06	0.0E+00
scurly30	†1 *3000*	*3004/0*	*2969.55*	*203*	*0/0*	[*0*]	-1.00316290E+06	0.0E+00
sineali	†1 *3000*	*5252/0*	*0.77*	*13*	*0/0*	[*0*]	-1.90096278E+03	0.0E+00
sineval	42	77/0	0.01	0	0/0	[0]	2.83150856E-41	0.0E+00
sinquad	19	27/0	6.94	7	0/0	[0]	5.81839093E-10	0.0E+00
sinrosnb	7	8/8	0.31	6	0/0	[0]	-9.99010000E+04	2.6E-16
sisser	18	19/0	0.01	2	0/0	[0]	1.21113886E-12	0.0E+00
smbank	17	18/18	0.05	8	0/0	[0]	-7.12929200E+06	1.2E-10
smmpsf	†8 *379*	*923/2003*	*5.56*	*204*	*12/43*	[*15584*]	1.04698738E+06	7.5E-08
snake	12	14/14	0.01	4	0/0	[0]	3.68984008E-06	8.9E-12
spanhyd	46	47/47	0.18	46	0/0	[0]	2.39738000E+02	1.3E-09
spiral	59	63/63	0.00	12	0/0	[0]	5.01180711E-09	7.2E-22
sreadin3	8	9/9	3.53	0	0/0	[0]	-6.45294901E-05	1.1E-08
srosenbr	21	33/0	2.52	0	0/0	[0]	1.87198782E-17	0.0E+00
ssebnln*	205	238/263	0.58	169	11/14	[925]	1.80474835E+07	5.5E-08
ssnlbeam*	22	23/23	0.01	18	0/0	[0]	3.41877235E+02	9.9E-12
stancmin	11	12/12	0.01	0	0/0	[0]	4.25000000E+00	2.2E-16
steenbrb*	59	60/60	13.21	59	0/0	[0]	9.07585537E+03	5.3E-13
steenbrc*	1236	1622/1633	8.07	1201	4/7	[7]	2.03575576E+04	4.5E-12
steenbrd*	235	243/247	55.57	233	1/3	[3]	9.43721949E+03	1.6E-10
steenbre	†7 *2203*	*2291/2337*	*1033.92*	*2090*	*21/25*	[*392*]	1.34864549E+10	3.7E-13
steenbrf	472	787/795	2.72	448	3/5	[21]	2.82679557E+02	7.3E-12
steenbrg*	146	149/149	87.61	146	0/0	[0]	2.74209296E+04	3.5E-12
svanberg	29	30/30	16.22	9	0/0	[0]	8.36142276E+03	1.0E-08
swopf	30	85/87	0.05	0	1/1	[154]	6.78601914E-02	3.2E-14
synthes1	12	13/13	0.01	0	0/0	[0]	7.59284421E-01	5.7E-09
trainf*	31	32/32	13.58	0	0/0	[0]	3.10340933E+00	2.3E-13
trainh	75	76/76	38.68	0	0/0	[0]	1.23119978E+01	2.0E-10
trimloss	174	955/1005	0.99	0	14/36	[5776]	9.06000027E+00	1.1E-08
try	14	18/18	0.00	4	0/0	[0]	1.56988806E-18	0.0E+00
twirism1	†7 *999*	*5794/6484*	*193.48*	*423*	*53/630*	[*150137*]	-5.28215719E-01	4.2E-13
twobars	13	14/14	0.01	0	0/0	[0]	1.50865242E+00	2.2E-16
ubh5	7	8/8	4.49	0	0/0	[0]	1.11600081E+00	4.7E-12
vardim	25	26/0	0.20	0	0/0	[0]	0.00000000E+00	0.0E+00
watson	13	14/0	0.07	13	0/0	[0]	1.01769229E-13	0.0E+00
weeds*	24	28/0	0.02	9	0/0	[0]	9.20543519E+03	0.0E+00
womflet*	12	13/13	0.01	0	0/0	[0]	6.05000000E+00	4.4E-16
woods	41	71/0	5.78	3	0/0	[0]	0.00000000E+00	0.0E+00
yfit	51	70/0	0.02	1	0/0	[0]	6.66973750E-13	0.0E+00
yfitu	36	50/0	0.02	1	0/0	[0]	6.66972064E-13	0.0E+00
zecevic3	26	30/36	0.02	7	1/4	[11]	9.73094501E+01	1.1E-12
zecevic4	12	13/13	0.02	0	0/0	[0]	7.55750777E+00	3.2E-17
zigzag	27	28/28	0.03	5	0/0	[0]	3.16173499E+00	6.4E-09
zy2	11	12/12	0.02	5	0/0	[0]	2.00000005E+00	3.8E-10

Table B.2: Numerical results of IPOPT on CUTE test set

Problem	#iter	$\#f/\#c$	CPU [s]	#reg	TRON #call/#it [#cgit]		$f(x_*)$	$\|\|c(x_*)\|\|$
bearing_200	16	17/0	79.88	0	0/0	[0]	-1.54793459E-01	0.0E+00
bearing_400	16	17/0	979.31	0	0/0	[0]	-1.54663515E-01	0.0E+00
camshape_10000	†6 *33*	*51/65*	*1495.00*	*0*	*4/10*	[*303076*]	-4.71483165E+00	7.5E-04
camshape_20000	67	89/95	346.70	0	3/3	[22132]	-4.15661282E+00	1.7E-12
catmix_10000	14	15/15	16.01	0	0/0	[0]	-4.80461940E-02	1.9E-11
catmix_20000	13	14/14	32.52	0	0/0	[0]	-4.80372208E-02	1.2E-10
chain_20000	7	8/8	30.94	1	0/0	[0]	5.06848022E+00	5.5E-10
chain_40000	7	8/8	108.49	1	0/0	[0]	5.06848013E+00	7.4E-09
channel_5000	3	4/4	18.87	0	0/0	[0]	1.00000000E+00	2.7E-09
channel_10000	3	4/4	39.65	0	0/0	[0]	1.00000000E+00	2.7E-09
elec_200	147	229/229	1060.64	140	0/0	[0]	1.84390170E+04	1.7E-12
elec_400	115	120/120	7300.95	109	0/0	[0]	7.55830970E+04	8.4E-11
gasoil_2500	15	36/36	74.84	0	0/0	[0]	5.23659583E-03	2.4E-09
gasoil_5000	19	88/88	306.72	0	0/0	[0]	5.23659583E-03	1.1E-14
glider_2500	†1 *3000*	*3479/5962*	*9372.19*	*597*	*40/2443*	[*811328*]	-1.18203726E+02	2.2E-02
glider_5000	‡ *0*	*0/0*	*0.00*	*0*	*0/0*	[*0*]	--	--
marine_1000	25	93/93	79.78	25	0/0	[0]	1.97465297E+07	2.9E-06
marine_2000	‡ *0*	*0/0*	*0.00*	*0*	*0/0*	[*0*]	--	--
methanol_5000	17	24/24	523.28	1	0/0	[0]	9.02229235E-03	1.9E-12
methanol_10000	12	13/13	1995.36	3	0/0	[0]	9.02229235E-03	8.9E-11
minsurf_200_200	117	455/0	1150.95	0	0/0	[0]	2.48548796E+00	0.0E+00
minsurf_300_300	205	748/0	7779.50	0	0/0	[0]	2.48436567E+00	0.0E+00
pinene_2500	‡ *0*	*0/0*	*0.00*	*0*	*0/0*	[*0*]	--	--
pinene_5000	66	188/194	898.80	3	3/3	[236]	1.98721669E+01	1.3E-13
polygon_200*	538	1437/1437	2838.46	528	0/0	[0]	-6.74980929E-01	8.9E-15
polygon_400	‡ *0*	*0/0*	*0.00*	*0*	*0/0*	[*0*]	--	--
robot_5000	‡ *0*	*0/0*	*0.00*	*0*	*0/0*	[*0*]	--	--
robot_10000*	51	59/59	1290.81	1	0/0	[0]	9.14098006E+00	6.4E-10
rocket_10000	48	70/70	252.44	1	0/0	[0]	-1.01280208E+00	5.6E-13
rocket_20000	49	61/61	803.63	1	0/0	[0]	-1.01276715E+00	3.7E-12
steering_10000	24	31/31	329.52	0	0/0	[0]	5.54570880E-01	6.1E-12
steering_20000	25	32/32	1295.33	0	0/0	[0]	5.54570878E-01	7.1E-15
torsion_200_200	14	15/0	74.74	0	0/0	[0]	-4.18436221E-01	0.0E+00
torsion_400_400	12	13/0	735.23	0	0/0	[0]	-4.18337120E-01	0.0E+00

Table B.3: Numerical results of IPOPT on COPS test set

Problem	#iter	#f/#c	CPU [s]	#reg	TRON #call/#it [#cgit]	$f(x_*)$	$\|\|c(x_*)\|\|$
cont5_1	17	18/18	1971.91	0	0/0 [0]	2.72155426E+00	1.2E-09
cont5_2_1	39	40/40	5165.02	1	0/0 [0]	6.57056636E-04	1.5E-11
cont5_2_2	47	48/48	6334.32	1	0/0 [0]	5.15824099E-04	8.0E-11
cont5_2_3	54	55/55	7180.00	1	0/0 [0]	5.15987516E-04	2.5E-11
cont5_2_4	13	14/14	1543.52	0	0/0 [0]	6.63714454E-02	3.8E-09
cont_p	21	22/22	80.57	1	0/0 [0]	2.31633886E+00	3.0E-10
ex1_80	13	14/14	15.43	0	0/0 [0]	6.10533233E-02	1.9E-12
ex1_160	10	11/11	248.09	0	0/0 [0]	6.38981163E-02	5.8E-10
ex2_80	17	18/18	14.64	0	0/0 [0]	5.53589470E-02	4.0E-13
ex2_160	15	16/16	290.42	0	0/0 [0]	5.81909441E-02	5.5E-14
ex3_80	15	16/16	14.04	0	0/0 [0]	1.10265620E-01	1.3E-12
ex3_160	14	15/15	270.86	0	0/0 [0]	1.10294268E-01	8.2E-12
ex4_80	14	15/15	13.37	0	0/0 [0]	7.78944721E-02	1.5E-12
ex4_160	11	12/12	212.80	0	0/0 [0]	7.83757517E-02	1.5E-09
ex4_2_80	†7 *590*	*4319/4475*	*5540.36*	*0*	*51/105*	[*702214*] 3.66772545E+00	3.9E-12
ex4_2_160	25	26/26	946.18	0	0/0 [0]	3.64611612E+00	2.4E-12
ex5_80	40	41/41	33.60	0	0/0 [0]	5.43472035E-02	1.3E-11
ex5_160	51	52/52	958.76	0	0/0 [0]	5.47653343E-02	3.7E-12
ex6_80	17	18/18	20.43	0	0/0 [0]	-4.25501059E+00	2.5E-10
ex6_160	20	21/21	439.63	0	0/0 [0]	-4.30667254E+00	1.1E-10
lukvle1	6	7/7	27.31	0	0/0 [0]	6.23245863E+00	8.4E-12
lukvle2	12	13/13	84.67	1	0/0 [0]	1.40923925E+06	5.6E-16
lukvle3	9	10/10	9.70	0	0/0 [0]	6.51214956E+01	8.9E-16
lukvle4	16	17/17	43.27	1	0/0 [0]	2.42907675E+05	3.8E-14
lukvle5	18	22/22	71.63	7	0/0 [0]	2.63928370E+00	4.4E-15
lukvle6	15	16/16	69.95	1	0/0 [0]	3.14422608E+06	4.4E-16
lukvle7	21	22/22	21.08	15	0/0 [0]	-6.61396154E+04	3.2E-13
lukvle8*	19	20/20	46.86	1	0/0 [0]	4.13006909E+06	7.5E-10
lukvle9	†8 *71*	*187/1225*	*222.70*	*29*	*6/11*	[*1109*] 1.08543084E+11	1.3E+00
lukvle10	10	11/11	23.71	2	0/0 [0]	1.76772385E+04	6.0E-10
lukvle11	8	9/9	13.82	0	0/0 [0]	4.41883326E-24	7.0E-12
lukvle12	6	7/7	11.22	0	0/0 [0]	7.72038783E+04	1.2E-09
lukvle13	22	23/23	35.42	9	0/0 [0]	4.01784151E+05	4.3E-14
lukvle14	26	27/27	28.59	0	0/0 [0]	3.80424848E+05	3.6E-15
lukvle15	40	52/52	73.77	30	0/0 [0]	6.18819459E-24	1.1E-11
lukvle16	17	27/27	30.71	9	0/0 [0]	6.42286851E-27	1.9E-14
lukvle17	8	9/9	11.67	0	0/0 [0]	7.14331493E+04	8.4E-12
lukvle18	10	11/11	16.79	2	0/0 [0]	5.99820079E+04	2.5E-09
lukvli1	†1 *3000*	*3752/3752*	*8720.08*	*8*	*0/0*	[*0*] 4.72949856E+02	2.8E-03
lukvli2	30	31/31	166.32	6	0/0 [0]	1.32665589E+06	5.9E-11
lukvli3	13	14/14	12.88	0	0/0 [0]	1.15775416E+01	1.5E-12
lukvli4	22	24/24	61.40	0	0/0 [0]	2.01020546E+04	1.9E-13
lukvli5*	37	47/47	128.75	3	0/0 [0]	4.89055906E-01	9.1E-12
lukvli6	20	21/21	77.41	0	0/0 [0]	3.14422608E+06	4.4E-16
lukvli7	33	40/40	29.42	18	0/0 [0]	-1.86338502E+04	3.6E-15
lukvli8	55	56/56	142.90	7	0/0 [0]	4.13048842E+06	4.5E-14
lukvli9	80	169/168	52.56	47	0/0 [0]	4.99466708E+03	1.8E-14
lukvli10	57	82/82	154.44	36	0/0 [0]	1.76795871E+04	4.2E-15
lukvli11	89	102/102	191.08	55	0/0 [0]	2.07964602E-05	3.4E-10
lukvli12	45	55/55	81.89	9	0/0 [0]	1.68010734E-09	8.1E-09
lukvli13	31	32/32	51.76	4	0/0 [0]	1.04302607E-05	3.7E-09
lukvli14	43	139/139	83.41	10	0/0 [0]	3.80424849E+05	2.5E-12
lukvli15*	86	87/87	182.77	48	0/0 [0]	4.69917742E-05	3.4E-09
lukvli16	26	27/27	36.05	0	0/0 [0]	4.69972613E-05	4.0E-09
lukvli17*	56	77/77	105.04	28	0/0 [0]	7.77008039E-01	9.3E-09
lukvli18	21	24/24	30.96	0	0/0 [0]	3.13339934E-05	8.2E-09

Table B.4: Numerical results of IPOPT on MITT test set

Code	Explanation
‡	The CPU time limit of 10,800 CPU seconds was exceeded.
†1	More than 3,000 iterations were taken.
†2	The trial step size $\alpha_{k,l}$ became smaller than 10^{-14}.
†3	The size of the search direction $\|d_k\|$ is less than $10 \cdot$ `macheps` times $\|x_k\|$ (only checked in filter option).
†4	The constraint violation is small (see (3.29)) and the switching condition (3.30) is satisfied, but $\nabla\varphi_\mu(x_k)^T d_k$ is positive, probably due to a poor solution to a very ill-conditioned linear system (3.3).
†5	The IEEE numbers `NaN` or `Inf` occurred in the KKT matrix (3.3).
†6	TRON converges to a point satisfying the convergence criterion for the feasibility problem (3.36) and $\|c(x_k)\|$ is small, but the point is not accepted by filter method.
†7	The restoration phase with TRON is invoked at an (almost) feasible point.
†8	Convergence problems within TRON (some trial step size in TRON becomes zero).
†9	The regularization parameter δ_1 in (3.6) exceeded 10^{20}.

Table B.5: Error codes for IPOPT

Appendix C

Results for KNITRO

The tables in this appendix document the runs of KNITRO on the test sets CUTE, COPS, and MITT for the comparison in Section 5.1.3. For each problem, they list the following numbers: number of iterations (#iter) and error code (as defined in Table D.4), if failed; number of evaluations of objective function ($\#f$) and Hessian of the Lagrangian ($\#H$; this corresponds to the number of successful iterations); required CPU time in seconds; final value of the objective function ($f(x_*)$) and (unscaled) constraint violation ($\|c(x_*)\|$). A problem name is marked with a star (*), if IPOPT or LOQO successfully terminated at a point with a different value of the objective function (see Eq. (5.3)).

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|c(x_*)\|$
airport	13	14/13	0.14	4.79527019E+04	2.4E-13
aljazzaf	146	147/129	0.01	7.50050002E+01	6.7E-13
allinit	†1*3000*	*3001/1394*	*0.35*	1.14115009E+01	1.8E+00
allinitc	†1*3000*	*3001/1394*	*0.37*	1.13135798E+01	1.7E+00
allinitu	7	8/6	0.01	5.74438491E+00	0.0E+00
alsotame	15	16/15	0.00	8.20850006E-02	1.1E-15
arwhead	6	7/6	0.49	0.00000000E+00	0.0E+00
avion2	13	14/13	0.01	9.46801293E+07	1.6E-06
bard	11	12/10	0.00	8.21487730E-03	0.0E+00
batch	308	309/172	0.25	2.59180350E+05	9.6E-09
bdexp	20	21/20	1.18	1.16133803E-06	0.0E+00
bdqrtic	†2*45*	*46/16*	*0.67*	3.98381795E+03	0.0E+00
beale	8	9/8	0.01	2.56379794E-30	0.0E+00
bigbank	655	656/655	112.59	-4.20569614E+06	1.4E-11
biggs3	10	12/9	0.01	6.98691272E-30	0.0E+00
biggs5	52	77/28	0.01	2.14274239E-18	0.0E+00
biggs6	45	46/26	0.01	2.59629369E-16	0.0E+00
box2	7	8/7	0.00	6.04232135E-25	0.0E+00
box3	8	9/8	0.00	8.91962715E-21	0.0E+00
brainpc0	2008	2016/1023	1401.73	1.73900973E-03	7.9E-09
brainpc1	415	425/301	392.28	4.07353194E-04	3.4E-06
brainpc2	69	79/44	226.33	4.13958062E-04	2.1E-06
brainpc3	176	177/113	186.06	3.65881954E-04	3.6E-07
brainpc4	94	104/70	74.91	3.51823338E-04	3.0E-06
brainpc5	199	200/128	217.17	3.33646490E-04	1.7E-06
brainpc6	511	523/305	405.17	3.47144665E-04	4.4E-06
brainpc7	732	739/427	565.06	3.52375370E-04	4.2E-07
brainpc8	177	179/107	257.01	3.54358389E-04	1.0E-07
brainpc9	753	763/466	589.83	3.51480779E-04	4.0E-07
bratu1d	†2*64*	*65/16*	*1.64*	-8.51892727E+00	0.0E+00
britgas	48	75/19	0.68	1.46509978E-06	1.5E-06
brkmcc	3	4/3	0.00	1.69042679E-01	0.0E+00
brownal	8	9/8	0.00	5.16009695E-26	0.0E+00
brownbs	1890	1891/1887	0.13	0.00000000E+00	0.0E+00
broydn7d	†2*205*	*206/87*	*2.09*	3.83895194E+02	0.0E+00
brybnd	11	12/11	2.57	4.93657750E-22	0.0E+00
bt1	6	7/6	0.00	-9.99999999E-01	1.3E-15
bt2	13	14/13	0.01	3.25682003E-02	1.7E-15
bt4	6	7/6	0.00	-4.55105507E+01	1.7E-08
bt5	6	7/6	0.00	9.61715172E+02	7.4E-11
bt6	10	12/9	0.01	2.77044788E-01	1.3E-11
bt7*	30	31/22	0.00	3.06499999E+02	2.1E-15
bt8	14	15/14	0.00	1.00000000E+00	3.7E-09
bt9	15	16/12	0.00	-1.00000000E+00	4.6E-14
bt11	7	8/7	0.01	8.24891778E-01	8.0E-12
bt12	6	8/6	0.00	6.18811881E+00	4.5E-13
bt13	53	54/43	0.01	2.04800033E-09	5.6E-09
byrdsphr	8	9/7	0.00	-4.68330013E+00	3.5E-09
camel6	†2*36*	*59/15*	*0.00*	-1.03162845E+00	3.3E-16
cantilvr	21	22/21	0.01	1.33995636E+00	8.8E-16
catena	68	70/45	0.02	-2.30777462E+04	3.3E-12
catenary	†1*3000*	*3003/2959*	*11.66*	-7.27478712E+08	2.6E+07
cb2	24	25/24	0.01	1.95222451E+00	1.4E-14
cb3	17	18/17	0.00	2.00000003E+00	4.5E-15
chaconn1	23	24/23	0.01	1.95222451E+00	1.3E-14
chaconn2	17	18/17	0.00	2.00000003E+00	4.1E-15
chebyqad	473	717/230	206.30	5.38656016E-03	1.1E-16
chnrosnb	74	75/44	0.02	1.50605650E-19	0.0E+00
cliff	†2*48*	*49/29*	*0.00*	1.99786613E-01	0.0E+00
clnlbeam*	15	16/15	1.24	3.54561508E+02	3.6E-08
clplatea	15	16/9	2.65	-1.25920948E-02	0.0E+00
clplateb	59	60/35	7.65	-6.98822201E+00	0.0E+00
clplatec	5	6/5	18.69	-5.02072422E-03	0.0E+00

Table C.1: Numerical results of KNITRO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|c(x_*)\|$
concon	270	271/270	0.05	-6.23079556E+03	1.6E-10
congigmz	29	30/27	0.00	2.80000000E+01	4.1E-08
core1	473	474/473	0.68	9.10562118E+01	7.0E-05
corkscrw	†1*3000*	*3001/3000*	*2313.89*	2.71576218E+01	1.2E+00
coshfun*	517	931/268	0.38	-7.98572820E-01	2.2E-09
cosine	†2*53*	*54/16*	*3.24*	-9.99900000E+03	0.0E+00
cragglvy	†2*53*	*54/18*	*2.75*	1.68821530E+03	0.0E+00
cresc100	‡*0*	*0/0*	*0.00*	--	--
cresc132	‡*0*	*0/0*	*0.00*	--	--
cresc4	†1*3000*	*3001/1744*	*0.57*	-3.50027981E-02	3.1E-02
cresc50	‡*0*	*0/0*	*0.00*	--	--
csfi1	31	39/24	0.01	-4.90751998E+01	2.1E-07
csfi2	53	54/53	0.01	5.50176057E+01	9.7E-08
cube	39	40/26	0.01	4.70358314E-29	0.0E+00
curly10	†2*63*	*64/14*	*1103.94*	-1.00316290E+06	0.0E+00
curly20	†2*58*	*59/14*	*2080.86*	-1.00316290E+06	0.0E+00
curly30	†2*55*	*56/13*	*2605.81*	-1.00316290E+06	0.0E+00
dallasl	440	441/440	64.93	-2.02604132E+05	9.5E-11
dallasm	‡*0*	*0/0*	*0.00*	--	--
dallass	‡*0*	*0/0*	*0.00*	--	--
deconvc*	105	121/91	0.49	1.01153772E-08	2.4E-14
demymalo	30	33/26	0.00	-2.99999996E+00	3.4E-16
denschna	6	7/6	0.00	1.10283709E-23	0.0E+00
denschnb	6	7/6	0.00	4.43734259E-31	0.0E+00
denschnc	12	13/11	0.01	9.75515256E-27	0.0E+00
denschnd	46	47/36	0.01	1.38597369E-11	0.0E+00
denschne	14	15/11	0.00	9.02606828E-24	0.0E+00
denschnf	6	7/6	0.01	6.51324621E-22	0.0E+00
dipigri	15	18/13	0.00	6.80630057E+02	1.1E-13
disc2	33	38/25	0.02	1.56250002E+00	1.2E-07
discs	†1*3000*	*3001/2872*	*11.09*	9.36339963E+00	8.0E-01
dittert	53	71/39	3.39	-1.99759674E+00	8.8E-10
dixchlng	9	10/9	0.00	2.47189781E+03	1.2E-11
dixchlnv	20	21/20	0.34	7.51226920E-20	8.8E-16
dixmaana	9	10/9	0.43	1.00000000E+00	0.0E+00
dixmaanb	9	10/9	0.72	1.00000000E+00	0.0E+00
dixmaanc	10	11/10	0.75	1.00000000E+00	0.0E+00
dixmaand	11	12/11	0.85	1.00000000E+00	0.0E+00
dixmaane	38	39/18	1.43	1.00000000E+00	0.0E+00
dixmaanf	35	36/17	1.97	1.00000000E+00	0.0E+00
dixmaang	35	36/22	2.18	1.00000000E+00	0.0E+00
dixmaanh	36	37/20	2.17	1.00000000E+00	0.0E+00
dixmaani	25	26/12	3.42	1.00000000E+00	0.0E+00
dixmaanj	73	74/34	9.11	1.00000000E+00	0.0E+00
dixmaank	66	67/28	10.98	1.00000000E+00	0.0E+00
dixmaanl	77	78/36	10.45	1.00000000E+00	0.0E+00
djtl	†2*147*	*148/41*	*0.02*	-8.95154472E+03	0.0E+00
dnieper	14	15/14	0.04	1.87440146E+04	6.5E-07
dqrtic	43	44/43	1.25	5.64156166E-09	0.0E+00
drcavty1	‡*0*	*0/0*	*0.00*	--	--
drcavty2	‡*0*	*0/0*	*0.00*	--	--
drcavty3	‡*0*	*0/0*	*0.00*	--	--
dtoc1l	10	12/9	5.99	1.25338129E+02	2.2E-16
dtoc1na	8	9/8	2.85	1.27020299E+01	3.4E-15
dtoc1nb	7	8/7	2.64	1.59377776E+01	1.6E-15
dtoc1nc	11	13/8	3.17	2.49698127E+01	3.3E-11
dtoc1nd*	24	35/15	2.60	1.26444577E+01	1.9E-11
dtoc2	6	7/6	4.69	5.08676209E-01	5.7E-11
dtoc4	6	7/6	2.68	2.86853821E+00	6.7E-12
dtoc5	6	7/6	1.36	1.53510891E+00	3.3E-09
dtoc6	13	14/13	2.86	1.34850616E+05	5.4E-09
edensch	10	11/10	0.33	1.20032845E+04	0.0E+00
eg1	15	16/15	0.00	-1.42930675E+00	2.2E-16

Table C.1: Numerical results of KNITRO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|\|c(x_*)\|\|$
eg2	3	4/3	0.05	-9.98947393E+02	0.0E+00
eg3*	41	44/34	0.19	3.26224219E-01	3.4E-10
eigena	70	90/51	0.78	1.00386417E-07	8.8E-16
eigena2	2	3/2	0.03	2.18908901E-29	0.0E+00
eigenaco	3	4/3	0.04	0.00000000E+00	0.0E+00
eigenals	34	35/21	0.46	1.89923224E-17	0.0E+00
eigenb	113	114/67	1.31	1.02348755E-17	0.0E+00
eigenb2*	2	3/2	0.03	1.80000000E+01	0.0E+00
eigenbco*	2	3/2	0.03	9.00000000E+00	0.0E+00
eigenbls	121	122/69	2.13	7.47068015E-17	0.0E+00
eigenc2	14	18/11	3.39	5.25734152E-19	4.4E-16
eigencco	13	17/8	0.02	3.82039292E-17	5.4E-12
engval1	†2 *56*	*57/16*	*1.54*	5.54866841E+03	0.0E+00
engval2	18	19/16	0.00	2.02540037E-28	0.0E+00
errinros*	93	94/58	0.02	3.99041539E+01	0.0E+00
expfit	12	13/8	0.00	2.40510593E-01	0.0E+00
expfita	36	38/35	0.02	1.13661997E-03	2.0E-14
expfitb	24	25/24	0.04	5.01940651E-03	8.4E-15
expfitc*	21	22/21	0.22	5.61320336E-02	3.3E-14
explin	61	65/58	0.21	-7.23756265E+05	7.9E-13
explin2	60	62/59	0.20	-7.24459142E+05	1.7E-15
expquad	23	24/23	0.03	-3.62459988E+06	1.7E-15
extrosnb	†3 *1*	*2/1*	*0.00*	0.00000000E+00	0.0E+00
fletcbv2	4	5/4	0.01	-5.14006786E-01	0.0E+00
fletcbv3	†1 *3000*	*3001/3000*	*296.01*	-5.61355412E+11	0.0E+00
fletchbv	†1 *3000*	*3001/3000*	*297.58*	-2.97872845E+19	0.0E+00
fletchcr	100	101/57	0.03	3.44461749E-18	0.0E+00
fletcher*	61	106/46	0.01	1.95253669E+01	1.7E-15
flosp2hh	†2 *448*	*450/448*	*157.04*	3.88730628E+01	0.0E+00
flosp2hl	†1 *3000*	*3001/3000*	*1901.85*	3.88705439E+01	0.0E+00
flosp2hm	†2 *2680*	*2688/2674*	*1049.07*	3.88712567E+01	0.0E+00
flosp2th	†1 *3000*	*3001/3000*	*1033.30*	1.00002724E+01	0.0E+00
flosp2tl	†1 *3000*	*3001/3000*	*1581.59*	1.00000000E+01	0.0E+00
flosp2tm	†1 *3000*	*3001/3000*	*1168.79*	1.00000006E+01	0.0E+00
fminsrf2	168	169/118	2.65	1.00000000E+00	0.0E+00
fminsurf	294	295/265	78.77	1.00000000E+00	0.0E+00
freuroth	†2 *52*	*53/14*	*2.47*	6.08159189E+05	0.0E+00
gausselm	690	706/666	4163.34	-1.63525883E+01	2.1E-07
genhumps	54	55/32	0.01	6.26529320E-18	0.0E+00
genrose	†2 *1112*	*1113/708*	*4.52*	1.00000000E+00	0.0E+00
gigomez1	31	32/27	0.01	-2.99999996E+00	0.0E+00
gilbert	34	35/30	0.56	4.82027299E+02	1.3E-13
gpp	19	20/19	17.24	1.44009271E+04	9.8E-09
growth	177	178/118	0.01	1.00404058E+00	0.0E+00
growthls	166	167/115	0.02	1.00404058E+00	0.0E+00
gulf	33	34/26	0.02	1.20555404E-25	0.0E+00
hadamals*	39	41/38	0.35	2.63083409E+01	6.6E-16
hadamard	3000	3001/164	73.35	NAN	NAN
hager2	5	6/5	1.47	4.32082250E-01	1.8E-12
hager4	18	19/18	17.80	2.79403653E+00	9.5E-12
haifam	216	414/142	1.16	-4.50003600E+01	2.4E-06
haifas	63	89/35	0.01	-4.49999969E-01	1.8E-11
hairy	†2 *62*	*63/27*	*0.00*	2.00000000E+01	0.0E+00
haldmads*	41	53/30	0.02	1.22383283E-04	2.7E-14
hanging	42	43/39	0.52	-6.20176038E+02	3.1E-11
hart6	17	20/15	0.01	-3.32288689E+00	1.1E-16
hatflda	45	54/37	0.00	1.20847640E-14	0.0E+00
hatfldb	18	19/18	0.00	5.57281104E-03	1.1E-16
hatfldc	12	13/12	0.00	4.91920547E-17	1.7E-15
hatfldd	30	31/24	0.00	6.61511391E-08	0.0E+00
hatfldе	39	40/30	0.01	4.43440235E-07	0.0E+00
heart6ls	1368	1369/944	0.15	6.42176864E-22	0.0E+00
heart8ls	1103	1104/774	0.20	6.52592151E-18	0.0E+00

Table C.1: Numerical results of KNITRO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|c(x_*)\|$
helix	17	18/13	0.00	1.53736012E-21	0.0E+00
himmelbb	13	14/8	0.00	7.66845976E-30	0.0E+00
himmelbf	†2 *47*	*48/15*	*0.01*	3.18571748E+02	0.0E+00
himmelbg	7	8/5	0.01	1.20187231E-18	0.0E+00
himmelbh	†2 *28*	*29/6*	*0.00*	-1.00000000E+00	0.0E+00
himmelbi	29	32/27	0.05	-1.75499999E+03	4.5E-13
himmelbj	†2 *156*	*157/29*	*0.05*	0.00000000E+00	4.4E+01
himmelbk	32	35/30	0.04	5.18143791E-02	1.9E-10
himmelp1	†2 *54*	*91/17*	*0.01*	-6.20538693E+01	3.5E-15
himmelp2	†2 *47*	*78/17*	*0.01*	-6.20538693E+01	3.5E-15
himmelp3	12	13/12	0.00	-5.90131229E+01	2.8E-14
himmelp4	13	14/13	0.00	-5.90131229E+01	7.3E-14
himmelp5	32	35/32	0.01	-5.90131234E+01	8.5E-14
himmelp6*	20	21/20	0.01	-5.90131235E+01	1.3E-12
hong	14	15/14	0.01	1.34730680E+00	0.0E+00
hs001	42	51/34	0.00	1.68191627E-17	0.0E+00
hs002	19	20/19	0.00	4.94122932E+00	0.0E+00
hs004	11	12/11	0.00	2.66666668E+00	1.0E-18
hs005	21	27/16	0.01	-1.91322295E+00	0.0E+00
hs006	12	14/9	0.00	1.97215226E-31	3.5E-15
hs007	8	9/7	0.00	-1.73205080E+00	2.6E-14
hs009	6	7/6	0.00	-5.00000000E-01	0.0E+00
hs010	18	19/18	0.00	-9.99999989E-01	1.8E-15
hs011	13	14/13	0.00	-8.49846417E+00	5.6E-16
hs012	14	15/14	0.01	-2.99999999E+01	2.0E-15
hs013*	25	26/25	0.00	1.00333615E+00	1.6E-27
hs014	15	16/15	0.00	1.39346499E+00	1.9E-16
hs015*	14	15/12	0.00	3.60379773E+02	6.8E-15
hs016	16	17/16	0.01	2.31446614E+01	1.1E-16
hs017	28	29/28	0.00	1.00000000E+00	1.1E-12
hs018	24	27/23	0.01	5.00000001E+00	2.8E-14
hs019	36	37/33	0.01	-6.96181355E+03	1.6E-12
hs020	14	15/14	0.01	4.01987323E+01	2.7E-16
hs023	14	15/14	0.00	2.00000002E+00	1.4E-14
hs024	15	16/15	0.00	-9.99999979E-01	4.3E-17
hs025*	38	43/34	0.03	1.42406726E-14	7.7E-16
hs026	22	23/22	0.00	1.52788759E-15	2.2E-08
hs027	23	35/16	0.01	4.00000000E-02	0.0E+00
hs029	15	16/15	0.00	-2.26274169E+01	0.0E+00
hs030	241	242/143	0.03	9.99999990E-01	1.0E-08
hs031	13	14/13	0.01	6.00000005E+00	2.0E-15
hs032	16	17/16	0.01	1.00000002E+00	7.2E-13
hs033	20	21/17	0.00	-4.58578628E+00	4.4E-16
hs034	24	26/24	0.00	-8.34032414E-01	1.7E-13
hs036	11	12/11	0.00	-3.29999999E+03	6.2E-16
hs037	11	12/11	0.00	-3.45599999E+03	1.9E-16
hs038	65	86/45	0.01	1.54031258E-18	1.7E-15
hs039	15	16/12	0.00	-1.00000000E+00	4.6E-14
hs040	3	4/3	0.00	-2.50000000E-01	5.4E-10
hs041	20	21/20	0.01	1.92592592E+00	2.2E-16
hs042	11	12/11	0.01	1.38578643E+01	0.0E+00
hs043	13	14/13	0.00	-4.39999998E+01	2.2E-14
hs045	20	21/20	0.00	1.00000005E+00	4.4E-16
hs046	22	23/22	0.00	7.91792371E-15	2.2E-08
hs047	20	22/18	0.00	1.21948212E-13	2.4E-09
hs049	20	21/20	0.00	7.00266269E-12	0.0E+00
hs050	9	10/9	0.00	4.97968446E-30	0.0E+00
hs054	15	16/15	0.00	1.92857142E-01	5.5E-17
hs056	24	25/20	0.00	-3.45600000E+00	2.2E-16
hs057*	12	13/12	0.00	3.06476190E-02	3.5E-15
hs059*	25	29/24	0.01	-6.74950527E+00	7.1E-15
hs060	11	12/11	0.00	3.25682002E-02	1.7E-15
hs061*	6	7/6	0.01	-1.43646142E+02	2.6E-09

Table C.1: Numerical results of KNITRO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|c(x_*)\|$
hs062	8	9/8	0.00	-2.62725144E+04	1.1E-16
hs063	12	13/12	0.00	9.61715172E+02	2.8E-14
hs064	22	23/22	0.01	6.29984242E+03	2.8E-14
hs065	53	75/36	0.01	9.53528867E-01	3.0E-14
hs066	24	25/24	0.00	5.18163294E-01	8.3E-14
hs067	10	12/10	0.01	-1.16202697E+03	4.5E-13
hs070	25	29/22	0.02	9.40197325E-03	1.4E-14
hs071	14	15/14	0.01	1.70140173E+01	1.0E-12
hs072	20	21/20	0.00	7.27679357E+02	4.3E-18
hs073	14	15/14	0.01	2.98943789E+01	2.4E-12
hs074	16	17/16	0.01	5.12649810E+03	4.3E-08
hs075	100	101/100	0.01	5.17441272E+03	1.1E-12
hs077	10	12/9	0.01	2.41505128E-01	2.1E-11
hs078	4	5/4	0.00	-2.91970040E+00	6.1E-11
hs079	6	7/6	0.00	7.87768208E-02	1.2E-11
hs080	13	14/13	0.00	5.39498477E-02	3.3E-14
hs081	13	14/13	0.01	5.39498477E-02	3.3E-14
hs083	8	9/8	0.00	-3.06655385E+04	7.8E-13
hs084	8	9/8	0.01	-5.28033511E+06	1.7E-07
hs085	30	54/22	0.02	-1.90515524E+00	1.8E-09
hs086	16	17/16	0.00	-3.23486779E+01	1.1E-15
hs087	183	184/183	0.04	8.82759837E+03	5.1E-07
hs088	106	111/68	0.16	1.36265682E+00	3.6E-15
hs089	†1*3000*	*3002/2834*	*7.13*	1.54557189E+00	4.8E-05
hs090	†1*3000*	*3002/2950*	*10.30*	1.56762475E+00	3.4E-06
hs091	†1*3000*	*3002/2965*	*14.73*	1.50414824E+00	4.4E-05
hs092	†1*3000*	*3002/2592*	*17.60*	1.57705578E+00	2.0E-05
hs093	8	9/8	0.00	1.35075965E+02	1.4E-13
hs095	14	15/14	0.00	1.56210611E-02	6.9E-10
hs096	14	15/14	0.00	1.56210611E-02	6.9E-10
hs097*	20	21/19	0.01	3.13581680E+00	2.0E-09
hs098*	21	22/20	0.01	3.13581680E+00	1.9E-09
hs099	12	13/12	0.00	-8.31079891E+08	4.3E-11
hs100	15	18/13	0.01	6.80630057E+02	1.1E-13
hs100lnp	9	12/7	0.00	6.80630057E+02	4.7E-09
hs100mod	16	20/14	0.00	6.78754727E+02	3.2E-12
hs101	211	219/118	0.06	1.80976508E+03	1.0E-09
hs102	249	258/135	0.07	9.11880667E+02	1.8E-10
hs103	284	287/146	0.08	5.43668090E+02	1.6E-09
hs104	36	37/27	0.01	3.95116348E+00	4.4E-15
hs105*	15	16/15	0.65	1.15139630E+03	1.0E-14
hs106	76	95/68	0.02	7.04924801E+03	3.2E-10
hs107	166	167/161	0.05	5.05501181E+03	3.2E-11
hs108*	39	45/31	0.02	-6.74981430E-01	1.3E-14
hs109	154	155/131	0.04	5.32685139E+03	8.7E-08
hs110	13	14/13	0.01	-4.57784697E+01	1.1E-16
hs111	18	19/12	0.01	-4.77610911E+01	1.2E-08
hs111lnp	12	15/9	0.00	-4.77610911E+01	1.3E-08
hs112	10	11/10	0.01	-4.77610908E+01	2.2E-16
hs113	14	15/14	0.01	2.43062106E+01	4.5E-12
hs114	19	21/19	0.01	-1.76880696E+03	9.7E-08
hs116	70	71/70	0.04	9.75874950E+01	2.8E-06
hs117	68	81/56	0.02	3.23486820E+01	1.7E-15
hs119	36	37/36	0.03	2.44899703E+02	2.2E-15
hs99exp	17	18/17	0.01	-1.00806250E+09	6.7E-09
hubfit	26	28/25	0.00	1.68934959E-02	0.0E+00
humps	343	344/205	0.02	4.58020978E-22	0.0E+00
hvycrash	†1*3000*	*3001/1499*	*11.82*	-4.41252558E-03	3.9E-04
hypcir	5	6/0	0.00	0.00000000E+00	1.6E-14
indef	†1*3000*	*3001/3000*	*15.04*	-5.82899423E+10	0.0E+00
jensmp	†2*32*	*33/12*	*0.00*	1.24362182E+02	0.0E+00
kissing	57	71/50	228.62	8.43331725E-01	8.4E-12
kiwcresc	22	24/19	0.01	2.04800033E-08	8.4E-16

Table C.1: Numerical results of KNITRO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|\|c(x_*)\|\|$
kowosb	14	15/9	0.01	3.07505603E-04	0.0E+00
lakes	†2*122*	*123/20*	*0.09*	7.34585760E+11	5.5E+02
launch	†2*1193*	*1194/626*	*0.86*	4.22100149E-01	7.5E+02
lch	1660	2253/1045	12.16	-4.31828879E+00	4.4E-16
liarwhd	14	15/14	2.15	4.84230273E-22	0.0E+00
lminsurf	†1*3000*	*3017/2984*	*1100.50*	1.80346838E+01	0.0E+00
loadbal	27	28/27	0.02	4.52851059E-01	1.7E-13
loghairy	1994	1995/1231	0.12	1.82321556E-01	0.0E+00
logros	67	89/46	0.01	2.22044604E-16	0.0E+00
lootsma	20	21/17	0.01	1.41421371E+00	4.4E-16
lsnnodoc	12	13/12	0.01	1.23112544E+02	2.2E-16
madsen	22	27/20	0.00	6.16432439E-01	2.1E-15
madsschj	46	53/36	3.40	-7.97283702E+02	2.7E-13
makela1	23	24/19	0.01	-1.41421355E+00	1.1E-16
makela2	16	17/16	0.00	7.20000002E+00	1.4E-15
makela3	21	22/20	0.01	4.09600067E-08	6.5E-18
mancino	†2*17*	*18/14*	*1.98*	7.62261070E-22	0.0E+00
manne	80	91/74	4.04	-9.74567847E-01	3.5E-07
maratos	3	4/3	0.00	-1.00000000E+00	1.5E-08
matrix2	34	37/29	0.00	2.42736463E-08	8.7E-18
maxlika*	15	16/15	0.63	1.15139630E+03	1.9E-14
mccormck	†1*3000*	*4508/2959*	*4886.03*	-4.56616135E+04	2.2E-17
mdhole	34	42/27	0.00	2.04800033E-09	7.6E-19
methanb8	734	735/734	1.31	1.59790569E-12	0.0E+00
methanl8	827	828/815	1.47	9.35712775E-12	0.0E+00
mexhat	4	5/4	0.00	-4.00999999E-02	0.0E+00
meyer3	†3*555*	*556/384*	*0.06*	8.79458551E+01	0.0E+00
mifflin1	14	15/14	0.00	-9.99999995E-01	4.0E-17
mifflin2	29	33/25	0.01	-9.99999979E-01	2.0E-14
minc44	43	46/40	3.12	2.57302913E-03	3.1E-11
minmaxbd	84	85/81	0.02	1.15706440E+02	1.8E-13
minmaxrb	34	35/26	0.01	3.96511806E-08	5.5E-09
minperm	130	147/73	439.55	3.62879999E-04	1.4E-10
minsurf	6	7/6	0.01	1.00000000E+00	0.0E+00
mistake	32	36/26	0.01	-9.99999959E-01	6.8E-10
morebv	1	2/1	31.96	9.91978649E-12	0.0E+00
msqrtals	55	56/26	559.78	6.47575225E-18	0.0E+00
msqrtbls	49	50/25	293.81	3.56275386E-17	0.0E+00
mwright	7	8/7	0.00	2.49788095E+01	8.8E-16
ngone*	53	70/53	3.78	-6.37636664E-01	3.5E-11
noncvxu2	469	470/384	4.78	2.31798346E+03	0.0E+00
noncvxun	†2*61*	*62/19*	*0.23*	2.31680841E+03	0.0E+00
nondia	7	8/7	0.98	4.53733530E-20	0.0E+00
nondquar	1670	1671/839	329.68	2.83124015E-08	0.0E+00
nonmsqrt	†2*221*	*222/138*	*0.02*	7.51825511E-01	0.0E+00
nonscomp	733	959/508	679.30	3.55253097E-06	1.4E-14
odfits	14	15/14	0.01	-2.38002677E+03	0.0E+00
oet2	108	177/79	31.38	8.71596399E-02	5.2E-16
oet7*	2728	3677/2166	145.96	2.06974434E-03	9.4E-10
optcdeg2	40	41/39	1.79	2.29573436E+02	2.1E-07
optcdeg3	35	36/35	1.76	4.61456762E+01	5.1E-08
optcntrl	41	42/41	0.02	5.49999951E+02	1.7E-07
optctrl3	35	36/22	0.04	2.04801654E+03	1.2E-15
optctrl6	35	36/22	0.04	2.04801654E+03	1.2E-15
optmass	23	24/23	0.01	-1.89542387E-01	2.1E-10
optprloc	83	95/60	0.06	-1.64197735E+01	6.7E-13
orthrdm2	6	7/6	1.12	1.55532815E+02	7.0E-10
orthrds2	†2*57*	*65/26*	*0.16*	3.05400434E+01	4.8E-13
orthrega*	7	8/7	0.16	1.66480113E+03	1.0E-06
orthregb	3	4/3	0.01	4.05749338E-17	1.1E-07
orthregc	21	27/12	7.97	1.89594597E+02	3.5E-09
orthregd	†2*68*	*69/37*	*19.62*	2.28221561E+03	1.1E-12
orthrege*	95	169/51	0.04	1.23940973E+00	1.4E-08

Table C.1: Numerical results of KNITRO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|c(x_*)\|$
orthrgdm	9	10/8	4.38	1.51380232E+03	1.5E-09
orthrgds	†2 *68*	*69/35*	*19.28*	2.86015358E+03	2.2E-13
osbornea	72	73/47	0.01	5.46489469E-05	0.0E+00
osborneb*	22	23/14	0.03	4.01377362E-02	0.0E+00
oslbqp	17	18/17	0.01	6.25000007E+00	8.8E-16
palmer1	†1 *3000*	*3949/2052*	*0.55*	1.17546025E+04	1.1E-16
palmer1a	†2 *146*	*216/79*	*0.03*	8.98836290E-02	0.0E+00
palmer1b	55	81/30	0.01	3.44735461E+00	0.0E+00
palmer1e	110	111/73	0.02	8.35268268E-04	0.0E+00
palmer2	†2 *186*	*303/71*	*0.02*	3.65108950E+03	0.0E+00
palmer2a	197	260/135	0.02	1.71607394E-02	0.0E+00
palmer2b	36	44/29	0.01	6.23394652E-01	5.5E-17
palmer2e	305	306/163	0.04	2.15352481E-04	0.0E+00
palmer3	†2 *142*	*234/53*	*0.02*	2.26595821E+03	0.0E+00
palmer3a	110	144/77	0.02	2.04314229E-02	0.0E+00
palmer3b	†2 *76*	*126/30*	*0.01*	4.22764725E+00	0.0E+00
palmer3e	299	300/165	0.05	5.07408418E-05	0.0E+00
palmer4	†2 *126*	*223/31*	*0.02*	2.28538322E+03	0.0E+00
palmer4a	130	174/87	0.02	4.06061393E-02	0.0E+00
palmer4b	†2 *58*	*91/27*	*0.01*	6.83513859E+00	0.0E+00
palmer4e	98	99/55	0.02	1.48004219E-04	0.0E+00
palmer5a	†1 *3000*	*3866/2135*	*0.47*	5.45306561E-02	6.6E-16
palmer5b	†2 *1574*	*2120/1030*	*0.27*	9.75249263E-03	0.0E+00
palmer5e	†1 *3000*	*3656/2345*	*0.46*	2.32243176E-02	0.0E+00
palmer6a	225	314/138	0.03	5.59488389E-02	0.0E+00
palmer6e	90	132/49	0.02	2.23955034E-04	0.0E+00
palmer7a	†1 *3000*	*3817/2184*	*0.40*	1.03764705E+01	9.0E-13
palmer7e	1577	2062/1093	0.25	1.01538986E+01	0.0E+00
palmer8a	100	130/71	0.02	7.40096979E-02	2.7E-17
palmer8e	46	65/28	0.00	6.33930743E-03	0.0E+00
penalty1	49	50/46	8.83	9.68617543E-03	0.0E+00
penalty2	†2 *47*	*48/20*	*0.10*	9.70960839E+04	0.0E+00
pentagon	39	50/29	0.00	1.36529870E-04	4.4E-16
pfit1ls*	631	632/427	0.05	1.94977489E-18	0.0E+00
pfit2ls	‡ *0*	*0/0*	*0.00*	--	--
pfit3ls*	269	270/182	0.03	2.77371292E-16	0.0E+00
pfit4ls	‡ *0*	*0/0*	*0.00*	--	--
polak1	17	18/17	0.00	2.71828184E+00	0.0E+00
polak2	30	31/21	0.01	5.45981505E+01	9.7E-11
polak3	77	86/50	0.03	5.93300337E+00	4.7E-11
polak4	1482	1629/1317	0.15	3.07200057E-08	2.2E-11
polak5	17	18/17	0.00	5.00000006E+01	1.5E-07
polak6	26	30/24	0.00	-4.39999998E+01	2.3E-13
power	10	11/10	0.52	4.35721690E-23	0.0E+00
probpenl	42	44/41	3.16	-4.23825956E+03	1.7E-15
prodpl0	22	23/22	0.02	6.09192385E+01	4.5E-13
prodpl1	22	23/22	0.02	5.30370171E+01	5.3E-14
pspdoc	24	29/20	0.00	2.41421356E+00	5.4E-17
qr3d	72	95/50	1.48	1.79675293E-15	1.1E-16
qr3dbd	53	65/42	0.68	1.69695510E-15	1.7E-15
qr3dls	74	96/53	1.74	1.82782455E-15	0.0E+00
qrtquad	42	53/32	0.04	-3.64808836E+06	9.1E-13
quartc	45	46/45	2.68	1.59648928E-08	0.0E+00
reading1	54	74/47	34.95	-1.60461950E-01	3.8E-08
reading3	49	50/26	0.28	-6.51524329E-09	9.0E-09
rk23	34	41/22	0.01	8.33333845E-02	2.1E-15
robot*	7	8/6	0.00	6.59329888E+00	4.4E-16
rosenbr	26	27/21	0.00	5.37138236E-21	0.0E+00
s365mod*	31	32/25	0.01	1.24238670E+02	1.5E-15
s368*	8	9/8	1.17	0.00000000E+00	5.5E-17
sawpath	12	14/11	39.99	1.81573788E+02	2.4E-08
scon1dls*	1965	3208/723	176.85	2.47938744E+02	1.1E-13
scosine	†1 *3000*	*3001/2812*	*1128.20*	2.64446347E+03	0.0E+00

Table C.1: Numerical results of KNITRO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|c(x_*)\|$
scurly10	‡*0*	*0/0*	*0.00*	--	--
scurly20	‡*0*	*0/0*	*0.00*	--	--
scurly30	‡*0*	*0/0*	*0.00*	--	--
sineali	$^{\dagger 1}$*3000*	*4050/1952*	*1.49*	-1.90096186E+03	0.0E+00
sineval	65	66/43	0.01	1.49827998E-19	0.0E+00
sinquad	319	320/208	46.66	3.78173688E-09	0.0E+00
sinrosnb	$^{\dagger 2}$*41*	*75/7*	*0.82*	-9.99010000E+04	8.8E-16
sisser	19	20/18	0.01	6.05207195E-13	0.0E+00
smbank	$^{\dagger 2}$*67*	*92/25*	*0.12*	-7.12929200E+06	5.8E-11
smmpsf	102	103/102	1.94	1.04710853E+06	4.8E-09
snake	$^{\dagger 1}$*3000*	*3001/2733*	*0.28*	-1.10982802E+09	2.8E+07
spanhyd	27	29/26	0.12	2.39738000E+02	3.7E-11
spiral	184	228/93	0.02	4.09600590E-09	0.0E+00
sreadin3	3	4/3	2.72	-3.07837695E-05	2.4E-09
srosenbr	27	28/23	2.06	1.48166963E-18	0.0E+00
ssebnln*	92	93/92	0.49	1.81906025E+07	7.0E-07
ssnlbeam*	11	12/11	0.01	3.37772472E+02	5.7E-13
stancmin	$^{\dagger 2}$*155*	*156/43*	*0.02*	3.61591084E+15	4.0E+00
steenbrb*	189	192/187	1.91	9.08618550E+03	7.2E-12
steenbrc*	218	265/172	1.85	2.02986138E+04	1.8E-12
steenbrd*	188	192/185	2.24	9.14991947E+03	1.8E-12
steenbre	302	310/295	4.10	2.75741831E+04	5.8E-11
steenbrf	79	87/72	0.61	2.82679952E+02	4.2E-09
steenbrg*	322	325/320	4.81	2.74712821E+04	1.8E-12
svanberg	17	18/17	1578.72	8.36142279E+03	4.0E-07
swopf	19	20/19	0.04	6.78603374E-02	1.2E-08
synthes1	14	15/14	0.01	7.59284648E-01	2.3E-15
trainf*	2159	2160/2159	2695.49	3.05848842E+00	1.8E-06
trainh	828	839/818	2016.07	1.23122591E+01	3.2E-09
trimloss	52	57/52	0.25	9.06000110E+00	2.9E-13
try	18	19/18	0.00	2.62144111E-17	2.2E-16
twirism1	1242	1251/802	192.29	-1.00513312E+00	4.5E-08
twobars	16	17/16	0.01	1.50865242E+00	4.4E-16
ubh5	$^{\dagger 1}$*3000*	*3001/2380*	*1645.49*	-1.13709069E+03	7.4E+01
vardim	26	27/26	0.02	3.73421817E-25	0.0E+00
watson	14	15/14	0.03	1.46704094E-11	0.0E+00
weeds*	3000	3002/5	0.40	NAN	NAN
womflet*	21	24/18	0.01	1.53600274E-07	7.6E-14
woods	64	65/41	5.34	0.00000000E+00	0.0E+00
yfit	80	105/57	0.02	6.84611152E-13	0.0E+00
yfitu	75	76/55	0.01	6.66972055E-13	0.0E+00
zecevic3	40	42/32	0.01	9.73094503E+01	6.8E-15
zecevic4	17	18/17	0.01	7.55750782E+00	2.6E-15
zigzag	46	50/40	0.04	3.16173497E+00	2.4E-09
zy2	12	14/12	0.02	2.00000015E+00	8.3E-14

Table C.1: Numerical results of KNITRO on CUTE test set

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|c(x_*)\|$
bearing_200	‡*0*	*0/0*	*0.00*	--	--
bearing_400	‡*0*	*0/0*	*0.00*	--	--
camshape_10000	‡*0*	*0/0*	*0.00*	--	--
camshape_20000	‡*0*	*0/0*	*0.00*	--	--
catmix_10000	19	20/19	28.60	-4.80191388E-02	6.6E-08
catmix_20000	19	20/19	55.70	-4.79857325E-02	4.7E-08
chain_20000	8	9/8	53.17	5.06848022E+00	6.3E-08
chain_40000	9	15/8	142.47	5.06848041E+00	3.5E-07
channel_5000	54	55/0	74.24	1.00000000E+00	1.1E-07
channel_10000	132	133/0	360.23	1.00000000E+00	2.8E-10
elec_200	70	127/36	32.59	1.84390521E+04	2.2E-16
elec_400	84	144/41	155.68	7.55832250E+04	3.3E-16
gasoil_2500	396	598/216	3224.25	5.23659583E-03	4.0E-11
gasoil_5000	‡*0*	*0/0*	*0.00*	--	--
glider_2500	†[1]*3000*	*4061/1905*	*6137.43*	-8.86411736E+02	1.5E-03
glider_5000	‡*0*	*0/0*	*0.00*	--	--
marine_1000	110	111/110	277.29	1.97465298E+07	3.3E-04
marine_2000	126	127/126	1205.35	1.97465299E+07	1.0E-03
methanol_5000	58	80/37	4759.15	9.02229189E-03	2.6E-11
methanol_10000	‡*0*	*0/0*	*0.00*	--	--
minsurf_200_200	†[1]*3000*	*3021/2980*	*5509.93*	1.70155430E+02	7.1E-15
minsurf_300_300	‡*0*	*0/0*	*0.00*	--	--
pinene_2500	6	7/6	327.40	1.98668203E+01	7.7E-07
pinene_5000	6	7/6	1482.12	1.98528294E+01	1.6E-06
polygon_200*	81	125/61	503.18	-7.32206184E-01	1.1E-07
polygon_400	‡*0*	*0/0*	*0.00*	--	--
robot_5000	16	17/16	241.62	9.16351253E+00	1.3E-06
robot_10000*	16	17/16	1140.79	9.41186007E+00	1.8E-06
rocket_10000	14	15/14	2136.08	-1.01265487E+00	1.3E-06
rocket_20000	‡*0*	*0/0*	*0.00*	--	--
steering_10000	14	15/14	140.40	5.54528353E-01	1.5E-05
steering_20000	13	14/13	760.55	5.54552974E-01	3.0E-06
torsion_200_200	‡*0*	*0/0*	*0.00*	--	--
torsion_400_400	‡*0*	*0/0*	*0.00*	--	--

Table C.2: Numerical results of KNITRO on COPS test set

Problem	#iter	$\#f/\#H$	CPU [s]	$f(x_*)$	$\|c(x_*)\|$
cont5_1	15	16/15	621.69	2.72155548E+00	6.6E-07
cont5_2_1	35	36/35	1546.03	6.55077316E-04	2.9E-09
cont5_2_2	50	55/50	2272.22	5.13450384E-04	2.8E-09
cont5_2_3	44	48/44	1921.60	5.15371385E-04	3.4E-07
cont5_2_4	†2 *30*	*31/30*	*3819.01*	6.63737777E-02	7.9E-15
cont_p	†2 *33*	*34/33*	*231.98*	2.31634290E+00	3.8E-14
ex1_80	237	238/237	318.36	6.10514304E-02	4.3E-11
ex1_160	121	122/119	2307.02	6.38725410E-02	2.0E-11
ex2_80	239	240/239	314.11	5.53569602E-02	9.0E-12
ex2_160	123	124/121	2328.35	5.82116045E-02	8.2E-12
ex3_80	21	22/21	40.09	1.10274800E-01	2.1E-11
ex3_160	21	22/21	482.25	1.10299153E-01	1.0E-11
ex4_80	17	18/17	42.85	7.78904832E-02	2.3E-11
ex4_160	15	16/15	341.73	7.83387208E-02	1.8E-10
ex4_2_80	12	13/12	71.48	3.66448742E+00	3.9E-06
ex4_2_160	13	14/13	1002.72	3.64744904E+00	8.7E-09
ex5_80	22	23/22	78.45	5.43568976E-02	3.3E-11
ex5_160	26	27/26	681.93	5.47639658E-02	1.3E-11
ex6_80	30	31/30	63.46	-4.25501474E+00	1.3E-10
ex6_160	32	33/32	673.40	-4.30666733E+00	3.9E-10
lukvle1	12	13/11	37.58	6.23245863E+00	1.0E-10
lukvle2	50	53/25	227.15	1.40922828E+06	8.2E-14
lukvle3	13	14/13	12.18	6.51214956E+01	8.8E-16
lukvle4	15	16/15	38.22	2.42907675E+05	4.8E-10
lukvle5	50	86/26	97.00	2.63928370E+00	6.5E-11
lukvle6	18	21/17	54.21	3.14422608E+06	1.2E-09
lukvle7	23	30/15	208.55	-6.61396154E+04	2.9E-14
lukvle8*	9	10/9	20.77	1.11165281E+07	2.2E-12
lukvle9	†2 *253*	*276/109*	*102.67*	0.00000000E+00	5.5E+22
lukvle10	9	10/9	19.78	1.76772385E+04	4.0E-08
lukvle11	10	11/10	16.91	2.90485500E-18	1.1E-15
lukvle12	8	9/8	10.94	7.72038795E+04	1.6E-07
lukvle13	29	40/19	30.84	4.01784151E+05	2.8E-14
lukvle14	22	23/22	29.24	3.80424848E+05	3.5E-15
lukvle15	42	51/24	36.83	3.04096065E-17	6.3E-13
lukvle16	22	31/12	21.15	3.57420915E-21	8.8E-16
lukvle17	17	18/12	15.82	7.14331493E+04	2.7E-08
lukvle18	13	14/11	13.66	5.99820079E+04	2.5E-09
lukvli1	‡ *0*	*0/0*	*0.00*	--	--
lukvli2	18	19/18	330.11	1.32665589E+06	7.1E-08
lukvli3	17	18/17	17.47	1.15775418E+01	0.0E+00
lukvli4	62	66/57	341.84	2.01020610E+04	5.4E-12
lukvli5*	61	66/47	233.26	3.70046480E-01	8.8E-13
lukvli6	18	19/18	84.73	3.14422616E+06	1.1E-11
lukvli7	31	48/15	201.98	-1.86338502E+04	1.3E-14
lukvli8	‡ *0*	*0/0*	*0.00*	--	--
lukvli9	258	472/156	159.96	4.99466708E+03	1.1E-08
lukvli10	101	123/78	179.40	1.76772536E+04	0.0E+00
lukvli11	103	136/86	559.81	1.70074270E-05	2.7E-07
lukvli12	193	252/130	1097.57	1.10330055E-08	6.3E-14
lukvli13	162	164/123	350.26	8.61187282E-06	6.2E-08
lukvli14	23	24/23	105.02	3.80424857E+05	2.0E-15
lukvli15*	45	46/45	177.72	3.84624825E-05	2.3E-08
lukvli16	23	24/23	113.31	7.43557685E-05	0.0E+00
lukvli17*	80	96/65	763.10	6.60456844E+00	5.3E-15
lukvli18	22	23/22	62.95	5.61599407E-05	2.3E-16

Table C.3: Numerical results of KNITRO on MITT test set

Code	Explanation
‡	The CPU time limit of $10,800$ CPU seconds was exceeded.
†1	More than $3,000$ iterations were taken.
†2	Halting because trust region radius $< 10 \cdot$ `macheps`.
†3	The size of the search direction $\|d_k\|$ is less than `macheps`.

Table C.4: Error codes for KNITRO

Appendix D

Results for LOQO

The tables in this appendix document the runs of LOQO on the test sets CUTE, COPS, and MITT for the comparison in Section 5.1.3. For each problem, they list the following numbers: number of iterations (#iter) and error code (as defined in Table D.4), if failed; number of evaluations of objective function ($\#f$) and constraints ($\#c$); required CPU time in seconds; final value of the objective function ($f(x_*)$) and *scaled* constraint violation ($\|\tilde{c}(x_*)\|$). A problem name is marked with a star (*), if IPOPT or KNITRO successfully terminated at a point with a different value of the objective function (see Eq. (5.3)).

Problem	#iter	#f/#c	CPU [s]	$f(x_*)$	$\|\tilde{c}(x_*)\|$
airport	20	39/39	0.19	4.79527016E+04	3.7E-11
aljazzaf	44	116/116	0.00	7.50049999E+01	1.3E-11
allinit	111	1616/1616	0.02	1.67059684E+01	2.4E-18
allinitc	367	7295/7295	0.11	3.04965495E+01	1.4E-19
allinitu	10	19/19	0.01	5.74438491E+00	3.1E-16
alsotame	11	21/21	0.00	8.20850009E-02	3.7E-10
arwhead	14	29/29	1.75	2.67928434E-09	3.1E-14
avion2	82	165/165	0.06	9.46801296E+07	7.2E-15
bard	17	34/34	0.00	8.21487730E-03	8.9E-16
batch	77	155/155	0.04	2.59180350E+05	2.0E-10
bdexp	33	65/65	3.39	4.54950613E-10	5.1E-14
bdqrtic	13	25/25	0.60	3.98381795E+03	1.0E-14
beale	10	19/19	0.00	6.47095106E-17	2.8E-16
bigbank	36	71/71	1.37	-4.20569614E+06	1.5E-10
biggs3	14	27/27	0.00	1.44901252E-17	4.6E-12
biggs5	36	80/80	0.01	1.75006359E-17	5.6E-16
biggs6	45	118/118	0.01	1.64098432E-18	2.0E-15
box2	†1*3000*	*6008/6008*	*0.93*	nan	nan
box3	11	21/21	0.00	2.61661740E-15	1.8E-15
brainpc0	†1*0*	*6012/6012*	*1388.61*	0.00000000E+00	0.0E+00
brainpc1	†4*1802*	*15359/15359*	*1068.26*	4.07106300E-04	8.2E-01
brainpc2	†4*1467*	*11943/11943*	*2232.99*	4.11429700E-04	1.7E-07
brainpc3	†2*574*	*2708/2708*	*287.08*	3.65877200E-04	4.1E-07
brainpc4	†1*3000*	*32657/32657*	*1952.88*	6.86410700E-04	2.7E-05
brainpc5	342	2344/2344	191.75	3.62667977E-04	1.0E-07
brainpc6	†1*3000*	*29199/29199*	*1927.99*	3.47901600E-04	4.7E-04
brainpc7	226	1020/1020	122.77	3.71789858E-04	2.2E-13
brainpc8	588	4390/4390	341.29	4.03303234E-04	6.2E-13
brainpc9	†4*532*	*2600/2600*	*337.66*	3.51483100E-04	1.6E-06
bratu1d	†2*55*	*190/190*	*2.72*	-2.66546400+166	2.1E-12
britgas	15	29/29	0.29	2.31376540E-08	3.2E-09
brkmcc	9	17/17	0.00	1.69042679E-01	5.0E-16
brownal	12	23/23	0.00	1.90597985E-19	8.4E-16
brownbs	35	72/72	0.00	6.10199884E-25	2.8E-10
broydn7d*	51	105/105	1.74	3.57687536E+02	1.7E-14
brybnd	15	29/29	4.82	6.17677961E-24	3.9E-15
bt1	18	36/36	0.00	-1.00000000E+00	4.1E-11
bt2	18	35/35	0.00	3.25682009E-02	1.7E-09
bt4	11	21/21	0.00	-4.55105507E+01	2.2E-10
bt5	9	17/17	0.00	9.61715171E+02	5.6E-09
bt6	12	24/24	0.00	2.77044783E-01	1.6E-09
bt7*	19	37/37	0.00	3.60379767E+02	5.5E-10
bt8	328	3884/3884	0.05	1.00000000E+00	3.4E-11
bt9	15	29/29	0.00	-1.00000000E+00	1.6E-09
bt11	12	23/23	0.00	8.24891764E-01	3.2E-09
bt12	13	25/25	0.01	6.18811881E+00	9.4E-11
bt13	39	79/79	0.01	7.67710802E-34	3.5E-07
byrdsphr	12	23/23	0.00	-4.68330020E+00	1.3E-08
camel6	10	19/19	0.01	-1.03162845E+00	2.4E-11
cantilvr	16	31/31	0.01	1.33995635E+00	3.6E-09
catena	27	53/53	0.02	-2.30777462E+04	1.1E-09
catenary	39	77/77	0.23	-3.48403157E+05	3.9E-10
cb2	11	21/21	0.00	1.95222443E+00	1.7E-08
cb3	11	21/21	0.00	1.99999999E+00	1.9E-09
chaconn1	11	21/21	0.00	1.95222446E+00	7.1E-09
chaconn2	11	21/21	0.00	2.00000002E+00	2.9E-09
chebyqad	67	151/151	60.09	5.38631531E-03	7.0E-16
chnrosnb	50	109/109	0.02	4.05599097E-20	3.6E-15
cliff	33	65/65	0.00	1.99786613E-01	8.9E-16
clnlbeam*	104	207/207	3.93	3.44876218E+02	1.4E-10
clplatea	9	17/17	1.97	-1.25920948E-02	1.3E-14
clplateb	12	24/24	2.51	-6.98822201E+00	1.8E-14

Table D.1: Numerical results of LOQO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\|\tilde{c}(x_*)\|\|$
clplatec	9	17/17	1.95	-5.02072422E-03	1.3E-14
concon	†4 *333*	*873/873*	*0.06*	-6.23079600E+03	2.2E-03
congigmz	33	65/65	0.00	2.80000008E+01	4.1E-08
core1	56	111/111	0.04	9.10562401E+01	5.4E-11
corkscrw	37	73/73	8.62	9.06878240E+01	9.0E-11
coshfun*	21	44/44	0.01	-7.73266589E-01	2.0E-07
cosine	13	25/25	2.91	-9.99900000E+03	4.4E-14
cragglvy	17	33/33	2.59	1.68821530E+03	1.8E-14
cresc100	239	1300/1300	1.66	5.67602709E-01	9.8E-11
cresc132	†1 *3000*	*6008/6008*	*313.02*	6.07729500E-01	9.9E-03
cresc4	56	111/111	0.01	8.71897585E-01	1.3E-09
cresc50	†1 *3000*	*6027/6027*	*4.63*	-5.25518200E-06	1.3E-01
csfi1	16	31/31	0.00	-4.90752002E+01	4.8E-10
csfi2	17	33/33	0.00	5.50176053E+01	4.4E-10
cube	34	75/75	0.01	7.32928436E-19	9.9E-16
curly10	18	35/35	8.19	-1.00316290E+06	3.5E-14
curly20	18	35/35	14.38	-1.00316290E+06	4.5E-14
curly30	18	35/35	21.31	-1.00316290E+06	4.0E-14
dallasl	58	115/115	1.21	-2.02604132E+05	4.9E-15
dallasm	66	180/180	0.25	-4.81981888E+04	2.7E-13
dallass	42	86/86	0.03	-3.23932257E+04	1.7E-14
deconvc*	32	63/63	0.11	2.56947712E-03	1.5E-11
demymalo	15	29/29	0.00	-2.99999999E+00	2.4E-09
denschna	9	17/17	0.00	9.08811806E-14	0.0E+00
denschnb	10	19/19	0.00	1.22133871E-17	1.7E-16
denschnc	16	31/31	0.00	1.91291888E-19	4.4E-16
denschnd	37	74/74	0.00	2.81743430E-10	7.1E-15
denschne	14	29/29	0.00	4.30711719E-14	3.2E-15
denschnf	12	23/23	0.00	6.02653181E-20	8.7E-16
dipigri	11	22/22	0.00	6.80630059E+02	9.7E-09
disc2	29	57/57	0.02	1.56249999E+00	3.5E-10
discs	†4 *1531*	*3780/3780*	*2.28*	4.96642500E+02	1.7E+02
dittert	179	357/357	9.89	-1.99759674E+00	5.7E-15
dixchlng	27	53/53	0.00	2.47189778E+03	2.1E-09
dixchlnv	†2 *271*	*595/595*	*3.95*	2.24419500E-25	1.1E-18
dixmaana	12	23/23	0.85	1.00000000E+00	2.0E-14
dixmaanb	13	25/25	1.54	1.00000000E+00	3.1E-14
dixmaanc	14	27/27	1.67	1.00000000E+00	2.5E-14
dixmaand	15	29/29	1.78	1.00000000E+00	2.8E-14
dixmaane	18	35/35	1.35	1.00000000E+00	5.1E-14
dixmaanf	18	35/35	2.32	1.00000000E+00	4.0E-14
dixmaang	20	39/39	2.54	1.00000000E+00	3.2E-14
dixmaanh	22	48/48	2.90	1.00000000E+00	3.1E-14
dixmaani	18	35/35	1.34	1.00000000E+00	7.1E-14
dixmaanj	22	43/43	2.80	1.00000000E+00	4.7E-14
dixmaank	24	47/47	2.97	1.00000000E+00	4.4E-14
dixmaanl	24	47/47	2.99	1.00000000E+00	5.4E-14
djtl	21	86/86	0.01	-8.95154472E+03	8.9E-15
dnieper	28	55/55	0.04	1.87440143E+04	8.9E-09
dqrtic	64	127/127	3.48	1.65455828E-17	3.3E-11
drcavty1	‡ *0*	*0/0*	*0.00*	--	--
drcavty2	‡ *0*	*0/0*	*0.00*	--	--
drcavty3	318	1171/1171	10350.23	2.02792795E-04	2.0E-14
dtoc1l	11	21/21	5.97	1.25338129E+02	3.7E-09
dtoc1na	11	21/21	3.36	1.27020299E+01	8.3E-10
dtoc1nb	10	19/19	3.17	1.59377776E+01	5.1E-09
dtoc1nc	16	47/47	5.47	2.49698130E+01	2.7E-07
dtoc1nd*	23	60/60	3.65	1.26444578E+01	4.1E-09
dtoc2	15	29/29	3.13	5.08676205E-01	4.1E-10
dtoc4	18	35/35	6.13	2.86853820E+00	6.4E-11
dtoc5	17	33/33	3.36	1.53511152E+00	5.0E-11
dtoc6	23	45/45	5.31	1.34850615E+05	7.6E-10
edensch	11	21/21	0.59	1.20032845E+04	2.0E-14

Table D.1: Numerical results of LOQO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\|\tilde{c}(x_*)\|\|$
eg1	10	19/19	0.01	-1.42930674E+00	3.9E-09
eg2	8	15/15	0.13	-9.98947393E+02	0.0E+00
eg3*	26	51/51	0.07	7.37420944E-18	3.2E-09
eigena	24	47/47	0.32	2.29623646E-09	5.1E-14
eigena2	15	29/29	0.11	4.29150330E-20	5.7E-12
eigenaco	21	41/41	0.31	1.64819236E-21	2.2E-13
eigenals	34	67/67	1.04	5.44183632E-21	3.9E-15
eigenb	84	192/192	1.22	3.03076683E-20	2.5E-15
eigenb2*	52	106/106	0.63	5.59222616E-02	5.5E-10
eigenbco*	68	145/145	1.19	7.51668111E-21	1.9E-12
eigenbls	83	191/191	2.08	1.17583598E-19	2.6E-15
eigenc2	48	108/108	23.87	1.53871767E-18	7.3E-11
eigencco	18	40/40	0.01	1.86991123E-18	1.3E-10
engval1	15	50/50	1.86	5.54866841E+03	3.4E-15
engval2	22	45/45	0.00	2.28050304E-21	1.2E-15
errinros*	50	125/125	0.03	3.99041539E+01	1.5E-14
expfit	11	23/23	0.00	2.40510593E-01	3.1E-16
expfita	23	45/45	0.01	1.13661243E-03	2.4E-15
expfitb	33	65/65	0.03	5.01936574E-03	8.3E-14
expfitc*	45	89/89	0.27	2.33025731E-02	9.3E-13
explin	23	45/45	0.01	-7.23756265E+05	2.1E-08
explin2	24	47/47	0.01	-7.24459142E+05	3.1E-07
expquad	32	63/63	0.03	-3.62459988E+06	3.1E-09
extrosnb	9	17/17	0.00	6.86422864E-20	1.4E-15
fletcbv2	9	17/17	0.01	-5.14006786E-01	2.2E-15
fletcbv3	†1*3000*	*6001/6001*	*1113.19*	nan	nan
fletchbv	†1*3000*	*6001/6001*	*1160.68*	nan	nan
fletchcr	51	102/102	0.05	1.14506820E-20	3.5E-15
fletcher*	14	27/27	0.00	1.95253663E+01	2.2E-09
flosp2hh	†1*3000*	*6231/6231*	*1107.95*	nan	nan
flosp2hl	10	19/19	0.50	3.88705439E+01	1.5E-08
flosp2hm	10	19/19	0.52	3.88712559E+01	1.5E-08
flosp2th	11	21/21	0.56	1.00000000E+01	7.0E-10
flosp2tl	10	19/19	0.49	1.00000000E+01	1.5E-08
flosp2tm	10	19/19	0.53	1.00000000E+01	1.5E-08
fminsrf2	†1*3000*	*10316/10316*	*153.94*	1.57998200E+07	4.1E-03
fminsurf	55	192/192	159.10	1.00000000E+00	4.5E-13
freuroth	14	28/28	2.36	6.08159189E+05	2.9E-15
gausselm	†1*3000*	*6048/6048*	*456.21*	-1.47918300E+00	1.4E-03
genhumps	108	227/227	0.01	2.13037341E-12	2.8E-13
genrose	†1*3000*	*6035/6035*	*59.43*	nan	nan
gigomez1	17	33/33	0.00	-3.00000000E+00	5.9E-10
gilbert	35	69/69	153.81	4.82027297E+02	6.7E-08
gpp	19	37/37	3.79	1.44009275E+04	6.9E-09
growth	80	192/192	0.01	1.00404058E+00	3.2E-13
growthls	78	176/176	0.01	1.00404058E+00	5.7E-14
gulf	27	57/57	0.03	1.08838278E-16	1.3E-14
hadamals	†4*431*	*861/861*	*5.38*	1.83059300E+02	1.7E+00
hadamard	15	29/29	0.06	1.00000000E+00	2.0E-13
hager2	†1*3000*	*6320/6320*	*1538.57*	nan	nan
hager4	†4*1546*	*3319/3319*	*504.29*	2.80065400E+00	2.8E+01
haifam	†1*3000*	*35148/35148*	*35.47*	-4.50003600E+01	1.2E-05
haifas	12	23/23	0.00	-4.50000079E-01	1.7E-07
hairy	53	130/130	0.01	2.00000000E+01	2.5E-15
haldmads*	26	56/56	0.01	3.26427063E-02	5.3E-10
hanging	16	31/31	0.11	-6.20176031E+02	6.6E-08
hart6	12	23/23	0.00	-3.32288689E+00	1.8E-16
hatflda	8	15/15	0.00	1.62036095E-15	9.9E-09
hatfldb	11	21/21	0.00	5.57280966E-03	1.3E-16
hatfldc	9	17/17	0.00	2.56709634E-21	1.3E-11
hatfldd	25	50/50	0.00	6.61511391E-08	7.1E-16
hatfldе	30	60/60	0.01	4.43440070E-07	3.3E-16
heart6ls	†1*3000*	*6760/6760*	*2.04*	nan	nan

Table D.1: Numerical results of LOQO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\|\tilde{c}(x_*)\|\|$
heart8ls	55	157/157	0.02	2.29787047E-19	5.2E-16
helix	14	27/27	0.00	3.97130470E-20	2.2E-16
himmelbb	13	29/29	0.00	1.72373395E-19	5.6E-16
himmelbf	23	47/47	0.00	3.18571748E+02	9.2E-13
himmelbg	8	16/16	0.00	1.80160247E-15	2.2E-16
himmelbh	10	19/19	0.00	-1.00000000E+00	2.2E-16
himmelbi	26	51/51	0.02	-1.75499999E+03	3.6E-14
himmelbj	575	5387/5387	0.71	-1.91034472E+03	5.3E-14
himmelbk	21	43/43	0.02	5.18143470E-02	5.3E-08
himmelp1	20	39/39	0.01	-6.20538693E+01	3.6E-13
himmelp2*	17	33/33	0.00	-6.20538693E+01	7.1E-12
himmelp3	18	35/35	0.00	-5.90131234E+01	4.9E-11
himmelp4	32	75/75	0.01	-5.90131234E+01	1.6E-11
himmelp5	55	155/155	0.01	-5.90131235E+01	7.7E-11
himmelp6*	33	97/97	0.01	-8.19803172E+00	5.8E-12
hong	20	39/39	0.00	1.34730661E+00	4.2E-14
hs001	32	64/64	0.00	6.14828293E-19	4.0E-16
hs002	22	43/43	0.00	4.94122931E+00	3.7E-14
hs004	10	19/19	0.00	2.66666672E+00	1.9E-09
hs005	10	19/19	0.00	-1.91322295E+00	4.1E-16
hs006	11	21/21	0.00	2.71727114E-20	6.1E-09
hs007	14	27/27	0.00	-1.73205081E+00	1.3E-09
hs009	10	19/19	0.00	-4.99999999E-01	4.9E-09
hs010	15	29/29	0.00	-1.00000000E+00	3.0E-09
hs011	13	25/25	0.00	-8.49846423E+00	1.2E-09
hs012	10	19/19	0.00	-2.99999976E+01	5.2E-08
hs013	†1*3000*	*35741/35741*	*0.23*	1.01960900E+00	9.3E-07
hs014	11	21/21	0.00	1.39346495E+00	4.5E-09
hs015*	34	68/68	0.00	3.06500000E+02	4.1E-09
hs016	17	33/33	0.00	2.31446609E+01	2.9E-10
hs017	33	65/65	0.00	1.00000001E+00	5.1E-11
hs018	16	31/31	0.00	5.00000001E+00	1.8E-09
hs019	27	54/54	0.01	-6.96181389E+03	2.1E-10
hs020	22	43/43	0.01	4.01987301E+01	2.0E-09
hs023	18	35/35	0.00	2.00000000E+00	1.3E-10
hs024	15	29/29	0.00	-9.99999998E-01	2.5E-16
hs025*	16	31/31	0.02	3.28350000E+01	7.4E-12
hs026	15	29/29	0.00	2.49476766E-10	9.5E-07
hs027	17	33/33	0.01	3.99999999E-02	4.2E-10
hs029	10	19/19	0.00	-2.26274169E+01	5.2E-09
hs030	10	19/19	0.00	1.00000000E+00	3.6E-11
hs031	9	17/17	0.00	5.99999999E+00	3.2E-10
hs032	23	46/46	0.00	1.00000002E+00	8.1E-13
hs033	11	21/21	0.00	-4.58578640E+00	1.5E-08
hs034	14	27/27	0.00	-8.34032404E-01	1.4E-09
hs036	16	35/35	0.00	-3.29999999E+03	6.7E-17
hs037	11	21/21	0.00	-3.45599998E+03	2.4E-16
hs038	44	99/99	0.00	3.32350040E-18	1.8E-15
hs039	15	29/29	0.00	-1.00000000E+00	1.6E-09
hs040	9	17/17	0.01	-2.50000004E-01	3.9E-09
hs041	16	31/31	0.01	1.92592592E+00	1.6E-10
hs042	9	17/17	0.01	1.38578643E+01	7.8E-09
hs043	11	21/21	0.01	-4.40000057E+01	2.1E-07
hs045	23	45/45	0.00	1.00000000E+00	7.3E-16
hs046	22	44/44	0.00	5.24551316E-15	3.0E-08
hs047	21	44/44	0.01	3.70273175E-11	1.6E-08
hs049	24	47/47	0.00	1.21310590E-11	5.5E-16
hs050	16	31/31	0.00	7.65690604E-15	1.1E-14
hs054	12	23/23	0.00	1.92857142E-01	1.3E-11
hs056	13	25/25	0.00	-3.45600000E+00	4.5E-10
hs057*	24	47/47	0.01	2.84596697E-02	8.4E-12
hs059*	22	47/47	0.01	-7.80278944E+00	5.8E-10
hs060	9	17/17	0.00	3.25682006E-02	1.1E-09

Table D.1: Numerical results of LOQO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\|\tilde{c}(x_*)\|\|$
hs061*	11	21/21	0.00	-1.43646142E+02	2.4E-08
hs062	13	25/25	0.00	-2.62725144E+04	1.0E-09
hs063	9	17/17	0.00	9.61715171E+02	1.0E-08
hs064	27	53/53	0.00	6.29984242E+03	1.4E-09
hs065	16	31/31	0.00	9.53528855E-01	2.4E-10
hs066	15	29/29	0.00	5.18163272E-01	1.0E-08
hs067	26	51/51	0.02	-1.16202698E+03	6.1E-13
hs070	23	47/47	0.02	9.40197325E-03	2.5E-09
hs071	13	25/25	0.00	1.70140172E+01	8.6E-10
hs072	26	51/51	0.01	7.27679349E+02	1.5E-15
hs073	19	39/39	0.00	2.98943781E+01	2.1E-10
hs074	16	31/31	0.00	5.12649809E+03	1.8E-09
hs075	18	35/35	0.00	5.17441268E+03	9.2E-10
hs077	13	25/25	0.00	2.41505128E-01	4.6E-11
hs078	9	17/17	0.00	-2.91970041E+00	6.4E-10
hs079	9	17/17	0.00	7.87768214E-02	9.0E-10
hs080	9	17/17	0.00	5.39498450E-02	2.3E-09
hs081	16	31/31	0.00	5.39498464E-02	1.2E-09
hs083	13	25/25	0.00	-3.06655394E+04	1.9E-08
hs084	21	64/64	0.00	-5.28033512E+06	2.1E-11
hs085	29	58/58	0.03	-1.90515525E+00	1.1E-12
hs086	14	27/27	0.00	-3.23486789E+01	7.2E-14
hs087	24	47/47	0.00	8.82759772E+03	6.9E-09
hs088	28	55/55	0.08	1.36265681E+00	3.5E-13
hs089	28	58/58	0.09	1.36265681E+00	2.8E-12
hs090	29	58/58	0.15	1.36265681E+00	1.3E-12
hs091	29	57/57	0.19	1.36265681E+00	2.0E-13
hs092	23	45/45	0.21	1.36265681E+00	3.1E-12
hs093	12	23/23	0.00	1.35075962E+02	3.8E-09
hs095	16	32/32	0.00	1.56195261E-02	2.4E-13
hs096	19	40/40	0.00	1.56195279E-02	1.0E-10
hs097*	18	36/36	0.00	4.07124635E+00	2.4E-10
hs098*	45	137/137	0.01	3.13580914E+00	6.5E-11
hs099	21	41/41	0.02	-8.31079891E+08	2.1E-09
hs100	11	22/22	0.01	6.80630059E+02	9.7E-09
hs100lnp	12	23/23	0.01	6.80630057E+02	2.2E-11
hs100mod	15	29/29	0.00	6.78754727E+02	2.8E-10
hs101	37	100/100	0.01	1.80976475E+03	2.1E-09
hs102	55	208/208	0.02	9.11880565E+02	1.0E-09
hs103	48	136/136	0.02	5.43667958E+02	5.3E-11
hs104	14	27/27	0.00	3.95116344E+00	9.6E-11
hs105*	22	43/43	0.92	1.13636098E+03	2.5E-15
hs106	24	47/47	0.00	7.04924804E+03	1.8E-09
hs107	47	93/93	0.01	5.05501179E+03	3.9E-10
hs108*	21	41/41	0.00	-8.66025433E-01	2.3E-08
hs109	33	91/91	0.01	5.32685133E+03	4.9E-10
hs110	9	17/17	0.01	-4.57784697E+01	5.8E-13
hs111	15	30/30	0.00	-4.77610911E+01	6.8E-09
hs111lnp	19	37/37	0.01	-4.77610909E+01	3.0E-09
hs112	17	35/35	0.01	-4.77610908E+01	1.7E-10
hs113	16	31/31	0.00	2.43062098E+01	1.6E-09
hs114	24	53/53	0.00	-1.76880696E+03	1.3E-08
hs116	31	63/63	0.00	9.75875095E+01	7.7E-13
hs117	19	37/37	0.00	3.23486789E+01	2.1E-10
hs119	29	57/57	0.01	2.44899697E+02	2.8E-15
hs99exp	556	1122/1122	0.27	-1.00806249E+09	4.5E-12
hubfit	12	23/23	0.01	1.68934998E-02	1.7E-11
humps	228	502/502	0.02	2.76489068E-12	3.2E-13
hvycrash	†1 *3000*	*25503/25503*	*32.57*	-1.70430000E-01	9.3E-05
hypcir	9	17/17	0.01	0.00000000E+00	2.7E-10
indef	†4 *299*	*602/602*	*3.40*	-3.15744400E+15	8.1E+05
jensmp	14	27/27	0.00	1.24362182E+02	3.1E-16
kissing	†1 *3000*	*6001/6001*	*124.65*	9.18478500E-01	4.3E-02

Table D.1: Numerical results of LOQO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\|\tilde{c}(x_*)\|\|$
kiwcresc	14	27/27	0.00	-1.94430919E-09	6.6E-09
kowosb	11	22/22	0.00	3.07505603E-04	5.9E-16
lakes	280	559/559	0.28	3.50524793E+05	7.3E-14
launch	†1 *3000*	*23820/23820*	*4.31*	1.56743500E+01	2.6E-03
lch	26	51/51	15.17	-4.31828879E+00	3.5E-10
liarwhd	22	43/43	6.13	7.97139002E-23	1.8E-13
lminsurf	†1 *3000*	*15516/15516*	*4486.86*	9.99924600E+06	2.2E+01
loadbal	23	45/45	0.01	4.52851040E-01	1.3E-16
loghairy	†1 *3000*	*6001/6001*	*0.38*	nan	nan
logros	294	740/740	0.02	0.00000000E+00	0.0E+00
lootsma	12	23/23	0.00	1.41421356E+00	7.2E-10
lsnnodoc	21	41/41	0.00	1.23112449E+02	3.2E-13
madsen	24	48/48	0.01	6.16432379E-01	9.2E-08
madsschj	24	47/47	0.49	-7.97283704E+02	2.1E-09
makela1	14	28/28	0.00	-1.41421356E+00	7.6E-09
makela2	12	23/23	0.00	7.19999993E+00	4.0E-09
makela3	18	35/35	0.00	1.64774205E-09	2.5E-08
mancino	19	37/37	2.80	2.48962260E-21	3.7E-13
manne	†1 *3000*	*6003/6003*	*50.59*	-9.74548300E-01	1.0E-06
maratos	8	15/15	0.00	-1.00000000E+00	9.2E-09
matrix2	26	51/51	0.00	9.34717613E-09	2.5E-09
maxlika*	21	41/41	0.89	1.13636098E+03	9.8E-16
mccormck	10	19/19	9.96	-4.56616135E+04	1.4E-14
mdhole	†2 *48*	*104/104*	*0.00*	1.84517800E-01	1.1E+09
methanb8	93	317/317	0.17	1.43326372E-20	3.3E-11
methanl8	160	604/604	0.34	3.78072374E-20	5.8E-11
mexhat	8	15/15	0.00	-4.01000000E-02	0.0E+00
meyer3	†3 *287*	*644/644*	*0.05*	8.79458600E+01	3.2E+05
mifflin1	9	17/17	0.00	-9.99999964E-01	4.0E-09
mifflin2	13	25/25	0.00	-1.00000001E+00	1.3E-08
minc44	†1 *3000*	*6001/6001*	*786.40*	nan	nan
minmaxbd	31	61/61	0.01	1.15706439E+02	4.4E-10
minmaxrb	13	25/25	0.01	1.45439470E-08	1.1E-07
minperm	23	45/45	51.23	3.62879997E-04	9.9E-12
minsurf	11	21/21	0.01	1.00000000E+00	2.2E-10
mistake	15	29/29	0.00	-1.00000001E+00	9.9E-09
morebv	8	15/15	1.35	9.29711973E-12	2.5E-09
msqrtals	33	81/81	449.22	2.69194843E-18	7.0E-15
msqrtbls	28	60/60	341.84	8.56377377E-21	6.6E-15
mwright	12	23/23	0.01	2.49788094E+01	1.0E-09
ngone	†1 *3000*	*17629/17629*	*163.40*	-6.36936500E-01	1.9E-16
noncvxu2	277	668/668	210.86	2.31741140E+03	5.2E-11
noncvxun	47	95/95	0.65	2.31680841E+03	3.3E-11
nondia	13	25/25	3.48	2.46347567E-22	0.0E+00
nondquar	24	47/47	5.01	7.17668965E-11	4.0E-14
nonmsqrt	†1 *3000*	*6106/6106*	*1.36*	nan	nan
nonscomp	33	67/67	6.70	2.40942120E-09	1.3E-11
odfits	15	29/29	0.00	-2.38002677E+03	3.1E-09
oet2	179	357/357	2.41	8.71596338E-02	2.1E-16
oet7	†3 *460*	*1146/1146*	*18.68*	4.44559800E-05	3.2E+05
optcdeg2	72	143/143	1.18	2.29573418E+02	5.7E-09
optcdeg3	53	105/105	0.89	4.61456692E+01	2.6E-10
optcntrl	48	126/126	0.02	5.49999974E+02	5.5E-09
optctrl3	36	131/131	0.12	2.04801654E+03	9.8E-09
optctrl6	36	131/131	0.11	2.04801654E+03	9.8E-09
optmass	15	29/29	0.01	-1.89542498E-01	5.6E-08
optprloc	23	45/45	0.01	-1.64197737E+01	2.7E-11
orthrdm2	†1 *3000*	*64787/64787*	*2598.23*	1.05387800E+04	6.4E-12
orthrds2	†1 *3000*	*56509/56509*	*47.09*	1.16206200E+03	1.7E-13
orthrega*	61	149/149	1.50	1.41405588E+03	9.0E-12
orthregb	10	19/19	0.01	4.50318011E-20	1.1E-09
orthregc	16	36/36	20.97	1.89609222E+02	6.5E-09
orthregd	†1 *3000*	*64932/64932*	*6524.44*	4.24577900E+04	5.6E-12

Table D.1: Numerical results of LOQO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\tilde{c}(x_*)\|$
orthrege	†1 *3000*	*6111/6111*	*7.58*	nan	nan
orthrgdm	†1 *3000*	*64478/64478*	*6374.12*	3.11824600E+04	3.7E-12
orthrgds	†1 *3000*	*64931/64931*	*6274.11*	2.59299700E+04	4.7E-12
osbornea	†1 *3000*	*6943/6943*	*3.87*	nan	nan
osborneb*	28	58/58	0.03	8.75947240E-02	3.1E-15
oslbqp	27	53/53	0.00	6.25000003E+00	2.0E-11
palmer1	21	41/41	0.01	1.17546025E+04	1.7E-15
palmer1a	46	93/93	0.01	8.98836290E-02	6.5E-14
palmer1b	28	79/79	0.00	3.44735461E+00	1.4E-14
palmer1e	85	187/187	0.02	8.35268268E-04	3.4E-14
palmer2	12	24/24	0.00	3.65108950E+03	5.3E-15
palmer2a	85	192/192	0.01	1.71607394E-02	1.2E-14
palmer2b	26	52/52	0.00	6.23394652E-01	1.4E-14
palmer2e	78	170/170	0.01	2.15352481E-04	2.7E-15
palmer3	15	32/32	0.01	2.26595821E+03	3.6E-15
palmer3a	73	153/153	0.02	2.04314229E-02	4.8E-15
palmer3b	19	37/37	0.00	4.22764725E+00	2.2E-15
palmer3e	106	247/247	0.03	5.07408418E-05	1.6E-14
palmer4*	16	35/35	0.01	2.28538322E+03	5.6E-17
palmer4a	70	151/151	0.01	4.06061393E-02	2.1E-14
palmer4b	18	35/35	0.01	6.83513859E+00	2.9E-15
palmer4e	43	87/87	0.01	1.48004219E-04	1.4E-14
palmer5a	†4 *40*	*79/79*	*0.00*	6.64342200E+02	2.4E+00
palmer5b	†4 *27*	*53/53*	*0.00*	2.34272800E+04	6.8E+02
palmer5e	†4 *294*	*653/653*	*0.05*	2.69984600E-02	6.3E+05
palmer6a	139	293/293	0.02	5.59488389E-02	3.6E-12
palmer6e	60	128/128	0.01	2.23955033E-04	1.1E-14
palmer7a	†1 *3000*	*6112/6112*	*1.41*	nan	nan
palmer7e	289	1080/1080	0.07	1.01538986E+01	1.7E-08
palmer8a	74	147/147	0.01	7.40096979E-02	5.5E-12
palmer8e	27	53/53	0.00	6.33930743E-03	7.7E-15
penalty1	56	113/113	179.59	9.68617543E-03	3.7E-11
penalty2	22	43/43	0.13	9.70960839E+04	2.7E-15
pentagon	28	55/55	0.01	1.36521683E-04	1.6E-13
pfit1ls*	196	408/408	0.02	2.19912367E-20	6.0E-16
pfit2ls*	73	150/150	0.02	1.97310021E-20	6.4E-16
pfit3ls*	109	243/243	0.02	1.52436769E-20	5.1E-16
pfit4ls*	208	477/477	0.02	1.44548780E-20	9.2E-16
polak1	14	27/27	0.01	2.71828188E+00	1.7E-09
polak2	24	47/47	0.00	5.45981490E+01	1.7E-08
polak3	22	43/43	0.01	5.93300349E+00	1.3E-07
polak4	13	25/25	0.00	2.28583287E-13	2.0E-15
polak5	37	103/103	0.00	5.00000000E+01	3.3E-08
polak6	27	53/53	0.00	-4.40000002E+01	5.8E-09
power	13	25/25	0.13	7.25022095E-20	8.8E-15
probpenl	†1 *3000*	*6006/6006*	*3024.81*	-9.56770800E+24	5.4E+00
prodpl0	23	46/46	0.01	6.09192365E+01	1.2E-13
prodpl1	20	40/40	0.01	5.30370157E+01	4.4E-12
pspdoc	12	23/23	0.00	2.41421357E+00	1.5E-10
qr3d	47	118/118	1.36	6.08118330E-20	5.3E-15
qr3dbd	31	67/67	0.47	1.14107098E-19	5.0E-15
qr3dls	48	118/118	1.62	3.58297110E-19	5.8E-15
qrtquad	21	41/41	0.03	-3.64753029E+06	1.7E-08
quartc	69	137/137	8.86	2.40605658E-18	9.3E-11
reading1	†2 *392*	*797/797*	*151.24*	-1.60447700E-01	1.2E-07
reading3	49	100/100	0.31	-6.74327290E-14	6.9E-11
rk23	13	25/25	0.01	8.33333344E-02	3.0E-10
robot*	21	41/41	0.01	6.59329888E+00	2.6E-11
rosenbr	26	52/52	0.00	4.62370031E-20	8.0E-16
s365mod*	22	46/46	0.00	5.21399044E+01	1.0E-08
s368*	10	19/19	1.96	0.00000000E+00	0.0E+00
sawpath	122	491/491	2.33	1.81572992E+02	4.1E-10
scon1dls*	287	1365/1365	11.73	5.97380915E-21	2.1E-12

Table D.1: Numerical results of LOQO on CUTE test set (*continued on next page*)

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\|\tilde{c}(x_*)\|\|$
scosine	80	159/159	19.19	-9.99899999E+03	1.9E-14
scurly10	103	236/236	58.26	-1.00316290E+06	2.3E-14
scurly20	94	219/219	86.14	-1.00316290E+06	2.1E-14
scurly30	88	198/198	117.05	-1.00316290E+06	2.2E-14
sineali	†1 *3000*	*6097/6097*	*1.99*	nan	nan
sineval	47	103/103	0.00	4.98691229E-18	8.9E-16
sinquad	53	129/129	25.90	4.45192520E-12	2.2E-14
sinrosnb	6	11/11	0.20	-9.99010000E+04	8.2E-11
sisser	16	31/31	0.00	8.70623343E-10	2.2E-16
smbank	27	53/53	0.03	-7.12929200E+06	4.7E-10
smmpsf	†2 *285*	*3166/3166*	*2.88*	1.04698500E+06	3.4E-13
snake	2033	23398/23398	0.22	-2.62457999E-08	1.7E-09
spanhyd	104	585/585	0.37	2.39738000E+02	3.4E-16
spiral	†4 *205*	*420/420*	*0.01*	1.24602400E+02	4.4E+09
sreadin3	25	51/51	8.86	-7.30798006E-05	2.6E-12
srosenbr	27	53/53	4.33	1.77349539E-21	4.7E-14
ssebnln*	602	1203/1203	0.93	1.61706000E+07	1.4E-13
ssnlbeam*	63	125/125	0.04	3.37772470E+02	4.2E-10
stancmin	17	33/33	0.00	4.25000008E+00	3.6E-17
steenbrb	†4 *167*	*364/364*	*24.58*	9.92219300E+03	9.9E-01
steenbrc	†1 *3000*	*33485/33485*	*23.29*	-7.72809100E+08	8.7E-01
steenbrd	†2 *394*	*863/863*	*30.32*	9.03008200E+03	1.5E-17
steenbre	†2 *445*	*969/969*	*61.64*	2.75679300E+04	9.6E-16
steenbrf	†1 *3000*	*33072/33072*	*21.58*	-4.95250200E+05	8.4E-01
steenbrg	†2 *576*	*1550/1550*	*53.52*	2.74209300E+04	2.8E-16
svanberg	20	39/39	7.59	8.36142280E+03	1.5E-08
swopf	19	38/38	0.03	6.78601826E-02	1.3E-08
synthes1	17	33/33	0.00	7.59284395E-01	9.1E-12
trainf*	77	153/153	35.91	3.10338433E+00	2.2E-11
trainh	†4 *854*	*4170/4170*	*774.94*	1.29593600E+01	2.7E+05
trimloss	50	110/110	0.06	9.06000001E+00	3.3E-07
try	18	35/35	0.00	3.64260966E-22	9.0E-11
twirism1	†2 *2666*	*11906/11906*	*217.64*	-1.00636300E+00	1.6E-16
twobars	10	19/19	0.00	1.50865241E+00	1.6E-09
ubh5	†1 *3000*	*6001/6001*	*5807.64*	nan	nan
vardim	37	77/77	0.13	3.77693010E-21	2.0E-15
watson	18	35/35	0.05	9.15855152E-13	1.3E-15
weeds*	34	74/74	0.01	2.58727739E+00	1.0E-14
womflet*	11	21/21	0.00	6.04999993E+00	1.9E-08
woods	48	106/106	9.62	4.47371053E-22	1.5E-13
yfit	41	84/84	0.01	6.66972074E-13	2.2E-16
yfitu	42	86/86	0.01	6.66972049E-13	8.9E-15
zecevic3	13	25/25	0.00	9.73094501E+01	2.4E-10
zecevic4	15	29/29	0.00	7.55750784E+00	1.2E-09
zigzag	30	59/59	0.02	3.16173497E+00	1.1E-09
zy2	13	26/26	0.00	2.00000000E+00	1.3E-10

Table D.1: Numerical results of LOQO on CUTE test set

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\|\tilde{c}(x_*)\|\|$
bearing_200	23	45/45	54.87	-1.54828675E-01	1.2E-15
bearing_400	24	47/47	508.12	-1.54818368E-01	2.2E-15
camshape_10000	†1 *3000*	*6001/6001*	*1316.86*	-4.92666800E+00	2.8E-04
camshape_20000	†1 *3000*	*6005/6005*	*2676.32*	-5.14288900E+00	1.2E-06
catmix_10000	26	51/51	29.17	-4.80556771E-02	7.2E-09
catmix_20000	27	53/53	58.00	-4.80556797E-02	3.7E-09
chain_20000	†5 *9999*	*0/0*	*351.57*	0.00000000E+00	0.0E+00
chain_40000	†5 *9999*	*0/0*	*60.96*	0.00000000E+00	0.0E+00
channel_5000	64	229/229	153.10	1.00000000E+00	3.7E-12
channel_10000	64	231/231	306.50	1.00000000E+00	4.7E-11
elec_200	‡ *0*	*0/0*	*0.00*	--	--
elec_400	‡ *0*	*0/0*	*0.00*	--	--
gasoil_2500	145	793/793	298.21	5.23659583E-03	2.4E-16
gasoil_5000	264	1499/1499	1187.84	5.23659583E-03	2.3E-16
glider_2500	†1 *3000*	*26214/26214*	*5825.77*	-1.37220000E+02	7.7E-01
glider_5000	‡ *0*	*0/0*	*0.00*	--	--
marine_1000	93	191/191	298.68	1.97465297E+07	1.2E-10
marine_2000	†4 *511*	*1167/1167*	*2326.11*	1.97465300E+07	6.9E+04
methanol_5000	‡ *0*	*0/0*	*0.00*	--	--
methanol_10000	†4 *122*	*501/501*	*5441.45*	9.03481500E-03	1.5E+05
minsurf_200_200	‡ *0*	*0/0*	*0.00*	--	--
minsurf_300_300	‡ *0*	*0/0*	*0.00*	--	--
pinene_2500	20	39/39	199.60	1.98721669E+01	4.7E-13
pinene_5000	25	49/49	924.52	1.98721669E+01	4.4E-13
polygon_200	†1 *3000*	*20503/20503*	*4826.21*	-7.59593700E-01	1.3E-06
polygon_400	‡ *0*	*0/0*	*0.00*	--	--
robot_5000	‡ *0*	*0/0*	*0.00*	--	--
robot_10000	‡ *0*	*0/0*	*0.00*	--	--
rocket_10000	52	135/135	178.07	-1.01283691E+00	5.9E-10
rocket_20000	56	143/143	565.03	-1.01283691E+00	3.0E-10
steering_10000	‡ *0*	*0/0*	*0.00*	--	--
steering_20000	‡ *0*	*0/0*	*0.00*	--	--
torsion_200_200	21	41/41	57.73	-4.18468661E-01	1.0E-14
torsion_400_400	22	43/43	493.90	-4.18488299E-01	2.0E-14

Table D.2: Numerical results of LOQO on COPS test set

Problem	#iter	$\#f/\#c$	CPU [s]	$f(x_*)$	$\|\|\tilde{c}(x_*)\|\|$
cont5_1	‡*0*	*0/0*	*0.00*	--	--
cont5_2_1	‡*0*	*0/0*	*0.00*	--	--
cont5_2_2	‡*0*	*0/0*	*0.00*	--	--
cont5_2_3	‡*0*	*0/0*	*0.00*	--	--
cont5_2_4	†4*50*	*109/109*	*1350.36*	6.63747200E-02	4.2E+04
cont_p	†4*180*	*359/359*	*343.93*	2.31635300E+00	8.1E+03
ex1_80	31	61/61	19.07	6.10491161E-02	2.6E-12
ex1_160	35	69/69	164.86	6.38548278E-02	1.3E-12
ex2_80	35	69/69	20.18	5.53434247E-02	1.2E-12
ex2_160	43	85/85	195.99	5.81291703E-02	1.3E-13
ex3_80	34	67/67	21.02	1.10258843E-01	1.1E-12
ex3_160	40	79/79	189.30	1.10267560E-01	1.4E-12
ex4_80	31	61/61	21.38	7.78894879E-02	1.0E-11
ex4_160	24	47/47	122.94	7.83339986E-02	2.9E-11
ex4_2_80	†2*431*	*2838/2838*	*842.51*	3.66771200E+00	9.5E-14
ex4_2_160	†2*310*	*728/728*	*5868.31*	3.64606500E+00	2.5E-13
ex5_80	†1*3000*	*29439/29439*	*2772.34*	5.43315800E-02	5.3E-13
ex5_160	†4*311*	*893/893*	*1610.23*	5.47023400E-02	1.6E-01
ex6_80	30	59/59	38.47	-4.25502418E+00	3.7E-11
ex6_160	33	65/65	457.87	-4.30672759E+00	5.3E-12
lukvle1	23	45/45	68.44	6.23245863E+00	2.8E-15
lukvle2	28	55/55	108.80	1.40923924E+06	1.3E-08
lukvle3	17	33/33	16.56	6.51214956E+01	1.7E-13
lukvle4	24	47/47	59.27	2.42907675E+05	8.9E-10
lukvle5	26	52/52	86.36	2.63928370E+00	4.6E-13
lukvle6	58	115/115	205.00	3.14422608E+06	1.6E-11
lukvle7	18	35/35	19.00	-6.61396154E+04	1.4E-07
lukvle8	‡*0*	*0/0*	*0.00*	--	--
lukvle9	†4*776*	*7334/7334*	*1459.91*	7.85801400E+10	1.5E+05
lukvle10	18	35/35	35.43	1.76772386E+04	8.6E-10
lukvle11	14	27/27	21.13	4.29515309E-21	1.2E-13
lukvle12	19	37/37	26.58	7.72038784E+04	3.3E-09
lukvle13	15	30/30	20.17	4.01792565E+05	7.3E-10
lukvle14	26	51/51	34.85	3.80424848E+05	2.5E-13
lukvle15	33	76/76	48.38	5.20548219E-23	5.9E-14
lukvle16	18	35/35	22.93	2.61088237E-22	6.9E-14
lukvle17	15	29/29	19.27	7.14331509E+04	1.9E-07
lukvle18	13	25/25	16.80	5.99820083E+04	5.1E-08
lukvli1	†4*136*	*279/279*	*361.89*	4.94283900E+02	1.2E+06
lukvli2	26	51/51	98.59	1.32665589E+06	5.6E-09
lukvli3	15	29/29	14.68	1.15775416E+01	3.5E-13
lukvli4	36	71/71	81.61	2.01020546E+04	3.4E-09
lukvli5	†3*116*	*248/248*	*327.56*	5.26762400E-01	2.7E+05
lukvli6	48	95/95	167.55	3.14422608E+06	3.3E-11
lukvli7	23	45/45	25.09	-1.86338502E+04	1.7E-07
lukvli8	‡*0*	*0/0*	*0.00*	--	--
lukvli9	33	65/65	22.33	4.99466708E+03	5.4E-07
lukvli10	28	55/55	53.99	1.76772387E+04	6.1E-10
lukvli11	45	89/89	65.29	9.06047748E-13	3.2E-08
lukvli12	23	45/45	30.55	4.32556328E-09	6.7E-07
lukvli13	34	67/67	40.60	7.79697552E-09	2.2E-08
lukvli14	32	63/63	39.52	3.80424848E+05	5.0E-09
lukvli15*	52	103/103	62.30	5.87188731E+00	5.2E-12
lukvli16	45	89/89	52.42	4.02122909E-09	1.6E-07
lukvli17*	77	153/153	112.34	1.94252005E+00	8.6E-10
lukvli18	35	69/69	41.05	1.91432678E-08	1.1E-07

Table D.3: Numerical results of LOQO on MITT test set

Code	Explanation
‡	The CPU time limit of $10,800$ CPU seconds was exceeded.
†1	More than $3,000$ iterations were taken.
†2	The dual problem is reported to be infeasible.
†3	The primal problem is reported to be infeasible.
†4	The primal or the dual problem is reported to be infeasible.
†5	More than 2 GB of memory was allocated.

Table D.4: Error codes for LOQO